Michael Hobbs
Andrzej M. Goscinski
Wanlei Zhou (Eds.)

LNCS 3719

Distributed and Parallel Computing

6th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP Melbourne, Australia, October 2005, Proceedings

Springer

Lecture Notes in Computer Science 3719

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Michael Hobbs   Andrzej M. Goscinski
Wanlei Zhou (Eds.)

Distributed and Parallel Computing

6th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP
Melbourne, Australia, October 2-3, 2005
Proceedings

Springer

Volume Editors

Michael Hobbs
Andrzej M. Goscinski
Wanlei Zhou
Deakin University
School of Information Technology
Geelong, Victoria 3217
Australia
E-mail:{mick,ang,wanlei}@deakin.edu.au

Library of Congress Control Number: 2005933043

CR Subject Classification (1998): D, F.1-3. C, I.6

ISSN 0302-9743
ISBN-10 3-540-29235-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-29235-7 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springeronline.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11564621 06/3142 5 4 3 2 1 0

Preface

There are many applications that require parallel and distributed processing to allow complicated engineering, business and research problems to be solved in a reasonable time. Parallel and distributed processing is able to improve company profit, lower costs of design, production, and deployment of new technologies, and create better business environments. The major lesson learned by car and aircraft engineers, drug manufacturers, genome researchers and other specialist is that a computer system is a very powerful tool that is able to help them solving even more complicated problems. That has led computing specialists to new computer system architecture and exploiting parallel computers, clusters of clusters, and distributed systems in the form of grids. There are also institutions that do not have so complicated problems but would like to improve profit, lower costs of design and production by using parallel and distributed processing on clusters.

In general to achieve these goals, parallel and distributed processing must become the computing mainstream. This implies a need for new architectures of parallel and distributed systems, new system management facilities, and new application algorithms. This also implies a need for better understanding of grids and clusters, and in particular their operating systems, scheduling algorithms, load balancing, heterogeneity, transparency, application deployment, which is of the most critical importance for their development and taking them by industry and business.

ICA3PP has been a premier conference that has brought together researchers and practitioners from academia, industry and governments around the world to advance the theories and technologies of parallel and distributed computing. Previously, ICA3PP conferences have been held successfully in Brisbane, Singapore, Melbourne, Hong Kong and Beijing.

ICA3PP 2005 returned to Melbourne with the main focus on the most critical areas of parallel and distributed computing: operating systems and middleware, fault-tolerant systems, scheduling and load balancing, algorithms, tools and environments, and communication and networks.

In total, the conference received 98 papers from researchers and practitioners from 15 countries. Each paper was reviewed by at least three internationally renowned referees and selected based on their originality, significance, correctness, relevance, and clarity of presentation. Among the high quality submissions, 27 long papers and 25 short papers were accepted. All of the selected papers are included in the proceedings. After the conference, the proceedings editors will recommend some high quality papers from the conference to be published in a special issue of an international journal.

We are delighted to be able to host two well-known international scholars, Professor Ian Foster and Professor Zhiwei Xu, who delivered the keynote speeches.

We would like to take this opportunity to thank all the authors for their submissions to the conference. Many of them have traveled some distance to participate in the conference. We also thank the Program Committee members and additional reviewers for their efforts in reviewing the large number of papers. Thanks also go to the local conference organizers for their great support.

Last but not least, we would like to express our gratitude to all of the organizations who have supported our efforts to bring the conference to fruition. We are grateful to the IEEE Technical Committee on Scalable Computing for the cooperation; and to Deakin University, NICTA and Alexander Technology for their sponsorships and assistance.

October 2005 Michael Hobbs, Andrzej Goscinski and Wanlei Zhou
Melbourne,

Organization

This conference was organized by the School of IT, Deakin University, Australia; and Martin Lack & Associates, Australia. Sponsorship was provided by the School of IT, Deakin University, Australia; the Faculty of Science and Technology, Deakin University, Australia; National ICT Australia (NICTA); and Alexander Technology, Australia.

Conference Chairs

Andrzej M. Goscinski (Deakin University, Australia)
Wanlei Zhou (Deakin University, Australia)

Program Chair

Michael J. Hobbs (Deakin University, Australia)

Program Committee

Jemal Abawajy (Deakin University, Australia)
David Abromson (Monash University, Australia)
Akkihebbal Ananda (National University of Singapore, Singapore)
Bill Appelbe (RMIT University, Australia)
Mark Baker (Portsmouth University, UK)
Amnon Barak (Hebrew University of Jerusalem, Israel)
Arndt Bode (Technical University of Munich, Germany)
Peter Brezany (University of Vienna, Austria)
Marian Bubak (AGH University of Science and Technology., Cracow, Poland)
Raj Buyya (University of Melbourne, Australia)
Jianning Cao (Hong Kong Polytechnic University, Hong Kong)
Samuel Chanson (Hong Kong University of Science and Technology, Hong Kong)
Jianer Chen (Texas A&M University, USA)
Francis Chin (University of Hong Kong, Hong Kong)
Toni Cortes (University of Politecnica de Catalunya, Spain)
Brian d'Auriol (University of Texas at El Paso, USA)
Xiaote Deng (City University of Hong Kong, Hong Kong)
Robert Dew (Deakin University, Australia)
Jack Dongarra (University of Tennessee, USA)
Ding-Zhu Du (University of Minnesota, USA)
Wen Gao (Inst. of Computing Technology, China)
Al Giest (Oak Ridge Nation Labs, USA)
Minyi Guo (University of Aizu, Japan)

Salim Hariri (Syracuse University, USA)
Louis Hertzberger (University of Amsterdam, The Netherland)
Shi-Jinn Horng (National Taiwan University of Science and Technology, Taiwan)
Ali Hurson (Pennsylvania State University, USA)
Weijia Jia (City University of Hong Kong, Hong Kong)
Xiaohua Jia (City University of Hong Kong, Hong Kong)
Hai Jin (Huazhong University of Science and Technology, China)
Peter Kacsuk (MTA SZTAKI Research Inst., Hungry)
Krishna Kavi (The University of North Texas, USA)
Zvi Kedem (New York University, USA)
Wayne Kelly (Queensland University of Tech., Australia)
Tohru Kikuno (Osaka University, Japan)
Jacek Kitowski (University of Mining and Metallurgy, Poland)
Domenico Laforenza (ISTI-CNR, Italy)
Laurent Leferve (INRIA, Lyon, France)
Keqin Li (State University of New York at New Paltz, USA)
Zhi Yong Liu (National Natural Science Foundation, China)
Thomas Ludwig (University of Heidelberg, Germany)
George Mohay (Queensland University of Tech., Australia)
Christine Morin (IRISA/INRIA, France)
Edgar Nett (Otto-von-Guericke University, Germany)
Yi Pan (Georgia State University USA)
Marcin Paprzycki (Oklahoma State University)
Sushil K. Prasad (Georgia State University USA)
Rajeev Raje (Purdue University, USA)
Michel Raynal (IRISA, France)
Justin Rough (Deakin University, Australia)
Srinivas Sampalli (Dalhousie University, Canada)
Edwin Sha (University of Texas at Dallas, USA)
Behrooz Shirazi (University of Texas Arlington, USA)
Jackie Silcock (Deakin University, Australia)
Beth Simon (University of San Diego, USA)
Chengzheng Sun (Griffith University, Australia)
Jiachang Sun (Institute of Software, China)
Vaidy Sunderam (Emory University, Atlanta, USA)
Yong-Meng Teo (National University of Singapore, Singapore)
Alistair Veitch (Hewlett Packard Labs, California, USA)
Greg Wickham (GrangeNet, Australia)
Jie Wu (Florida Atlantic University, USA)
Yue Wu (University of Electronic Science and Technology, China)
Roman Wyrzykowski (Czestochowa University of Technology, Poland)
Jingling Xue (University of New South Wales, Australia)
Zhiwei Xu (Inst. of Computing Technology, China)
Laurence T. Yang (St. Francis Xavier University, Canada)
Chung-Kwong Yuen (National University of Singapore, Singapore)
Si Q Zheng (University of Texas Dallas, USA)
Weimin Zheng (Tsinghua University, China)
Wei Zhao (Texas A&M University, USA)
Jun Zou (Chinese University of Hong Kong, China)
Albert Zoymaya (University of Sydney, Australia)

Table of Contents

Author Index

Improving Concurrent Write Scheme in File Server Group

Fengjung Liu[1] and Chu-sing Yang[2]

[1] Department of Management Information Systems, Tajen University,
Pingtung, 907, Taiwan
fjliu@mail.tajen.edu.tw
[2] Department of Computer Science and Engineering, National Sun Yat-sen University,
Kaohsiung, 80424, Taiwan
csyang@cse.nsysu.edu.tw

Abstract. The explosive growth of the Web contents has led to increasing attention on scalability and availability of file system. Hence, the ways to improve the reliability and availability of system, to achieve the expected reduction in operational expenses and to reduce the operations of management of system have become essential issues. A basic technique for improving reliability of a file system is to mask the effects of failures through replication. Consistency control protocols are implemented to ensure the consistency among replicas. In this paper, we leveraged the concept of intermediate file handle to cover the heterogeneity of file system and proposed an efficient data consistency control scheme supporting dependence checking among writes and management of out-of-ordered requests for file server group. Finally, the results of experiments proved the efficiency of the proposed consistency control mechanism. Above all, easy to implement is our main design consideration.

1 Introduction

The explosive growth of the Web contents has led to increasing attention on scalability and availability of file system. A basic technique for improving reliability of file system is to mask the effects of failures by replication. There are two major approaches of building a highly reliable file system: hardware replication approach and software replication approach. In a distributed environment, it is not always necessary to use special hardware for improved reliability. The computers connected by the high-speed network are a natural resource of duplicates. The software replication approach replicates file systems on workstations in the network. Consistency control protocols are designed to ensure the consistency among replicas.

2 Related Works

Many Distributed File systems, such as intermezzo[1], Coda[2], Deceit[3], Ficus [4], RNFS[5] and Pangaea[6], implemented reliable file system services through software

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 1 – 10, 2005.

replication approach. In particular, FSG[7,9,10], RNFS and Deceit are NFS-based systems. JetFile[8] and Coda are the instances of multicast-based file systems.

2.1 Overviews of Network File System

The Network File System, NFS, is the most popular distributed file system developed by SUN Microsystems. Each server computer can serve an arbitrary number of the sub-tree in its local file system. Clients are able to mount the exported sub-trees, linking then to its own file system, using the same semantics valid while mounting physical local devices. Its main features are

I. It is a "stateless" protocol. A server does not need to maintain any protocol state information about any of its clients to function correctly.
II. The NFS protocol is idempotent. Because of this and the statelessness property of NFS, what a client has to do for recovery from the crashed server is simply trying the failed RPC until the server reappears.
III. A client accesses a file using a handle, called *fHandle,* obtained from the server as a result of a LOOKUP operation.
IV. Updates are synchronous with respect to failures. If a write RPC completes, the client is assumed that data has been written. Again, this is a property not met by all NFS implementations.

Because of the stateless property of NFS, it could reduce the overhead of recovery after system crashing and make our implementation easier.

2.2 Multicast

In IP multicast [11] there are 2^{28} (2^{112} in IPv6) distinct multicast channels. Channels are named with IP addresses from a subnet of the IP address space. IP packets are only delivered with best effort. To multicast a packet, the sender uses the name of the multicast channel as the IP destination address.

Scalable Reliable Multicast, SRM, [12] is designed to meet only the minimal definition of reliable multicast, i.e., eventual delivery of all data to all group members. As opposed to ISIS [13], SRM does not enforce any particular delivery order. Delivery order is to some extent orthogonal to reliable delivery and can instead be enforced on top of SRM. SRM is logically layered above IP multicast and also relies on the same lightweight delivery model. To be scalable, it does not make use of any negative or positive packet acknowledgements, nor does it keep any knowledge of receiver group membership. Applications will often be able to recover after a period pf packet loss by only requesting to current data. Thus, it is not always necessary to catch up on every missed application data packets.

3 System Design

In system design, we assumed the fail-stop property be approximated by the kernel and the hardware. Additionally, we also assumed that the servers are connected by a Local Area Network and the network is not subject to partitions.

3.1 Overview of File Server Group, FSG

In designing system, the collection of replicated servers is treated as a group, assigned a group IP address. The IP address will be used by the underlying multicast protocol to deliver messages to all servers in this group. The system model is shown in Fig. 1. The nodes in this model are not limited to be homogeneous processors.

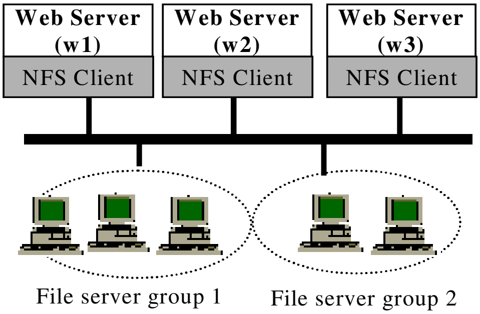


Fig. 1. System Model

In FSG system, a user on the client machines uses the "mount" command to connect to the sever group. The main difference from the traditional UNIX mount command is that the "server:pathname" parameter is replaced by "multicast IP address:pathname". An example is made below, which [aa.bb.cc.dd] is broadcast IP address, /usr1 is an exported directory in server group and /mnt is the mount directory.

Syntax :
#mount [-F nfs] [-mrO] [-o suboptions] server:pathname mount_point
Example :
#mount aa.bb.cc.dd:/usr1 /mnt

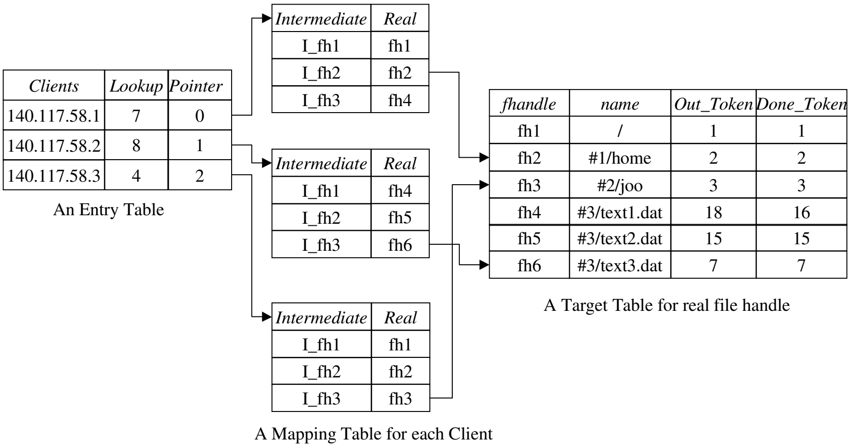
After the client is connected the server group, the user can read/write files in the replicated servers just like normal UNIX local files. In UNIX system, users perform read/write operations through file handles, *fhandle*. Each opened file is assigned a *fHandle* which is a unique file index assigned by the server to identify an opened file. Since files are replicated in the FSG system, each replicated server will generate its own *fHandle* for that opened file. However, each client can only recognize one. To solve this problem, we leveraged the concept of intermediate file handle, *I_fHandle*, proposed previously in papers [7,9]. The illustration of the scenario of the new mount procedure is not repeated here for the space limitation.

3.2 The Structure of Mapping Table

The traditional content of a file handle is composed of device number, the *inode* number, and a generation number for the *inode*. Obviously, it is machine-dependent. So, we proposed the intermediate file handles to mask the heterogeneity of file systems. An *I_fHandle* consists of 4 items, client's IP address, a mount number, a sequence number and an incremental number. The *Client_IP_addr* is used to distinguish different clients. The *Seq_number* and the *Mount_number* respectively represent different files in the mount directory and the order of different mount. The *Inc_number* item is to represent different components in the multi-component LOOKUP request [14,20,21].

Each replicated server maintains a mapping table to map *I_fHandle* into corresponding *fHandle*. While a client tries to mount a remote directory, it has to issue firstly a mount command to the server group. As receiving the mount request from a client, the server creates an Entry Table for the client as shown at the most left hand side of Fig. 2. Within the Entry Table, the LOOKUP column is used to keep the latest token for LOOKUP requests. To ensure that the unique and consistent *I_fHandle* be generated in each server, the LOOKUP operations must be performed sequentially.

In the mapping table for each client, it contains two items, *I_fhandle* and *fhandle*. In general, a file server uses the *fhandle* to locate the corresponding information in the target table. A file server used the *Out_Token* field in the target table to keep the latest updated tokens of each files and the *name* field to represent the file/directory name. The *Done_Token* field is deployed to record the maximum token of completed requests for the implementation of consistency control scheme.



Clients	*Lookup*	*Pointer*
140.117.58.1	7	0
140.117.58.2	8	1
140.117.58.3	4	2

An Entry Table

Intermediate	*Real*
I_fh1	fh1
I_fh2	fh2
I_fh3	fh4

Intermediate	*Real*
I_fh1	fh4
I_fh2	fh5
I_fh3	fh6

Intermediate	*Real*
I_fh1	fh1
I_fh2	fh2
I_fh3	fh3

A Mapping Table for each Client

fhandle	*name*	*Out_Token*	*Done_Token*
fh1	/	1	1
fh2	#1/home	2	2
fh3	#2/joo	3	3
fh4	#3/text1.dat	18	16
fh5	#3/text2.dat	15	15
fh6	#3/text3.dat	7	7

A Target Table for real file handle

Fig. 2. The structure of a Mapping Table

3.3 The Process of a GETTOKEN Request

In this section, we illustrated how the sequencer constructs the tokens for *GETTOKEN* requests. Since NFS is an UDP-based protocol, timeout and retransmissions are used to take care of lost messages and transient network failure. However, retransmissions can cause the same request to be executed twice on the server and this is unacceptable for non-idempotent requests. A skill to detect duplicate requests is to package token numbers into all request messages.

Table 1. RPC request list of a File Server Group

RPC request	Action	Nature	Idempotent
GETATTR	To get file attributes	Unicast	Yes
SETATTR	To set file attributes	Multicast	Yes
LOOKUP	To look up file name	Multicast	No
READLINK	To read from symbolic link	Unicast	Yes
READ	To read from file	Unicast	Yes
WRITE	To write to file	Multicast	Yes
CREATE	To create file	Multicast	Yes
REMOVE	To remove file	Multicast	No
RENAME	To rename file	Multicast	No
LINK	To create link to file	Multicast	No
SYMLINK	To create symbolic link	Multicast	Yes
MKDIR	To create directory	Multicast	No
RMDIR	To remove directory	Multicast	No
READDIR	To read from directory	Unicast	Yes
STATFS	To get file system attributes	Unicast	Yes
GETTOKEN	To get a token for write	Multicast	No
SYNC	To sync. Server group.	Multicast	Yes

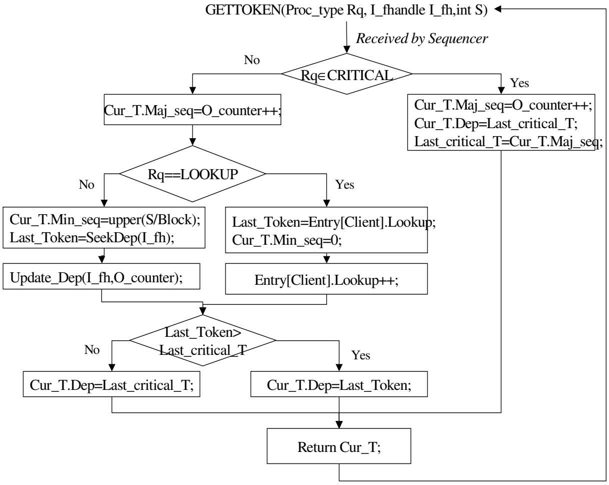

GETTOKEN(Proc_type Rq, I_fhandle I_fh,int S)
Received by Sequencer
No
Rq∈CRITICAL
Yes
Cur_T.Maj_seq=O_counter++;
Cur_T.Maj_seq=O_counter++;
Cur_T.Dep=Last_critical_T;
Last_critical_T=Cur_T.Maj_seq;
No
Rq==LOOKUP
Yes
Cur_T.Min_seq=upper(S/Block);
Last_Token=SeekDep(I_fh);
Last_Token=Entry[Client].Lookup;
Cur_T.Min_seq=0;
Update_Dep(I_fh,O_counter);
Entry[Client].Lookup++;
No
Last_Token>
Last_critical_T
Yes
Cur_T.Dep=Last_critical_T;
Cur_T.Dep=Last_Token;
Return Cur_T;

Fig. 3. Flow Chart of a GETTOKEN procedure of Mode 2 in the Sequencer

In Table 1, these RPC calls are classified into 2 types, idempotent and non-idempotent. Before all of update requests including *LOOKUP* request are performed, they must issue a *GETTOKEN* request to ensure these requests be executed in the same order. For simplicity, these are restricted to be executed sequentially. However, such a simple scheme causes the poor performance. Therefore, based on the token-based mechanism, we proposed an efficient consistency control scheme. We classified these update requests in Table 1, into two sets, ***CRITICAL*** and ***DEPENDENCY***. The servers cannot decide efficiently the dependency of requests in the ***CRITICAL*** set with merely message header. But, the target files of requests in the ***DEPENDENCY*** set are determinate. These sets are listed below.

Set ***CRITICAL*** *={REMOVE, RENAME, LINK, MKDIR, RMDIR, GETTOKEN}*
Set ***DEPENDENCY*** *={SETATTR, LOOKUP, WRITE, CREATE, SYMLINK}*

```
typedef struct Token {                  typedef struct Out_Token {
        int Gen_Number;                         Token Out;
        int Maj_Seq;                            Token Dep;
        int Min_Seq;                    } Out_Token;
} Token;
```

```
int Block;                          //file system 's Block Size or MTU considered
int SeekDep(I_fhandle I_fh);        //To return the out_token in Target Table with I_fh.
Update_Dep(I_fhandle I_fh, Token T);   //Update the Out_token in Target Table
Out_Token GETTOKEN(Proc_type Rq,I_fhandle I_fh, int S);
   // Proc_type R : NFS operation type;
   //I_fhandle I_fh : which file a client want to access.
   // int S: Data Size
Token Last_critical_T; // The last token assigned to CRITICAL requests.
Out_Token Cur_T;       // GETTOKEN() procedure return to caller
Token Last_Token;
Entry[clients];// Entry table for the client's state as shown in Fig. 3
```

The processing of a GETTOKEN procedure in the sequencer is shown in Fig. 3. When receiving a *GETTOKEN* request, the sequencer will firstly check if *Proc_type* parameter, Rq, is a *CRITICAL* request. If true, the *O_counter* increases one, the *Cur_T.Out. Maj_Seq* is assigned with the *O_counter*, the *Cur_T.Dep* is set to the *Last_critical_T*, assigned to last *CRITICAL* request and the *Last_critical_T* is set to the *O_counter*. At last, the *Cur_T* is returned. Otherwise, if *Proc_type* parameter, *Rq*, is a *LOOKUP* request, the *Cur_T.Dep* is set to the last token, which the sequencer ever dispatched for *LOOKUP* request, and update the lookup field of the client's entry table in Fig. 3 with the new token, *Cur_T.Out*. Else, the *Out_Token* counter, the last token ever assigned for accessing the *I_fh*, will be found out in the target table and the *Cur_T.Dep* is assigned with the maximum of the *Out_Token* and *Last_critical_T*. Then, if the required data size is greater than 0, the sequencer will set the minor sequence number, *Cur_T.Min_seq*, to *upper(Required Size/Block)*. Finally, the *Cur_T* is returned to the caller.

3.4 Consistency Control Mechanism

Concurrent write sharing is achieved in some variants of NFS [15,16,17,18]. The monolithic server system [7] suffered from the poor system utilization due to the lack of dependence checking among writes and management of out-of-ordered requests. Based on the deployment of control window [10], there are 3 modes designed to keep the data consistency among replicas and illustrated below:

Mode 1: Strictly global sequential write scheme

In this mode, proposed previously in [7], the Sequencer generates tokens for each GETTOKEN requests without considering which file or directory is accessed. And, each server executes sequentially each WRITE requests. Such a scheme is simple and easy to implement but gets poor performance.

Mode 2: Sequential write scheme

In Mode 2, proposed in [10], the Sequencer considers which file or directory to be accessed to generate one *Out_token* for each GETTOKEN requests. In Fig. 4, it describes the executions of multiple writes with Mode 2. Client 1 acquires an *Out_Token* (5,4) which means the request with *Token* (5) is executed only when *I_Counter* is greater than *Token(4)*. Thus, the *Out_Token(6,2)* can be executed before *Out_Token(5,4)*.

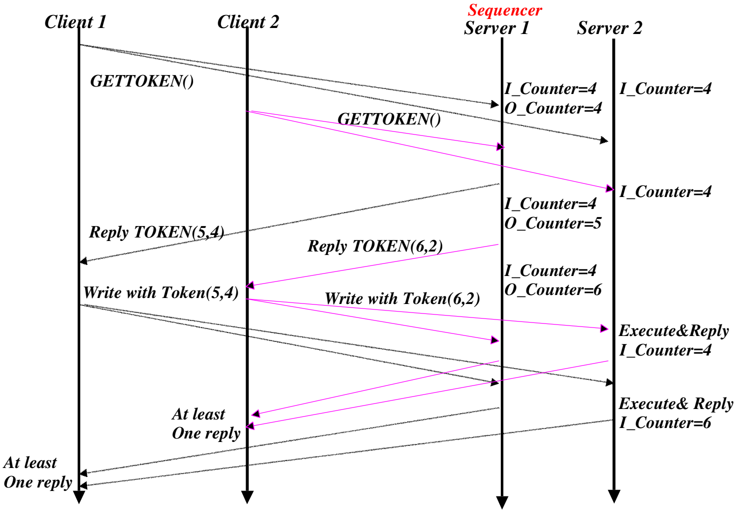
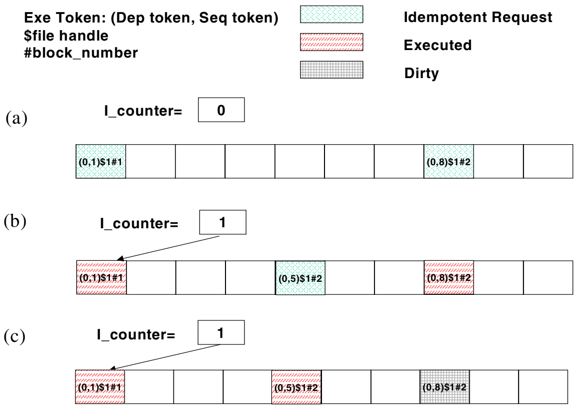


Fig. 4. The executions of multiple WRITE requests. (Mode 2)

Mode 3: Concurrent write scheme

The new consistency control scheme in Fig. 5 is designed to support "concurrent write" on the server side. It mainly utilized the idempotent property of NFS write operations. While an idempotent write request comes, the server will check if the *Dep* token is completed or not. If done, the server will execute this request ahead and check if any executed ahead requests with the greater tokens exist, which updated the same blocks as the current request did. If exists, the status of these requests are set to *Dirty*. That is, if the *Done_token* field in Target Table is greater than the current token, it means the current update request may be conflict with the executed ahead update requests. The executed ahead

Updates are set to *Dirty*. In this mode, its main difference of the GETTOKEN processing from Mode 2 is the *Dep* token is set to the latest token, ever assigned to last *CRITICAL* requests, not the token acquired for the same file.



(a). Two WRITEs with (0,1) and (0,8) tokens arrive and attempt to write different blocks at same file. They are independent, so they can be executed individually

(b). One WRITE with (0,5) token arrives and attempts to write block 2 in a file whose handle is 1.

(c). The server will look forward to check if the assess area is same and set the status of request to dirty.

Fig. 5. Examples of concurrent write

4 Experimental Results for Consistency Control Schemes

Due to the synchronization property of RPC calls, each client needs to send-and-wait for each requests processing. On our experimental programs, the average time for writing a 200K-byte file and the averaged RPS, a client contributing to a server, are shown in Table 2.

Table 2. The averaged time and RPS for a client contributing to a server on writing a 200K-byte file. (Timeout=200ms)

File Size (byte)	Avg. RPS	Time (sec)
200K	66.7	3.0

In this experiment, there are five Pentium-4 computers which are all homogenous, running on Win2K server OS and connected with Fast Ethernet Switch. Within them, two act as the clients, two as servers and one as the sequencer respectively. Each client host is assigned separately with 2 and 4 processes as NFS Clients. Each client writes sequentially 5 files, chosen randomly from 4 files. Additionally, the data size is set to 1K bytes and the timeout value is 200 ms.

Table 3. The average time to write 5 200K and 400K bytes files using different consistency control Modes. (Timeout=200ms)

Size (byte)	No of Clients	Mode 1(sec)	Mode 2 (sec)	Mode 3 (sec)
200k	1	3.0	3.0	3.0
	2	5.1	3.8	3.8
	4	12.0	7.8	7.1
	8	68.7	12.8	10.6
400k	1	6.0	6.0	5.9
	2	9.1	8.2	7.3
	4	25.1	15.4	14.1
	8	181.2	24.0	22.9

The average time to write 5 200K/400K bytes files using different consistency control modes are shown in Table 3. The 5 files are randomly selected from previously installed 10 files in client hosts. It explains that the more the number of clients increase, the worse the access time using strictly global sequential write scheme, Mode 1, gets. But, the Mode 2 and Mode 3, which support dependence checking, have much better performance than Mode 1.

5 Conclusion

In this paper, we had leveraged the concept of intermediate file handle to cover the heterogeneity of replicated file system. Based on this concept, a decentralized consistency control scheme is designed to achieve concurrent writing and improvement of utilization in File Server Group. The results of experiment revealed that the new consistency control schemes, Mode 2 and Mode 3, are able to improve the system efficiency. As illustrated in the paper [19], most instances of write sharing can be predicted easily, and they demand consistency only within a window of minutes. Thus, in FSG system, the SYNC request is deployed to keep the consistency among duplication when an out-of-ordered request comes. Above all, easy to implement is our main design consideration.

References

1. Peter J. Braam, “File Systems for Clusters from a Protocol Perspective”, http://www.intermezzo.org
2. M. Satyanarayanan, J.J. Kistler, P.Kumar, M.E. Okasaki, E.H. Siegel and D.C.Steere "Coda: A highly available file system for a distributed workstation environment" IEEE Transactions on Computers, 39(4), pp.447-459, April 1990
3. A. Siegel, K. Birman and K. Marzullo. "Deceit: A flexible distributed file systems" In Summer 1990 USENIX Conference, pages 51-61, Anaheim, CA, June 1990

4. R.G. Guy, J.S. Heidemann, W. Mak, W. Page and G.J. Popek "Implementation of the Ficus replicated file system". In Proceedings of Summer 1990 USENIX Conference, June 1990, Pages 63-71
5. M.M. Leboute and Taicy Weber, "A reliable distributed file system for UNIX based on NFS", UFRGS, Brazil, IFIP International Workshop on Dependable Computing and Its Applications (DCIA 98) January 12 - 14, 1998, Johannesburg, South Africa
6. Yasushi Saito and Christos Karamanolis, "Pangaea: a symbiotic wide-area file system," ACM SIGOPS European Workshop, Sep 2002.
7. C. S. Yang, S. S. B. Shi and F. J. Liu, "The Design and Implementation of a Reliable File Server", Newsletter of the Technical Committee on Distributed Processing, summer 1997.
8. Bjorn Gronvall, Assar Westerlund, and Stephen Pink. "The design of a multicast-based distributed file system". In Proc. of Operating Systems Design and Implementation, pages 251-264, 1999.
9. F.J.Liu and C.S.Yang, "THE DESIGN AND ANALYSIS OF A HIGHLY-AVAILABLE FILE SERVER GROUP", IEICE Transactions on Information and System, Vol.86-E, No.11, pp. 2291-2299, 2003.
10. F.J.Liu, C.S.Yang and Y.K.Lee, "The Design of An Efficient and Fault-tolerant Consistency Control Scheme in File Server Group", IEICE Transactions on Information and System, Vol.E87-D No.12, pp.2697-2705, 2004.
11. S.Deering, Host Extensions for IP Multicasting, RFC 1112, Internet Engineering Task Force, 1989.
12. S.Floyd, V. Jacobson, C.Liu, S. McCanne, L.Zhang, A Reliable Multicast Framework for Light-weigh Sessions and Application Level Framing, IEEE/ACM Transactions on Networking, 5(6), Dec. 1997.
13. K.Birman, A.Schiper, P.Stephenson, Light-weight Causal and Atomic Group Multicast, ACM Transactions on Computer Systems, 9(3), Aug. 1991.
14. The NFS Version 4 Protocol.
15. M.Nelson, B.Welch, and J.Ousterhout. Caching in the Sprite Network File System. ACM Transactions on Computer Systems, 6(1):pp.134-154,Feb.1988.
16. V. Srinivasan and J. Mogul. Spritely NFS: Experiments with Cache Consistency Protocols. In Proceedings of the Twelfth ACM Symposium on Operating Systems Principles, pp. 45--57, Dec. 1989
17. J. Mogul, Recovery in Spritely NFS, Computing Systems, 7(2): pp. 201-262, 1994
18. Macklem, Rick, "Not Quite NFS, Soft Cache Consistency for NFS," Winter USENIX Conference Proceedings, USENIX Association, Berkeley, CA, Jan. 1994.
19. Susan Spence, Erik Riedal and magnus Karlsson, "Adaptive consistency –patterns of sharing in a networked world," Technical Report HPL-SSP-2002-10, HP labs, Feb. 2002.
20. Callaghan, B., "WebNFS Client Specification," RFC 2054, October 1996. http://www.ietf.org/rfc/rfc2054.txt
21. Callaghan, B., "WebNFS Server Specification," RFC 2055, October 1996. http://www.ietf.org/rfc/rfc2055.txt

A Comparative Performance Study of Distributed Mutual Exclusion Algorithms with a Class of Extended Petri Nets

Alexander Kostin[1], Ljudmila Ilushechkina[2], and Erhan Basri[1]

[1] Department of Computer Engineering, Eastern Mediterranean University,
Magusa, via Mersin 10, Turkey
{Alexander.Kostin, Erhan.Basri}@emu.edu.tr
[2] Department of Software Engineering, Moscow Institute of Electronic Technology,
Zelenograd, Moscow, Russia
Ljuda_il@fromru.com

Abstract: A few algorithms of distributed mutual exclusion are discussed, their unified model in terms of a finite-population queuing system is proposed, and their simulation performance study is presented with the assumption that they use multicast communication if possible. To formally represent the algorithms for simulation, a class of extended Petri nets is used. The simulation was done in the simulation system Winsim based on this class of Petri nets.

1 Introduction

Distributed mutual exclusion serializes the access of a group of processes, running in different nodes of a distributed system, to a shared resource (SR), with the exclusive use of SR by no more than one process at a time. The part of the process program which deals with SR is usually called a *critical* section.

During more than past 20 years, a number of approaches and solutions to the problem have been proposed. One of the first solutions is due to Lamport [1]. In subsequent works of Ricart and Agrawala [2], Maekawa [3], Suzuki and Kasami [4], Sanders [5], Trehel and Naimi [6], Raymond [17], Agrawal and Abbadi [7], and Singhal [8], new schemes and algorithms for distributed mutual exclusion were described, and the theoretical foundations of the problem were laid down. Different classification frameworks and comparative analysis of distributed mutual exclusion algorithms proposed up to the beginning of 1990's can be found in [9], [10] and [11].

Since the middle of 1990's, more distributed mutual exclusion algorithms have been developed. Some of them are extensions or modifications of the previously designed algorithms [12], [13], while others represent new proposals and schemes [14], [15].

Since the theoretical analysis of distributed mutual exclusion algorithms often does not yield exact comparative information, much attention of researchers was paid to simulation studies [11], [16]. As a rule, in analytical and simulation models of distributed mutual exclusion algorithms, the *unicast* (point-to-point) mode of communication between processes was assumed. The use of this mode was dictated

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 11 – 22, 2005.

by the state of the network communication technology of the past. The progress in this area opens new possibilities for construction of distributed mutual exclusion algorithms. In particular, the use of *multicast* communication [19], [20] gives the possibility to make many previously developed mutual exclusion algorithms much more efficient with respect to communication traffic.

In this paper, four often cited distributed mutual exclusion algorithms are analyzed and simulated with the assumption that they use *multicast* communication between involved processes where possible, instead of *unicast* communication as was assumed by the authors. For the study, the algorithms of Ricart and Agrawala [2], Suzuki and Kasami [4], Singhal [8], and Naimi, Trehel and Arnold [12] were chosen. In addition, a novel distributed mutual algorithm is described and compared with the above listed algorithms. All the algorithms are considered in terms of a *finite-population queuing system*. For the description of simulation models, a class of *extended Petri nets* was used, and simulation was carried out in a simulation system based on this class of Petri nets.

The rest of the paper is structured in the following way. Section 2 presents a system model and assumptions used in the analysis and simulation of algorithms of distributed mutual exclusion. Section 3 outlines four published algorithms of distributed mutual exclusion. In Section 4, a novel distributed mutual exclusion algorithm is described in some detail. Finally, Section 5 contains the results of simulation study of all the algorithms and the discussion of these results.

2 System Model and Assumptions

When a group of N processes competes for the mutually exclusive use of some shared resource (SR), the whole system can be logically modeled as a finite-population queuing system with N clients and one non-preempted SR server. According to this model, each client process performs some application-specific task as a sequence of steps (steps of main work or thinking). At the end of each step of main work, the process generates a request for SR, sends it to the SR server for subsequent handling, and goes to sleep. Handling of the request by the SR server can be viewed as the use of SR by the process. The SR server extracts requests from its input queue for servicing according to the FIFO order or based on some priority scheme that depends on the concrete algorithm. After the servicing of the request has been completed, the SR server sends its response back to the client to awake it and to force it to proceed with the next step of main work. It should be noted that, in the distributed mutual exclusion system, there is actually no explicit SR server. The functionality of the server is carried out by a client process every time its request for SR is granted.

The model of distributed mutual exclusion in terms of a finite-population queuing system simplifies analysis and comparison of different algorithms, because well known performance measures of a queuing system can be used to characterize them. However, having the same general model in terms of a queuing system, distributed mutual exclusion algorithms differ in the way the clients' requests are actually communicated to the SR server in the underlying network and how the server informs the clients about the completion of the service. Therefore, a queuing system can be considered as an idealized model of distributed mutual exclusion algorithms. Since, in theory, queuing systems are considered as centralized objects, they do not take into account communication overheads. This means, that distributed mutual exclusion algorithms will always have longer response time than the corresponding centralized queuing system.

In this study, it is assumed that processes in a distributed system are identical in that the probability distributions for a step of main work (or thinking step) and for a step of using SR are the same for all processes. Without loss of generality, we assume also that these probability distributions are exponential, with mean values chosen to get the desired load of the SR server. With these assumptions, the idealized model of distributed mutual exclusion is an $M/M/1/N/N$ queuing system [18]. The assumption of exponential probability distributions is necessary only to validate the simulation models. The mutual exclusion algorithms under study do not rely on this assumption in their work.

Finally, it is implicitly assumed in all algorithms under study that processes use reliable multicast for communication [19], [20]. Semantics of reliable multicast is defined in [21].

3 Distributed Mutual Exclusion Algorithms Chosen for the Study

This section outlines distributed mutual exclusion algorithms of Ricart and Agrawala, Suzuki and Kasami, Singhal, and Naimi, Trehel and Arnold chosen for the study in this paper. These algorithms were chosen, first of all, because they are frequently cited in literature and often used for comparative purposes. The second reason is that they have been described by their authors in full detail which is essential for the development of their correct simulation models. One more reason is that these algorithms have been carefully simulated in [11] with the use of *unicast* communication, so that it would be interesting to see the difference in the performance of these algorithms when they use *multicast* communication where possible.

The *permission-based algorithm of Ricart and Agrawala* [2] is probably the most often cited distributed mutual exclusion algorithm. It assumes that each involved process knows the number N of all processes in the group. When a process i wants to access an SR, it sends a REQUEST message to other $N-1$ processes. The message contains the identifier of the sending process and a sequence number constructed according to the Lamport's timestamp [1] that is used, together with the process identifier, in a priority resolution scheme. Every other process $j \neq i$, after receiving a REQUEST message, immediately sends a REPLY message to process i or stores the received message in its queue for the deferred reply. Process i may access SR only after it has received $N-1$ replies from other processes. If the REQUEST message can be sent in the multicast mode, then the algorithm will need exactly N messages per use of SR: one multicast REQUEST message and $N-1$ unicast REPLY messages.

The *algorithm of Suzuki and Kasami* [4] belongs to the class of token-based algorithms. A token is represented by a PRIVILEGE message. A message of type REQUEST is used to inform all processes in the group about the desire of this process to access SR. In addition to these two types of messages, the algorithm uses, in each involved process, two one-dimensional integer arrays RN and LN of size N each. These two arrays are necessary to order requests from different processes and to keep the ordered requests in a system-wide queue passed in each PRIVILEGE message. If it happens that the process, wanting to access SR, keeps the PRIVILEGE token already, then no request message is sent in the network, and the process can immediately start using SR. With the use of unicast communication, the algorithm requires *approximately N* messages per use of SR. But when the REQUEST message is sent in the multicast mode (as was done in our simulation experiments), the algorithm will require less than two messages per use of SR.

The *permission-based algorithm of Singhal* [8] uses, in each involved process, a dynamic information structure that evolves with time as processes learn about the state of the whole system. The information structure in each process i is actually a particular implementation of a general information structure proposed in [5]. It consists of the request set R_i and the inform set I_i, with the first set of identifiers of the processes from whom process i must get permission before accessing SR, and with the second set of identifiers to which process i must send its permission to access SR. The algorithm works with two types of messages – a request for SR and a reply. The number of messages sent in the network depends on the load of SR and varies from $N - 1$ for low load to $3(N - 1)/2$ for high load, with N processes. Since request and reply messages are sent selectively according to the request set and inform set, the algorithm cannot benefit from multicast mode of communication.

Finally, the *token-based algorithm of Naimi, Trehel and Arnold* [12] uses a dynamic rooted tree to pass a token between processes. Its first version was published in [6]. Initially, all involved processes are structured in a simple tree, with process 1 as father in the root and all other processes as children, with a pointer to the root. During the work of the algorithm, the configuration of the tree of processes will vary, but every time each process knows its new father process. The algorithm keeps the distributed queue of requests and uses two types of messages – the REQUEST for SR and TOKEN. Every time a process finishes the use of SR, it will send the TOKEN message to its next process in the distributed queue. It is claimed that the algorithm requires $O(\log N)$ messages per use of SR. However, it cannot exploit multicast communication to decrease the network traffic.

4 A Novel Algorithm of Distributed Mutual Exclusion

As all the algorithms outlined in the previous section, the proposed algorithm assumes a *reliable* communication between involved processes. All its messages are transmitted in the multicast mode. This means that communication between processes is *anonymous*, so that each process needs to know only the group address. In contrast with the above described algorithms, the proposed algorithm does not need to know the number of processes in the group, although the knowledge of an *approximate* size of the group can be used by the algorithm to improve its performance. It is assumed also that each process in the group knows the approximate duration of its intended use of SR and includes this information in its request. Further, each process uses a few time-outs and delays and is capable to measure the passage of its time-outs and delays with some accuracy.

The proposed algorithm uses three types of messages: a request for the SR (message of type R), the SR is free (message of type F), and the SR is being used (message of type E). Each process is either performing a step of main work, or accessing the SR, or waiting for the SR to become free.

Consider the behavior of some process *P* informally. After finishing a step of main work, process *P* tests the current state of SR by the use of its local state variable RSTA. If, from the point of view of process *P*, SR is not free or is being already negotiated for the access by some other processes, process *P* will enter its waiting state until SR becomes free.

If SR is free then the process multicasts, *with some probability*, a message R to inform all other processes in the group about its *desire* to access SR and starts its time-out T_1. Probability of multicasting a message R is not fixed, it depends on the approximate size of the group of processes and on the load of the SR server. Message R includes the estimated duration of the intended use of SR. If, according to the calculated probability, the process makes decision not to request SR, then it delays for some time (*slot* time) and, after elapsing this time, can make a new attempt if SR is free. The probabilistic decision helps reduce the competition for SR between processes and makes the algorithm more *scalable* with respect to the number of processes in the group.

If, after transmission of the request message, process *P* did not receive any *conflicting* message R from any other process during time-out T_1, it deduces that no other process intends to access SR and starts using SR. In this case, due to reliable multicast, all other processes in the group receive the message R from *P*, learn the estimated duration of the use of SR by *P* from this message, and set the corresponding state of SR in their local variable RSTA accordingly. The duration of the use of SR will be used by all other processes to calculate the upper limit of a crash-recovery time-out T_2. Process *P*, after finishing the use of SR, multicasts a message of type F which forces all other processes in the group to set the free state of SR.

On the other hand, if process *P* receives at least one message R from some other process during time-out T_1, it understands that there is a conflict with another process, and starts a random back-off delay T_3. The same action will be done by each conflicting process. After elapsing of T_3, the behavior of process *P* and of all other processes involved in the conflict, is determined by the state of SR as seen by each conflicting process.

The detailed specification of the algorithm is presented in the form of a state diagram in Fig. 1. States of the diagram have the following meaning: 0 – process is performing a step of its main work (or *thinking* step); 1 – process is running time-out T_2; 2 – process is delaying during a slot time; 3 – process is delaying during time-out T_1; 4 – process is using SR (is in its critical section); 5 – process is delaying during a random back-off interval T_3.

In the diagram, expressions over transition arcs are predicates in terms of events, logical statements, and logical variables. Expressions under transition arcs represent actions performed by the algorithm if the corresponding logical expression over this transition arc is true. There are three self-loops in states 0, 2 and 5 which are not shown to save space for the figure. The first self-loop takes place when message of type R or E is received, in this case the reaction is RSTA $\leftarrow$ 2 and *calc*(T_2). The second self-loop corresponds to receiving of message F in the situation RSTA = 2, with the reaction RSTA $\leftarrow$ 0. Finally, the third self-loop is receiving message F in the situation RSTA = 0, with the reaction *warn*(SR).

The events, predicates and logical variables have the following meaning: $\uparrow$X – a message of type X is received from some other process; *elapsed* (*T*) – delay or time-out *T* elapsed; *finished*(main) – the process completed a step of its main work and estimated the duration of the intended use of SR; *finished* (SR) – the process completed the use of SR; *SR* – the process probabilistically decided to request SR; ~*SR* – the process decided not to request SR; RSTA = *s* – state variable RSTA has value *s*.

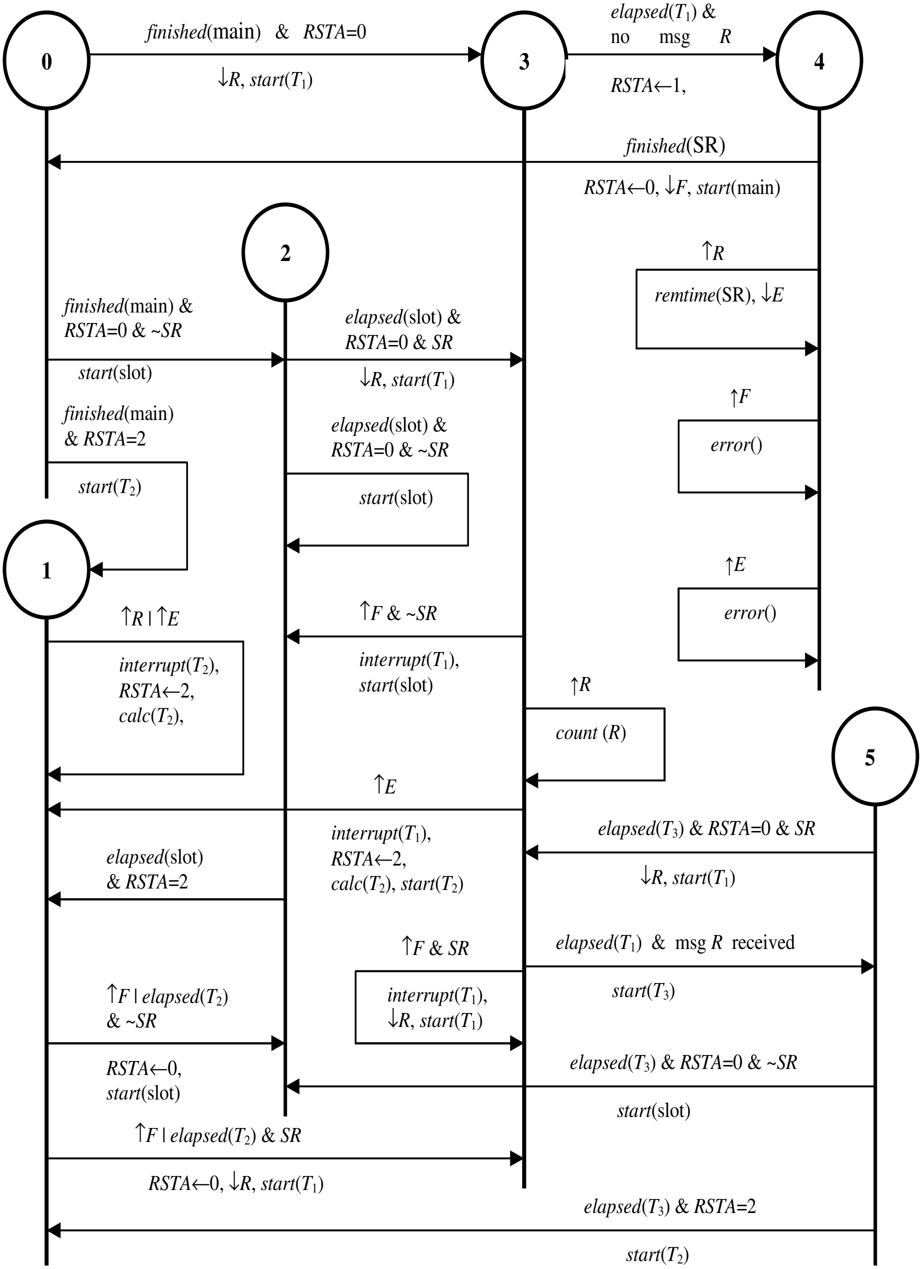


Fig. 1. State diagram of the proposed distributed mutual exclusion algorithm

The algorithm can perform the following actions when it transits from one state to another: ↓X – sending a message of type X (in messages of types R and E, the protocol includes the estimated duration of use of SR and the remaining time of use of SR, respectively); *start* (*T*) – starting a delay or time-out *T*; *start* (main) – calculating the duration of a step of main work and starting the step; *start* (SR) – starting the use of SR; *interrupt* (*T*) – interrupting time-out *T*; *calc* (*T*) – calculating the value of delay or time-out *T*; *remtime* (SR) – calculating the remaining time of use of SR; *count* (X) – counting received messages of type X; *warn* (SR) – outputting a warning message “SR is free already”; *error* () – outputting an error message “Protocol error” and exiting. Actions *warn*(SR) and *error*() are included for completeness. In the correct behavior of the algorithm, these actions should never appear.

The state variable RSTA represents the process’s knowledge of the state of SR. The variable can have the following values: 0 – SR is free ; 1 – SR is being used by this process; 2 – SR is being negotiated by some other processes or it is being used by some other process.

The algorithm uses two time-outs and two random delays: T_1 – conflict-detection time-out, T_2 – deadlock-resolution and crash-handling time-out, T_3 – back-off delay, and T_4 – so called *p-persistence* delay.

The choice of the values for these time-outs and delays is important for the correct operation of the algorithm. The use of these time-outs and delays could be understood from the informal description of the algorithm and from its state diagram. Due to space limitation, a more detailed discussion of the time-outs and delays of the proposed algorithm is omitted.

5 Performance Study

The performance study of the described algorithms was carried out with the use of simulation in the system Winsim, based on a class of extended Petri nets [22], [23], [25]. Being a universal algorithmic system, these nets support attributed tokens, timing, control functions and data transformation. With these Petri nets, the model is structured as a collection of interconnected *elementary nets*, each of which is represented by a single transition with incident places. There are five different types of elementary nets T, X, Y, G, and I, each type having strictly defined properties and a graphical form of the involved transition. In particular, the elementary net of the type T performs data transformation and delaying functions when its transition fires. The elementary net of the type X is used to route a token from some input place to a selected output place. The net of the type Y performs multiplexing of input tokens to output places. The net of the type G combines the properties of nets of types X and Y. Finally, the net of the type I implements the interruption of the fired transition. The detailed description of this class of Petri nets is given in [23].

For simulation, each distributed mutual exclusion algorithm was represented in Winsim as a collection of *segments*, with each segment expressing activity of one process. All these segments were connected to a model of network of Ethernet type. For all the algorithms under study, the simulation model of the network was the same. According to this model, the time to transmit a message in the network was assumed to be random, with the uniform probability distribution in the range (2, 4) ms. This is in agreement with actual delivery time of frames of size about 500 bytes in a LAN of Ethernet type [24].

As an example, the user manual of Winsim [23] contains the complete description of a Petri-net-based simulation model of the Ricart and Agrawala algorithm. This description includes the Petri net schemes of a process segment and a network segment, their texts in the Model Description Language of Winsim, for three involved processes, and a file of simulation parameters. Section 8 of the manual [23] explains all the steps in the preparation of the model and presents the numerical results of its running.

In the models of all algorithms, mean time of using SR 1/μ was fixed at 500 ms, while mean time of a step of main work (or mean thinking time of a process) 1/λ was

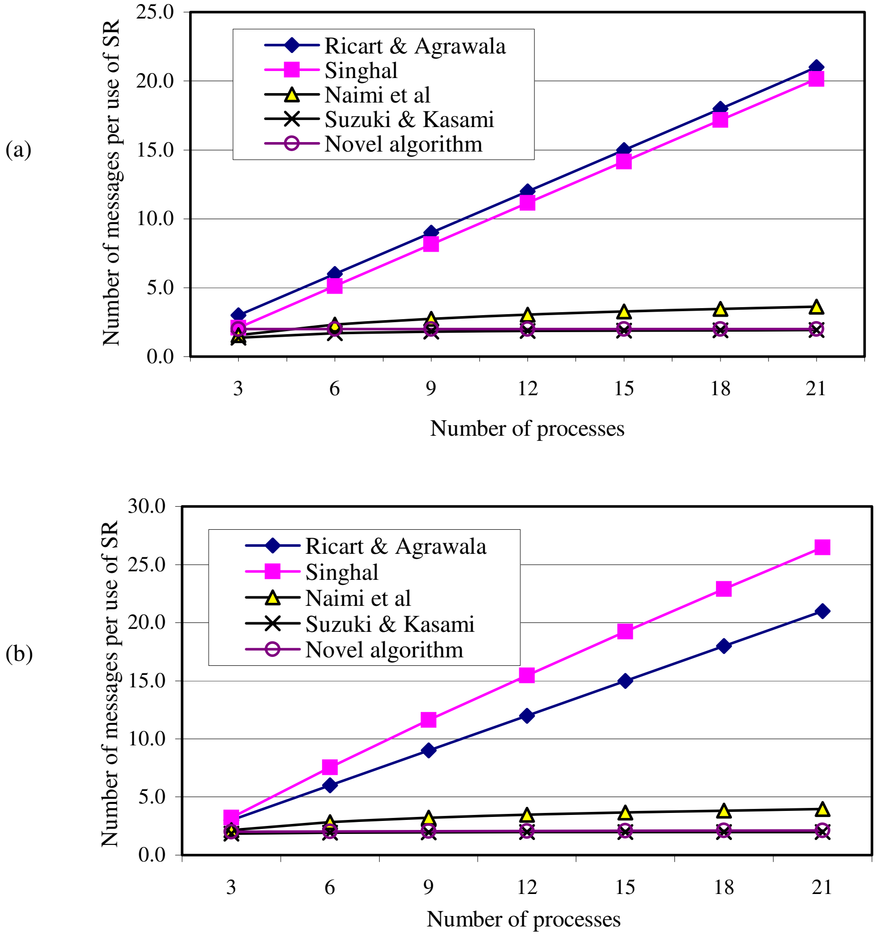


Fig. 2. Average number of messages per use of SR for low (a) and high (b) load

varied to get three different loads of SR server – low (0.1), medium (0.5) and high (0.9). To get the desired load of SR server, the value of $1/\lambda$ was set using the known expressions for a finite-population queuing system $M/M/1/N/N$ [18]. For example, with $N = 3$ processes and high load of 0.9, $1/\lambda = 635$ ms.

The following values of time-outs and random delays were fixed in simulation experiments with the proposed novel algorithm: $T_1 = 20$ ms, T_2 is uniformly distributed between $2T_1$ and 100 ms, T_3 is uniformly distributed between $T_1/2$ and 100 ms, and T_4 is uniformly distributed between $T_1/2$ and T_1.

As performance measures of each algorithm, the *average number of messages* per use of SR and the *relative unfairness* in the access of SR were used. The first performance measure reflects the communication complexity of the protocol.

The second performance measure is intended to evaluate how the algorithm is *unfair* to processes. Let r_1, r_2, .., r_N be the numbers of the use of SR by N processes in a simulation run of an algorithm. As a measure of relative unfairness of the algorithm, the expression $u = (r_{\max} - r_{\min})/r^*$ was used, where $r_{\max}$ and $r_{\min}$ are maximal and minimal values among r_i, and r^* is the total average over all r_i, $i = 1, 2, \ldots, N$.

For each load, the number of involved processes N was varied as 3, 6, 9, …, 21. The value $N_{\max} = 21$ was chosen to compare our simulation results with the published results of simulation [11] where the algorithms outlined in Section 3 used only unicast communication.

Since, from a simulation point of view, each algorithm under study can be considered as a *non-terminating* system, only steady-state performance measures are of interest. To neutralize the effect of the transient state, each simulation run was done long enough to ensure that each process accesses SR about 5000 times. This corresponds to quite narrow 95% confidence intervals (not shown in the presented results).

Graphs in Fig. 2 show, for each algorithm under study, the average number of messages per use of SR versus the number of processes, for low and high loads. Table 1 summarizes the average number of messages per use of SR, for $N = 21$ processes and three different loads – low, medium, and high. Table 2 contains information on the relative unfairness of the algorithms, for the high load of an SR server, with varying number of processes.

From the results of simulation, the following observations and conclusions can be drawn:

1. Among the algorithms, outlined in Section 3, those of Ricart & Agrawala and Suzuki & Kasami benefited most of all from the use of multicast communication. With unicast communication, the number of messages per use of SR in the Ricart & Agrawala algorithm is $2(N - 1)$ [2], while with multicast communication this number reduces to N.

2. Among the investigated algorithms, the algorithm of Suzuki & Kasami has the lowest multicast communication traffic that is less than two messages per use of SR, instead of N with unicast communication.

3. The algorithms of Singhal and Naimi *et al.* actually cannot exploit multicast communication. With high load, the Singhal's algorithm has higher communication complexity than the algorithm of Ricart & Agrawala (see Fig. 2, *b*).

4. Although the algorithm of Naimi *et al.* does not benefit from multicast communication, its average number of messages per use of SR grows very slowly with the number of processes.

Table 1. Average number of messages per use of SR, for 21 processe

Algorithm	Load		
	0.1	0.5	0.9
Ricart & Agrawala	21	21	21
Singhal	20.14	21.30	26.48
Naimi, Trehel & Arnold	3.62	3.72	3.96
Suzuki & Kasami	1.92	1.95	1.99
Novel algorithm	**2.01**	**2.05**	**2.12**

Table 2. Relative unfairness in the use of SR by processes, for high load (0.9)

Algorithm	Number of processes						
	3	6	9	12	15	18	21
Ricart & Agrawala	0.007	0.015	0.017	0.046	0.043	0.079	0.068
Singhal	0.026	0.056	0.054	0.040	0.046	0.046	0.042
Naimi *et al.*	0.013	0.026	0.012	0.037	0.031	0.047	0.060
Suzuki & Kasami	0.100	0.128	0.114	0.084	0.081	0.085	0.089
Novel algorithm	**0.018**	**0.012**	**0.013**	**0.032**	**0.023**	**0.033**	**0.034**

5. With respect to communication complexity, the proposed novel algorithm is the second best after the algorithm of Suzuki & Kasami. However, as Table 2 shows, the relative unfairness of the use of SR is much higher in the Suzuki & Kasami's algorithm than in the proposed one since, in the former algorithm, processes with low identifiers have a priority over processes with high identifiers. One more drawback of the Suzuki & Kasami's algorithm is that each its PRIVILEGE message carries varied-size queue and array depending on the number of processes, so that this algorithm does not scale well. In addition, the crash of a process that holds a PRIVILEGE will result in the complete failure of this algorithm.

6. According to Table 2, relative unfairness depends on the number of processes, but this dependence has no clear tendency. It was found that the algorithms, which use process identifiers to resolve conflicting requests, give a preference to processes with low identifiers. This is especially true for the algorithm of Suzuki and Kasami.

6 Conclusion

A few algorithms of distributed mutual exclusion were represented in terms of a finite-population queuing system and their performance study was carried out with the assumption that they use the multicast communication where possible. The algorithms in the study are those of Ricart and Agrawala, Singhal, Naimi *et al.*, Suzuki and Kasami and a novel algorithm proposed in this paper. The models of these algorithms, in the exact correspondence with their original description, were presented in terms of a class of extended Petri nets. The models were run in the simulation system Winsim based on these nets. The results of simulation provide detailed information for evaluation of performance of these algorithms and for their comparison when they use multicast communication, instead of unicast communication.

References

1. Lamport, L.: Time, Clocks, and Ordering of Events in a Distributed System. Communications of the ACM, vol.21, no.7 (1978) 558 – 565
2. Ricart, G. and Agrawala, A.K.: An Optimal Algorithm for Mutual Exclusion in Computer Networks. Communications of the ACM, vol. 24, no. 1 (1981) 9 – 17
3. Maekawa, M.A.: $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. ACM Transactions on Computer Systems, vol. 3, no. 2 (1985) 145 – 159
4. Suzuki, I. and Kasami, T.: A Distributed Mutual Exclusion Algorithm. ACM Trans. on Computer Systems, vol. 3, no. 4 (1985) 344 – 349
5. Sanders, B.A.: The Information Structure of Distributed Mutual Exclusion Algorithms. ACM Transactions on Computer System, vol. 5, no. 3 (1987) 284 – 299
6. Trehel, M. and Naimi, M.: A Distributed Algorithm for Mutual Exclusion Based on Data Structures and Fault Tolerance. Proc. 1987 Phoenix Conference on Computer and Communications, IEEE Computer Society Phoenix (1987) 35 – 39
7. Agrawal, D and El Abbadi, A.: An Efficient and Fault-Tolerant Solution for Distributed Mutual Exclusion Algorithm. ACM Trans. on Computer Systems, vol. 9, no. 1 (1991) 1 – 20
8. Singhal, M.: A Dynamic Information Structure Mutual Exclusion Algorithm for Distributed Systems. IEEE Trans. on Parallel and Distributed Systems, vol. 3, no. 1 (1992) 121 – 125.
9. Raynal, M.: A Simple Taxonomy for Distributed Mutual Exclusion. ACM Operating Systems Review, vol. 25, no. 2 (1991) 47 – 51
10. Singhal, M.: A Taxonomy of Distributed Mutual Exclusion. Journal of Parallel and Distributed Computing, vol. 18, no. 1 (1993) 94 – 101
11. Chang, Y.-I.: A Simulation Study on Distributed Mutual Exclusion. Journal of Parallel and Distributed Computing, vol. 33, no. 2 (1996) 107 – 121
12. Naimi, M., Trehel, M., and Arnold, A.: A Log (N) Distributed Mutual Exclusion Algorithm Based on the Path Reversal. Journal of Parallel and Distributed Computing, vol. 34, no. 1 (1996) 1 – 13
13. Lodha, S. and Kshemkalyani, A.: A Fair Distributed Mutual Exclusion Algorithm. IEEE Trans. on Parallel and Distributed Systems, vol. 11, no. 6 (2000) 537 – 549
14. Jayaprakash. S. and Muthukrishnan, C.R.: Permission-based Fault-tolerant Distributed Mutual Exclusion Algorithm. International Journal of Computer Systems Science and Engineering, vol. 14, no. 1 (1999) 51 – 56
15. Attiya, H. and Bortnikov, V.: Adaptive and Efficient Mutual Exclusion. Distributed Computing, vol. 15, no. 3 (2002) 177 – 189
16. Makki, K., Been, K. and Pissinou, P.: A Simulation Study of Token-Based Mutual Exclusion Algorithms in Distributed Systems. Int'l Journal in Computer Simulation, vol. 4, no. 1 (1994) 65 – 88
17. Raymond, K.: A Tree-based Algorithm for Distributed Mutual Exclusion. ACM Transactions on Computer Systems, vol. 7, no. 1 (1989) 61 – 77
18. Banks, J., Carson, J.S., Nelson, B.L. and Nicol, D.M.: Discrete-Event System Simulation, 3rd ed., Prentice-Hall (2001)

19. Hamad, A.M. and Kamal, A.E.: A survey of Multicasting Protocols for Broadcast-and-Select Single-Hop Networks. IEEE Network, vol. 16, no. 4 (2002) 36 – 48
20. Maxemchuk, N.F.: Reliable Multicast with Delay Guarantees. IEEE Communications Magazine, vol. 40, no. 9 (2002) 96 – 102
21. Hadzilacos, V. and Tueg, S.: Fault-Tolerant Broadcasts and Related Problems. In: Mullender, S. (Ed.), Distributed Systems, 2nd ed., Addison-Wesley (1993) 97 - 145
22. Kostin, A.E. and Savchenko, L.V.: Modified E-Nets for Distributed Information Processing System Performance Analysis. Automatic Control and Computer Sciences, vol. 22, no. 6 (1988) 27 – 35
23. Simulation System Winsim Based on Extended Petri Nets: User Manual, http: //www.daimi.au.dk/PetriNets/tools/db/winsim.html
24. Held, G.: Local Area Network Performance: Issues and Answers, John Wiley & Sons (1994)
25. Kostin, A. and Ilushechkina, L.: Winsim: A Tool for Performance Evaluation of Parallel and Distributed Systems. LNCS, vol. 3261, (2004) 312 – 321

A Practical Comparison of Cluster Operating Systems Implementing Sequential and Transactional Consistency

Stefan Frenz[1], Renaud Lottiaux[2], Michael Schoettner[1],
Christine Morin[2], Ralph Goeckelmann[1], and Peter Schulthess[1]

[1] Ulm University, 89069 Ulm, Germany
frenz@vs.informatik.uni-ulm.de
[2] IRISA/INRIA, 35042 Rennes, France
renaud.lottiaux@irisa.fr

Abstract. Shared Memory is an interesting communication paradigm for SMP machines and clusters. Weak consistency models have been proposed to improve efficiency of shared memory applications. In a programming environment offering weak consistency it is a necessity to worry about individual load and store operations and about proper synchronization. In contrast to this explicit style of distributed programming hared memory systems implementing strong consistency models are easy to program and consistency is implicit. In this paper we compare two representatives: Kerrighed and Plurix implementing sequential and transactional consistency respectively. Kerrighed is a single system image operating system (OS) based on Linux whereas Plurix is a native OS for PC clusters designed for shared memory operation. The measurements presented in this paper show that strong consistency models implemented at the OS level are competitive.

1 Introduction

Many projects in the distributed systems area have aimed at simplifying the development of applications. The proposed systems typically fall into two main categories: message passing and shared memory approaches. Message passing systems typically use explicit data distribution, exchange and synchronization, e.g. MPI, RMI, .NET.

Shared memory libraries implement implicit communication and can automatically guarantee consistency for all objects stored within the distributed shared memory (DSM). For the latter numerous weak memory consistency models have been proposed to minimize synchronization and improve efficiency [3]. Unfortunately, these consistency models put an additional burden on the programmer. Explicit synchronization primitives, like *acquire* and *release*, must be used very carefully and the programmer has to reason about single load and store operations.

In this paper we describe and compare two Operating Systems (OS) implementing a page-based DSM at the kernel level [1], [2], [5], [7]. Kerrighed is a single system image OS based on Linux whereas Plurix is a native OS for PC clusters designed for shared memory operation. Both OSs implement a strong consistency model.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 23 – 33, 2005.

Kerrighed implements sequential and Plurix transactional consistency. The measurements discussed in this paper show that strong shared memory consistency models can be efficient and convenient when implemented at the kernel level.

2 The Kerrighed DSM

Kerrighed is a single system image (SSI) operating system based on Linux for high performance computing on clusters. For the users and programmers it creates the illusion that a cluster is a single shared memory multiprocessor machine. The Kerrighed DSM is based on a global memory management service implementing the concept of containers.

The key idea is that a container creates the illusion to system services that the cluster physical memory is shared as in an SMP machine. In a cluster, each node executes its own operating system (OS) kernel, which can be coarsely divided into two parts: (1) system services and (2) device managers. We propose a generic service inserted between the system services and the device manager layers called *container* [7]. Containers are integrated in the core kernel thanks to *linkers*, which are software pieces inserted between existing device managers and system services and containers.

Several services, such as the virtual memory service, in a core kernel rely on the handling of physical pages. Linkers divert some functions of these services to ensure data sharing through containers. To each container is associated one or several high level linkers called *interface linkers* and a low level linker called *input/output* linker. The role of interface linkers is to divert device accesses of system services to containers while an I/O linker allows a container to access a device manager.

A container is a software object storing and sharing data between stations. A container is a kernel level mechanism and it is completely transparent to user level software. Data is stored in a container at the request of the host OS of one node and can be shared and accessed by the host OS of other cluster nodes. Pages handled by a container are stored in page frames and can be used by the host kernel as any other page frame. Container pages can for instance be mapped in a process address space.

By integrating this generic sharing mechanism into each host system, it is possible to give the illusion to the kernel that it is managing and using physically shared memory. On top of this virtual physically shared memory the traditional services offered by a standard operating system can be extended to the cluster scale. The existing OS interface is preserved while taking advantage of the low level local resource management mechanisms implemented by the standard node OS.

The containers implement a sequentially consistent memory model using a write invalidation protocol. The memory I/O linker ensures input and output of physical memory pages in and out of containers.

When a container is linked to a memory I/O linker, it becomes a memory container. The memory I/O linker is very simple since it consists in allocating and releasing page frames like the host kernel does for the management of memory segments.

Everything together provides the sight of a single SMP machine, even though the processors are distributed on several nodes in a cluster. The nodes are connected by standard hardware, which is Fast Ethernet for the measurements presented in this paper.

More details on Kerrighed DSM can be found in [8].

3 The Plurix DSM

The Plurix project implements a native distributed OS for PC clusters customized for DSM operation. Instead of using special functions for allocating data in DSM memory the Plurix DSM is managed as a heap and accessed like local memory. The benefits of a heap organization have also been identified in other systems but Plurix goes one step beyond by also storing code and runtime structures in the DSM. Thus we extend the SSI concept by storing OS, kernel, and all drivers in the DSM.

Distributed garbage collection relieves programmers from explicit memory management. Unreferenced objects can be collected very easily using the compiler-supported bookkeeping of references [4].

Because weaker consistency models are hard to program and because weak consistency might jeopardize OS integrity we have introduced a strong model called transactional consistency. Memory pages are distributed and read-only copies are replicated in the cluster. When writing to a memory page all read-only copies are invalidated and the writing node becomes the new owner of that page. Inconsistencies are avoided by synchronizing memory accesses from different nodes using our transactional consistency model [5].

In contrast to existing memory consistency models we do not synchronize memory after each write access but bundle several operations within a transaction (TA). In case of a conflict between two transactions we rely on the ability to reset changes made by a TA. This conflict resolution scheme is known in the database world as optimistic concurrency control. Optimistic concurrency control occurs in three steps: the first step is to monitor the memory access pattern of a TA. For this purpose we use the built-in facilities of the memory management unit (MMU) of the processor.

The next step is to preserve the old state of memory pages before modifications. Shadow images are created, saving the original page state before the first write operation within a TA. These shadow pages are used to restore the memory in case of a collision, as described in the next step.

During the validation phase of a terminating TA the access patterns of all concurrent TAs in the cluster are compared. In case of a conflict at least one TA is rolled back using the shadow pages otherwise the latter are discarded.

Currently, we have implemented “first-wins” collision resolution using on a circulating token. Only the current owner of the token is allowed to commit. During a commit the write-set of the TA is broadcast to all nodes in the Fast-Ethernet LAN. All nodes in the cluster compare the write set with their running TA to detect conflicts and to abort voluntarily. In future we plan to integrate other conflict resolution strategies to improve fairness.

Instead of having traditional processes and threads the scheduler in Plurix works with transactions. We have adopted the cooperative multitasking model from the Oberon system, [6]. In each station there is a central loop (the scheduler) executing a number of registered transactions with different priorities. Any TA can register further transactions. System TAs, e.g. the garbage collector, are automatically registered by the OS. Furthermore, the OS automatically encapsulates all user commands within a transaction.

Transactions should be short to minimize collision probability. For long running transactions like the tested calculations, the programmer has to split the calculation in multiple steps appropriate to transactions.

4 Comparison of Sequential and Transactional Consistency

In this section, we present a performance evaluation of Kerrighed and Plurix. To evaluate performance of both systems, we used two parallel applications: SOR and ray tracer, programmed using a shared memory paradigm.

4.1 Experimental Platform

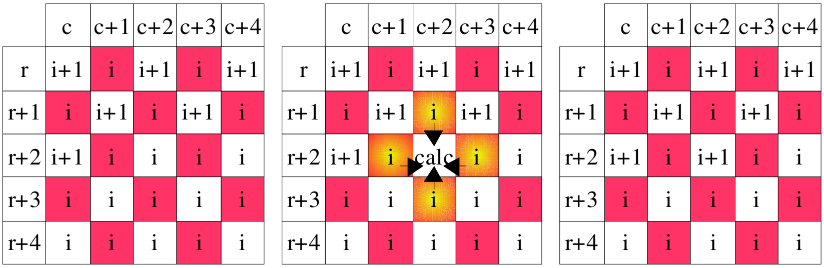
The measurements for both systems have been carried out using 12 nodes:

- Single AthlonXP 2500+ 1833 MHz
- Asus A7V8X-X mainboard with KT400A chipset
- 512 MB DDR-333-RAM
- 3Com 905 B and C Fast Ethernet network cards
- Allied Telesyn AT-8024 switch

The Plurix cluster doesn't require any cluster-outside connection, but Kerrighed needs an additional NFS-server for the shared file system, which is connected in the same manner. In both settings there was no additional traffic on the cluster-network during measurements.

4.2 Succesive-Over Relaxation (SOR)

A single n,n-matrix with randomized shared data is transmitted at the start of calculation and then changed during iterated calculation, where border elements are not changed. The matrix is red-white-coloured, and the calculation of the next iteration is done in two phases, where first all white and then all red elements are iterated. The calculation of an element (see figure 1) requires the four bordering neighbours, so all source-values are derived from the same iteration-step:



	c	c+1	c+2	c+3	c+4
r	i+1	i	i+1	i	i+1
r+1	i	i+1	i	i+1	i
r+2	i+1	i	i	i	i
r+3	i	i	i	i	i
r+4	i	i	i	i	i

before calculating i→i+1 of (c+2 | r+2)

	c	c+1	c+2	c+3	c+4
r	i+1	i	i+1	i	i+1
r+1	i	i+1	i	i+1	i
r+2	i+1	i	calc	i	i
r+3	i	i	i	i	i
r+4	i	i	i	i	i

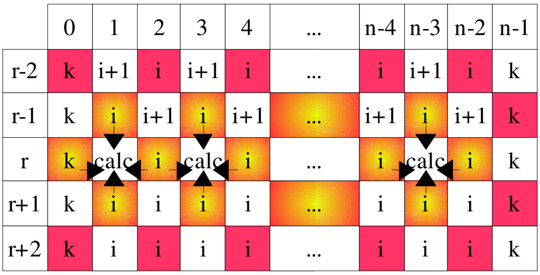
during calculating i→i+1 of (c+2 | r+2)

	c	c+1	c+2	c+3	c+4
r	i+1	i	i+1	i	i+1
r+1	i	i+1	i	i+1	i
r+2	i+1	i	i+1	i	i
r+3	i	i	i	i	i
r+4	i	i	i	i	i

after calculating i→i+1 of (c+2 | r+2)

Fig. 1. Calculation of an Element Accessing Its Four Element-Neighbors

Distribution is achieved by splitting up calculation in bands of lines. There is no intrinsic write-conflict on elements, but there is the need of synchronization after each phase of iteration because of the small overlapping read-area of one line at top and bottom frontier, where reading across the borders needs the other values to correspond to the reader's phase. Figure 2 shows the calculation of line *r*, which requires read-access of lines *r-1* and *r+1*. Synchronization is achieved with barriers both in Kerrighed and Plurix.



	0	1	2	3	4	...	n-4	n-3	n-2	n-1
r-2	k	i+1	i	i+1	i	...	i	i+1	i	k
r-1	k	i	i+1	i	i+1	...	i+1	i	i+1	k
r	k	calc	i	calc	i	...	i	calc	i	k
r+1	k	i	i	i	i	...	i	i	i	k
r+2	k	i	i	i	i	...	i	i	i	k

accessed red fields during calculating white fields
of row r (constant outline fields k with n,n-matrix)

Fig. 2. Calculating a Line Accessing Neighbor-Lines

If *r* resides at the top border of a node's band, line *r-1* is inside the band of the previous node, and if *r* resides at the bottom border of a node's band, line *r+1* is inside the band of the next node. These nodes will calculate their elements in parallel, so on page-based distribution such as with Kerrighed and Plurix, there is read access to elements that reside on pages, which are written by another station in the same phase. But as all nodes start calculating with their first line of their band, this is not an indispensable bottle neck: calculation takes long enough to disperse accesses to first line of node t+1 and to last line of node t, so nodes running Kerrighed with a MESI-like protocol for pages do not get in the way of nodes. Within Plurix reads and writes occur atomically during the commit of a TA, so each phase is split up in two sub-phases for bisection of calculated bands. Thereby the borders are not crossed mutual, so the nodes do not read and write concurrently to the same line in the same phase.

Because of synchronization after each phase and because all nodes calculate the same amount of phases, all nodes finish after nearly the same time. The synchronization is not very data intensive and therefore mainly depending on network latency. In contrast the network bandwidth is important for data exchange after synchronization, because for each node two rows of the matrix (for the measurements one row is between 8 and 28 kilobytes) have to be transmitted.

SOR is implemented from scratch both for Plurix (java compiled with the Plurix Java Compiler) and Kerrighed (C compiled with gcc) based upon the Splash-II-Suite.

4.3 Ray Tracer

The calculation of the ray tracer starts with an empty shared result-matrix as container for the image. Each node calculates each element independently from other nodes or elements based upon a shared scene definition supporting spheres and triangles with colored and reflective surfaces as well as multiple and different light sources. For this measurements, the scene-definition (see picture RAY) contains 99 spheres and 8 triangles illuminated by three light sources. Each result pixel can be calculated without information about other pixels and as a consequence there is no need of synchronization and the result-matrix is write-only during calculation. Apart from the transfer of the scene definition and of the accesses to the result-matrix there is no intrinsic communication and therefore the ray-tracer demonstrates the limits of system distribution and system dependent scaling. The ray tracer is implemented from scratch both for Plurix (using the Plurix Java Compiler) and Kerrighed (C compiled with gcc) based upon project 5 of class 6.837 at MIT [14].

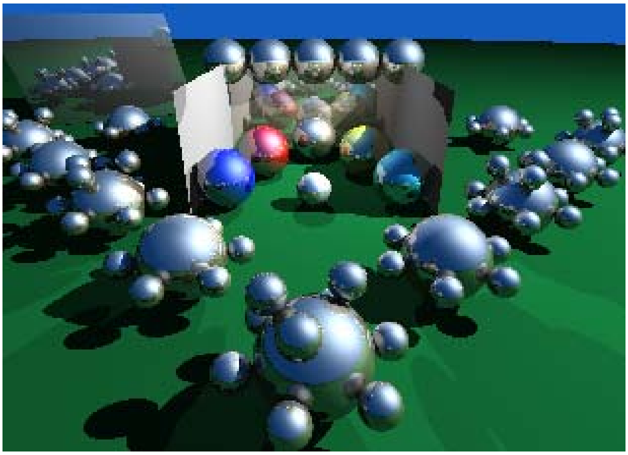


Fig. 3. Calculated Scene

4.4 Experimental Results with SOR

The SOR-algorithm has been tested on both systems using 1, 2, 4, 8 and 12 nodes and with the following matrix sizes: 2048x2048, 3584x3584, 4096x4096, 5068x5068, 6144x6144, 7168x7168. Measurement results are presented in tables SSK and SSP. Figure 4 presents the speed-up of the SOR algorithm on Kerrighed. Figure 5 presents the speed-up on Plurix.

We can observe on fig. 4 and fig. 5 that the matrix size has little impact on speed-up for Kerrighed but more so for Plurix, which will be explained later. The best speed-up achieved is 5.8 (Kerrighed) respectively 6.7 (Plurix) on 12 nodes, which is far from ideal.

The main reason for less than ideal performance on both systems is the large gap between processor speed and network bandwidth. While the processors are very powerful, the network is comparatively slow. The SOR algorithm exchanges border rows between each phase of the computation inducing communications. On the available hardware platform, the network-bandwidth/processor-speed ratio is not good enough to reach high speed-ups. Some experiments using a faster network such as gigabit Ethernet or Myrinet would be of interest.

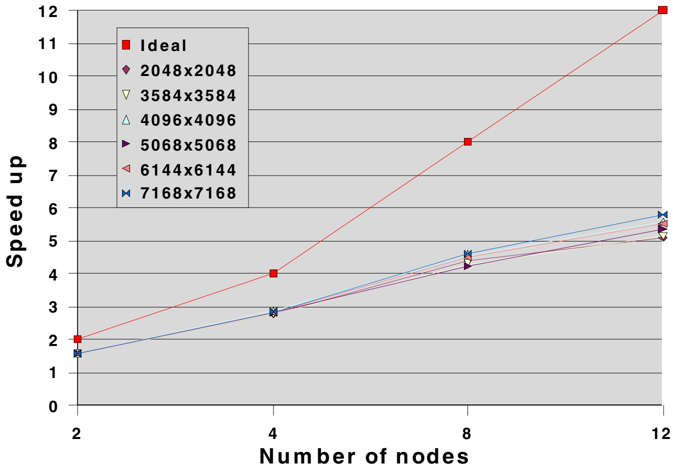


Fig. 4. Speed-up with SOR on Kerrighed

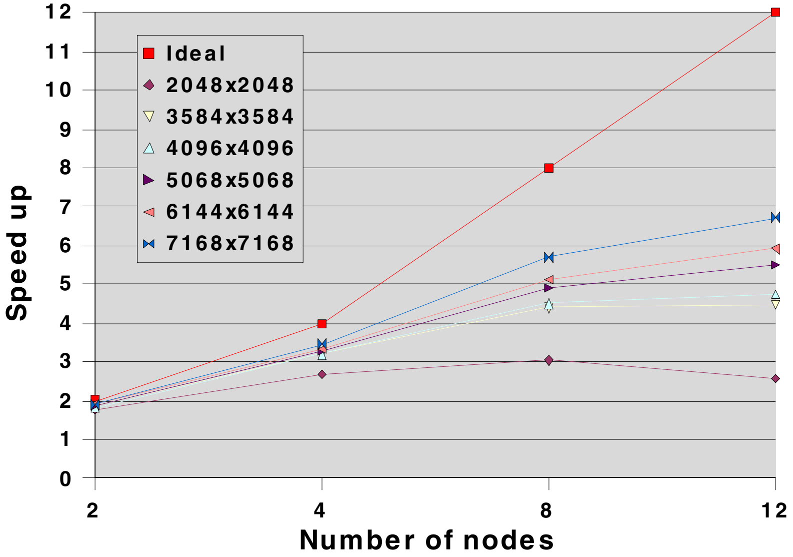


Fig. 5. Speed-up with SOR on Plurix

The main difference between the systems is not the difference in speed-up for a large matrix but the different behavior for a small matrix: even with the smallest matrix Kerrighed has a speed-up on 12 nodes similar to the largest matrix, whilst Plurix has a point of reversal on 8 nodes. This is because of the synchronization mechanisms used on Plurix, that are expensive compared to the few calculations that have to be done for small matrices. The barrier implementation on Plurix is in a not fully developed state and still subject of research.

4.5 Experimental Results with Ray Tracer

The ray tracer has been tested on both systems using 1, 2, 3, 4, 6, 8, 10 and 12 nodes and with the following image sizes: 2048x1536, 4096x3072 and 5792x4344. Measurement results are presented in tables RSK and RSP. Figure 6 presents the speed-up of the ray tracer on Kerrighed. Figure 7 presents the speed-up on Plurix.

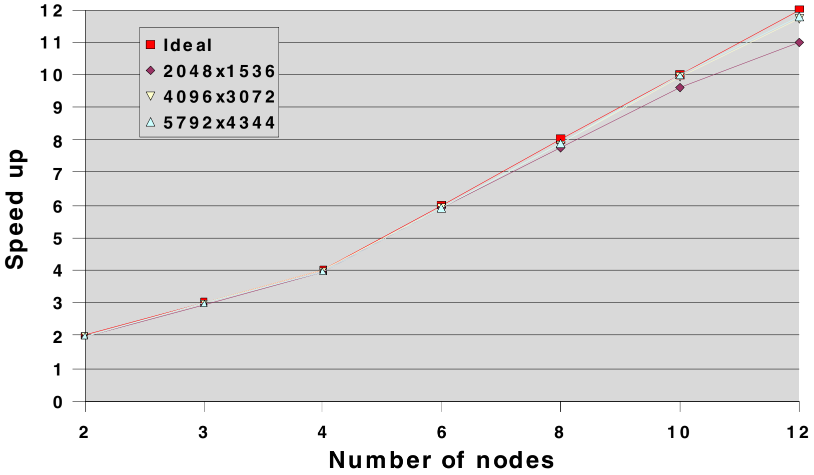


Fig. 6. Speed-up with ray tracer on Kerrighed

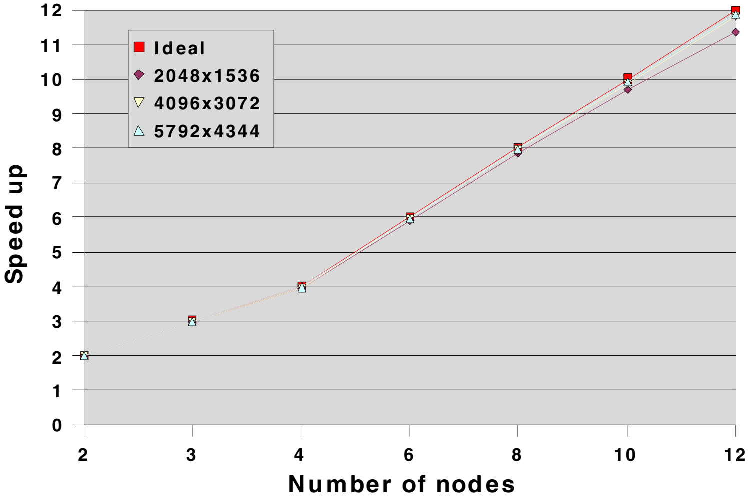


Fig. 7. Speed-up with ray tracer on Plurix

We can observe in figure 6 and in figure 7 that the image size has little impact on speed-up. The best speed-up achieved is 11.77 (Kerrighed) respectively 11.88 (Plurix) on 12 nodes, which is fairly good result. The network is not the bottleneck for the ray tracer application because there is much more calculation than communication.

Furthermore, as there is no need for barrier-synchronization between steps of calculation, Plurix can fully utilize its optimistic synchronization model within the transactional consistency, because even the first write access to a page does not require the affirmation of all other nodes.

4.6 Comparison of Kerrighed and Plurix

Kerrighed and Plurix are completely different systems: Kerrighed uses Linux and NFS to share files, whilst Plurix is a operating system from scratch without the need for any file system. Both Kerrighed and Plurix use page-based distributed shared memory, but Kerrighed uses an extended MESI protocol to exchange pages, whereas Plurix always transfers the most recently committed version of a page and uses the semantic group of a transaction to invalidate multiple pages.

Nevertheless, both systems perform almost linear with the ray tracer up to 12 nodes and show mature communication models, which are hidden from the application programmer completely, even though system knowledge will of course help in writing well performing applications as in any.

The SOR-algorithm uses high-volume communication in comparison to the time needed for calculation, as a consequence Fast Ethernet becomes the bottleneck.

5 Related Work

Numerous DSM projects have implemented a global memory management service on top of an existing OS, e.g. Solaris, Linux, and Windows NT. IVY was the first page-based DSM implementation (sequential consistency) followed by others implementations with weaker consistency models, e.g. TreadMarks (lazy release consistency), [9], [11]. To the researcher and to the students from these user-level systems provide important insight, especially about the relative merits of different consistency models [3]. But this approach introduces many programming constraints and limits performance. Specific run-time functions might be called by the programmer to data in the DSM or special storage classes might be defined.

The Single System Image idea has also been addressed by several projects in the past. The Sprite OS for example is written from scratch and provides a distributed file system and a process migration facility [12]. But Sprite does not allow to migrate threads and does not implement a global memory management mechanism.

The Mosix project extends Linux with a kernel-level process-migration facility. However, it does not provide any data sharing mechanism. Thus, processes can n ot share memory and threads can not be migrated in Mosix [13].

Plurix is the first OS tailored to a transactional DSM. Furthermore, there is no other system utilizing the DSM heap to distribute both data and code. Transactional consistency in the context of distributed computing is also proposed in [11]. The ideas discussed are similar to the transactiopnal consistency in Plurix but the authors simulate a new CPU design for SMP machines rather than a cluster implementation.

6 Conclusions

The comparison of sequential and transactional consistency in Kerrighed and Plurix respectively shows that both perform adequately in spite of their strong consistency models. Efficiency is ensured in both systems by implementing the DSM at the kernel level and by avoiding the overhead of expensive context switches. Furthermore, Plurix benefits from the fact that several write operations are bundled into one transaction.

The SOR measurements revealed noticeable costs for the barrier synchronization in Plurix caused by a currently not optimal barrier implementation. Nevertheless, it is encouraging that a DSM organized as a heap storing code and data (Plurix) can compete with a traditional DSM approach (Kerrighed) allocating only dedicated data in DSM. Experiments with faster networks like Gigabit Ethernet, Myrinet, and Infiniband are planed in future work.

Acknowledgement

This work has been funded by the German DAAD within the PROCOPE program and the French ministry of foreign affairs. The work on Kerrighed has been partly financed by the Direction Générale de l'Armement (DGA) for the COCA project and by EDF R&D.

References

1. www.kerrighed.org
2. www.plurix.de
3. D. Mosberger, “Memory Consistency Models”, *ACM Operating Systems Review*, 27(1), 18-26, January 1993.
4. R. Goeckelmann, S. Frenz, M. Schoettner, P. Schulthess, “Compiler Support for Reference Tracking in a type-safe DSM”, Proceedings of the Joint Modular Languages Conference Klagenfurt, Austria, 2003.
5. M. Wende, M. Schoettner, R. Goeckelmann, T. Bindhammer, P. Schulthess, “Optimistic Synchronization and Transactional Consistency”, Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, Berlin, Germany, 2002.
6. N. Wirth and J. Guteknecht, *Project Oberon - The Design of an Operating System and Compiler*, Addison-Wesley, 1992.
7. Renaud Lottiaux and Christine Morin, “Containers: A Sound Basis For a True Single System Image”, in: Proceeding of IEEE International Symposium on Cluster Computing and the Grid (CCGrid '01), Brisbane, Australia, 2001.
8. Geoffroy Vallée, Renaud Lottiaux, Louis Rilling, Jean-Yves Berthou, Ivan Dutka-Malhen, and Christine Morin, “A Case for Single System Image Cluster Operating Systems: Kerrighed Approach “, Parallel Processing Letters, 13(2), June 2003.
9. K. Li., “IVY: A Shared Virutal Memory System for Parallel Computing”, in: Proceedings of the International Conference on Parallel Processing, 1988.
10. P. Keleher et al., “TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems”, in: USENIX Winter 1994, 1994.

11. Lance Hammond, Brian D. Carlstrom, Vicky Wong, Ben Hertzberg, Mike Chen, Christos Kozyrakis, and Kunle Olukotun, "Programming with Transactional Coherence and Consistency ", in: Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, USA, 2004.
12. John Ousterhout, A. Cherenson, Fred Douglis, M. Nelson, Brent Welch, "The Sprite network operating system", Computer, 21(2):23-36, February, 1988.
13. Amnon Barak, S. Guday, Richard Wheeler, "The MOSIX Distributed Operating System", volume 672 of Lecture Notes in Computer Science, Springer, 1993.
14. http://graphics.csail.mit.edu/classes/6.837/F01/Project05/project5.html

Clock Synchronization State Graphs Based on Clock Precision Difference

Ying Zhao[1], Wanlei Zhou[2], Yingying Zhang[1], E.J. Lanham[2], and Jiumei Huang[1]

[1] School of Information Science and Technology,
Beijing University of Chemical Technology, Beijing, 100029, P.R. China
{zhaoy, huangjm}@mail.buct.edu.cn
[2] School of Information Technology, Deakin University,
211 Burwood HWY, Burwood, VIC 3125, Australia
{wanlei, lanham}@deakin.edu.au

Abstract. Consistent and stable global states of clock synchronization are very important in distributed and parallel systems. This paper presents an innovative strategy and method to obtain stable global clock synchronization state graphs in asynchronous Internet environments. Our model will introduce the concept of clock precision difference as a means to evaluate running states of all clocks in this system and make this system self-adaptive well. Finally, we introduce the concept of clock precision difference into global states analysis of clock synchronization and construct clock synchronization state graphs in order to evaluate distributed clock synchronization states. We also present detailed simulations of the strategy and mathematical analysis used on real Internet environments.

Keywords: Clock Synchronization, Precision Difference, Self-Adaptive, State graphics.

1 Introduction

Computing technologies and network technologies have developed at an explosive, but steady rate in the last decade, and many applications have been built on distributed network environments. Nevertheless, distributed and parallel systems without a global clock are common nowadays. PC clusters, the GRID, industrial process control systems, and mobile communications are all examples of this, as without an identical clock system, these applications cannot perform their tasks well. Consistent Global States (CGS) proved to be useful in this field [1]. Since Lamport published a paper [2] to introduce logical clocks and the ordering of events in 1978, logical clocks have been playing a dominant role in distributed clock synchronization systems. In the process of the implementation, we will use a software logical clock concept in order to separate physical clocks, and easily adjust the logical clock. Sometimes we refer to the logical clock as a virtual clock.

How to describe and evaluate a clock synchronization system in distributed environments has been a very difficult problem. In order to solve this classical problem, we introduce the concept of synchronization state graphs based on clock precision differences. We have also shown that it not only describes a clock synchronization

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 34 – 39, 2005.

system clearly, but also provides in a uniform manner with both uncertainty of transmission times and the uncertainty due to the clock drifts. Based on this type of graph, each node in this system will have a global view of this clock synchronization system and can easily be observe the node failure.

The paper is organized as follows: Section 2 introduces the related work about clock synchronization. A linear mathematical trend analysis for clock precision differences is proposed and an important theorem and a conclusion about clock synchronization are presented in section 3. Section 4 proposes the important concept of clock synchronization state graphs and gives a detailed definition. A real simulation of clock synchronization state graph is discussed in section 5. Finally, Section 6 describes some extensions of this model and conclusions.

2 Related Works

In the research area of Clock Synchronization, algorithms and models should satisfy some synchronization conditions, such as: bounded skew and bounded rate of communication. However, due to the communication uncertainty it is not reasonable to assume that in practical applications, the skew and rate of communication are bounded.

As there are different clock precisions for nodes, the issue of clock synchronization occurs. If an instantaneous time is applied to a synchronization process, the message delay must be considered seriously. However, questions such as: how to calculate this delay, and how to predict every delay in the system, are very complicated and difficult to solve. Some papers use round trip delay to estimate the transmission delay, but the irregular feature of networks frustrates this solution[3],[4]. In this paper, we first use one way timed transmission to estimate the precision difference, and then use a mathematical analysis method to estimate clock precision difference in order to tune the clock in a local node.

With a distributed clock, synchronizing, monitoring, and describing its state is an important task. Normally, clock synchronization graphs can be viewed as an extension of the graphs used by Lamport to describe the execution of completely asynchronous system [3]. But Lamport's graphs are unweighted, and the main property of interest regarding a pair of points is whether one is reachable from the other. The dissertation [5] considered systems with clocks, and define graphs which are weighted. The main property of interest regarding two points is the distancebetween them.

Because of the imperfect manufacturing process of oscillators, each clock has a slightly different drift rate and therefore each clock will have a slightly precision difference. This precision difference is relatively stable and can be used in clock synchronization state graphs as a weighted value. The global state graphs of clock synchronization can not only provide consistent views of clocks running in this system, but can also evaluate a clock's faults easily.

3 Clock Precision Difference Evaluation

On the surface, the occurrence of this transmission delay is random; therefore it is difficult to measure. In fact, if the previous timestamp arrives at destination late, then the next timestamp will have a tendency to come early. We can use the coming timestamps to get its trend.

From the perspective of the slave, they have their own clocks, and know the sending interval of T_{period}. If a slave clock has the same precision as the master clock, then the arrival interval calculated by the slave node will waver near the interval of T_{period}. Hence, if there is a long interval this time, a short interval shall be expected next time. Even though, in some cases, this next interval may in fact be longer than previous one. This also means that the next or future interval shall be expected shorter time period than the interval we are currently expecting. In the long run, its trend line should be a horizontal line.

Based on above analysis, we can create the following theorem.

Theorem 1. We assume that a master node sends a timestamp per T_{period} interval, and the drift rate of the slave clock meets assumption 2. Based on the samples of the slave nodes collected, we can achieve a linear trend formula, which is shown as follows:

$$y = b_0 + b_1 x \tag{1}$$

Here, b1, as a slope, denotes the drift speed of the slave clock, and has a close relationship with drift rate τ_i of this clock. As the number of samples η increases, the following formula holds:

$$\tau_i = \lim_{\eta \to \infty} (b_1 / T_{period}) \tag{2}$$

Based on above theorem and analysis, a conclusion is given as follows:

Conclusion 1. If we have calculated the slope b_1 of linear equation from the sampling data, then the following statements prove correct:

If $b_1<0$, the *ith* slave clock runs slower than the reference clock.
If $b_1=0$, the *ith* slave clock runs same speed as the reference clock.
If $b_1>0$, the *ith* slave clock runs faster than the reference clock.

4 Clock Synchronization State Graphs

The introduction of clock synchronization state graphs based on clock precision difference is an important contribution to the research of distributed clock synchronization system. Abstract graph-theoretic methods allow us to analyze clock synchronization problems more practically. The network is modeled as a collection of links which facilitates communication amongst computers. From the general point of view, the traditional clock synchronization graphs usually use transmission delays as weight values of edges. Even if the clock nodes run correctly, the state graphs for clock synchronization are still difficult to reach a stable state, due to communication uncertainty and jittering of networks. In this paper, we extend the concept of clock synchronization graphs, and present an innovative concept of clock state graphs. This means that each node in the clock synchronization system has a consistent view of all clocks running states in the system. The definition of this concept is described as following:

Definition 1. The clock synchronization state graph is a pair *(G, local_time)*, where *G=(V,E)* is a weighted bidirectional directed graph with *arc<p,q>* $\in E$ if and only if *arc<p,q>* $\in E$, and local time includes logical time (*LT*) and physical time (*PT*) with each point $p \in E$. For any two points $p,q \in V$, we define *arc〈p, q〉* as relative clock precision difference τ_{pq} of p node compared to q node.

Each node has a clock synchronization state graph which shows the global view of states in the clock synchronization system. It not only includes a graph G, but also is labeled by the logical time and physical time. The following is an example of the clock synchronization state graph with two nodes *p* and *q*.

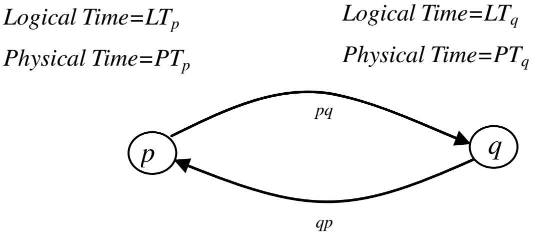


Fig. 1. The clock synchronization state graph with two nodes *p* and *q*

Where the weight value of arc <q,p> is τ_{pq}; the weight value of arc <p,q> is τ_{qp}. The value τ can be computed by Theorem 1.

Property 1. Let p,q,r be any three points of the state graph. If the clock synchronization state graph steps in a stable state, then the draft rate on the edge meets the tranmissive rule:

$$\tau_{qp} = \tau_{qr} + \tau_{rp} \qquad (3)$$

In order to prove this reasoning, we have designed a simulation environment. The simulation environment is presented in the next section.

5 Simulation

Our simulation is based on the Internet environment in our campus. There are 4 nodes communicating with each other through the Internet. The system uses logical time to exchange messages. In order to get results quickly, we use a program in each node to slow down or speed up the running of the clock. The running state of each clock is described as follows:

Node 1: We think this node is a standard clock and runs normally. Its logical clock is equal to physical clock.

Node 2: This node slows down 1 ms per 30 seconds.

Node 3: This node speeds up 1 ms per 30 seconds.

Node 4: This node speeds up 2 ms per 30 seconds.

For the interval of 30 seconds in logical time, every node broadcasts a timestamp to other nodes. After we collect 2000 samples, each node should know its running precision differences relating to other nodes. For example node1, it has three trend equations.

Node2: Y=1.000x-1.8109 $\tau_{21}=1/30000$;
Node3: Y=-1.000x+2.3304 $\tau_{31}=-1/30000$;
Node4: Y=-2.000x+1.4963 $\tau_{41}=-2/30000$;

As node1, it thinks that other nodes should be standard clock and send out a time stamp every 30 seconds.

After the system runs near 13 hours, a detailed clock synchronization state graph for this simulation is described as follows:

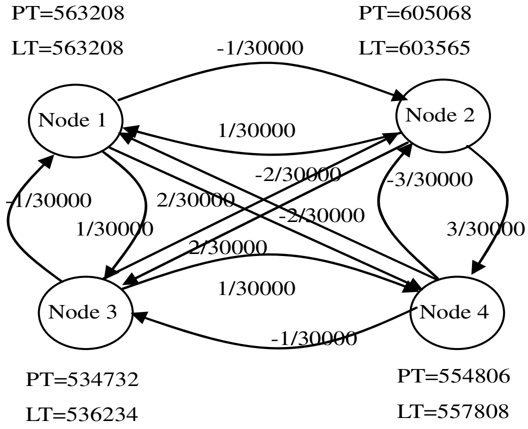


Fig. 2. The clock synchronization state graph

If this system meets assumption 2 and all clocks have a linear drift rate, then we can achieve a stable state graph of clock synchronization quickly and easily.

In order to validate property 1, we consider a relative draft rate of node 2 compared to node 4. We have:

$$\tau_{42}=\tau_{43}+\tau_{32}=(-1/30000)+(-2/30000)= -3/30000$$

6 Conclusions

This paper presents an innovative concept of clock synchronization state graph to describe and analyze clock synchronization systems. All edges are directed and labeled by the relative clock precision differences, rather than transmission delays or instantaneous values. In order to improve and stabilize the differences of clock precision, a continuous communication model and the method of trend analysis are adopted in this paper. The application of clock precision difference not only reduces the effect of network delay, but also provides important information for local clocks. Based on this information, a local clock cannot only predict the active-synchronization time, but can also apply a self-adaptive state when its connection crashes.

From a long-term viewpoint, more information of precision differences means more accurate clock synchronization and state graphs**.**

References

1. Janusz Borkowski, Hierarchical Detection of Strongly Consistent Global States, Proceedings of the Third International Symposium on Parallel and Distributed Computing /Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks(ISPDC/HeteroPar'04), July 05-07, 2004, Cord, Ireland. pp. 256-261
2. L. Lamport, Time, Clocks, and the ordering of events in a distributed system, Communications of the ACM, vol 21, no 7,1978. pp. 558-564
3. Masato T., Tetsuya T., Yuji O., Estimation of Clock Offset From One-Way Delay Measurement on Asymmetric Paths, Proceeding of the 2002 Symposium on Application and the Internet (SAINT'02w), pp. 126-133, Nara, Japan,2002
4. http://www.faqs.org/rfcs/rfc2681.html
5. Boaz Patt, A Theory of Clock Synchronization, Massachusetts Institute of Technology, 1994, USA

A Recursive-Adjustment Co-allocation Scheme in Data Grid Environments*

Chao-Tung Yang[1,**], I-Hsien Yang[1], Kuan-Ching Li[2], and Ching-Hsien Hsu[3]

[1] High Performance Computing Laboratory,
Department of Computer Science and Information Engineering,
Tunghai University, Taichung 40704 Taiwan ROC
ctyang@thu.edu.tw, g922906@thu.edu.tw
[2] Parallel and Distributed Processing Center,
Department of Computer Science and Information Management,
Providence University, Taichung 43301 Taiwan ROC
kuancli@pu.edu.tw
[3] Department of Computer Science and Information Engineering,
Chung Hua University, Hsinchu 300, Taiwan
chh@chu.edu.tw

Abstract. The co-allocation architecture was developed in order to enable parallel downloads of datasets from multiple servers. Several co-allocation strategies have been coupled and used to exploit rate differences among various client-server links and to address dynamic rate fluctuations by dividing files into multiple blocks of equal sizes. However, a major obstacle, the idle time of faster servers having to wait for the slowest server to deliver the final block, makes it important to reduce differences in finish time among replica servers. In this paper, we propose a dynamic co-allocation scheme, namely *Recursive-Adjustment Co-Allocation* scheme, to improve the performance of data transfer in Data Grids. Our approach reduces the idle time spent waiting for the slowest server and decreases data transfer completion time.

Keywords: Data Grid, Globus, GridFTP, Co-allocation, Recursive-adjustment, Data transfer.

1 Introduction

Data Grids aggregate distributed resources for solving large-size dataset management problems [1, 2, 4, 7, 9]. Most Data Grid applications execute simultaneously and access large numbers of data files in the Grid environment. Certain data-intensive scientific applications, such as high-energy physics, bioinformatics applications and virtual astrophysical observatories, entail huge amounts of data that require data file management systems to replicate files and manage transfers and distributed data ac-

[*] This paper is supported in part by National Science Council, Taiwan ROC, under grants no. NSC92-2213-E-029-025, NSC92-2119-M-002-024, NSC93-2119-M-002-004, and NSC93-2213-E-029-026.

[**] Corresponding author.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 40 – 49, 2005.

cess. The data grid infrastructure integrates data storage devices and data management services into the grid environment, which consists of scattered computing and storage resources, perhaps located in different countries/regions yet accessible to users [2, 9].

Replicating popular content in distributing servers is widely used in practice [11, 13, 15]. Recently, large-scale, data-sharing scientific communities such as those described in [1, 4] used this technology to replicate their large datasets over several sites. Downloading large datasets from several replica locations may result in varied performance rates, because the replica sites may have different architectures, system loadings, and network connectivity. Bandwidth quality is the most important factor affecting transfers between clients and servers since download speeds are limited by the bandwidth traffic congestion in the links connecting the servers to the clients[17, 18].

One way to improve download speeds is to determine the best replica locations using replica selection techniques [15]. This method selects the best servers to provide optimum transfer rates because bandwidth quality can vary unpredictably due to the shared nature of the internet. Another way is to use co-allocation technology [13] to download data. Co-allocation of data transfers enables the clients to download data from multiple locations by establishing multiple connections in parallel. This can improve the performance compared to the single-server cases and alleviate the internet congestion problem [13]. Several co-allocation strategies were provided in the previous work [13]. An idle-time drawback remains since faster servers must wait for the slowest server to deliver its final block. Therefore, it is important to reduce the differences in finish time among replica servers.

In this paper, we propose a dynamic co-allocation scheme based on co-allocation Grid data transfer architecture called *Recursive-Adjustment Co-Allocation* that reduces the idle time spent waiting for the slowest server and improves data transfer performance. Experimental results show that our approach is superior to previous methods and achieved the best overall performance.

The remainder of this paper is organized as follows. Related studies are presented in Section 2 and the co-allocation architecture is introduced in Section 3. Our research approaches are outlined in Section 4, and experimental results and a performance evaluation of our scheme are presented in Section 5. Section 6 concludes this research paper.

2 Related Work

Data grids consist of scattered computing and storage resources located in different countries/regions yet accessible to users [7]. In this study we used the grid middleware Globus Toolkit [8, 10, 12] as the data grid infrastructure. The Globus Toolkit provides solutions for such considerations as security, resource management, data management, and information services. One of its primary components is MDS [5, 8, 10, 12, 20], which is designed to provide a standard mechanism for discovering and publishing resource status and configuration information. It provides a uniform and flexible interface for data collected by lower-level information providers in two modes: static (e.g., OS, CPU types, system architectures) and dynamic data (e.g., disk availability, memory availability, and loading). And it uses GridFTP [1, 8, 12], a reliable, secure, and efficient data transport protocol to provide efficient management and transfer of terabytes or petabytes of data in a wide-area, distributed-resource environment.

As datasets are replicated within Grid environments for reliability and performance, clients require the abilities to discover existing data replicas, and create and register new replicas. A Replica Location Service (RLS) [3, 15] provides a mechanism for discovering and registering existing replicas. Several prediction metrics have been developed to help replica selection. For instance, Vazhkudai and Schopf [14, 16, 17] used past data transfer histories to estimate current data transfer throughputs.

In our previous work [19], we proposed a replica selection cost model and a replica selection service to perform replica selection. In [13], the author proposes a co-allocation architecture for co-allocating Grid data transfers across multiple connections by exploiting the partial copy feature of GridFTP. It also provides *Brute-Force*, *History-Base*, and *Dynamic Load Balancing* for allocating data block. *Brute-Force Co-Allocation* works by dividing file sizes equally across available flows without addressing bandwidth differences among the various client-server links. The *History-based Co-Allocation* scheme keeps block sizes per flow proportional to predicted transfer rates.

The *Conservative Load Balancing* dynamic co-allocation strategy divides requested datasets into “k” disjoint blocks of equal size. Available servers are assigned single blocks to deliver in parallel. When a server finishes delivering a block, another is requested, and so on, till the entire file is downloaded. The loadings on the co-allocated flows are automatically adjusted because the faster servers will deliver more quickly providing larger portions of the file. The *Aggressive Load Balancing* dynamic co-allocation strategy presented in [13] adds functions that change block size deliveries by: (1) progressively increasing the amounts of data requested from faster servers, and (2) reducing the amounts of data requested from slower servers or ceasing to request data from them altogether.

The co-allocation strategies described above do not handle the shortcoming of faster servers having to wait for the slowest server to deliver its final block. In most cases, this wastes much time and decreases overall performance. Thus, we propose an efficient approach called *Recursive-Adjustment Co-Allocation* and based on a co-allocation architecture. It improves dynamic co-allocation and reduces waiting time, thus improving overall transfer performance.

3 Co-allocation Architecture

Figure 1 shows the co-allocation of Grid Data transfers, which is an extension of the basic template for resource management [6] provided by Globus Toolkit. The architecture consists of three main components: an information service, broker/co-allocator, and local storage systems. Applications specify the characteristics of desired data and pass the attribute description to a broker. The broker queries available resources and gets replica locations from information services [5] and replica management services [15], and then gets a list of physical locations for the desired files.

The candidate replica locations are passed to a replica selection service [15], which was presented in a previous work [19]. This replica selection service provides estimates of candidate transfer performance based on a cost model and chooses appropriate amounts to request from the better locations. The co-allocation agent then downloads the data in parallel from the selected servers. In this research, we use

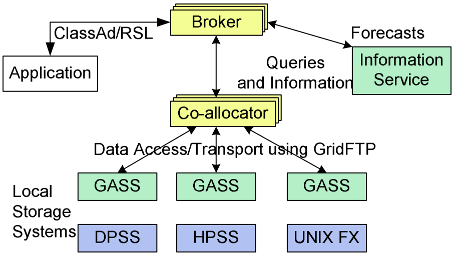


Fig. 1. Data Grid co-allocation architecture

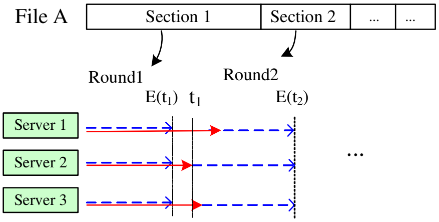


Fig. 2. The adjustment process.

GridFTP [1, 8, 12] to enable parallel data transfers. GridFTP is a high-performance, secure, reliable data transfer protocol optimized for high-bandwidth wide-area networks. Among its many features are security, parallel streams, partial file transfers, third-party transfers, and reusable data channels. Its partial file transfer ability allows files to be retrieved from data servers by specifying the start and end offsets of file sections.

4 Dynamic Co-allocation Strategy

Dynamic co-allocation, described above, is the most efficient approach to reducing the influence of network variations between clients and servers. However, the idle time of faster servers awaiting the slowest server to deliver the last block is still a major factor affecting overall efficiency, which *Conservative Load Balancing* and *Aggressive Load Balancing* [13] cannot effectively avoid. The approach proposed in the present paper, a dynamic allocation mechanism called "*Recursive-Adjustment Co-Allocation*" can overcome this, and thus, improve data transfer performance.

4.1 Recursive-Adjustment Co-allocation

Recursive-Adjustment Co-Allocation works by continuously adjusting each replica server's workload to correspond to its real-time bandwidth during file transfers. The goal is to make the expected finish time of all servers the same. As Figure 2 shows, when an appropriate file section is first selected, it is divided into proper block sizes according to the respective server bandwidths. The co-allocator then assigns the blocks to servers for transfer. At this moment, it is expected that the transfer finish time will be consistent at $E(T_1)$. However, since server bandwidths may fluctuate during segment deliveries, actual completion time may be dissimilar (solid line, in Figure 2). Once the quickest server finishes its work at time T_1, the next section is assigned to the servers again. This allows each server to finish its assigned work-load by the expected time at $E(T_2)$. These adjustments are repeated until the entire file transfer is finished.

The *Recursive-Adjustment Co-Allocation* process is illustrated in Figure 3. When a user requests file *A*, the replica selection service responds with the subset of all available servers defined by the maximum performance matrix. The co-allocation service gets this list of selected replica servers. Assuming *n* replica servers are selected, S_i

denotes server "*i*" such that $1 \leqq i \leqq n$. A connection for file downloading is then built to each server.

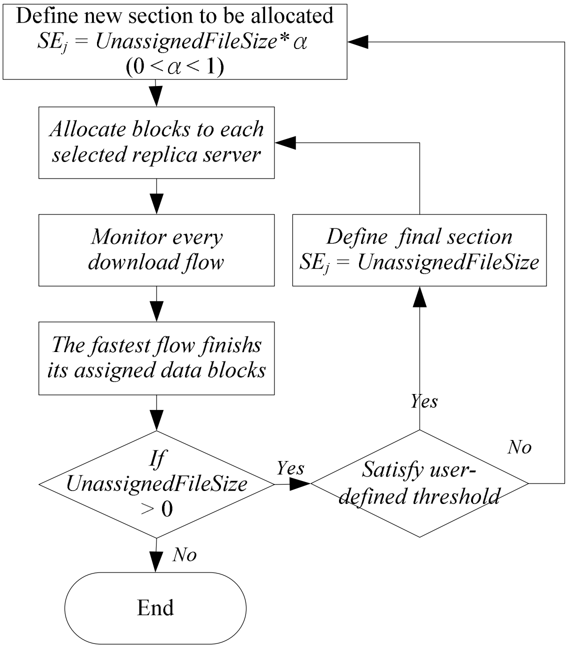

Define new section to be allocated
$SE_j = UnassignedFileSize * \alpha$
$(0 < \alpha < 1)$
Allocate blocks to each selected replica server
Monitor every download flow
The fastest flow finishs its assigned data blocks
If UnassignedFileSize > 0
Yes
No
End
Satisfy user-defined threshold
Yes
No
Define final section
$SE_j = UnassignedFileSize$

Fig. 3. The Recursive-Adjustment Co-Allocation process

The *Recursive-Adjustment Co-Allocation* process is as follows. A new section of a file to be allocated is first defined. The section size, "SE_j", is:

$$SE_j = UnassignedFileSize \times \alpha, \ (0 < \alpha < 1), \tag{1}$$

where SE_j denotes the section *j* such that $1 \leqq j \leqq k$, assuming we allocate *k* times for the download process, and thus, there are *k* sections, while T_j denotes the time section *j* allocated. *UnassignedFileSize* is the portion of file *A* not yet distributed for downloading; initially, *UnassignedFileSize* is equal to the total size of file *A*. α is the rate that determines how much of the section remains to be assigned.

In the next step, SE_j is divided into several blocks and assigned to "*n*" servers. Each server has a real-time transfer rate to the client of B_i, which is measured by the Network Weather Service (NWS) [18]. The block size per flow from SE_j for each server "*i*" at time T_j is:

$$S_i = (SE_j + \sum_{i=1}^{n} UnFinishSize_i) \times B_i \Big/ \sum_{i=1}^{n} B_i - UnFinishSize_i \tag{2}$$

where $UnFinishSize_i$ denotes the size of unfinished transfer blocks that is assigned in previous rounds at server "*i*". $UnFinishSize_i$ is equal to zero in first round. Ideally, depending to the real time bandwidth at time T_j, every flow is expected to finish its workload in future.

This fulfills our requirement to minimize the time faster servers must wait for the slowest server to finish. If, in some cases, network variations greatly degrade transfer rates, *UnFinishSize*$_i$ may exceed $(SE_j + \sum_{i=1}^{n} UnFinishSize_i) * B_i \Big/ \sum_{i=1}^{n} B_i$, which is the total block size expected to be transferred after T_j. In such cases, the co-allocator eliminates the servers in advance and assigns SE_j to other servers.

After allocation, all channels continue transferring data blocks. When a faster channel finishes its assigned data blocks, the co-allocator begins allocating an unassigned section of file *A* again. The process of allocating data blocks to adjust expected flow finish time continues until the entire file has been allocated.

4.2 Determining When to Stop Continuous Adjustment

Our approach gets new sections from whole files by dividing unassigned file ranges in each round of allocation. These unassigned portions of the file ranges become smaller after each allocation. Since adjustment is continuous, it would run as an endless loop if not limited by a stop condition.

However, when is it appropriate to stop continuous adjustment? We provide two monitoring criteria, *LeastSize* and *ExpectFinishedTime*, to enable users to define stop thresholds. When a threshold is reached, the co-allocation server stops dividing the remainder of the file and assigns that remainder as the final section. The *LeastSize* criterion specifies the smallest file we want to process, and when the unassigned portion of *UnassignedFileSize* drops below the *LeastSize* specification, division stops. *ExpectFinishedTime* criterion specifies the remaining time transfer is expected to take. When the expected transfer time of the unassigned portion of a file drops below the time specified by *ExpectFinishedTime*, file division stops. The expected rest time value is determined by:

$$UnAssignedFileSize \Big/ \sum_{i=1}^{n} B_i \qquad (3)$$

These two criteria determine the final section size allocated. Higher threshold values will induce fewer divisions and yield lower co-allocation costs, which include establishing connections, negotiation, reassembly, etc. However, although the total co-allocation adjustment time may be lower, bandwidth variations may also exert more influence. By contrast, lower threshold values will induce more frequent dynamic server workload adjustments and, in the case of greater network fluctuations, result in fewer differences in server transfer finish time. However, lower values will also increase co-allocation times, and hence, increase co-allocation costs. Therefore, the internet environment, transferred file sizes, and co-allocation costs should all be considered in determining optimum thresholds.

5 Experimental Results and Analyses

In this section, we discuss the performance of our Recursive-Adjustment Co-Allocation strategy. We evaluate four co-allocation schemes: (1) *Brute-Force (Brute)*,

(2) *History-based (History)*, (3) *Conservative Load Balancing (Conservative)* and (4) *Recursive-Adjustment Co-Allocation (Recursive)*. We analyze the performance of each scheme by comparing their transfer finish time, and the total idle time faster servers spent waiting for the slowest server to finish delivering the last block.

In our example, we assumed that a client site at Tunghai University (THU), Taichung city, Taiwan, was fetching a file from three selected replica servers: one at Providence University (PU), one at Li-Zen High School (LZ), and one at Da-Li High School (DALI). We monitored the bandwidth variations from THU to each server using NWS [18] probes. Network environment variations of each connection are shown in Figure 4.

We assign α = 0.5 and experiment it over several file sizes, such as 500MB, 1000MB, 2000MB, and 4000MB. We set the *LeastSize* limit threshold to 100MB, which result in 12, 15, 17, and 19 block numbers. As mater of comparison, we use the equal block numbers above to calculate the performance of each size, when using the *Conservative Load Balancing*. In Figure 5, we show the cost time of each scheme that transfers different file sizes. Obviously, Figure 5 shows that our approach reduces the time efficiently when compared with the other three schemes.

For each of schemes, we analyzed the effect of faster servers waiting for the slowest server to deliver the last block. In Figure 6, we calculate the total waiting idle time with different file sizes, and it shows that our *Recursive-Adjustment Co-Allocation* scheme offers significant performance improvements in every file size case when compared with other schemes. This result is due to our approach reduces the difference of each server's finished time efficiently.

For the *Recursive-Adjustment* technique, we study the effects of various α values on the block numbers and the total idle times. Figures 7 and 8 ,show for an assigned file size of 10MB to LeastSize, the total idle time increased and the total block number decreased as the α value increased. When the α value was greater then 0.7, the wait time grew rapidly, and although the wait time performance was good when the α value was less than 0.4, it resulted in a great increase in block numbers, which may cause high co-allocation costs. This experiment indicates that the assigned α value should be neither too large nor too small.

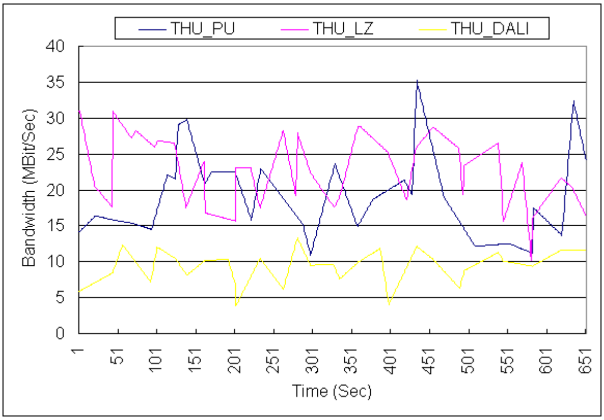
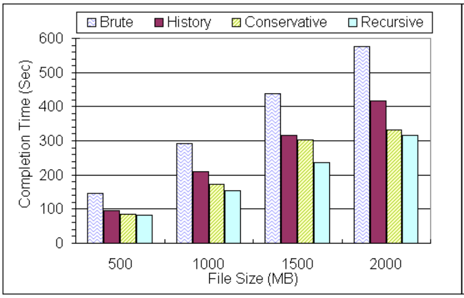


Fig. 4. Network variation between client and each server



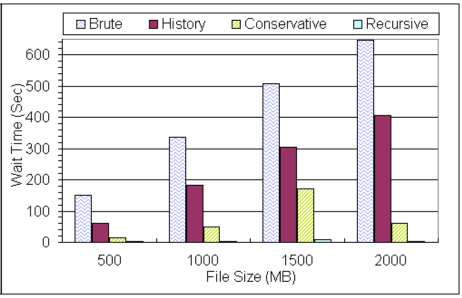
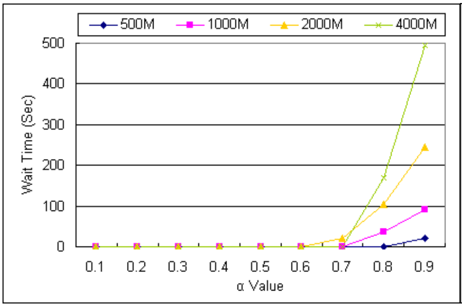


Fig. 5. Completion times for various methods

Fig. 6. Idle times for various methods



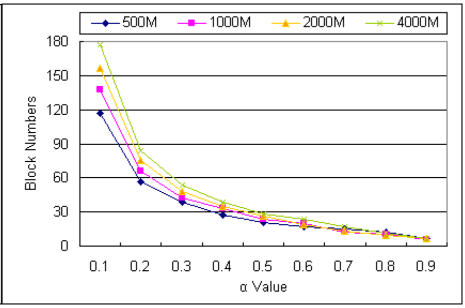
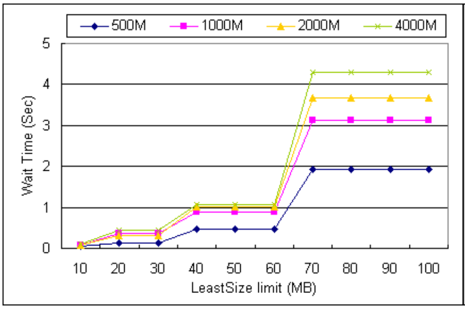


Fig. 7. Idle times for various *α* values

Fig. 8. Block numbers for various α values

Figures 9 and 10 show that the *LeastSize* threshold value in our *Recursive-Adjustment* method is also an important factor affecting total wait time and block numbers. In this experiment, we set the *α* value to 0.5 and tested various *LeastSize* values. The results indicate that decreasing the *LeastSize* threshold value effectively reduces the total wait time. Although this results in more block numbers, the increase is not excessive. Figure 9 indicates we may infer that the *Recursive-Adjustment* scheme performs better with smaller *LeastSize* threshold values for most file sizes because smaller size final blocks are less influenced by network variations.



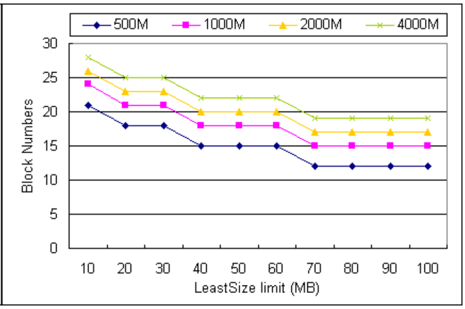


Fig. 9. Idle times for various *LeastSize* values

Fig. 10. Block numbers for various *LeastSize* values

6 Conclusions

Using the parallel-access approach to downloading data from multiple servers reduces transfer time and increases resilience to servers. The co-allocation architecture provides a coordinated agent for assigning data blocks. A previous work showed that the dynamic co-allocation scheme leads to performance improvements, but cannot handle the idle time of faster servers, that must wait for the slowest server to deliver its final block. This study proposes the *Recursive-Adjustment Co-Allocation* scheme to improve data transfer performances using the co-allocation architecture in [13]. In this approach, the workloads of selected replica servers are continuously adjusted during data transfers, and we provide a function that enables users to define a final block threshold, according to their data grid environment. Experimental results show the effectiveness of our proposed technique in improving transfer time and reducing overall idle time spent waiting for the slowest server.

References

1. B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 28(5):749-771, May 2002.
2. B. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, S. Tuecke, "Secure, efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing," *Proc. of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies*, pp. 13-28, 2001.
3. A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunszt, and M. Ripeanu, "Giggle: A Framework for Constructing Scalable Replica Location Services," *Proc. of SC 2002*, Baltimore, MD, 2002.
4. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, S. Tuecke, "The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets," *Journal of Network and Computer Applications*, 23:187-200, 2001.
5. K. Czajkowski, S. Fitzgerald, I. Foster and C. Kesselman, "Grid Information Services for Distributed Resource Sharing," *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10'01)*, 181-194, August 2001.
6. K. Czajkowski, I. Foster, C. Kesselman. "Resource Co-Allocation in Computational Grids," *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8'99)*, August 1999.
7. F. Donno, L. Gaido, A. Ghiselli, F. Prelz, M. Sgaravatto, "DataGrid Prototype 1," *TERENA Networking Conference*, June 2002, http://www.terena.nl/conferences/tnc2002/Papers/p5a2-ghiselli.pdf
8. Global Grid Forum, http://www.ggf.org/
9. Wolfgang Hoschek, Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, Kurt Stockinger, "Data Management in an International Data Grid Project," *First IEEE/ACM International Workshop on Grid Computing - Grid 2000,* Bangalore, India, December 2000.
10. IBM Red Books, "Introduction to Grid Computing with Globus," IBM Press, www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf
11. Heinz Stockinger, Asad Samar, Bill Allcock, Ian Foster, Koen Holtman, Brain Tierney, "File and Object Replication in Data Grids," *Journal of Cluster Computing*, 5(3):305-314, 2002.

12. The Globus Alliance, http://www.globus.org/
13. S. Vazhkudai, "Enabling the Co-Allocation of Grid Data Transfers," *Proceedings of Fourth International Workshop on Grid Computing,* pp. 41-51, November 2003.
14. S. Vazhkudai, J. Schopf, "Using Regression Techniques to Predict Large Data Transfers," *International Journal of High Performance Computing Applications (IJHPCA)*, 17:249-268, August 2003.
15. S. Vazhkudai, S. Tuecke, I. Foster, "Replica Selection in the Globus Data Grid," *Proceedings of the 1st International Symposium on Cluster Computing and the Grid (CCGRID 2001),* pp. 106-113, May 2001.
16. S. Vazhkudai, J. Schopf, "Predicting Sporadic Grid Data Transfers," *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 '02)*, pp. 188-196, July 2002.
17. S. Vazhkudai, J. Schopf, and I. Foster, "Predicting the Performance of Wide Area Data Transfers," *Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, pp.34-43, April 2002, pp. 34 – 43.
18. R. Wolski, N. Spring and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computer Systems*, 15(5-6):757-768, 1999.
19. Chao-Tung Yang, Chun-Hsiang Chen, Kuan-Ching Li, and Ching-Hsien Hsu, "Performance Analysis of Applying Replica Selection Technology for Data Grid Environments," *PaCT 2005, Lecture Notes in Computer Science*, vol. 3603, pp. 278-287, Springer-Verlag, September 2005.
20. X. Zhang, J. Freschl, and J. Schopf, "A Performance Study of Monitoring and Information Services for Distributed Systems", *Proceedings of 12th IEEE International Symposium on High Performance Distributed Computing (HPDC-12 '03)*, pp. 270-282, August 2003.

Reducing the Bandwidth Requirements of P2P Keyword Indexing

John Casey and Wanlei Zhou

School of Information Technology, Deakin University,
221, Burwood, HWY, Burwood, VIC 3125, Australia
{jacasey, wanlei}@deakin.edu.au

Abstract. This paper describes the design and evaluation of a federated, peer-to-peer indexing system, which can be used to integrate the resources of local systems into a globally addressable index using a distributed hash table. The salient feature of the indexing systems design is the efficient dissemination of term-document indices using a combination of duplicate elimination, leaf set forwarding and conventional techniques such as aggressive index pruning, index compression, and batching. Together these indexing strategies help to reduce the number of RPC operations required to locate the nodes responsible for a section of the index, as well as the bandwidth utilization and the latency of the indexing service. Using empirical observation we evaluate the performance benefits of these cumulative optimizations and show that these design trade-offs can significantly improve indexing performance when using a distributed hash table.

1 Introduction

Current, peer-to-peer keyword indexing systems that are built on top of distributed hash tables (DHT), map keywords and documents into a globally accessible index of federated resources. To facilitate load balance and scalability many global indexing systems fragment index data by keyword so that a particular node will only index a fraction of the total keyword indices. The postings lists of such an index will most likely be broken into a series of fixed size blocks which will provide efficient means to update, and read document indices. The keyword indices that a particular node is responsible for are identified deterministically using a hash value derived from the keyword.

This indexing strategy presents two problems. First, the node responsible for a particular keyword index has to be located. Second, the keyword indexing data has to be exported to a remote peer. Therefore, if a node wanted to export its local index so that its files and resources were globally accessible. The nodes indexing service would have to lookup the remote systems responsible for every <term, postings list> pair in the index, forwarding keyword index data to the appropriate systems. As a consequence, data indexing operations may consume excessive amounts of bandwidth, locating nodes and sending data across the network.

Index duplication is another problem that a DHT based indexing service will have to cope with. In a homogenous local-area network, which is the target environment of such an indexing system we expect index data to be heavily duplicated between dif-

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 50 – 59, 2005.

ferent systems. A recent study carried out by Bolosky. Et al in [1] showed that across 550 desktop file systems at Microsoft nearly 47% of the aggregate disk space was consumed by duplicate documents. Therefore, if duplicate document indices could be eliminated from the indexing process before they are sent to a remote host, the resources involved with the indexing process could be significantly reduced.

In this paper, we use a combination of batch indexing, bloom filter duplicate detection, aggressive index pruning, compression and leaf set forwarding to reduce the latency and bandwidth required to locate and store term-document indices in a DHT. To accomplish this we have developed a prototype indexing system that is based upon the Bamboo DHT [2] and have incorporated these optimizations into the indexing services lookup and put methods.

2 Background and Related Work

In this section, we provide a brief introduction to distributed hash tables (DHTs), and describe how they can be used to locate, store, and distribute data items in a reliable manner. Moreover, we review some of the indexing techniques used by similar projects and analyse their short comings to provide a motivation for this project.

2.1 Distributed Hash Tables

Structured P2P networks use distributed hash tables to map objects into a large virtual identifier space in which individual peers assume responsibility for a small range of the key space. Objects are uniformly mapped to key identifiers using a hash function on the contents or label of an object. This improves load balance, scalability and retrieval performance over unstructured P2P networks as queries can be directly routed to the peers that are responsible for items of interest. DHTs such as Chord [3], Bamboo [2], and Pastry [4] are able to route query requests in O(log n) steps for a network of n hosts. DHT implementations commonly use a hash function on a node's IP address to generate the nodeID or address of a node in the ring topology. In the Chord and Pastry systems, key-value pairs are stored on the host which has a nodeID that immediately succeeds a keys identifier, whereas the Bamboo system stores data items on the numerically closest node (modulo 2^{160}).

2.2 Routing in a Distributed Hash Table

To efficiently route messages to one another, each node participating in the ring maintains a routing table of log N rows, where N is the number of nodes in the system. Each row *n* of the routing table maintains information (address and node id) relating to peers which share the present node's nodeID in the first *n* digits but whose *n + 1* entries do not match the *n + 1* digits of the present node's nodeID. Nodes use this information to route messages deterministically around the ring by forwarding messages to nodes in the routing table whose nodeID matches the largest number of digits in the messages destination hash address. This process is called prefix routing, and theoretically allows nodes to route messages in O(log N) time.

To improve the routing performance and fault tolerance of the ring topology many distributed hash table implementations maintain a set of pointers or a leaf set to the nodes immediately to the left of a peer's nodeID and immediately to the right of a peer's nodeID. These are called the predecessor and successor nodes respectively, and most DHT implementations typically maintain a leaf set of two to four neighbouring systems. This parameter can be tuned to support different performance trade offs, such as routing performance, memory utilization and leaf set maintenance.

2.3 Related Work

Several projects have recently considered using distributed hash tables to support keyword indexing and search and have attacked the problem of efficient index dissemination from several different angles. The research projects that are the most pertinent to our research project are reviewed below. These papers can be divided roughly into three different categories: data indexing, query processing, and duplicate file detection. Many of the papers that relate to data indexing concentrate their efforts on efficient query execution in a distributed environment. Therefore, in a lot of these systems the underlying index implementation is often lacking and will more often than not consume excessive amounts of bandwidth to index and update postings lists. Similarly, the research projects which deal with duplicate file detection often concentrate their efforts on detecting redundant data in a distributed file system. In this paper, we seek to exploit the distinct properties of distributed inverted indexes to prevent redundant index data being re-indexed.

In [5] Reynolds and Vahdat simulate a vertically distributed inverted index which caches popular indices along the lookup paths of queries. The system exchanges bloom filter summaries of document postings lists to reduce the bandwidth requirements of conjunctive "and" queries which require remote document listings to be intersected. Our research project uses a similar bloom filter summary of the documents stored on a node to prevent redundant duplicate document indices being re-indexed. The major difference between this work and ours is that we initially use a smaller term filter to reduce the number of documents summarized in the resultant document bloom filter. This helps to reduce the bandwidth requirements of the bloom filter document summaries which are exchanged between indexing systems and remote hosts.

Burkard [6] extended the routing mechanism of Chord to support ring based forwarding, which reduces many of the lookup operations required to locate the system responsible for particular data items. The salient feature of ring based forwarding is that routing costs can be eliminated if the node the index service is currently sending data to forward its successor node back to the indexing system. In this way the indexer can "walk" around the DHT ring and distribute term-document indices with out having to lookup the host responsible for every term. Our prototype system uses a similar indexing strategy to forward a list of successor nodes from a remote system's leaf set back to the indexing system.

A possible drawback to using leaf set or ring based forwarding is that a remote system's leaf set maybe out of date due to nodes joining and leaving the system. Currently, we assume a relatively stable system state and have left issues such as leaf set inaccuracies for future work.

Apoidea [7] a decentralized web crawler which uses a distributed hash table maintains a large in memory bloom filter of the current URLs that have been processed by a peer to prevent peers re-processing a URL. The system filters URL listings that are to be processed and prevents previously indexed URLS being re-indexed. However, because of the large size of the in memory bloom filter URL listings are only filtered after a URL listing has been sent to a peer. Therefore, a lot of redundant URL listings still have to be transmitted across the network.

In [8] Muthitacharoen et. Al developed a low bandwidth file system (LBFS) which exploits the similarities between different versions of files to save bandwidth. To minimize redundant network transfer LBFS divides files into content based chunks, and indexes each of the chunks by a 64-bit hash value. Network bandwidth is reduced by identifying the chunks of data that a remote system already has so as to avoid having to transmit redundant data across the network. The duplicate detection process used by LBFS is similar in spirit to bloom filter summaries used in this research project. However, using the LBFS scheme a client indexing system would have to exchange several chunk identifiers with a remote system before redundant indexing data can be eliminated. Moreover, we expect the chunk identifiers used by LBFS to be too coarse to index the differences between new documents and duplicate documents in various postings lists.

3 Efficient Data Indexing in a DHT

In this section we detail the design and optimizations of the indexing service used in this paper and discuss some of the advantages and potential draw backs of the methods that we have used. We also discuss the two different index routing methods we use: the baseline batch indexing algorithm, leaf set forwarding and how duplicate detection using bloom filter digests can help reduce redundant network traffic.

3.1 Batch Indexing

The first indexing optimization that this paper uses is batch indexing. The basic idea behind batch indexing is that the postings lists for a particular term will be batched together so that a lot of smaller messages do not have to be routed along the same path. This works well, but performance can be improved by further aggregating those term-document indices which are destined for the same remote system. For example, the following index terms all generate a hash address which shares a common prefix of at least two digits. Instead of routing a single RPC put request for each of these terms indexes it makes more sense to batch them together and send them as a single unit.

Table 1. SHA-1 Term Images

base	0x1405df66cbe219b0bf6355bc3d60361a8376b6b4
input	0x140f86aae51ab9e1cda9b4254fe98a74eb54c1a1
mar	0x1418c40237ee713b2752a18beb0b3335c688b68b

In our prototype system we implement batch indexing using a two step process. Using a temporary inverted index structure we transform document indices into a series of <hash, term > listings, which are sorted in sequential order. Once this temporary index has been constructed, the indexer will iterate through the <hash, term> listings, and locate the nodes responsible for hosting a particular terms postings list. The indexing service will either look to its leaf set to find the remote system responsible for a term, or lookup the node responsible for a term using the underlying DHT. Once the node responsible for a particular term has been found the term and its associated document postings list are put on the indexing queue for that node. This indexing queue is emptied periodically and the data it contains is sent to the DHT in compressed form to be serialized. The current batch indexing prototype serializes queued indexing data at regular intervals in blocks of 500 terms. The number of terms that the indexer serializes can be tuned to reduce the indexer's memory requirements or increase the number of terms that are batched together.

In addition, compression can be used to minimise network utilization. The intuition behind this is that the network will be the biggest bottleneck in distributing document indices, so the extra processing and memory resources required to queue and compress the indices should have a small impact on the indexing service. The current prototype system uses generic GZIP compression.

Finally, to reduce the number of index terms and the number of documents the prototype indexing system utilises aggressive index pruning, stemming, and stop word removal to select the most informative terms and document postings from local indexes. These techniques improve the performance of index construction and retrieval by eliminating uninformative words and document postings from the index. Terms are selected for indexing based upon their document frequency (df) value which quantifies the number of documents a term occurs in. The document frequency filter in the prototype indexing system has a lower limit of df >=6 and an upper limit of df<=1000.

3.2 Leaf Set Forwarding

The major problem associated with batch indexing is that the node responsible for a particular keywords postings list has to be looked up using a DHTs routing substrate. Not only is the expensive in terms of messages disseminated though out a system. But, it may also be inaccurate as the state of a system may change, invalidating previously batched indices. Therefore, our system makes use of ring based forwarding as proposed by Burkard [6] in his thesis about data indexing and web crawling in a DHT environment. The ring based forwarding indexing process is largely complementary to our work on duplicate detection.

3.3 Duplicate Index Elimination Using Bloom Filters

Distributed indexes integrate a wide range of different document resources into a single, globally addressable, distributed index. As a consequence, it is natural to assume that some of these integrated resources are also mirrored or duplicated on a number of systems. Therefore, to reduce bandwidth consumption and redundant RPC messages between systems it makes sense to utilize some form of duplicate detection to ensure duplicate term-document indices are not needlessly re-indexed.

Bloom filters are a lossy indexing scheme, used to compactly represent a set or index as a series of superimposed bitmap patterns, which are used to represent the items stored in the set [9]. Using this technique, the amount of memory (or bandwidth) required to perform membership queries on an index can be significantly reduced in comparison to storing the entire set. However, this comes at the price of an adjustable false positive which is dependent on the number of bits used to construct the bloom filter, and the number of items represented by the bloom filter. Bloom filters are defined using two parameters: *m* which specifies the size of the filter in bits, and *k* which defines the number of independent hash functions used to derive a data items bitmap pattern. These parameters determine the amount of space required to encode *n* items and the associated false positive *p* of the bloom filter. The false positive rate of a bloom filter can be calculated using the following formula.

$$p = \left[1 - e\frac{-kn}{m}\right]^k \tag{1}$$

Data items can be inserted into the bloom filter by hashing the label of an item or the contents of an item with the bloom filter's *k* hash functions $h_1, h_2, \ldots h_k$, each of which map into the range *{0, ...,m-1}*. The corresponding bit locations as generated by the *k* hash functions are then set to one within the bloom filter. Successive, updates to the bloom filter may set the same bit location to one multiple times as different data items may hash to the same bit locations. Once a bloom filter has been populated membership queries can be accomplished by hashing a data item using a bloom filter's *k* hash functions and checking that the *k* bit locations in the bloom filter are set to one. If all the corresponding bit locations are set to one, then the item has probably been stored in the bloom filter with a small chance of false positive. If any of the bit locations are set to zero, then the data item is definitely not in the bloom filter. A diagram illustrating the insertion of an item into a bloom filter using four hash functions is shown below in figure 1.

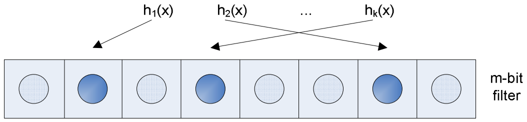


Fig. 1. Inserting Data Item X into the Bloom Filter

In this paper, we use bloom filters to summarize the term and document indices stored at a particular node and derive the *k* hash functions used to perform index and lookup operations from the SHA-1 hash of an index items label. This is accomplished by dividing the result of the SHA-1 hash function into several equally sized blocks, where the content of each block corresponds to the result of one of the *k* hash functions to be used by the bloom filter. The bloom filter index summaries used in this paper are created using a two step process to reduce the number of bloom filter entries that are to be transferred between the indexing system and the remote host. Initially the indexing system will create a bloom filter summary of the indexing terms that are to be sent to a remote system taking into account the number of documents indexed under a specific term. Once the remote system has received the indexing clients list of terms, it will create a bloom filter digest which summaries the documents that are indexed under the terms specified by the indexing service. In this way, the number of documents that are transferred between the two systems can be reduced, especially when the number of terms to be indexed is small in comparison to the number of terms stored on the remote node. Finally, if a duplicate document is found it is not completely discarded from the indexing process and it is still indexed in the replica location index as a replica document.

4 Experimental Evaluation

In this section, we present the preliminary results of these optimizations using a prototype indexing service that has been integrated into the Bamboo DHT's [10] lookup and put remote procedure calls. In particular, we quantify the potential bandwidth savings using a realistic document set and compare the optimizations to a baseline batch indexing system. The experiments have been carried out on a network of 30 Pentium II nodes, where each system has 384 MB of memory and is connected to the network using a 100 Mbps network card.

To evaluate the performance of the indexing system we have indexed the "Large Web Document Clustering Collection" [10] which in total constitutes 10,000 documents and 27,620 terms. Each of the indexing terms of this collection on average indexes ≈ 57 document postings. The frequency rank distribution of these indexing terms is shown on the next page in figure 2. The relationship between the index terms and document postings approximates Zipf's law. Therefore, a few terms will index many document postings whilst the majority will only index a few.

For each benchmark we measure the time it takes to index this document set and record the number of RPC operations used to locate the host responsible for a particular index term as well as the amount of bandwidth consumed by the indexer. To minimise the effect of various errors or bias in the experiment, we repeat each experiment 10 times and present the average result in the following figures and analysis.

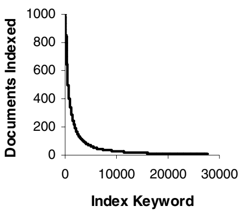


Fig. 2. Index Distribution

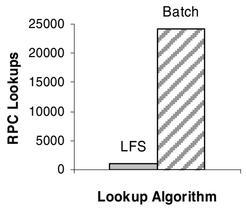


Fig. 3. RPC Lookups

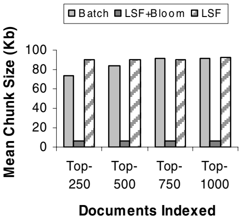


Fig. 4. Mean Chunk Size

A similar benchmark is used to evaluate the performance of the bloom filter duplicate detection system. However, this time we measure the bandwidth used to transfer a bloom filter and compare this to the bandwidth saved by eliminating duplicate documents from the index. To determine the basic performance and overhead of the bloom filter duplicate detection system, we repeatedly re-index the local text database so that the maximum number of duplicate documents may be found by the system. This means the duplicate detection system will flag every document as being a duplicate as exactly the same index is being re-indexed again. For each experiment we have also varied the maximum number of document postings that can be indexed, by a constant increment of 250 documents. However, this doesn't seem to have had much effect on the overall results of the indexing algorithms. The reason for this isn't immediately clear but we surmise this is due to the heavy tailed distribution of postings lists as shown figure 2.

4.1 RPC Lookups

Figure 3 presents the results of the RPC lookup benchmark which we use to compare the performance of the batch indexing and leaf set forwarding lookup algorithms. The batch indexing lookup procedure roughly performs 25 thousand lookup operations to locate the remote systems responsible for individual indexing keywords and their document posting lists. The reason for this is that the batch indexing algorithm has to perform a lookup operation for nearly every term in its database except for those destined for its leaf set. In comparison, the leaf set forwarding (LSF) algorithm significantly reduces the number of RPC lookup operations required to locate a remote systems. The reason for this is that the leaf set forwarding algorithm is able to amortize the costs of looking up a node in a DHT by simply forwarding a remote system's successor list back to the indexing service.

4.2 Mean Chunk Size

Figure 4 depicts the average size of a compressed indexing chunk as used by the batch indexing, leaf set forwarding, and bloom filter indexing schemes. In this experiment an indexing chunk represents a compressed group of serialized term-document indices which are to be sent to the same remote system. On average, the leaf set forwarding and batch indexing algorithms roughly transmit ≈ 80 Kb to remote indexing systems. In comparison, the bloom filter indexing scheme roughly transmits ≈ 6 Kb of data to remote indexing systems which is much less. This 6 Kb of data represents the overhead of the bloom filter duplicate detection system in terms of the term-document summaries used by the indexing service to identify duplicate documents.

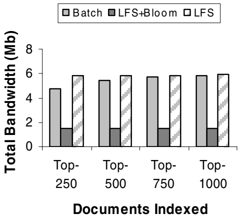


Fig. 5. Aggregate Bandwidth

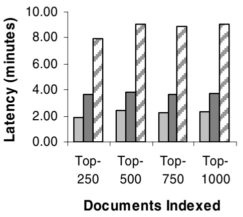


Fig. 6. Indexing Latency

4.3 Aggregate Bandwidth

In figure 5 we examine the differences between the aggregate bandwidth utilized by the leaf set, batch and duplicate detection indexing algorithms. For each of these algorithms the aggregate bandwidth metric is used to quantify the amount of data transferred between a system's indexing service and the DHT. Moreover, for the duplicate detection algorithm the aggregate bandwidth metric also quantifies the size of the term-document filters exchanged between the indexing service and remote host. The batch indexing and leaf set forwarding algorithms in aggregate consume a similar amount of bandwidth as they index exactly the same number of documents. In Comparison, the duplicate detection algorithm in aggregate uses far less bandwidth as it suppresses many of the redundant indices that have already been indexed. Despite the significant bandwidth savings of the duplicate detection algorithm more research needs to be carried out to more accurately quantify the bandwidth savings under different levels of duplication. In addition, using a bloom filter to summarize the contents of a node for each indexing transfer is expensive and in some cases will probably be unnecessary. Therefore, different strategies need to be developed to reduce the size and computational cost of creating a bloom filter document summary.

4.4 Indexing Latency

Figure 6 shows the average latency of the various algorithms used in the indexing process. The fastest algorithm by far is the leaf set forwarding (LSF) algorithm. The duplicate detection algorithm is almost as fast. But, the overhead of dynamically creating a bloom filter increases the latency of the algorithm by a constant factor. The latency of the batch algorithm is significantly worse than the other algorithms and we attribute this to large number of RPC requests made by the algorithm.

5 Conclusion and Further Work

In this paper, we have proposed a set of mechanisms to significantly reduce the number of lookup operations, and bandwidth required to index term-documents in a distributed hash table. Our results show that the combination of the different techniques such as index compression, batching, leaf set forwarding and specifically duplicate detection can help to improve the performance and bandwidth utilization of indexing operations.

We are currently conducting more extensive investigations to address some of the current limitations of the system and plan to develop a comprehensive index distribution which accurately reflects the distribution of duplicate and non-duplicate documents in a networked environment. Finally, we would also like to perform a larger scale evaluation of the indexer using more nodes to more accurately gauge the scalability of the system.

References

[1] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer, "Feasibility of a Serverless Distributed File System Deployed on an existing set of Desktop PCs",SIGMetrics'00.

[2] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT",Usenix'04.

[3] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications",SIGComm'01.

[4] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems",Middleware'01.

[5] P. Reynolds and A. Vahdat, "Efficient Peer-to-Peer Keyword Searching",Middleware'03.

[6] T. Burkard, "Herodotus: A Peer-to-PeerWeb Archival System",in *Department of Electrical Engineering and Computer Science*. Cambridge: Massachusetts Institute of Technology, 2002.

[7] A. Singh, M. Srivatsa, L. Liu, and T. Miller, "Apoidea: A Decentralized Peer-to-Peer Architecture for Crawling the World Wide Web",SIG2003.

[8] A. Muthitacharoen, B. Chen, and D. Mazières, "A Low-bandwidth Network File System",18th SOSP, 2001.

[9] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Comm. of the ACM*, vol. 13, pp. 422-426, 1970.

[10] M. Sinka and D. Corne, "A large benchmark dataset for web document clustering," *Soft Computing Systems: Design, Management and Applications*, vol. 87, pp. 881-890, 2002.

A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids

Srikumar Venugopal and Rajkumar Buyya

Grid Computing and Distributed Systems (GRIDS) Laboratory,
Department of Computer Science and Software Engineering,
The University of Melbourne, Australia
{srikumar, raj}@cs.mu.oz.au

Abstract. In this paper, we present an algorithm for scheduling of distributed data intensive Bag-of-Task applications on Data Grids that have costs associated with requesting, transferring and processing datasets. The algorithm takes into account the explosion of choices that result due to a job requiring multiple datasets from multiple data sources. The algorithm builds a resource set for a job that minimizes the cost or time depending on the user's preferences and deadline and budget constraints. We evaluate the algorithm on a Data Grid testbed and present the results.

1 Introduction

Multi-institutional scientific projects in domains such as high energy physics, astronomy and bioinformatics are increasingly generating data in the range of Tera Bytes (TB) which is replicated at various sites for improving reliability and locality. Grid computing [1] has made it possible to aggregate heterogeneous, geographically distributed compute and storage resources for executing large-scale applications in such eScience [2] projects. Data Grids [3] are instances of Grids where access and management of distributed data resources have equal or higher priority than computational requirements. A well-cited example of a Data Grid is the one being set up for processing the output of the ATLAS experiment at the Large Hadron Collider(LHC) at CERN [4].

The execution of data-intensive applications involves requirements for discovering, processing, storing and managing large distributed datasets and is guided by factors such as cost and speed of accessing, transferring and processing data. There may be multiple datasets involved in a computation, each replicated at multiple locations that are connected to each other and to the compute resources by networks with varying cost and capability. Consequently, this explosion of choices makes it difficult to identify the most optimal resources for retrieving and performing the required computation on the selected datasets.

In large collaborations that form Data Grids, there can be a lot of pressure on the network, storage and processing elements. This can lead to overloading of resources and appearance of network ”hot spots” as is commonly observed in the World Wide Web [5]. Previous work has suggested a computational economy metaphor for resource management within computational grids [6]. Resource providers price their goods to reflect supply and demand in order to make a profit or to regulate consumption. On the consumer

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 60–72, 2005.

side, users specify their deadlines for completing their jobs, the budget available to them and their preference for the cheapest or the fastest processing according to their needs and priorities. While such strategies have been proposed and evaluated for computational grids [7], no study has yet been made for similar requirements on Data Grids.

In this paper, we introduce an algorithm for scheduling a Bag-of-Tasks(BoT) application on a set of geographically distributed, heterogeneous compute and data resources. Each of the tasks within the application depends on multiple datasets that may be distributed anywhere within the grid. Also, there are economic costs associated with the movement and processing of datasets on the distributed resources. The algorithm minimizes either the overall cost or the time of execution depending on the user's preference subject to two user-defined constraints - the deadline by which the processing must be completed and the overall budget for performing the computation.

The rest of this paper is organised as follows. In Section 2, we survey previous work in data-intensive Grid scheduling. We extend the notion of user-driven deadline and budget constrained scheduling within computational grids to data grids in Section 3. In Section 4, the proposed algorithm is evaluated on a real Grid testbed and the results are reported. Finally, we conclude our paper and outline future work.

2 Related Work

Several approaches have been proposed to schedule data-intensive applications on distributed resources. In [8], the authors evaluate various heuristics for parameter-sweep jobs which have files as input. They introduce a new heuristic, *XSufferage*, that takes into account file locality by scheduling jobs to those clusters where the files have already been transferred for a previous job. Takefusa, et. al [9] explore various combinations of scheduling and replication algorithms and come to the conclusion that for large files, moving computation close to the source of data is the best strategy. Ranganathan and Foster [10] have simulated job scheduling and data scheduling algorithms and recommend that it is best to decouple data replication from the job scheduling. In Chameleon [11], the scheduling strategy executes a job on one site while taking into account computation and communication costs. Kim and Weissman [12] explore a Genetic Algorithm-based approach for decomposing and scheduling a parallel data-intensive application. In previous work [13], we have proposed an adaptive algorithm that schedules jobs while minimizing data transfer. It evaluates all known replica locations of the file and submits the job to the compute resource which is located closest to one of the replica locations. However, in the case of applications having to deal with data from multiple sources, the problem is executing the application such that it is optimal with respect to all the data sources rather than a single source as has been considered in the works presented before.

The problem of scheduling BoT applications on distributed systems is a very well-studied one [14][8][15]. Deadline and budget constrained scheduling algorithms for compute-intensive BoT applications were proposed and evaluated in [7]. In this paper, we extend the same notion to data-intensive BoT applications in the following manner. This paper proposes a detailed cost model for distributed data-intensive applications that builds on the models for system costs (processing and transferring overheads) previously discussed in [12][11][16] and takes into account expenses for storage, transfer and processing of data. It then proposes a new algorithm based on the Min-Min heuristic described in [14] that takes into account the deadline and budget constraints of the users and produces a schedule that minimises either cost or time depending on their requirements. The proposed algorithm also explicilty deals with the explosion of choices in scheduling Data Grid applications as is mentioned in the previous section. While this is similar in intent to the work presented in [17], there is a lot of difference in the methodologies. In [17], the search space is pruned by grouping resources into collections and then sorting the collections in the order decided by a certain performance metric. As will be shown later, within our algorithm, the resource sets are created in an incremental fashion and the search space is limited to only those resources that minimze the objective function.

3 Scheduling

Fig. 1 shows a typical Data Grid environment which is composed of storage resources, or *data hosts*, which store the data and compute resources which run the jobs that execute upon the data. This is based on the scenarios drawn up for users of the production Data Grid projects such as LHC Grid [18]. It is possible that the same resource may contain both storage and computation capabilities. For example, it could be a supercomputing center which has a Mass Storage Facility attached to it. The datasets may be replicated at various sites within this data grid depending on the policies set by the administrators of the storage resources and/or the producers of data. The scheduler is able to query a data directory such as a Replica Catalog [19] or the SRB [20] Metadata Catalog for information about the locations of the datasets and their replicas. We associate economic costs with the access, transfer and processing of data. The processing cost is levied upon by the computational service provider, while the transfer cost comes on account of the access cost for the data host and the cost of transferring datasets from the data host to the compute resource through the network.

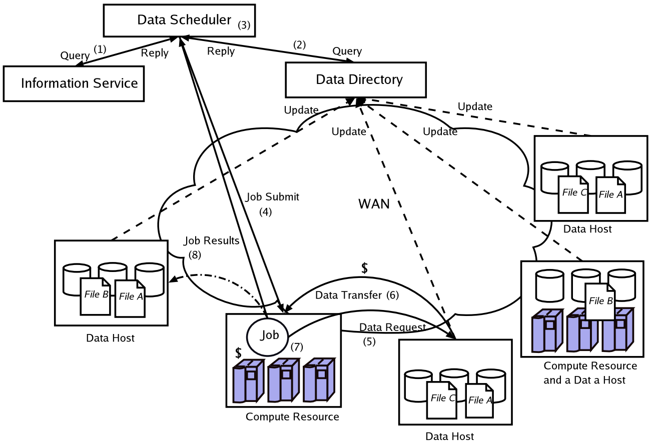


Fig. 1. An economy-based data grid environment

We consider a job (equivalent to a task in a BoT) as the atomic unit of computation within this model. Each job requires one or more datasets as input. Each dataset is available through one or more data hosts. The steps for submitting a job to the grid shown in Fig. 1 are as follows: The scheduler gathers information about the available compute resources through a resource information service (1) and about the data through the data directory (2). It then makes decision on where to submit the job based on the availability and cost of the compute resource, the minimization preference and the location, access and transfer costs of the data required for the job (3). The job is dispatched to selected the remote compute resource (4) where it requests for the dataset from the replica location selected by the scheduler (5 & 6). After the job has finished processing (7), the results are sent back to the scheduler host or another storage resource which then updates the data directory(8). This process is repeated for all the jobs within the set. Here, only resources that meet minimum requirements of the application such as architecture(instruction set), operating system, minimum free memory and storage threshold are considered as suitable candidates for job execution.

We consider, therefore, a set of N independent jobs $J = \{j_1, j_2, \ldots, j_N\}$ which have to be scheduled on M computational resources $R = \{r_1, r_2, \ldots, r_M\}$. Typically, $N \gg M$. Each job $j, j \in J$ requires a subset $F_j = \{f_{j1}, f_{j2}, \ldots, f_{jK}\}$ of a set of datasets, F, which are each replicated on a subset of P data hosts,$D = \{d_1, d_2, \ldots, d_P\}$. For $f \in F$, $D_f \subseteq D$ is the set of data hosts on which f is replicated.

The time taken to execute a job is the sum of the execution time and the times taken to transfer each of the datasets required for the job from their respective data hosts to the compute node. If the execution time for job j on compute resource r is denoted by t_{jr} and the transfer time for a dataset $f_j \in F_j$ from a location $d_{f_j} \in D_{f_j}$ to compute resource r is denoted by $t_{f_j r}$, then the total time required for executing the job j is given by $t_j = t_{jr} + \sum_{f_j \in F_j} t_{f_j r}$ where $t_{f_j r}$ is the sum of the *response time* of d_{f_j} and the time taken for the actual data movement. We define *response time* as the difference between the time when the request was made to the data host and the time when the first byte of the file was received at the compute resource. It is an increasing function of the load on the data host. The time taken for the data movement is the size of the data divided by the available bandwidth between the data host and the compute resource. While we have considered a case of sequential data transfer in this model, it can be modified to consider a parallel data transfer model as presented in [12].

To calculate the economic cost of executing the job, we denote the economic cost of executing the job j on the compute resource r by e_{jr} and cost of transferring the dataset f by $e_{f_j r}$. Therefore, the total execution cost for job j is given by $e_j = e_{jr} + \sum_{f_j \in F_j} e_{f_j r}$ where $e_{f_j r}$ is the sum of *access cost*, which is the price levied by the data host for serving the requested dataset and network transfer cost dependent on the size of the file and the cost of transferring unit data from data host to compute resource . The access cost can be an increasing function of either the size of the requested dataset or the load on the data host or both. This cost regulates the size of the dataset being requested and the load which the data host can handle. The cost of the network link may increase with the Quality of Service(QoS) being provided by the network. For example, in a network supporting different channels with different QoS as described in [21], a channel with a higher QoS may be more expensive but the data may be transferred faster.

We associate two constraints with the schedule, the deadline by which the entire set must be executed (denoted by $T_{Deadline}$) and the maximum budget, $Budget$, for processing the jobs.The deadline constraint can therefore be expressed in terms of job execution time as $max(t_j) \leq T_{Deadline}, \forall\ j \in J$. The budget constraint can be expressed as $\sum_J e_j \leq Budget$.

3.1 Algorithm

Depending on the user-provided deadline, budget and scheduling preference, we can have two objective functions, viz:

- **Cost minimization** We try to execute the jobs in the schedule that causes least expense while keeping the execution time within the deadline provided.
- **Time minimization** Here, the jobs are executed in the fastest time possible with the budget for the execution acting as the constraint.

In both cases the same algorithm can be applied to solve the different objective functions. This is done by means of a switch Min which allows us to change the decision variables depending on the minimization chosen within the algorithm. We define a function f_{min} that returns the smallest value within a set of values, A, depending on the minimization applied. Formally,

$$f_{min}(Min, CVar, TVar, A) = \begin{cases} min(CVar, A) \\ \text{if } Min = Cost \\ min(TVar, A) \\ \text{if } Min = Time \end{cases}$$

Here, $CVar$ and $TVar$ represents variables deal with cost and time respectively. The functions $min(CVar, A)$ and $min(TVar, A)$ will return the element of A with the smallest value of $CVar$ and $TVar$ respectively. Hence, by changing the value of Min we can determine the objective function to be minimized by the algorithm. Consequently, Min is a parameter to the scheduling algorithm.

The listing for the algorithm is given in Figure 2. J_U, J_A, J_C and J_F are subsets of the set of jobs J consisting of jobs in $Unsubmitted$, $Active$, $Completed$ and $Failed$ states respectively. Jobs initially are in the $Unsubmitted$ state, once they are submitted, they become $Active$ and finally end up being $Completed$ or $Failed$. The scheduling algorithm exits if all jobs are in the final states or if the deadline or budget constraints are violated. The initial part of the loop does bookkeeping. At every polling interval, we update the performance data of the compute resources and calculate the allocation for the current polling interval. For each data resource, we update the network conditions between itself and the computational resources. Then, we sort the computational resources either by the cost of the network link or the bandwidth between the compute resource and the data host depending on the minimization required. The rest of the algorithm is in two parts : the first part maps the jobs to the selected compute resources depending on selected minimization objective (cost or time) while the second dispatches the jobs while enforcing the deadline and budget constraints. These are described as follows:

```
while $J \neq J_C \cup J_F$ $OR$ $T_{current} < T_{Deadline}$ $OR$ $Budget\_spent < Budget$ do
    Update $Budget\_spent$ by taking into account the jobs completed in the last interval;
    for each $r \in R$ do
        Calculate performance data on the basis of resource performance in previous polling interval;
    end
    for each $d \in D$ do
        Based on current network values, sort $R$ in the increasing order of $Cost(Link_{dr})$ or $1/BW(Link_{dr}))$ depending on whether $Min$ is $Cost$ or $Time$;
        Maintain this list as $R_d$;
    end
    MAPPING SECTION;
    for $j \in J_U$ do
        $S_j, Temp_j \leftarrow \{\}$;
        for $f_j \in F_j$ do
            Select $\{r, d_{f_j}\}$ such that, depending on $Min$, either $e_{f_j r}$ or $t_{f_j r}$ is minimised;
            if $S_j = \{\}$ then
                $S_j \leftarrow S_j \cup \{r, d_{f_j}\}$;
                $Temp_j \leftarrow S_j$;
            end
            else
                $S_j \leftarrow (S_j - \{r_{prev}\}) \cup \{r, d_{f_j}\}$;
                $Temp_j \leftarrow Temp_j \cup \{d_{f_j}\}$;
            end
            $S_j \leftarrow f_{min}(Min, e_j, t_j, \{S_j, Temp_j\})$;
            $Temp_j \leftarrow S_j$;
            $r_{prev} \leftarrow r \in S_j$;
        end
    end
    DISPATCHING SECTION;
    Sort $J_U$ in the ascending order of $e_j$ or $t_j$ depending on $Min$;
    $Expected\_Budget \leftarrow Budget\_spent$;
    for $j \in J_U$ do
        Take the next job $j \in J_U$ in sorted order;
        $r \leftarrow r \in S_j$;
        if r can be allocated more jobs then
            if $Min = Cost$ AND $(T_{Current} + t_j) < T_{Deadline}$ then
                if $(Expected\_Budget + e_j) \leq Budget$ then submit $j$ to $r$;
                else stop dispatching and exit to main loop
            end
            if $Min = Time$ AND $Expected\_Budget + e_j \leq Budget$ then
                if $(T_{Current} + t_j) < T_{Deadline}$ then submit $j$ to $r$;
                else stop dispatching and exit to main loop
            end
            $Expected\_Budget = Expected\_Budget + e_j$;
            Remove $j$ from $J_U$;
        end
    end
    Wait for the duration of the polling interval;
end
```

Fig. 2. Pseudo-code for Deadline and Budget Constrained Cost-based Scheduling of Data Intensive Applications.

Mapping: We require one compute resource to execute the job and one data host each for every dataset required by the job. That is, for each job j, we create a **resource set** $S_j = \{r_j, d_{j1}, d_{j2}, \ldots, d_{jK}\}$ that represents the compute and data resources to be accessed by the job in execution. However, if we try all possible combinations of compute and data resources for each job, this results in a $O(N(MP)^K)$ mapping where K is the maximum number of datasets for each job.

We, therefore, decrease the complexity by making a choice at each step within the mapping section. For a job, we iterate through the list of datasets it requires. For each dataset, we pick the combination of a compute resource and a data host that returns the lowest value for expected transfer cost($e_{f_j r}$) or time($t_{f_j r}$) depending on either cost or time minimization. This is done in $O(P)$ time as for each data host, we only have to pick the first compute resource out of its sorted list of compute resources. Then we create two resource sets, S_j and $Temp_j$, the former with the current selected compute and data resources and the latter with the current selected data resource but with the compute resource selected in the previous iteration, r_{prev} (lines 15 - 22) . Then, we compare the two sets on the basis of the expected cost or execution time and select the resource set which gives us the minimum value (line 23). This procedure ensures that the choice of the compute resource and the resource set so formed at the end of each iteration is better than those selected in all previous iterations. For N jobs, therefore, the above mapping loop runs in $O(NKP)$ time.

Dispatching: In the dispatching section, we first sort all the job in the ascending order of the value of the minimization function for their respective combinations. Then, starting with the job with the least cost or least execution time, we submit the jobs to the compute resources selected for them in the mapping step if the allocation for the resources as determined in the initial part has not been exhausted. For cost minimization, we see if the deadline is violated by checking whether the current time($T_{Current}$) plus the expected execution time exceeds $T_{Deadline}$ (line 35). If so, the job goes back into the unsubmitted list in the expectation that the next iteration will produce a better combination. If $Budget$ is exceeded by the current job then we stop dispatching any more jobs and return to the main loop since the rest of the jobs in the list will have higher cost (lines 36-37). For time minimization, we check if the budget spent (including the budget for all the jobs previously submitted in current iteration) plus the budget for the current job exceeds $Budget$. If the deadline is violated by the current job then we stop dispatching and return to the main loop.

4 Experiments and Results

We have implemented the scheduling algorithm presented in Section 3 within the Gridbus Broker [13]. The testbed resources used in our experiments is detailed in Table 1. The *cost per sec* denotes the rate for performing a computation on the resource in Grid Dollars (G$). It can be seen that some of the resources were also used to store the replicated data and therefore, were performing the roles of both data hosts and compute resources. The average available bandwidth between the compute resources and the data hosts is given in Table 2. We have used NWS (Network Weather Service) [22]

Table 1. Resources within Belle testbed used for evaluation and their costing

Organization	Resource details	Role	Compute Cost(G$ /sec)	Total Jobs Done	
				Time	Cost
Dept. of Computer Science, University of Melbourne	*belle.cs.mu.oz.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Broker Host, Data Host, Compute resource, NWS Server	6	94	2
School of Physics, University of Melbourne	*fleagle.ph.unimelb.edu.au* 1 Intel 2.6 Ghz CPU, 512 MB RAM, 70 GB HD, Linux	Replica Catalog host, Data host, NWS sensor	N.A.*	–	–
Dept. of Computer Science, University of Adelaide	*belle.cs.adelaide.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data host, NWS sensor	N.A. *	–	–
Australian National University, Canberra	*belle.anu.edu.au* 4 Intel 2.6 GHz CPU, 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	6	2	4
Dept of Physics, University of Sydney	*belle.physics.usyd.edu.au* 4 Intel 2.6 GHz CPU(1 avail), 2 GB RAM, 70 GB HD, Linux	Data Host, Compute resource, NWS sensor	2	2	119
Victorian Partnership for Advanced Computing, Melbourne	*brecca-2.vpac.org* 180 node cluster (only head node utilised)	Compute resource, NWS sensor	4	27	0

*Not used as a compute resource but only as a data host

for measuring the network bandwidths between the computational and the data sites. We have used only the performance data and not the bandwidth forecasts provided by NWS. It has been shown that NWS measurements with 64 KB probes cannot be correlated with large data transfers[23][24]. However, we consider the NWS measurements are indicative of the actual available bandwidth in our case. In the future, we hope to use regression models for more accurate measurements as has been shown in [23][24] .The broker itself was extended to consider the price of transferring data over network links between the compute resources and the data hosts while scheduling jobs. In our experiments, although we have artificially assigned data transmission costs shown in Table 3, they can be linked to real costs as prescribed by ISPs (Internet Service Providers). During scheduling, data movement cost and time were explicitly taken into account when data and compute services were hosted on different resources.

Within the performance evaluation, we wanted to capture various properties and scenarios of Data Grids and applications. Accordingly we devised a synthetic application application that requests K datasets that are located on distributed data sources and are registered within a replica catalog. The datasets are specified as Logical File Names (LFNs) and resolved to the actual physical locations by the broker at runtime. The application then processes these datasets and produces a small output file (of the order of KB). In this particular evaluation, the datasets are files registered within the catalog. There are 100 files of size 30 MB each, distributed between the data hosts listed in Table 1. The BoT application here is a parameter-sweep application consisting of 125

Table 2. Avg. Available Bandwidth between Data Hosts and Compute Resources as reported by NWS(in Mbps)

	Compute Resources			
Data Hosts	UniMelb CS	ANU	UniSyd	VPAC
ANU	6.99	–	10.242	6.33
Adelaide	3.45	1.68	2.29	6.05
UniMelb Physics	41.05	6.53	2.65	20.57
UniMelb CS	–	6.96	4.77	36.03
UniSyd	4.78	12.57	–	2.98

Table 3. Network Costs between Data Hosts and Compute Resources (in G$/MB)

	Compute Resources			
Data Hosts	UniMelb CS	ANU	UniSyd	VPAC
ANU	34.0	0	31.0	38.0
Adelaide	36.0	34.0	31.0	33.0
UniMelb Physics	40.0	32.0	39.0	35.0
UniMelb CS	0	30.0	36.0	33.0
UniSyd	33.0	35.0	0	37.0

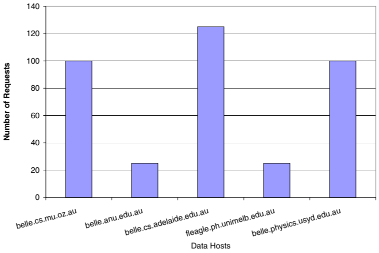


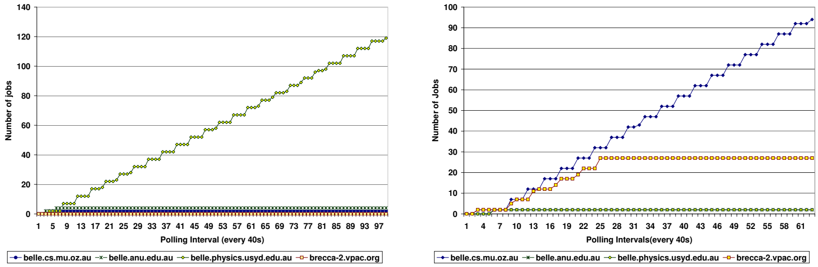
Fig. 3. Distribution of file access

Table 4. Summary of Evaluation Results

Minimization	Total Time (mins.)	Compute Cost (G$)	Data Cost (G$)	Total Cost (G$)
Cost	80	31198.27	39126.65	70324.93
Time	54	76054.90	43821.64	119876.55

jobs, each job an instance of the application described before requiring 3 files (that is, $K = 3$ for all the jobs in this evaluation). Fig. 3 gives the distribution of the number of requests for data made by the jobs versus the data hosts. The distribution is the same for both cost and time minimization. The datasets were transferred in sequence, that is, the transfer of one dataset was started after the previous had completed. The computation times for the jobs were randomly distributed within 60-120 seconds.

There are two measures of performance that we are interested in: the first is the relative usage of the computational resources under cost and time minimization which indicates how the choice of minimization criteria impacts resource selection and the second, is the distribution of jobs with respect to the computational and data transfer costs and times incurred within each minimization which tells us the how effective the algorithm was in producing the cheapest or the fastest schedule. The experiments were carried out on 29th November 2004 between 6:00 p.m. and 10:00 p.m. AEDT. The deadline and budget values for both cost and time minimization were 2 hours and 500,000 G$ respectively. Table 4 shows the summary of the results that were obtained. The total time is the wall clock time taken from the start of the scheduling procedure up to the last job completed. All the jobs completed successfully in both the experiments. The average costs per job incurred during cost and time minimization are 562.6 G$ and 959 G$ with standard deviations of 113 and 115 respectively. Mean wall clock time taken per job(including computation and data transfer time) was 167 secs for cost minimization and 135 secs for time minimization with standard deviations 16.7 and 19 respectively.



(a) cost minimization scheduling

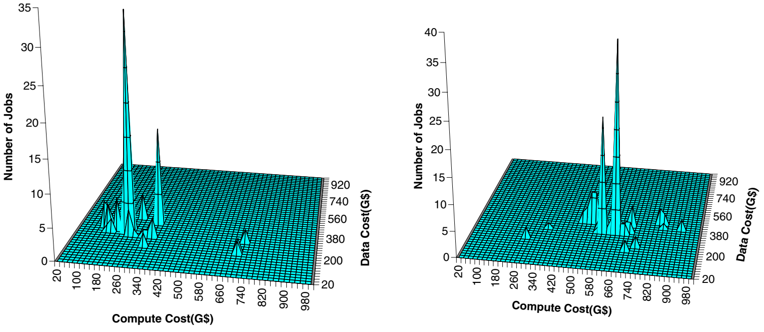
(b) time minimization scheduling

Fig. 4. Cumulative number of jobs completed vs time

As expected, cost minimization scheduling produces minimum computation and data transfer expenses whereas time minimization completes the experiments in the least time. The graphs in Figs. 4 and 4 show the number of jobs completed versus time for the two scheduling strategies for data grids. Since the computation time was dominant, within cost minimization, the jobs were executed on the least economically expensive compute resource. This can be seen in Fig. 4 where the compute resource with the least cost per sec, the resource at University of Sydney, was chosen to execute 95% of the jobs. Since a very relaxed deadline was given, no other compute resource was engaged by the scheduler as it was confident that the least expensive resource alone would be able to complete the jobs within the given time. Within time minimization, the jobs were dispatched to the compute resources which promised the least execution time even if they were expensive as long as the expected cost for the job was less than the budget per job. Initially, the scheduler utilised two of the faster resources, the University of Melbourne Computer Science(UniMelb CS) resource and the VPAC resource. However, as seen from Fig. 3, 26.67% of the requests for datasets were directed to the UniMelb CS resource. A further 6.67% were directed to the resource in UniMelb Physics. Hence, any jobs requiring one of the datasets located on either of the above resources were scheduled at the UniMelb CS resource because of the low data transfer time. Also, the UniMelb CS resource had more processors. Hence, a majority of the jobs were dispatched to it within time minimization.

Figs. 5(a) and 5(b) show the distribution of the jobs with respect to the compute and data costs respectively. For cost minimization, 95% of the jobs have compute costs less than or equal to 400 G$ and data costs between 250 G$ to 350 G$. In contrast, within time minimization, 91% of the jobs are in the region of compute costs between 500 G$ to 700 G$ and data costs between 300 G$ to 400 G$. Hence, in time minimization, more jobs are in the region of high compute costs and medium data costs. Thus, it can be inferred that the broker utilized the more expensive compute and network resources to transfer data and execute the jobs within time minimization.

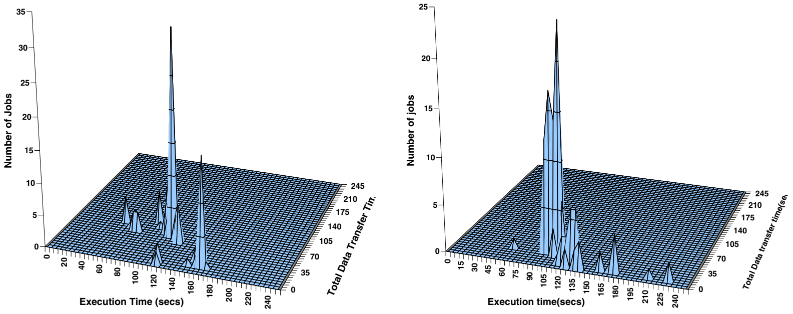
Figs. 6(a) and 6(b) show the distribution of the jobs with respect to the total execution time and the total data transfer time for cost minimization and time minimization respectively. The execution time excludes the time taken for data transfer. It can be seen that within time minimization 6(b) the maximum data transfer time was 35s as com-



(a) cost minimization scheduling

(b) time minimization scheduling

Fig. 5. Distribution of jobs against compute and data costs



(a) cost minimization scheduling

(b) time minimization scheduling

Fig. 6. Distribution of jobs against execution time and data transfer time

pared to 75s for cost minimization. Also, there are more jobs within time minimization that have had transfer time less than 10s which implies that the jobs were scheduled close to the source of the data. Therefore, from the results, it can be seen that given cost or time minimization, the algorithm presented in this work does minimize the objective function for upto 90% of the set of jobs.

5 Conclusion and Future Work

We have presented here a model for executing jobs on data grids which takes in to account both processing and data transfer costs. We have also presented an algorithm which greedily creates a resource set, consisting of both compute and data resources, that promises the least cost or least time depending on the minimization chosen. We have presented empirical results obtained from evaluating the algorithm on a Data Grid testbed.

We plan to conduct further evaluations to conclusively state that the algorithm minimizes its objective functions. We also plan to evaluate the algorithm with a testbed with different levels of replication of data and with varying resource prices.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1999)
2. Hey, T., Trefethen, A.E.: The UK e-Science Core Programme and the Grid. Journal of Future Generation Computer Systems(FGCS) **18** (2002) 1017–1031
3. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. Journal of Network and Computer Applications **23** (2000) 187–200
4. Lebrun, P.: The Large Hadron Collider, A Megascience Project. In: 38th INFN Eloisatron Project Workshop on Superconducting Materials for High Energy Colliders, Erice, Italy (1999)
5. Mahajan, R., Bellovin, S.M., Floyd, S., Ioannidis, J., Paxson, V., Shenker, S.: Controlling high bandwidth aggregates in the network. Computer Communications Review **3** (2002)
6. Buyya, R., Giddy, J., Abramson, D.: A Case for Economy Grid Architecture for Service-Oriented Grid Computing. In: 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), In conjunction with IPDPS 2001, San Francisco, California, USA (April 2001)
7. Buyya, R., Giddy, J., Abramson, D.: An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications. In: The Second Workshop on Active Middleware Services (AMS 2000), Pittsburgh, USA (2000)
8. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for Scheduling Parameter Sweep Applications in Grid environments. In: 9th Heterogeneous Computing Systems Workshop (HCW 2000), Cancun,Mexico, IEEE CS Press (2000)
9. Takefusa, A., Tatebe, O., Matsuoka, S., Morita, Y.: Performance Analysis of Scheduling and Replication Algorithms on Grid Datafarm Architecture for High-Energy Physics Applications. In: Proceedings of the 12th IEEE international Symposium on High Performance Distributed Computing(HPDC-12), Seattle, USA, IEEE CS Press (2003)
10. Ranganathan, K., Foster, I.: Decoupling Computation and Data Scheduling in Distributed Data-Intensive Applications. In: Proceedings of the 11th IEEE Symposium on High Performance Distributed Computing (HPDC), Edinburgh, Scotland, IEEE Computer Society (2002)
11. Park, S.M., Kim, J.H.: Chameleon: A Resource Scheduler in a Data Grid Environment. In: Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2003 (CCGrid 2003), Tokyo, Japan, IEEE CS Press (2003)
12. Kim, S., Weissman, J.: A GA-based Approach for Scheduling Decomposable Data Grid Applications. In: Proceedings of the 2004 International Conference on Parallel Processing (ICPP 04), Montreal, Canada, IEEE CS Press (2003)
13. Venugopal, S., Buyya, R., Winton, L.: A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids. In: Proceedings of the 2nd Workshop on Middleware in Grid Computing (MGC 04) : 5th ACM International Middleware Conference (Middleware 2004), Toronto, Canada (2004)
14. Maheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing(JPDC) **59** (1999) 107–131

15. Beaumont, O., Legrand, A., Robert, Y., Carter, L., Ferrante, J.: Bandwidth-Centric Allocation of Independent Tasks on Heterogeneous Platforms. In: Proceedings of the 2002 International Parallel and Distributed Processing Symposium(IPDPS '02), Fort Lauderdale, California, USA, IEEE CS Press (2002)
16. Stockinger, H., Stockinger, K., Schikuta, E., Willers, I.: Towards a Cost Model for Distributed and Replicated Data Stores. In: 9th Euromicro Workshop on Parallel and Distributed Processing PDP 2001, Mantova, Italy, IEEE Computer Society Press (2001)
17. Dail, H., Casanova, H., Berman, F.: A Decoupled Scheduling Approach for the GrADS Environment. In: Proceedings of the 2002 IEEE/ACM Conference on Supercomputing (SC'02), Baltimore, USA, IEEE CS Press (2002)
18. Hoschek, W., Jaen-Martinez, F.J., Samar, A., Stockinger, H., Stockinger, K.: Data management in an international data grid project. In: Proceedings of the First IEEE/ACM International Workshop on Grid Computing(GRID '00), Bangalore, India, Springer-Verlag, Berlin (2000)
19. Vazhkudai, S., Tuecke, S., Foster, I.: Replica Selection in the Globus Data Grid. In: Proceedings of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001), Brisbane, Australia (2001)
20. Baru, C., Moore, R., Rajasekar, A., Wan, M.: The SDSC Storage Resource Broker. In: Procs. of CASCON'98, Toronto, Canada (1998)
21. Hui, T., Tham, C.: Reinforcement learning-based dynamic bandwidth provisioning for quality of service in differentiated services networks. In: Proceedings of IEEE International Conference on Networks (ICON 2003), Sydney, Australia (2003)
22. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. Journal of Future Generation Computing Systems **15** (1999) 757–768
23. Vazhkudai, S., Schopf, J.: Using Regression Techniques to Predict Large Data Transfers. International Journal of High Performance Computing Applications **17** (2003) 249–268
24. Faerman, M., Su, A., Wolski, R., Berman, F.: Adaptive Performance Prediction for Distributed Data-Intensive Applications. In: Proceedings of the 1999 IEEE/ACM Conference on Supercomputing (SC'99), Portland, Oregon, USA, IEEE CS Press (1999)

A Survivability Model for Cluster System

Khin Mi Mi Aung[1], Kiejin Park[2], and Jong Sou Park[1]

[1] Computer Engineering Dept., Hankuk Aviation University
{maung, jspark}@hau.ac.kr
[2] Division of Industrial and Information System Engineering, Ajou University
kiejin@ajou.ac.kr

Abstract. Even in an intrusion tolerant system, the resources will be fatigued if the intrusion is long lasting because of compromising iteratively or incrementally. In due course, the system will not provide even the minimum critical functionality. Thus we propose a model to increase the cluster system survivability level by maintaining the essential functionality. In this paper, we present the cluster recovery model with a software rejuvenation methodology, which is applicable in security field and also less expensive. Firstly, we perform the steady-state analysis of a cluster system and then study the 4th Generation Security Mechanism: Restore system with cold standby cluster. The basic idea is investigate the consequences for the exact responses in face of attacks and rejuvenate the running software or/and service, or/and reconfigure it. It shows that the system operates through intrusions and provides continued the critical functions, and gracefully degrades non-critical system functionality in the face of intrusions.

1 Introduction

After intrusion protection, detection and tolerant systems mechanisms, the next 4th security mechanism is the restore system. Restore system includes diagnosis, learning, reconfiguration, software rejuvenation, natural immunity and reflection [1]. Thus we propose a restore system model to increase the cluster system survivability level by maintaining the essential functionality. In this paper, we present the cluster recovery model with a software rejuvenation methodology, which is applicable in security field and also less expensive. An attacker carries out a DoS attack by making a resource out of action. The nature of attacks is very dynamic because attackers have the specific intention to attack and well prepare their steps in advance. So far no respond technique able to cope with all types of attacks has been found. In most attacks, attackers overwhelm the target system with a continuous flood of traffic designed to consume all system resources, such as CPU cycles, memory, network bandwidth, and packet buffers. These attacks degrade service and can eventually lead to a complete shutdown. In this work, we address attacks mainly related to CPU usage, physical memory and swap space usage, running processes, network flows and packets. It will automatically detect potential weaknesses and reconfigure with attack patterns, which are characterizing an individual type of attack and attack profiles. We had analyzed the attack datasets and injected the attacks events into a system, and

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 73–82, 2005.

learned the prior knowledge. The next step is to restore the system to a healthy state within a set time following the predictive alerts [2]. Software rejuvenation is a proactive fault management technique aimed at cleaning up the internal system state to prevent the occurrence of more severe future crash failures. It involves occasionally terminating an application or a system, cleaning its internal state and restarting it. IBM Software Rejuvenation is a tool to help increase server availability by proactively addressing software and operation system aging [3]. The effect of aging is captured as crush/hang failures [4].

In the current literature, there are significant numbers of researches, which are mainly concerned with survivability analysis. Jha et. al. [5]and Nikolopoulos et. al. [6] have studied reliability, latency and cost benefit model. Jha et. al. have analyzed survivability of network systems, which are service dependent; therefore a system architect should focus on the design of the system by analyzing only the service required of that system. They use a Constrained Markov Decision Process (CMDP) to form the basis of the survivability analysis, which is composed of reliability, latency, and cost benefit.

Liew et. al. [7] had presented a survivability function model. In their study, a survivability function is used as the measure instead of a single value for survivability. They evaluate network survivability in terms of nodes connected after a failure (disaster) that results in unavailable or destroyed nodes. The survivability function is described as the probability that a fraction of the nodes are connected to the central node. The function allows for different quantities to be calculated based on the network characteristics such as type of failure (disaster) and goodness of the network. The survivability function can calculate expected, worst-case, r-percentile, and probability of zero survivability. Newport [8] built node and link connectivity models. The terms connectivity and survivability are used interchangeably in their research. They measure survivability using Node Connectivity Factor (NCF) and Link Connectivity Factor (LCF). A modified cut-saturation algorithm in conjunction with Floyd's algorithm is used in the design process for networks. Moitra et. al. [9][10] simulated the model for managing survivability of network information systems. They propose a model to assess the survivability of a network system. Different parameters affect survivability such as the frequency and impact of attacks on a network system. The authors finally conclude that there is no absolute survivability and sites other measures of survivability such as relative survivability, worst-case survivability, and survivability with expected compromise. Simulations to analyze survivability used the Poisson model.

In this paper, we present a model to increase the cluster system survivability level using software rejuvenation. The organization of the paper is as follows. In Section 1, we define the problem and address related research. Section 2 presents a proposed model which can be used to analyze and proactively manage the effects of cluster network faults and attacks, and recover accordingly and in the following section, the model is analyzed and experiment results are given to validate the model solution. Finally, we conclude that software rejuvenation is a viable method and present further research issues.

2 Proposed Model

Significant features of various system resources may differ between specific attacks. And the response and restore methods would differ as well. In this work, the system has divided into three stages; healthy stage, restoration stage and failure stage (refer to Figure. 1). The model consists of five states: healthy state (H), infected state (I), rejuvenation state (R_j), reconfiguration state (R_c) and failure state (F). The healthy state represents the functioning and service providing phases. In the healthy stages, the systems aware to resist by various policies and offer proactive managements which are periodic diagnostics and automatic error log analysis, scheduled tasks (checking routine) based on experiences to assess the approximate frequency of unplanned outages due to resources exhaustion, monitoring server subsystems and software processes to ascertain common trends accompanying regular failures, error logging and alerts (error logging controls).

At the rejuvenation performing state, we need to be able to weigh the risk of policy with further damage against the policy of shutting the system in an emergency stage. In this case, the tools not only detect an attacker's presence but also support to get the information containments. The events are preconditions and are related to compromised system states. Susceptible to attack is an action or series of actions that lead to a compromise. Multiple defense mechanisms are the set of actions that may be taken to correct vulnerable conditions existing

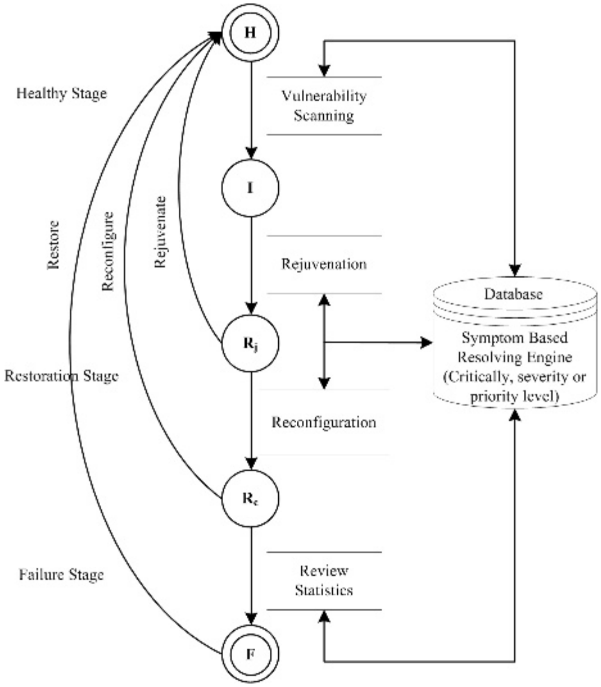


Fig. 1. Proposed Model

on the system or to move the system from a more compromised state to a less compromised state. To this end, software rejuvenation methodologies are reviewed and synthesized by the policies. The main strategies are occasionally stopping the executing software, cleaning the internal state and restarting by means of effectiveness of proactive managements, degrading mechanism, service stop, service restart, reboot and halt.

At the restoration stage, they may be decomposed into three types according to their specific attacks such as

- Performing rejuvenation only
- Performing reconfiguration only and
- Performing both rejuvenation and reconfiguration

For example, if an attacker carries out attack by overloading processes, causing resources to become unavailable, we will perform a rejuvenation process by gracefully terminating processes causing the resource overload and immediately restarting them in a clean state. But for the other kinds of attacks, we have to reconfigure the system according per their impact. In this case we have considered the reconfiguration state with various reconfiguration mechanisms, such as

- Patching (operating system patch, application patch),
- Version control (operating system version, application version),
- Anti virus (vaccine),
- Access control
  (IP blocking, port blocking, session drop, contents filtering), and
- Traffic control (bandwidth limit)

As an example, performing rejuvenation only could deter the attacks, which cause the process degradation such as spawn multiple processes, fork bombs, CPU overload etc. For the cases of process shutdown and system shutdown attacks, the attackers intend to halt a process or all processing on a system. Normally it happens by exploiting a software bug that causes the system to halt could cause system shutdown. In this case, just as with software bugs that are used to penetrate, so until the software bug is reconfigured, all systems of a certain type would be vulnerable. An example of attacks called mail bombardment or mail spam, the attacker accomplishes this attack by flooding the user with huge message or with very big attachments. Depending on how the system is configured, this could be counteracted by performing both reconfiguration and rejuvenation processes. To perform the various reconfiguration mechanisms, we have implemented the event manager, which contains the various strategies with respect to the various impact levels of the specific infected cases. Each type of event has its own routine, to be run when the attack takes place [2].

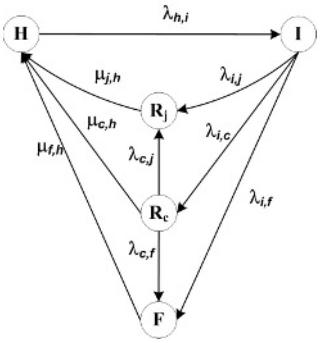


Fig. 2. State Transition Diagram of non-cluster system

3 A Survivability Model with Cold Standby Cluster

3.1 Steady-State Analysis on a Single Node Through Markov Process

According to the state transition diagram of Figure. 2 We denote as,
$\lambda_{h,i}$ = infected rate from the healthy state
$\lambda_{i,j}$ = rejuvenation rate from the infected state
$\mu_{j,h}$ = rejuvenation service rate to the healthy state
$\lambda_{i,c}$ = reconfiguration rate from the infected state
$\mu_{c,j}$ = reconfiguration service rate to the rejuvenation state
$\lambda_{c,f}$ = failure rate from the reconfiguration state
$\mu_{c,h}$ = reconfiguration service rate to the healthy state
$\lambda_{i,f}$ = failure rate from the infected state
$\mu_{f,h}$ = service rate from the failure state

And let the steady-state probabilities of the state of the system be
π_h = the probability that the system is in Healthy State
π_i = the probability that the system is in Infected State
π_r = the probability that the system is in Rejuvenation State
π_c = the probability that the system is in Reconfiguration State
π_f = the probability that the system is in Failure State
Using principle of the rate at which the process enters each state with the rate at which the process leaves can derive the balance equations for the system (refer to Figure 3).

$$\lambda_{h,i}\pi_h = \mu_{j,h}\pi_j + \mu_{c,h}\pi_c + \mu_{f,h}\pi_f \tag{1}$$

$$\pi_i = E\pi_h \tag{2}$$

$$\pi_c = \frac{\lambda_{i,c}}{F}E\pi_h \tag{3}$$

$$\pi_j = \left(\lambda_{i,j} + \frac{\lambda_{c,j}\lambda_{i,c}}{F}\right)E\frac{1}{\mu_{j,h}}\pi_h \tag{4}$$

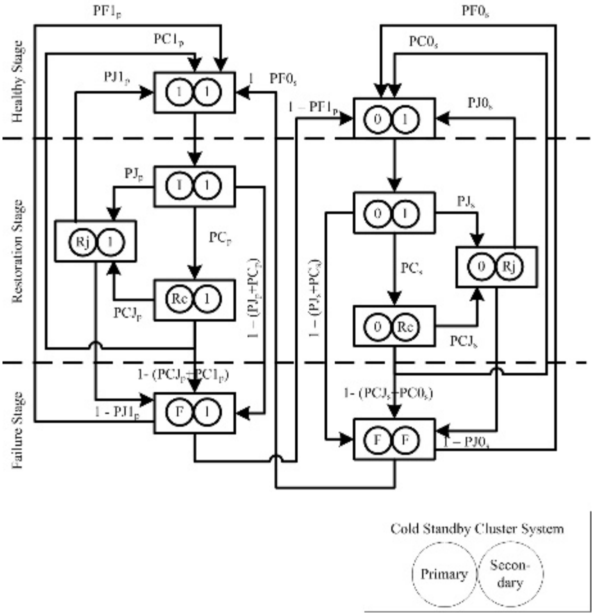


Fig. 3. Restoration system with cold standby cluster

$$\pi_f = \left(\lambda_{i,f} + \frac{\lambda_{c,f}\lambda_{i,c}}{F}\right) E \frac{1}{\mu_{j,h}} \pi_h \tag{5}$$

By solving above equation in terms of π_h and the condition $\pi_h + \pi_i + \pi_r + \pi_c + \pi_f$, we get

$$\pi_h = \frac{1}{1 + E + \frac{\lambda_{i,c}}{F} E} + \frac{1}{\left(\lambda_{i,j} + \frac{\lambda_{c,j}\lambda_{i,c}}{F}\right)} E \frac{1}{\mu_{j,h}} + \frac{1}{\left(\lambda_{i,f} + \frac{\lambda_{c,f}\lambda_{i,c}}{F}\right)} E \frac{1}{\mu_{j,h}} \tag{6}$$

Where $E = \frac{\lambda_{h,i}}{\lambda_{i,j} + \lambda_{i,c} + \lambda_{i,f}}$ and $F = \lambda_{c,f} + \lambda_{c,j} + \mu_{c,h}$

The availability for the steady-state analysis on a single node through Markov Process can be expressed as:

$$A = 1 - (\pi_f + \pi_j + \pi_c) \tag{7}$$

3.2 Steady-State Analysis with Two Nodes Through Semi-Markov Process

When only one of the states in the diagram violates the memoryless property, which means that soj0urn time in a state does not follow exponential distribution, the diagram is classified as a semi-Markov process. Semi-Markov models contain a Markov chain, which describes the stochastic transitions from state to state, and transition or 'sojourn' times, which describe the duration that the process takes to transition from state to state. We address the survivability model with semi-Markov process. We consider a cold standby cluster with two nodes through Semi-Markov process. One node is as an active (primary) and other as a standby (secondary) unit. The failure rate of the primary node and secondary node are different, and also the effect of failure of the primary node is different from that of secondary node. The state transition diagram is shown in Figure 3. Initially the system is in state (1,1). In the infected state, the system has to figure out whether rejuvenate or reconfigure to recover or limit the damage that may happen by an attack. If the primary node has to reconfigure, the system enters state $(R_c, 1)$ otherwise enters rejuvenation state $(R_j, 1)$. If both strategies fail then the primary system enters the fail state. When the primary is infected by active attacks, the system enters state (I, 1).When the primary node fails a protection switch successfully restores service by switching in the secondary unit, and the system enters state (0,1). If the node failure occurs when the system is in one of the states : (0,I) or $(0, R_c)$, the system fails and enters state (F,F).To calculate the steady-state availability of the proposed model, the stochastic process of equation 1 was defined.

$$X(t) : t > 0 \tag{8}$$

$XS = \{(1,1), (I,1), (R_j,1), (R_c,1), (F,1), (0,1), (0,R_j), (0,R_c), (F,F)\}$ Through SMP (Semi-Markov Process) analysis applying M/G/1, whose service time is general distribution; we have calculated the steady-state probability in each state. Healthy Stage: $\pi_{1,1} + \pi_{0,1}$ Restoration Stage: $\pi_{R_j,1} + \pi_{R_c,1} + \pi_{0,R_j} + \pi_{0,R_c}$ Failure Stage: $\pi_{F,1} + \pi_{F,F}$ As all the states shown in Figure 3 are attainable to each other, they are irreducible. Additionally, as they do not have a cycle and can return to a certain state, they satisfy the ergodicity (Aperiodic, Recurrent, and Nonnull) characteristics. Therefore, there is a probability in the steady-state of SMP for each state and each corresponding SMP can be induced by embedded DTMC (Discrete-time Markov Chain) using transition probability in each state. If we define the mean sojourn times in each state of SMP as $h_i's$ and define DTMC steady-state probability as $d_i's$, the steady-state probability in each state of SMP π_i can be calculated by equation 2 [11].

$$\pi_i = \frac{d_i h_i}{\sum_j d_j h_j}, i, j \in XS \tag{9}$$

The system availability in the steady-state is defined as equation 3, which is the same as the exclusion of the probability of being in (F, 1) and (F, F) in the state transition diagram.

$$A = 1 - (\pi_{F,1} + \pi_{F,F}) \tag{10}$$

The cluster systems are not survive in all of the rejuvenation process in the normal state (1), all of the switchover states, and the failure state (0). The survivability of cold standby cluster systems is defined as follows:

$$S = A - ((1 - \pi_{R_j,1}) + (1 - \pi_{R_c,1}) + (1 - \pi_{0,R_j}) + (1 - \pi_{0,R_c})) \tag{11}$$

4 Numerical Results

We perform the experiments using the same system operating parameters with [12]. Failure rate of the server is one time per year and repair time is fifteen hours. Rejuvenation is scheduled at every month. The rejuvenation and switchover time are 10 and 3 minutes, respectively. The expected downtime cost per unit is 100 times greater than that of the scheduled rejuvenation cost. The number of servers is varied from simplex to multiplex (n=4), at the same time we perform software rejuvenation with the interval from 10 days (rate = 3) to infinity (rate=0: no rejuvenation).

From the graph (Figure 4), the amount of survivability level increment from simplex to duplex is significant but from duplex to multiplex very little is shown. As infected states are removed frequently with high rejuvenation rates, the survivability of the cluster systems with simplex configuration increases. However, as the degree of redundancy is larger than or equal to 3, the improvement of availability is not significant. According to the required survivability level, the decision making of a rejuvenation rate is possible under consideration of various evaluation criteria such as state probabilities and downtime cost.

Figure 5 shows the relationship between switchover time and rejuvenation rate with survivability level. When switchover time is less than 15 minutes, a high rejuvenation rate is beneficial for improving survivability level. However when switchover time exceeds 15 minutes, frequent rejuvenation is not beneficial.

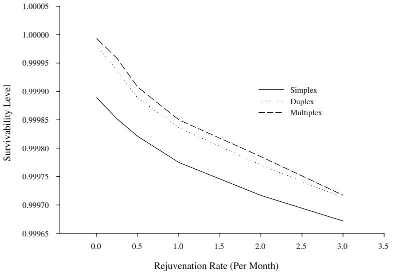


Fig. 4. Survivability level changes due to rejuvenation

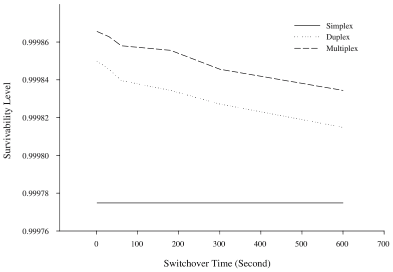


Fig. 5. Survivability level changes due to switch over time

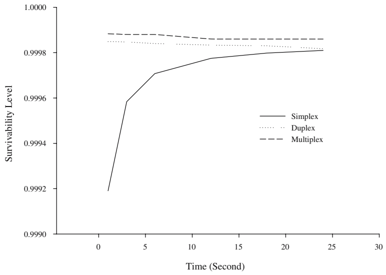


Fig. 6. Survivability Level Changes Due to Failure Rate

Due to this fact, switchover time must be considered carefully when determining the rejuvenation policy. The influence of failure rates along with rejuvenation rates on survivability level is shown in Figure 6. In the duplex configuration, failure rates are less sensitive to rejuvenation rate for availability. These results suggest that software reliability is more important than hardware reliability in improvement of the survivability of cluster systems.

5 Conclusion

In this paper, we have presented a survivability model for cold standby system. This study defined 10 states for a cold-standby cluster system, computed DTMC steady-state probability and SMP steady-state probability using the transition probability and the mean sojourn time in each state and based on the results, defined the availability of general systems. We have demonstrated the model can be used to analyze and proactively manage the effects of cluster network faults and attacks, and restore accordingly. According to the system operating parameters, we have modeled and analyzed steady-state probability and survivability level of cluster systems under DoS attacks by adopting a software rejuvenation technique. The result shows that the system operates through intrusions and provides continued the critical functions, and gracefully degrades non-critical system functionality in the face of intrusions. As an ongoing work, we are performing our model with the real sojourn times of specific attacks in order to generalize it with various attacks. We are analyzing a variety of probability distributions in the real attack data, which is, described the attackers' transitions and the sojourn time that they spend in every state. The integration of response time and throughput with downtime cost will provide a more accurate evaluation measure.

Acknowledgment. This research is granted by Ajou University and Regional Research Center(RRC) Program, a research program of Korea Science and Engineering Foundation.

References

1. J. Lala: Introduction to the Proceedings of Foundations of Intrusions Tolerant Systems (OASIS 03), Dec. 2003.
2. J. Park and K. Aung: Transient Time Analysis of Network Security Survivability Using DEVS, Lecture Notes in Computer Science, Springer, Vol. 3397, ISBN 3-540-24476, pp.607-616, 2005.
3. Y. Huang, C. Kintala, N. Kolettis and N. Fulton: Software Rejuvenation: Analysis, Module and Applications, Proc. of FTCS-25 Pasadena, CA pp.381-390, 1995.
4. S. Garg, A. Puliafito, M. Telek and K. S. Trivedi: Analysis of Software Rejuvenation Using Markov Regenerative Stochastic Petri Nets, International Symposium on Software Reliability Engineering, Oct. 1995.
5. S. Jha and J. Wing: Survivability Analysis of Networked Systems, Proc. of the 23rd International Conference on Software Engineering, IEEE, pp.872-874, 2001.
6. S. Nikolopoulos, A. Pitsillides and D. Tipper: Addressing Network Survivability Issues by Finding the Kbest Paths through a Trellis Graph, 16th Annual Joint Conference of the IEEE Computer and Communications Societies, Vol. 1, pp.370-377, 1997.
7. S. Liew and K. Lu: A Framework for Network Survivability Characterization, IEEE International Conference on Communications, pp.441-451, 1992.
8. K. Newport: Incorporating Survivability Considerations Directly into the Network Design Process, 9th Annual Joint Conference of the IEEE Computer and Communication Societies, pp.1963-1970, 1990.
9. Moitra, D. Soumyo and S. Konda: Survivability of Network Systems: An Empirical Analysis, SEI, Dec 2000.
10. Moitra, D. Soumyo and S. Konda: A Simulation Model for Managing Survivability of Networked Information Systems, SEI, Dec 2002.
11. K. Trivedi: Probability and Statistics with Reliability Queueing and Computer Science Applications, John Wiley and Sons, Inc. 2003.
12. S. Garge, "Analysis of Preventive Maintenance in Transactions Based Software Systems," IEEE Transactions on Computers Vol. 47(1), pp.96-107, 1998.

Localization Techniques for Cluster-Based Data Grid

Ching-Hsien Hsu[1], Guan-Hao Lin[1], Kuan-Ching Li[2], and Chao-Tung Yang[3]

[1] Department of Computer Science and Information Engineering,
Chung Hua University, Hsinchu 300 Taiwan
chh@chu.edu.tw
[2] Department of Computer Science and Information Management,
Providence University, Taichung 43301 Taiwan
kuancli@pu.edu.tw
[3] Department of Computer Science and Information Engineering,
Tunghai University, Taichung 40704 Taiwan
ctyang@mail.thu.edu.tw

Abstract. In this paper, we present an efficient method for optimizing localities of data distribution when executing data parallel applications. The data to logical grid nodes mapping technique is employed to enhance the performance of parallel programs on cluster grid. Cluster grid is a typical computational grid environment consists of several clusters located in multiple campuses that are distributed globally over the Internet. Objective of the proposed technique is to reduce inter-cluster communication overheads and to speed the execution of data parallel programs in the underlying distributed cluster grid. The theoretical analysis and experimental results show improvement of communication costs and scalable of the proposed techniques on different hierarchical cluster grids.

1 Introduction

Computing grid system [8] integrates geographically distributed computing resources to establish a virtual and high expandable parallel machine; cluster grid is a typical paradigm in which each cluster is geographically located in different campus and is connected by software of computational grids through the Internet. In cluster grid, computers might exchange data through network to other computers to run job completion. This consequently incurs two kinds of communication between grid nodes in a cluster grid. If the two grid nodes are geographically belong to different clusters, the messaging should be accomplished through the Internet. We refer this kind of data transmission as *external communication*. If the two grid nodes are geographically in the same space domain, the communications take place within a cluster; we refer this kind of data transmission as *interior communication*. Intuitionally, the external communication is usually with higher communication latency than that of the interior communication sine the data should be routed through numbers of layer-3 routers or higher-level network devices over the Internet. Therefore, to efficiently execute parallel programs on cluster grid, it is extremely critical to avoid large amount of external communications.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 83 – 92, 2005.

This paper presents a generalized processor reordering technique for minimizing external communications of data parallel program on cluster grid. We employ the problem of data alignments and realignments in data parallel programming languages to examine the effective of the proposed data to logical processor mapping technique. As many research discovered that many parallel applications require different access patterns to meet parallelism and data locality during program execution. This will involve a series of data transfers such as array redistribution. For example, a 2D-FFT pipeline involves communicating images with the same distribution repeatedly from one task to another. Consequently, the computing nodes might decompose local data set into sub-blocks uniformly and remapped these data blocks to designate processor group. From this phenomenon, we propose a processor-reordering scheme to reduce the volume of external communications of data parallel programs in cluster grid. The key idea is that of distributing data to grid/cluster nodes according to a mapping function at data distribution phase initially instead of in numerical-ascending order. The theoretical analysis and experiments results of the processor-reordering technique on mapping data to logical grid nodes show improvement of volume of external communications and conduce to better performance of data alignment in different cluster grid topologies.

2 Related Work

Research works on computing grid have been broadly discussed on different aspects, such as security, fault tolerance, resource management [10, 4], job scheduling [2, 18, 19, 20], and communication optimizations [6]. From the issue of communication optimizations, Dawson *et al.* [6] addressed the problems of optimizations of user-level communication patterns in local space domain for cluster-based parallel computing. Plaat *et al.* analyzed the behavior of different applications on wide-area multi-clusters [17, 3]. Similar research works were studied in the past years over traditional supercomputing architectures [13, 14]. Guo *et al.* [12] eliminated node contention in communication step and reduced communication steps with schedule table. Y. W. Lim *et al.* [16] presented an efficient algorithm for block-cyclic data realignments. A processor mapping technique presented by Kalns and Ni [15] can minimize the total amount of communicating data. Namely, the mapping technique minimizes the size of data that need to be transmitted between two algorithm phases. Lee *et al.* [11] proposed similar method to reduce data communication cost by reordering the logical processors' id. They proposed four algorithms for logical processor reordering. They also compared the four reordering algorithms under various conditions of communication patterns.

There is significant improvement of the above research for parallel applications on distributed memory multi-computers. However, most techniques applicable for applications running on local space domain, like single cluster or parallel machine. For a global grid of clusters, these techniques become inapplicable due to various factors of Internet hierarchical and its communication latency. In this following discussion, our emphasis is on minimizing the communication costs for data parallel programs on cluster grid and on enhancing data distribution localities.

3 Data Distribution on Cluster Grid

Appropriate data distribution is critical for balancing the computational load. A typical function to decompose the data equally can be accomplished via the BLOCK distribution directive in many data parallel programming languages. However, a good mapping of data to logical processors must change adaptively in order to ensure good data locality and reduce inter-processor communication during program running. For example, a global array could be equally allocated to a set of processors at beginning in `BLOCK` distribution manner. As the algorithm goes into another phase that requires to access fine-grain data patterns, processors might divide their own data set into sub-blocks locally and then exchange these sub-blocks with corresponding processors over the cluster grid. To explicitly define the problem, upon the number of clusters (C), number of computing nodes in each cluster (n_i) and the degree of refinement, we consider two models of cluster grid when performing data reallocation.

3.1 Identical Cluster Grid Model

Identical cluster grid is composed by several clusters in which each cluster provides the same number of computing nodes (identical n_i) involved in the computation. Figure 1 shows an example of this scenario. Cluster 1 owns logical grid nodes P_0, P_1, …P_{m-1}, cluster-2 owns P_m, P_{m+1}, …, P_{2m-1}, and so on. If the number of nodes provided by cluster i is n_i, we have $n_i = m$, for all i=1~C. Considering the data reallocation problem described above, in this model, each node is initially responsible to hold a complete block. When algorithm goes into a refinement phase (assume the degree of refinement is k), it will partition its own block of data into k sub-blocks locally and redistribute them over the global grid processor set. This process will bring volumes of inter-processor message exchange during program execution. These exchanges could be intra-cluster or inter-cluster communications. Because of network latency of inter-cluster message passing, how to increase data localities and transform inter-cluster communications as intra-clusters becomes an important subject to the performance of these applications.

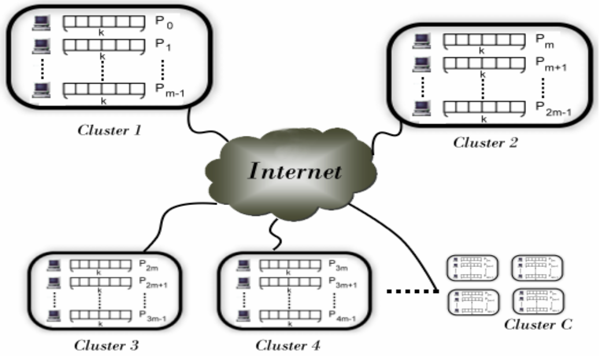


Fig. 1. Identical cluster grid model

3.2 Non-identical Cluster Grid Model

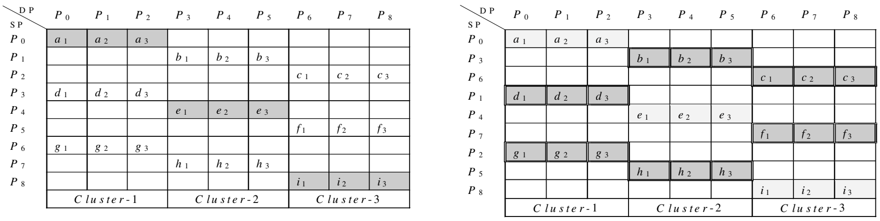
Non-identical cluster grid is composed by several clusters in which each cluster may provides different number of computing nodes (non-identical n_i) involved in the computation. Because of the unequal n_i, the total number of grid nodes can be denoted as $P = \sum_{i=1}^{C} n_i$, where C is the number of clusters in grid. Since cluster grid is composed of heterogeneous cluster systems over the internet, the overheads of interior and external communications among different clusters should be identified individually. To formulate cost model for evaluating the communication costs in cluster grid, let T_i represents the time of two nodes both in *Cluster-i* to transmit per unit data; I_i is the total number of interior communications within cluster i; for external communication between cluster i and cluster j, T_{ij} is used to represent the time of processor p in cluster i and processor q in cluster j to transmit per unit data; similarly, the total number of external communications between cluster i and cluster j is denoted by E_{ij}. According to these declarations, we can have equation $T_{comm} = \sum_{i=1}^{C}(I_i \times T_i) + \sum_{i,j=1,\, i\neq j}^{C}(E_{ij} \times T_{ij})$. This equation explicitly defines the communication costs of a parallel program running on a cluster grid. However, there are various factors might cause unstable communication delay over internet; it is difficult to estimate accurate costs. As the need of a criterion for performance modeling, integrating the interior and external communications among all clusters into points is an alternative mechanism to get legitimate evaluation. Therefore, we totaled the number of these two terms as $|I| = \sum_{i=1}^{C} I_i$, the number of interior communications, and $|E| = \sum_{i,j=1,i\neq j}^{C} E_{ij}$, the number of external communications for the following discussion and analysis.

4 Localized Data Mapping

4.1 Motivating Example

To motivate the proposed localization technique for data reallocation, we use the example shown in Figure 2 for explanation. As demonstrated in section 3, the degree of data refinement is set to three (K = 3). This example also assumes an identical cluster grid that consists of three clusters and each cluster provides three nodes to join the computation. In algorithm phase, to accomplish the fine-grained data distribution, processors partition its own block into K sub-blocks and distribute them to corresponding destination processors in ascending order of processors' id that specified in most data parallel programming languages. For example, processor P_0 divides its data block A into a_1, a_2, and a_3; it then distributes these three sub-blocks to processors P_0, P_1 and P_2, respectively. Because processors P_0, P_1 and P_2 belong to the same cluster with P_0; therefore, these three communications are interior. However, the same situation on processor P_1 generates three external communications. Because processor P_1 divides its local data block B into b_1, b_2, and b_3. It then distributes these three sub-blocks to processors P_3, P_4 and P_5, respectively. As processor P_1 belongs to *Cluster* 1 and processors P_3, P_4 and P_5 belong to *Cluster* 2. Therefore, this results three external communications. Figure 2(a) summarizes all messaging patterns of this example into communication table. We noted that messages {a_1, a_2, a_3}, {e_1, e_2, e_3} and {i_1, i_2, i_3} are interior communications (| I | = 9); all the others are external communications (| E | = 18).

The idea of changing data to logical processor mapping [11, 15] is employed in our implementation. Such techniques were used in many previous research works to minimize data transmission time of runtime array redistribution. In cluster grid, we can derive a mapping function to produce a realigned sequence of logical processors' id for grouping communications into local cluster. Given an identical cluster grid with C clusters, a new logical id for replacing processor P_i can be determined by $New(P_i) = (i \bmod C) * K + (i / C)$, where K is the degree of data refinement. Figure 2(b) shows the communication table of the same example after applying the above reordering scheme. The source data is distributed according to the reordered sequence of processors' id, i.e., <P_0, P_3, P_6, P_1, P_4, P_7, P_2, P_5, P_8> which is computed by mapping function. In the target distribution, processor P_0 distributes three sub-blocks to processors P_0, P_1 and P_2 in the same cluster. Similarly, processor P_3 sends three sub-blocks to processors P_3, P_4 and P_5 that are in the same cluster with $P_{3;}$ and processor P_6 sends e_1, e_2 and e_3 to processors P_6, P_7 and P_8 that causes three interior communications. All other processors generate three interior communications as well. Therefore, we have |I|= 27 and | E | = 0.



DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_1				b_1	b_2	b_3			
P_2							c_1	c_2	c_3
P_3	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_5							f_1	f_2	f_3
P_6	g_1	g_2	g_3						
P_7				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			Cluster-2			Cluster-3		

(a)

DP / SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
P_0	a_1	a_2	a_3						
P_3				b_1	b_2	b_3			
P_6							c_1	c_2	c_3
P_1	d_1	d_2	d_3						
P_4				e_1	e_2	e_3			
P_7							f_1	f_2	f_3
P_2	g_1	g_2	g_3						
P_5				h_1	h_2	h_3			
P_8							i_1	i_2	i_3
	Cluster-1			Cluster-2			Cluster-3		

(b)

Fig. 2. Communication tables of data reallocation over cluster grid. (a) Without data mapping (b) With data mapping.

4.2 Identical Cluster Grid

For the case of K (degree of refinement) is not equal to n (the number of grid nodes in each cluster), the mapping function becomes impracticable. In this subsection, we propose a grid node replacement algorithm for optimizing distribution localities of data reallocation. According to the relative position of the first of consecutive sub-blocks that produced by each processor, we can determine the best target cluster as candidate for node replacement. Combining with a load balance policy among clusters, this algorithm can effectively improve data localities. Figure 3 gives an example of data reallocation on cluster grid, which has four clusters. Each cluster provides three processors. The degree of data refinement is set to five. Figure 3(a) demonstrates an original reallocation communication patterns. We observe that | I | = 12 and | E | = 36.

If we change the distribution of block B to processors reside in cluster 2 (P_3, P_4 or P_5) or cluster 3 (P_6, P_7 or P_8) in the source distribution, we find that the communications could be centralized in local cluster for some parts of sub-blocks. Because cluster 2 and cluster 3 will be allocated the same number of sub-blocks in the target distribution, therefore processors belong to these two clusters have the same priority for node replacement. In our algorithm, P_3 is first assigned to replace P_1. For block C, most sub-blocks will be reallocated to processors in cluster 4, therefore the first available node P_9 is assigned to replace P_2. Similar determination is made to block D and results P_1 replace P_3. For block E, cluster 2 and cluster 3 have the same amount of sub-blocks. Processors belong to these two clusters are candidates for node replacement. However, according to the load balance policy among clusters, cluster 2 remains two available processors for node replacement while cluster 3 has three; our algorithm will select P_6 to replace P_4. Figure 3(b) gives the communication tables when applying data to logical grid nodes mapping technique. We obtain | I | = 28 and | E | = 20.

DP SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
P_0	a_1	a_2	a_3	a_4								
P_1					b_1	b_2	b_3	b_4				
P_2									c_1	c_2	c_3	c_4
P_3	d_1	d_2	d_3	d_4								
P_4					e_1	e_2	e_3	e_4				
P_5									f_1	f_2	f_3	f_4
P_6	g_1	g_2	g_3	g_4								
P_7					h_1	h_2	h_3	h_4				
P_8									i_1	i_2	i_3	i_4
P_9	j_1	j_2	j_3	j_4								
P_{10}					k_1	k_2	k_3	k_4				
P_{11}									l_1	l_2	l_3	l_4
	Cluster-1			Cluster-2			Cluster-3			Cluster-4		

(a)

DP SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}
P_0	a_1	a_2	a_3	a_4								
P_3					b_1	b_2	b_3	b_4				
P_9									c_1	c_2	c_3	c_4
P_1	d_1	d_2	d_3	d_4								
P_6					e_1	e_2	e_3	e_4				
P_{10}									f_1	f_2	f_3	f_4
P_2	g_1	g_2	g_3	g_4								
P_4					h_1	h_2	h_3	h_4				
P_{11}									i_1	i_2	i_3	i_4
P_5	j_1	j_2	j_3	j_4								
P_7					k_1	k_2	k_3	k_4				
P_8									l_1	l_2	l_3	l_4
	Cluster-1			Cluster-2			Cluster-3			Cluster-4		

(b)

Fig. 3. Communication tables of data reallocation on identical cluster grid. (C=4, n=3, K=4) (a) Without data mapping (b) With data mapping.

4.3 Non-identical Cluster Grid

Let's consider a more complex example in non-identical cluster grid, the number of nodes in each cluster is different. We need to add global information of cluster grid into algorithm for estimating the best target cluster as candidate for node replacement.

Figure 4 shows a non-identical cluster grid composed by four clusters. The number of processors provided by these clusters is 2, 3, 4 and 5, respectively. We also set the degree of refinement as *K*=5. Figure 4(a) presents the table of original communication patterns that consists of 19 interior communications and 51 external communications. Applying our node replacement algorithm, the derived sequence of logical grid nodes is <P_2, P_5, P_9, P_3, P_6, P_{10}, P_4, P_{11}, P_0, P_7, P_{12}, P_1, P_8, P_{13}>. This data to grid nodes mapping produces 46 interior communications and 24 external communications. This result reflects the effectiveness of the node replacement algorithm in term of minimizing inter-cluster communication overheads.

DP SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}
P_0	a_1	a_2	a_3	a_4	a_5									
P_1						b_1	b_2	b_3	b_4	b_5				
P_2	c_5										c_1	c_2	c_3	c_4
P_3		d_1	d_2	d_3	d_4	d_5								
P_4							e_1	e_2	e_3	e_4	e_5			
P_5	f_4	f_5										f_1	f_2	f_3
P_6			g_1	g_2	g_3	g_4	g_5							
P_7								h_1	h_2	h_3	h_4	h_5		
P_8	i_3	i_4	i_5										i_1	i_2
P_9				j_1	j_2	j_3	j_4	j_5						
P_{10}									k_1	k_2	k_3	k_4	k_5	
P_{11}	l_2	l_3	l_4	l_5										l_1
P_{12}					m_1	m_2	m_3	m_4	m_5					
P_{13}										n_1	n_2	n_3	n_4	n_5
	Cluster1		Cluster2			Cluster3				Cluster4				

(a)

DP SP	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}
P_2	a_1	a_2	a_3	a_4	a_5									
P_5						b_1	b_2	b_3	b_4	b_5				
P_9	c_5										c_1	c_2	c_3	c_4
P_3		d_1	d_2	d_3	d_4	d_5								
P_6							e_1	e_2	e_3	e_4	e_5			
P_{10}	f_4	f_5										f_1	f_2	f_3
P_4			g_1	g_2	g_3	g_4	g_5							
P_{11}								h_1	h_2	h_3	h_4	h_5		
P_0	i_3	i_4	i_5										i_1	i_2
P_7				j_1	j_2	j_3	j_4	j_5						
P_{12}									k_1	k_2	k_3	k_4	k_5	
P_1	l_2	l_3	l_4	l_5										l_1
P_8					m_1	m_2	m_3	m_4	m_5					
P_{13}										n_1	n_2	n_3	n_4	n_5
	Cluster1		Cluster2			Cluster3				Cluster4				

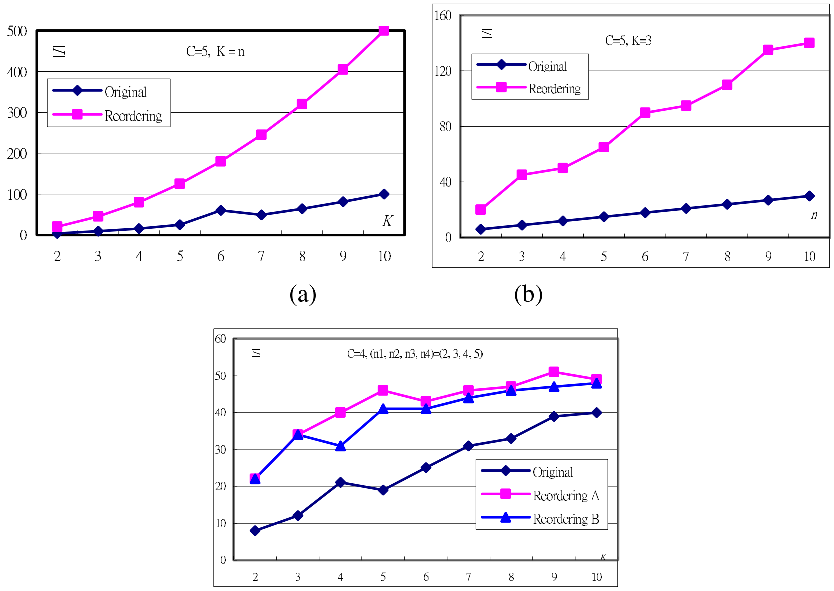
(b)

Fig. 4. Communication tables of data reallocation on non-identical cluster grid. (a) Without data mapping (b) With data mapping.

5 Performance Evaluation

5.1 Theoretical Estimate

This section presents the theoretical value of processor reordering technique in different hierarchy of cluster grid. For the case of data reallocation on an identical cluster grid that consists of five clusters (*C*=5) and *K* = *n*, the values of *K* vary from 2 to 10. The results in Figure 5(a) show that the processor reordering technique provides more interior communications than the method without processor reordering. Figure 5(b) gives the number of interior communications for both methods when $n \neq K$ and values of *n* vary from 2 to 10. Figure 5(c) compares the amount of interior communications for three methods, the original method, reordering algorithm A and reordering algorithm B. The difference between algorithms A and B is that algorithm A uses load balance policy to select target cluster for node replacement while algorithm B does not. As the results shown in Figure 5(c), reordering algorithm A has better performance than the other methods. Overall, the reordering technique enhances localities of data distribution and conduce lower communication overheads. We emphasize that the result reported in Figure 5 is improvement ratio come from theoretical estimate which will not be affected by network latency.



(a) (b)

(c)

Fig. 5. Improvement of interior communications (a) *C*=5 and *K* = *n* (b) *C*= 5, *K*=3 (c) non-identical cluster grid C=4, n_1= 2, n_2= 3, n_3= 4, n_4= 5

5.2 Experimental Results

To evaluate the performance of the proposed technique, we have implemented processor reordering technique with the application of data reallocation on *Taiwan UniGrid* [1], eight universities' clusters are geographically internet-connected. Each cluster owns different number of computing nodes. The programs were written in the single program multiple data (*SPMD*) programming paradigm with C+MPI codes.

Figure 6 shows the execution time of the methods with and without processor reordering to perform data reallocation on an identical cluster grid with *C*=*n*=4 and *K*=3. The size of test data is 10 MB that required remote I/O access. Different combinations of cluster grid denoted as *NTCH*, *NTCI*, *NTCD*, etc. were tested. The composition of these labels is summarized in Table 1.

In this experiment, method with processor reordering technique outperforms the method that without processor reordering. Compare to the results given in Figure 6, this experiment matches the theoretical estimation. It also satisfying reflects the efficiency of the processor reordering technique. This experimental result shows the proposed localization methods provide significant improvement.

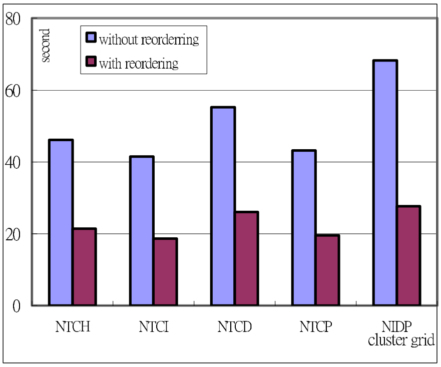


Table 1. Labels of different cluster grid

Label	Organization
N	*National Center for High Performance Computing*
T	*National Tsing Hua University*
C	*Chung Hua University*
H	*Tung Hai University*
I	*Institute of Information Science, Academia Sinica*
D	*National Dong Hwa University*
P	*Providence University*

Fig. 6. Execution time to perform data reallocation on different combination of cluster grids in Taiwan *UniGrid*

6 Conclusions

In this paper, we have presented a generalized processor reordering method for localizing distributions of parallel data. The data to logical grid nodes mapping technique is employed to enhance the performance of parallel programs on cluster grid. Effectiveness of the proposed technique is to reduce inter-cluster communication overheads and to speed the execution of data parallel programs in the underlying distributed cluster grid. The theoretical analysis and experimental results show improvement of communication costs and scalable of the proposed techniques on different hierarchical cluster grids.

Acknowledgement

The authors would like to acknowledge the National Center for High-Performance Computing for sponsoring the Taiwan UniGrid project, under the national project, “Taiwan Knowledge Innovation National Grid”. This research is also supported partially by National Science Council, Taiwan, under grant number NSC-93-2213-E-216-029.

References

1. Taiwan UniGrid, http://unigrid.nchc.org.tw
2. O. Beaumont, A. Legrand and Y. Robert, ”Optimal algorithms for scheduling divisible workloads on heterogeneous systems,” *Proceedings of the 12th IEEE Heterogeneous Computing Workshop,* 2003.

3. Henri E. Bal, Aske Plaat, Mirjam G. Bakker, Peter Dozy, and Rutger F.H. Hofman, "Optimizing Parallel Applications for Wide-Area Clusters," *Proceedings of the 12th International Parallel Processing Symposium IPPS'98*, pp 784-790, 1998.
4. M. Faerman, A. Birnbaum, H. Casanova and F. Berman, "Resource Allocation for Steerable Parallel Parameter Searches," *Proceedings of GRID'02*, 2002.
5. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta and K. Vahi, "The role of planning in grid computing," *Proceedings of ICAPS'03*, 2003.
6. J. Dawson and P. Strazdins, "Optimizing User-Level Communication Patterns on the Fujitsu AP3000," *Proceedings of the 1st IEEE International Workshop on Cluster Computing*, pp. 105-111, 1999.
7. I. Foster, "Building an open Grid," *Proceedings of the second IEEE international symposium on Network Computing and Applications*, 2003.
8. I. Foster and C. Kesslcman, "The Grid: Blueprint for a New Computing Infrastructure," Morgan Kaufmann, ISBN 1-55860-475-8, 1999.
9. I. Foster and C. Kesslcman, "Globus: A metacomputing infrastructure toolkit," *Intl. J. Supercomputer Applications*, vol. 11, no. 2, pp. 115-128, 1997.
10. James Frey, Todd Tannenbaum, M. Livny, I. Foster and S. Tuccke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids," *Journal of Cluster Computing*, vol. 5, pp. 237 – 246, 2002.
11. Saeri Lee, Hyun-Gyoo Yook, Mi-Soon Koo and Myong-Soon Park, "Processor reordering algorithms toward efficient GEN_BLOCK redistribution," *Proceedings of the 2001 ACM symposium on Applied computing*, 2001.
12. M. Guo and I. Nakata, "A Framework for Efficient Data Redistribution on Distributed Memory Multicomputers," *The Journal of Supercomputing*, vol.20, no.3, pp. 243-265, 2001.
13. Florin Isaila and Walter F. Tichy, "Mapping Functions and Data Redistribution for Parallel Files," *Proceedings of IPDPS 2002 Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, Fort Lauderdale*, April 2002.
14. Jens Koonp and Eduard Mehofer, "Distribution assignment placement: Effective optimization of redistribution costs," *IEEE TPDS*, vol. 13, no. 6, June 2002.
15. E. T. Kalns and L. M. Ni, "Processor mapping techniques toward efficient data redistribution," *IEEE TPDS*, vol. 6, no. 12, pp. 1234-1247, 1995.
16. Y. W. Lim, P. B. Bhat and V. K. Parsanna, "Efficient algorithm for block-cyclic redistribution of arrays," *Algorithmica*, vol. 24, no. 3-4, pp. 298-330, 1999.
17. Aske Plaat, Henri E. Bal, and Rutger F.H. Hofman, "Sensitivity of Parallel Applications to Large Differences in Bandwidth and Latency in Two-Layer Interconnects," *Proceedings of the 5th IEEE High Performance Computer Architecture HPCA'99*, pp. 244-253, 1999.
18. Xiao Qin and Hong Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proceedings of the 30th ICPP,* Valencia, Spain, 2001.
19. S. Ranaweera and Dharma P. Agrawal, "Scheduling of Periodic Time Critical Applications for Pipelined Execution on Heterogeneous Systems," *Proceedings of the 30th ICPP,* Valencia, Spain, 2001.
20. D.P. Spooner, S.A. Jarvis, J. Caoy, S. Saini and G.R. Nudd, "Local Grid Scheduling Techniques using Performance Prediction," *IEE Proc. Computers and Digital Techniques*, 150(2): 87-96, 2003.

GridFTP and Parallel TCP Support in NaradaBrokering

Sang Boem Lim[1], Geoffrey Fox[2], Ali Kaplan[2],
Shrideep Pallickara[2], and Marlon Pierce[2]

[1] Korea Institute of Science and Technology Information,
(KISTI), Daejeon, Republic of Korea
slim@kisti.re.kr
[2] Pervasive Technology Labs at Indiana University,
Bloomington, IN 47404-3730
{gcf, alikapla, spallick, marpierc}@indiana.edu

Abstract. Many of the key features of file transfer mechanisms like reliable file transferring and parallel transferring are developed as part of the service. It makes very hard to re–use the same code for the different systems. We are trying to overcome this disadvantage by decoupling useful features of file transfer mechanisms from the implementation of the service and protocol, and instead placed into the messaging substrate. We may thus treat file transfer operations as a specific usage case for a more general messaging environment. This will allow us to provide file transfer quality of service to other file transfer tools that does not have same features.

1 Introduction

Today's network environments require people to download many things on a daily basis. Especially new technologies developed recently, like Grid environments, require reliable, secure high performance file transfer as the most important services. GridFTP [2] [6] is the one of the most common data transfer services for the Grid and is a key feature of Data Grids [1]. This protocol provides secure, efficient data movement in Grid environments by extending the standard FTP protocol. In addition to the standard FTP features, the GridFTP protocol supports various features offered by the Grid storage systems currently in use.

Even though GridFTP has good features of file recovery technologies,many interesting features of GridFTP are tied to its protocol and implementation. Providing these features to other file transfer services (such as those based on Web Services, for instance) requires re–implementation and re-engineering. These shortcomings may be addressed by inserting a reliable, high performance messaging substrate between the client and service. This addresses specific problems in GridFTP client lifetimes, but more generally will allow us to extend GridFTP-like features to other services without extensive re–implementation. According to specification of GridFTP [2], GridFTP also has a restriction that the client needs to remain active at all the times until the transfer finishes.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 93–102, 2005.

This in turn implies that we cannot use the rich set of recovery features of GridFTP when the client state has been lost. In the event of client state loss, transfer has to restart from scratch.

2 Related Work

We are using many different file transfer mechanisms on daily basis. One of the most commonly used file transfer mechanism is File Transfer Protocol (FTP) [5]. This is the simplest way to exchange files between computers. FTP is an application protocol that uses the TCP/IP protocols. A more secure replacement for the common FTP, protocol is Secure Copy (SCP), which uses the Secure Shell (SSH) as the lower-level communication protocol. From the popularity of World Wide Web, we are also commonly using Hypertext Transfer Protocol (HTTP) as mechanism for transferring files. Even though some of file transfer mechanisms are quite reliable, these mechanisms do not provide guaranteed, reliable file transfer features like automatic recovery from failures.

Issues about reliable file transfer mechanism are more actively discussed and developed from the Grid community recently. More relevant service to our project is Reliable File Transfer (RFT) [8] [11] service developed by the Globus. RFT service provides reliable file transfer mechanisms like automatic failure recovery. In the next section we will discuss more about behaviors of RFT.

The RFT is developed with automatic failure recovery while overcoming the limitation of its predecessor technology, GridFTP. The most important idea added to the RFT service is automatic failure recovery mechanism when any problems are occurred during file transfer like dropped connections and temporary network outage. The RFT is dealing with problem by performing a retry until the problem is resolved. The RFT also will inherit all the features that GridFTP has since it is built on top of existing GridFTP. The RFT will inherit most of the automatic recovery features like restart support and remote problems of the RFT service and it also will not lose performance of GridFTP.

The RFT service resolved a strict restriction of its predecessor GridFTP. The client of GridFTP needs to remain active at all the times until the transfer finishes. However, the RFT no longer requires this restriction. The RFT introduced a non-user-based service. This service will store the transfer state in a persistent manner and this state will be used to recover transfer from the last marker recorded for that transfer when failure occurs including the client state failure.

3 NaradaBrokering

NaradaBrokering [9] [10] is messaging middleware designed to run on a large network of cooperating broker nodes (we avoid the use of the term servers to distinguish it clearly from the application servers that would be among the sources/sinks to messages processed within the system). Communication within NaradaBrokering is asynchronous and the system can support large client configurations publishing messages at a very high rate. The system places no restrictions on the number, rate and size of messages issued by clients.

In NaradaBrokering entities can also specify constraints on the Quality-of-Service (QoS) related to the delivery of messages. Among these services is the reliable delivery service, which facilitates delivery of events to interested entities in the presence of node and link failures. Furthermore, entities are able to retrieve any events that were issued during an entity's absence (either due to failures or an intentional disconnect). The scheme can also ensure guaranteed exactly-once ordered delivery.

Another service, relevant to this paper, is NaradaBrokering's Fragmentation/Coalescing service. This service splits large files into manageable fragments and proceeds to publish individual fragments. Upon receipt at a consuming entity these fragments are stored into a temporary area. Once it has been determined (by the coalescing service) that all the fragments for a certain file have been received, these fragments are coalesced into one large file and a notification is issued to the consuming entity regarding the successful receipt of the large file. The fragmentation/reliable delivery service combination can be used to facilitate transfer of large files reliably. Access to these capabilities is available to entities through the use of QoS constraints that can be specified. This facilitates exploiting these capabilities with systems such as GridFTP.

We emphasize here that NaradaBrokering software is a message routing system which provides QoS capabilities to any messages it sends. The NaradaBrokering system may be the messaging layer between many different applications, such as Audio/Video conferences [4]. The QoS features provided by the NaradaBrokering system are independent of the implementation details of the endpoint applications that use it for messaging. Thus applications do not need to implement (for example) reliable messaging.

Furthermore, NaradaBrokering provides capabilities for communicating through a wide variety of firewalls and authenticating proxies while supporting different authenticating-challenge-response schemes such as Basic, Digest and NTLM (a proprietary Microsoft authenticating scheme).

4 Enhancing GridFTP

On the previous papers ([3] [7]) we already described enhancing mechanisms. In this paper we will briefly describe enhancing GridFTP with NaradaBrokering. And we will focus more on how reliable mechanism works in the NaradaBerokering.

GridFTP and other file transfer mechanisms may already incorporate a number of reliability features on there implementation of service and protocol. However, the most important weakness of these architectures is all the great features can not be used outside of its own architecture. This means whenever people want develop new file transfer mechanism and if they want existing features of other mechanisms, they have to re-develop same features within the service implementation. It is our goal to show that these reliability features can be decoupled from the implementation of the service and protocol, and instead placed into the messaging substrate. This will allow us to provide file transfer quality of service comparable to GridFTP in other file transfer tools (such as normal FTP, SCP, HTTP uploads, and similar mechanisms).

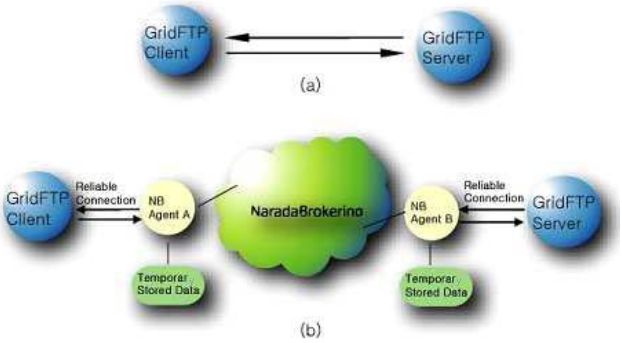


Fig. 1. (a) Traditional GridFTP (b) GridFTP with NaradaBrokering

Figure 1 is present the basic architecture of integration between GridFTP and NaradaBrokering. For initial testing we developed the router approach even though proxy approach is the more preferred method. Main difference of those two approaches is usage of NaradaBrokering Agent A. The router approach will use NaradBrokering Agent A as simple router to transfer requests to the remote server. Key to the proxy approach is the remote GridFTP server is simulated by the NaradaBrokering Agent A. Since NaradaBrokering Agent A is a simple router on the router approach, it is easier than the proxy approach to implement. However, the router approach also has disadvantages like we have to change the user application, even though change is minor and also requires some minor extensions to FTP/GridFTP client codes to communicate with NaradaBrokering Agent A. The client and server communicate solely with the agents on the edge of the broker cloud. For the GridFTP client point of view, NaradaBrokering Agent A is a server and NaradaBrokering Agent B is a client for GridFTP server point of view. The proxy approach is the preferred method since the GridFTP client code and user application do not have to change. All existing GridFTP code and user application can be used in our architecture without any changes once this method is implemented. Disadvantage of this approach is it is harder to implement and time consuming process since we have to create GridFTP server from the scratch.

4.1 Reliable Mechanism in NaradaBrokering

We will describe in depth about how reliable mechanism of NaradaBrokering works. As we mentioned earlier we assumed that any of our architecture nodes could be go down during transfer except GridFTP server. To achieve this idea we are using acknowledgements and database. As we can see from Figure 2, the

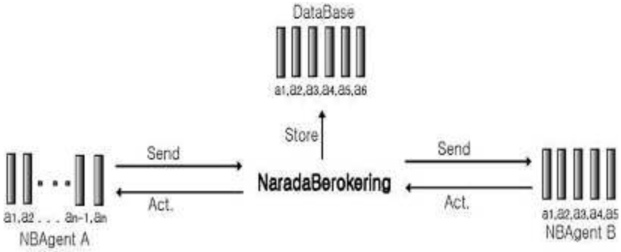


Fig. 2. Reliable Mechanisms in NaradaBrokering

first step is that we divide large file into small pieces $(a_1, a_2, \ldots, a_{n-1}, a_n)$ of same size except last piece that may truncated. Once NaradaBrokering get a piece from NaradaBrokering Agent A, it stores the piece into the database for any failure cases while NaradaBrokering is also sending same file to NaradaBrokering Agent B. An acknowledgment of receiving a piece on the NaradaBrokering from NaradaBrokering Agent A is taking place when NaradaBrokering is finished storing piece into the database. Also, there is an acknowledgment to NaradaBrokering after NaradaBrokering Agent B received and stored a piece into the temporary local directory. Those acknowledgments will be stored in the local file system and will be used when any failures occur during transferring a file. Once failure is fixed NaradaBrokering Agent A, and/or NaradaBrokering is looking for acknowledgment file and figure out the start point of resume transmission. For example, we have a machine failure on NaradaBrokering Agent A during sending a_7 with a_6 on acknowledgment file. After machine is re-started, NaradaBrokering Agent A is looking in the acknowledgment file and fined start point as a_7 since there are receive acknowledgment until a_6. This is goes to same between NaradaBrokering and NaradaBrokering Agent B.

Database on the NaradaBrokering will be used as storage of small pieces of files. In this way we can transfer file from NaradaBrokering Agent A to NaradaBrokering without any guarantee of NaradaBrokering Agent B running and it is true for sending file form NaradaBrokering to NaradaBrokering Agent B. Even NaradaBrokering server itself can be go down. NaradaBrokering server is smart enough to know resuming point to NaradaBrokering Agent B after recovered from failure.

4.2 Multiple Stream Transfer Mechanism in NaradaBrokering

Advancement in network technologies is providing increasing data rates, but current TCP implementation prevents us to use maximum bandwidth across high-performance networks. This problem becomes very clear especially when transferring data happens on a high-speed wide area network. Either increasing the TCP window size by tuning network settings or using multiple TCP streams in parallel can be used to overcome this problem and achieve optimal TCP performance. Since lack of automatic network tuning and tuning network settings is different in each every operating system, it cannot be considered as cross platform solution. Hence, we chose multiple parallel TCP streams to achieve maximum bandwidth usage and we will describe in depth about our implementation in this section.

Our idea of multiple parallel TCP streams consists of splitting data into sub small packets at sender side and sending these sub small packets over the network by using multiple Java socket streams in parallel. Although the default socket buffer size is not set to value of the bandwidth delay product, using multiple parallel TCP streams gives better transfer rate by aggregating each socket bandwidth.

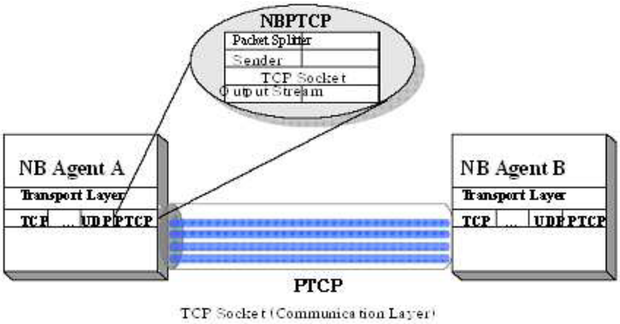


Fig. 3. NaradaBrokering PTCP Architecture

Figure 3 illustrates the architecture of NaradaBrokering Parallel TCP (NBPTCP) transport layer, and NBPTCP usage as communication layer between NaradaBrokering Agent A and NaradaBrokering Agent B. Like all other NaradaBrokering transport protocols, NBPTCP is implemented in the NaradaBrokering's transport layer as multi stream protocol, and it uses our Parallel TCP Socket (PTCPSocket) implementation. PTCPSocket can handle multiple sockets' input and output streams and it is derived from Java.net.Socket. It consists of packet splitter, packet merger, senders, receivers, and TCP sockets, and it has two types of channels; communication and data channels. All control information and negotiations are sent over the communication channel, which stays open till the end of whole data transfer, and data channels are used for actual user data transfer. For example, both sender side and receiver side agree on the number of streams, which will be used during the data transfer by using communication channel. Sender side is responsible for deciding the number of parallel streams before initiating the actual user data transfer.

After the setting parallel streams number, packet splitter starts diving user data into small packets. These packets are passed to senders layer and senders send them to receiver side by writing these packets into TCP sockets' output streams (data channels). The number of senders and receivers are same as the number of parallel streams. At receiver side, receivers read packets from the TCP sockets' input streams (data channels) then pass these packets to upper layer, which is called packet merger. The packet merger combines these incoming packets by checking their packet number, which is given by the packet splitter. Since TCP uses a checksum computed over the whole packet to verify that the protocol header and the data in each received packet have not been corrupted, there is no need to check data integrity at the packet merger layer again.

5 Benchmarks

In this section, we will discuss how well our reliable middleware architecture is performing in the existing services. To increase realities, we are done performance tests between Cardiff University at United Kingdom and Indiana University at United State. We are also using multiple platform environments to show inter–operability of the NaradaBrokering. For example, we are running NaradaBrokering server on the Windows platform and NB Agents on the Linux platform. The experimental setup is described below (see Figure 1 for each parts):

- **GridFTP Client**: Dual Pentium III 1GHz CPU with 1.5 GB of RAM on Red Hat Linux 7.2. Located at Cardiff University.
- **NB Agent A**: Dual Pentium III 1GHz CPU with 1.5 GB of RAM on Red Hat Linux 7.2. Located at Cardiff University.
- **NaradaBrokering Server**: Pentium 4 2.53GHz CPU with 512 MB of RAM on Windows XP Professional Operating System. Located at Indiana University.
- **NB Agent B**: Intel(R) Xeon(TM) CPU 2.40GHz CPU with 2GB of RAM on Red Hat Linux 7.2. Located at Indiana University.
- **GridFTP Sever**: Dual AMD Athlon(tm) MP 1800+ CPU with 513 MB on Red Hat Linux 7.3. Located at Indiana University.

In our performance measurements, we wish to examine the performance penalty represented by adopting the architecture of Figure 1. Again, the routing approach allows us to provide reliability features (such as recovery from network failures) on top of the basic GridFTP file transfer mechanisms. This will create some additional overhead, which we determine below.

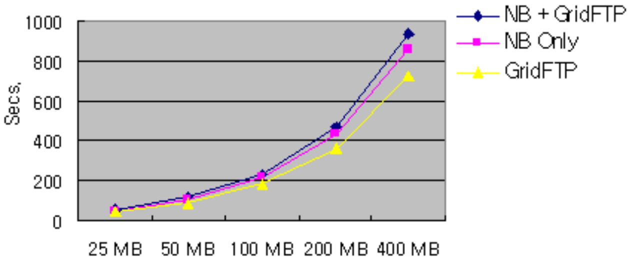


Fig. 4. File Transfer Results with 1 Stream

We will present performance results up to 2 streams since there are virtually no differences beyond 2 streams. This kind of behavior is due to the network setting between Cardiff University at UK and Indiana University at USA, which is beyond our control. Figure 4 shows the performance result of 1 stream of GridFTP, NBGridFTP, and NaradaBrokering. As we can see on this Figure, NBGridFTP is slower by 22.22% (25 MB) to 28.76% (400 MB) range. Those percentages of delays come from inside NaradaBrokering like dividing large file, writing to database, and temporarily copying data on the NaradaBrokering Agent A and NaradaBrokering Agent B. Result of NB only represent the performance result of between NaradaBrokering Agent A and NaradaBrokering Agent B. This means that we remove timing for temporary file store and NaradaBrokering Agent A is worked as GridFTP Client and NaradaBrokering Agent B is worked as NBGridFTP server. This result gives us idea about how well our NaradaBrokering network implemented. As actual network stand point of view it is only about 11.91% to 18.52% slower compared with GridFTP, plus our NaradaBrokering system has reliable mechanisms are there.

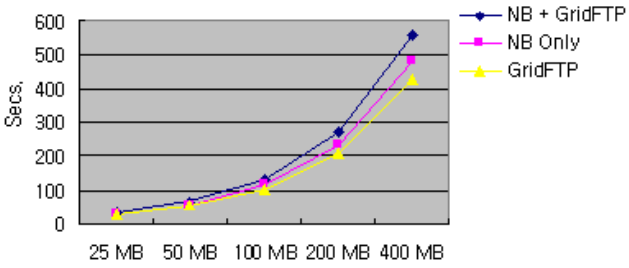


Fig. 5. File Transfer Results with 2 Streams

As we can see on the Figure 5, we also have similar results for 2 streams case. In this case our architecture is slower compared with GridFTP by 25.44% to 30.91% for NB + GridFTP case and about 7.56% to 13.45% for NB only case. We also can see the rate of second dropping from the 1 stream case is very similar to GridFTP-GridFTP dropped 42.36% and NaradaBrokering dropped 44.57%. This means our implementation of multiple streams is as effective as what GridFTP has currently. For the future optimization issues, we will discuss about the matters that delays our architecture in the next section.

5.1 NaradaBrokering Timing

We will look deeply into the time spent in our architecture for further optimization (see Table 1). We divide NaradaBrokering with GridFTP into 2 parts; Timing for internal and external NaradaBrokering time. Internal NaradaBrokering time is divided into initialization, deleting temporary file, writing to database, actual transferring, and merging file. For the external time, we measured file

Table 1. Detailed timing for NaradaBrokering + GridFTP with 2 streams in seconds

MB	Temp. Transfer	Init	Del.	DB	Merging	Network	NB + GridFTP	GridFTP
25	4.82	0.95	0.02	1	0.36	25.52	25.52	26.95
50	9.16	1.80	0.05	2	0.72	52.24	52.24	54.18
100	17.54	3.88	0.11	4	1.66	106.05	106.05	103.93
200	36.42	17.28	0.22	8	3.15	206.63	206.63	208.66
400	74.20	41.04	0.43	16	5.97	418.56	418.56	424.85

transfer between GridFTP client to NaradaBrokering Agent A and between NaradaBrokering Agent B to GridFTP server. A large file will be divided into small pieces of fixed size and will be stored into temporary directory in the Initialization phase and after done transfer, timing for the cleanup those temporary files are measured on the Delete phase. Those small pieces of a file will be stored into the database that located on the NaradaBrokering server first. This time is estimated timing based on the experimental benchmark. Actual file transferring time is measured on the Network phase. After NaradaBrokering Agent B gets all the small pieces of file it will reconstruct original file using those pieces. As we can see for this table, most of the time is either negligible (delete, database, and merging) or non-avoidable (temporary file transfer). And also actual timing for the transferring file is reasonable. According last two measurements of Table 1, actual file transfer rates are as good as GridFTP file transfer rates. GridFTP is little bit slower because we did not separate authentication from the actual file transfer.

One part we believe we can optimize is initialization. Table 1 shows that it is not taking much time if dealing with small file size. However it takes more then necessary when dealing with larger file size. Initialization phases will be deeply investigated for the future optimization.

6 Conclusions

We discussed reliable transfer mechanism in NaradaBrokering using GridFTP as an example. NaradaBrokering system is an event brokering system designed to run on a large network of cooperating broker nodes and is used here as a general purpose messaging substrate. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. . Decoupling desirable features of existing systems like file recovery technologies in GridFTP from the implementation of the service and instead placing into the reliable, high performance messaging substrate between the client and service will allow us to extend to other services without extensive reimplementation.

We also discussed deploying NaradaBrokering in GridFTP and its performance tests. As we can see from the performance tests we have reasonable file transfer rates with added features like reliable transfer and multiple stream file transfer. We show the possibilities of our goal of decoupling reliability features from the implementation of the service and protocol, and instead placed into the software messaging substrate without great lose of performances.

For future work, the brokering system is by design a many-to-many messaging system, so we may exploit this to support simultaneous delivery of files to multiple endpoints. Finally, we will develop more examples of using other file transfer mechanisms that will mimic RFT-like features without reimplementation

References

1. Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak and Ian Foster, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, and Steven Tuecke. Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal*, 28(5):749–771, May 2002.
2. W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. Gridftp: Protocolextensions to ftp for the grid. Technical Report JGF-TR-03, Argonne National Laboratory, April 2002.
3. Geoffrey Fox, Sang Boem Lim, Shrideep Pallickara, and Marlon Pierce. Message-based cellular peer-to-peer grids: foundations for secure federation and autonomic services. *Future Generation Computer Systems*, 21(3):401–415, 2005.
4. Geoffrey C. Fox, Wenjun Wu, Ahmet Uyar, and Hasan Bulut. Design and implementation of audio/video collaboration system based on publish/subscribe event middleware. In *CTS04*, January 2004.
5. Rfc 765 - file transfer protocol specification. http://www.faqs.org/rfcs/rfc765.html.
6. Gridftp: Universal data transfer for the grid. http://www.globus.org/datagrid/gridftp.html.
7. Sang Boem Lim, Geoffrey Fox, Shrideep Pallickara, and Marlon Pierce. Web service robust gridftp. In *The 2004 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA04)*, June 2004.
8. Ravi K Madduri. Reliable file transfer in grid environments. In *the 27th Annual IEEE Conference on Local Computer Networks (LCN'02)*, November 2002.
9. The naradabrokering system. http://www.naradabrokering.org.
10. Shrideep Pallickara and Geoffrey Fox. Naradabrokering: A middleware framework and architecture for enabling durable peer-to-peer grids. In *ACM/IFIP/USENIX International Middleware Conference Middleware-2003*, June 2003.
11. Reliable file transfer service. http://www-unix.mcs.anl.gov/ madduri/RFT.html.

2-Layered Metadata Service Model in Grid Environment*

Muzhou Xiong, Hai Jin, and Song Wu

Cluster and Grid Computing Lab,
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract. Data intensive task is becoming one of the most important applications in grid environment. The scale of data sets has been hundreds of terabytes and soon will be petabytes. The primary problem we face is how to organize the geographical distributed storage devices to support the collaborative operations on data in those resources. On the one hand, performance is critical to such application, but on the other hand the diverse network conditions prevent users from getting the same service quality. This paper focuses on how to resolve the above problem, and presents a 2-layered metadata service model in grid environment which utilizes the special locality of users' distribution and provides a platform for grid data management. We have implemented that 2-layered metadata service model in *ChinaGrid Supporting Platform* (CGSP) - the grid middleware for ChinaGrid project.

1 Introduction

Advances in science are made possible largely through the collaborative efforts of many researchers in a particular domain. We see collaborations of hundreds of scientists in areas such as gravitational-wave physics [1], high-energy physics [2], astronomy [3] and many others coming together and sharing a variety of resources within collaboration in pursuit of common goals. These resources are geographically distributed and can encompass people, scientific instruments, computer and network resources, applications and data. It is common to see datasets on the order of terabytes today, and soon be petabyte-scale. Grid technologies [4, 12, 13] enable efficient resource sharing in collaborative distributed environments. In this paper, we focus on the area of data management of the grid, with a particular emphasis on metadata management so that data can be well placed to provide high access efficiency.

One challenge in these shared environments is how to put data sets to proper storage resource so that the relevant researchers can fetch them with high efficiency. Usually data are transparently stored in system and users need not care how to select storage resources. Also the data must be easy to be discovered and accessed. But because of the complexity of grid components and their connectivity, geographical distributed users can hardly get the same performance from the grid environment. Perhaps a set of users gets much high transfer speed but the others can get very poor performance. Such conditions will not be tolerable especially when the scale of required data sets getting much larger.

* This paper is supported by National Science Foundation of China under grant 60125208 and 90412010, and ChinaGrid project of Ministry of Education of China.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 103 – 111, 2005.

For example, thousands of scientists from worldwide collaboratively work on the same project in grid environment and data are shared among them. They often get and store data from and into the grid. All of them want to get high transfer speed to improve the efficiency. However, the difference of logical distance between the scientists and the storage resources with the data makes the different performance of data transfer. How to build a common data management platform in grid environment so that all the users interested in the same domain will get alike performance is a big barrier in the development of grid technology.

Traditional replica mechanism is one solution to eliminate the performance gap among users. But with the data scale getting larger and the frequency of operations on the data becoming higher, the maintenance of data consistency will be a very complex work, which constrains users waiting for the data updating or working in degrading model in a long time. An alternative solution of such problem may be an intelligent data scheduling algorithm, which can store data in proper location to satisfy users' access requirement as much as possible according to users' access pattern. When the pattern changes, data will be migrated to another location to adapt to the new condition. This method contains two main shortcomings. One is that it is hard to find a common algorithm to satisfy all the requirements of application. The distribution of storage resources and users may cause different data scheduling mechanism, and the cost of scheduling is very expensive. The other is that when the access pattern changes frequently, data will be migrated in the grid environment very frequent. This will cause the whole system unstable and make the probability of data error higher.

This paper addresses the issue on how to build a metadata management model in grid environment to get high performance. In our model, we divide the traditional metadata manager in grid environment into two parts. One is responsible for building a global namespace for users in a special domain, and the other is responsible for collecting a set of storage resources to store data in that domain. We have implement this data management model in ChinaGrid project [10] as a part of *ChinaGrid support platform* (CGSP) [10]. Through that users in the same domain can get almost the same transfer speed.

This paper is organized as follows. We discuss the related work in section 2. In section 3, we give the overview of the data management in CGSP, and the two-layered metadata service in CGSP in section 4. Section 5 is the use case study of the model. We perform the performance study in section 6, and conclude this paper in section 7.

2 Related Works

Storage Resource Broker (SRB) [5] and its associated *Metadata Catalog* [6] provide metadata and data management services. SRB supports a logical name space that is independent with physical name space. The logical objects, logical files in SRB can also be aggregated into collections. SRB provides various authentication mechanisms to access metadata and data within SRB.

Replica Metadata (RepMec) [14] catalog is built upon the Spitfire database service. The RepMec Catalog stores logics and physical metadata. It is used within the EDG project to map user-provided logical names of data items to unique identifiers called GUIDs. RepMec is used in the Reptor system in cooperation with a replica location service.

Internet Backplane Protocol (IBP) [7] is a middleware for managing and using remote storage. It supports logistical networking in large scale, distributed systems and applications. It defines logistical networking as the global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources. IBP provides a mechanism for using distributed storage for logistical purposes and also provides strategies of data depots and repositories, and replica management.

Majority of data grid projects, such as Particle Physics Data Grid [15] and LHC Computing Grid Project [16], are focusing on designing higher-level services on top of the basic Globus infrastructure. Some of the services being developed are: replica management, which combines replica catalog with file transfer; replica selection, which chooses the "best" replica with respect to network and storage performance; and broker services, which seek out available resources to schedule jobs.

3 Overview of Data Manager in CGSP

ChinaGrid [10] integrates all the resources among participant universities in China, and makes users and heterogeneous grid resources work cooperatively. It provides transparent grid services with high performance, high reliability for all kinds of science computing and research. *ChinaGrid Support Platform* (CGSP) [10] is the core middleware for ChinaGrid, which also provides development environment for grid application.

CGSP integrates all kinds of resources in education and research environments, makes the heterogeneous and dynamic nature of resource transparent to the users, and provides high performance, high reliable, secure, convenient and transparent grid service for the scientific computing and engineering research. CGSP provides both ChinaGrid service portal, and a set of development environment for deploying various grid applications.

The current version, CGSP 1.0, is based on the core of Globus Toolkit 3.9.1, and is WSRF [8] and OGSA [9] compatible. CGSP contains five building blocks: grid portal, grid development toolkits, information service, grid management and grid security. Grid management has four parts: service container, data manager, job manager, and domain manager [10].

Data management is the core service in CGSP. It manages heterogeneous storage resources and data in grid environment. When a user sends a request to data manager for storing a set of data, data manager selects a proper storage resource to receive the data set and in the meaning while, it also recodes the relevant metadata. When a user wants to fetch a set of data, data manager will find the relevant metadata according to the data identifier and return to the user with the best replica. Data logical domain manager and data domain manager are in charge of metadata maintenance. Different from the traditional metadata management mechanism, it uses the 2-layered metadata management model to record the relevant metadata. We will discuss this model in detail in next section. In section 5 we also give a use case study to descript how this model works in grid environment.

4 2-Layered Metadata Services

4.1 Requirements of Metadata Services in CGSP

Basically, metadata manager records a variety of information of the related data. Some information is application dependent, such as the creation time, author, described in Dublin Core [6]. It also provides data location service. For example, if a user wants to fetch a data file with a given logical file name in his data namespace, the metadata service finds the corresponding metadata associated with the file name and provides the URL of the storage location storing the data. The metadata service records *Access Control List* (ACL) for every data file. Data are shared among different users and the author does not want to let all users have the same access right to the data. Receiving a data access request, the metadata service first checks the ACL to tell whether the user has the right to do such operation.

Metadata service must also have a proper data scheduling mechanism to confirm all the transfer speed to users. In grid environment, it is hard to find such an algorithm to satisfy all the users' requirement. We need to change to traditional metadata organization to satisfy such requirement. We design 2-layered metadata service model to guarantee the service quality of metadata management.

4.2 2-Layered Metadata Service Model

This design is adaptive to the applications in grid environment, especially to the data-intensive applications. In these applications, data are stored in storage resources and shared by users, which are all geographically distributed. Collaborative work makes the shared data becoming the key component among the users. But all applications have space locality, that is to say users and data usually distributed in some special sites. Using this locality, we provide the 2-layered metadata service model which can largely improve the data transfer speed. The design of 2-layered metadata service model focuses on how to resolve the data transfer speed in grid environment. Such solution does not make emphasis on data transfer protocol, but on the organization of data and storage resources. In such a model, system creates a data logical domain for every application containing a set of users. In order to confirm data near to users, data logical domain also contains a set of storage resources logically near to the users and data are always stored in them. Through that, proper data are confirmed to be stored on the proper storage resources, making high effective data transfer speed between users and resources.

The 2-layered metadata service model in CGSP has two parts: the lower part organizes distributed storage resources into a manageable storage pool and also provides data location service, which is called *Data Domain* (DD); the upper part with the name of *Data Logical Domain* (DLD) connects special applications and a set storage resources, also it maintains the domain independent metadata. DD takes care of the management of storage resources with the dynamic state of resources, which supports resource selection strategy. DLD does not manage any storage resource, but it contains a list of resources for a special data logical domain. All data in the DLD are stored in those resources, which are selected to satisfy almost all users' transfer speed requirement.

4.3 Data Domain

The tremendous storage resources are organized as where they are, that is, all storage resources in that region are collected together and managed by DD. Every storage resource sends its own state information such as CPU load, storage capacity, to DD periodically. Through such information, DD knows status of each resource so that upper component can decide where to store data when receiving data storing request. If a resource does not send its information to DD in a certain internal, it will be considered as out-of-work and the next task will not refer to this resource. Through this mechanism, DD dynamically knows which resource adapts to the type of task. For example, if a task is a data back-up, usually it will focus on the capacity of the resource but not on CPU processing capability.

Replica catalog is another function of DD. The main task of replica catalog is to provide data location service and maintain the integrity of data. It implements mapping from data uniform identifier to URLs where user can fetch data. Because a data can have multiple replicas, replica catalog must also choose the best one. Through this mechanism, users can transparently access data, without knowing where to store or fetch data and what type of data transfer protocol is. When updating a replica, replica catalog will maintain the consistent among the replicas.

4.4 Data Logical Domain

DLD is the main feature in data management of CGSP, through which performance consistency can be achieved. DLD is a triple-tuple: <*application, user, resource*>, which describes an application stores data on which storage resources and which users work together for the application. The element *application* indicates the task to which DLD provides storage service. A given *application* usually has fixed users working in some special places, and also *application* decides the type of resource for it. The element *user* is a collection of user IDs with an administrator in it. When a DLD is created, *user* has only one element: administrator. The administrator takes care of user management. The user joining the DLD is added into the *user* collection and has the right to access data in DLD. The administrator manages storage resources. It selects storage resources from resource list, and the selected resource is the last element of DLD: *resource*. Because of the locality of user's geographical distribution, administrator can choose resources logically for almost every user, but not with tremendous users with disordered distribution. Through this the data storage quality of service is confirmed.

DLD builds a uniform namespace for each user. When launching a DLD, user sees a tree structure directory. A group of data operations are supported, such as data upload, download. For example, a user can upload a data file with a given file name in any directory of the namespace. Insides the system the relevant metadata information including the logical file name (the complete path in the namespace) is recorded. A global uniform ID is corresponding to the logical file name. In DD the ID is mapped to a set of URLs. The two translations make data operations transparent, and user does not need to know the location of data. Every application has its own data shared by all the collaborative users, and DLD provides a mechanism to publish the data to all. DLD records metadata of all shared data and adds them to every user's namespace. In user's namespace, a fixed directory called *common* in a DLD contains all the shared data. Any user can operate these data under certain access control rule.

A user has a namespace even if he is not belonged to a DLD. Each user is in a default data logical domain. Default data domain is created during system initialization, and contains all users registered in the system. The difference between a DLD and default DLD is that default DLD contains all the resources of the DD, that is, data are distributed in all the resources, which will decrease the transfer performance without considering the locality. Default DLD is convenient to users when they work alone, but the performance seems worse than that of DLD (seen in the performance study section). Metadata in DLD and default DLD together compose a uniform namespace for a user.

In some circumstances, no storage resource can satisfy users for the performance requirement. DLD will duplicate data for them. DLD chooses the most frequently used data replica near to those users, and makes consistent between the replica and original data. Through this, all users in DLD will get relative high performance.

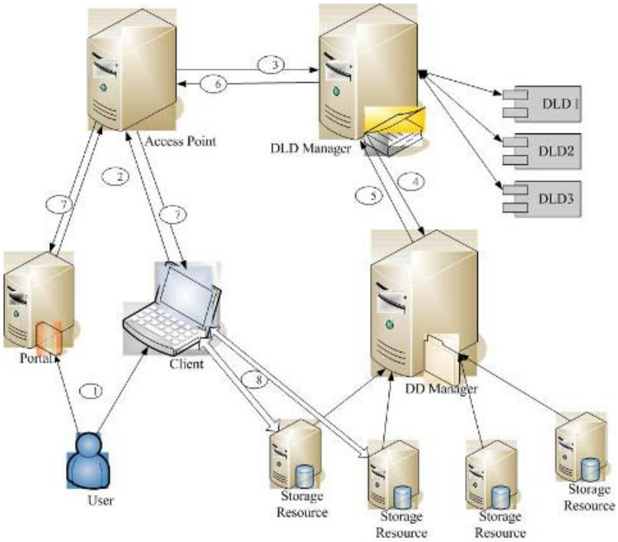


Fig. 1. Workflow of 2-layered metadata services in CGSP

5 Use Case Study

When a user uses data management in CGSP, he may first login from the portal or the data management client. Through the client or portal, the user sends request to operate data. The request which contains the data path in the user's namespace is sent to the Access Point and the Access Point forwards it to the corresponding DLD manager. In DLD manager the request is resolved to the following format: <*request type, DLD name, data path*>. If the request is a query operation, DLD manager will search the metadata in the corresponding DLD according to the DLD name and data path, which will be returned to Access Point and sent to user. Fig. 1 illustrates a fetching operation workflow. Like the query operation, DLD manager finds the uniform identifier (UID) and sends it to DD manager which will get URLs of data from replica catalog.

When a portal or a client gets the URLs, it can get data from the relevant storage resource directly. When creating a new file, DLD manager will create a new UID for the file, record metadata of the file and choose a proper resource to store the file from the DLD resource list. UID is sent to DD manager. Also the URL reserved in replica catalog is created according to the selected storage resource. Data will be uploaded from portal or client to the site indicated by URL.

6 Performance Evaluation

The data management of CGSP is run on a cluster with 16 nodes and each node is configured with 1GHz Xeon CPU with 512MB memory and 40GB disk, with Red Hat 7.3 kernel version 2.4.9 as its operating system. All nodes are connected by 100Mb Ethernet. All the data management modules including DD and DLD are installed in the cluster. Clients are distributed in the wide area network.

With this study we aim to address two issues: 1) the performance of metadata service model; and 2) the performance of data transfer speed in DLD and default DLD. The key measurement of metadata services is the operation rate to metadata. The operation contains two aspects: metadata read which is used to query or list data and metadata write when creating new data file. We also compare the performance between data in DLD and default DLD.

We used 1 to 12 concurrent threads to perform the read/write operation to metadata. The result is showing in Fig. 2. From the result we can see that metadata read rate is about 15% higher than that of write. That is because we use OpenLDAP to optimize read operation to store metadata information.

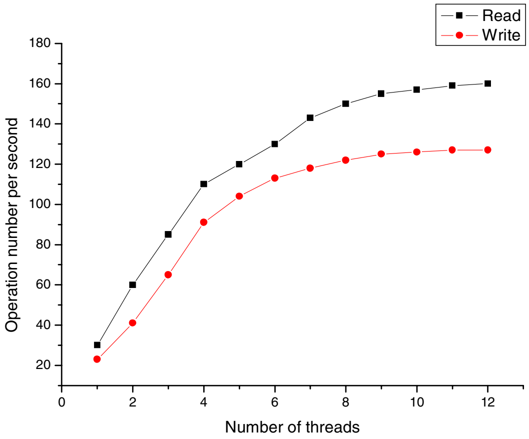


Fig. 2. Performance of metadata operation (read, write)

For data transfer test, we fetch data in DLD and default DLD with the size range from 50 to 1000 MB, respectively. We use GridFTP [12] as the data transfer protocol. From Fig. 3, we find that the performance of data transfer speed in DLD is averagely 60% higher than that in default DLD. The data transfer speed is much steady because users are distributed in relative small scale sites and the storage resources are logically near them. For data in default DLD, data transfer speed may be high, but for the most circumstances, we just get poor performance. The performance difference shows that using geographical distribution locality, we can get much better result.

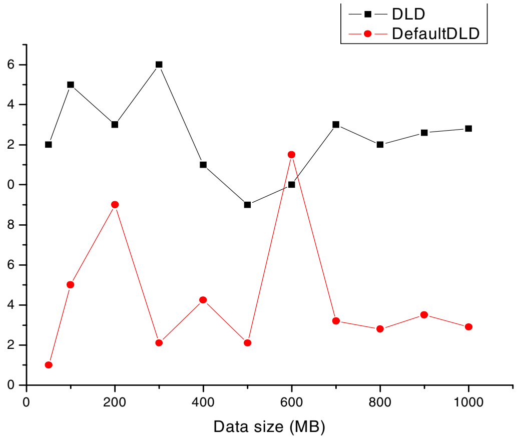


Fig. 3. Performance of data transfer speed

7 Conclusion and Future Work

In this paper we present the 2-layered metadata service model in grid environment. The model is based on the locality distribution of users. With the separation of traditional metadata service, we manage metadata in DLD and DD. DLD manages a set of resources and a user's namespace and in DD it maintains replica catalog. Without complex resources scheduling algorithm, we can get a good data transfer speed. The model is implemented in CGSP 1.0 of ChinaGrid project.

Replica mechanism is another important aspect in the model. We have implemented some simple algorithms for replica creating, resource selection and replica shifting, but they are not enough for the complex conditions in grid environment. In the future we plan to design efficient replica algorithms to satisfy users' distribution in larger area.

References

1. B. C. Barish and R. Weiss, "LIGO and the Detection of Gravitational Waves", *Physics Today*, Vol.52, p.44, 1999.
2. C.-E. Wulz, "CMS – Concept and Physics Potential," *Proceedings II-SILAFAE*, San Juan, Puerto Rico, 1998.
3. NVO, 2004. http://www.us-vo.org/.
4. I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, vol.15, 2001.

5. C. Baru, R. Moore, A. Rajasekar, and M. Wan, “The SDSC Storage Resource Broker”, *Proceedings of CASCON’98 Conference*, 1998.
6. MCAT, *MCAT – A Meta Information Catalog* (Version 3.0), http://www.npaci.edu/DICE/SRB/mcat.html.
7. M. Beck, Y. Ding, T. Moore, and J. S. Plank, “Transnet Architecture and Logistical Networking for Distributed Storage”, *Proceedings of Workshop on Scalable File Systems and Storage Technologies*, San Francisco, CA, Sept. 2004.
8. The Web Services Resource Framework, http://www.globus.org/wsrf/.
9. Open Grid Service Architecture, http://www.ggf.org/Public_Comment_Docs/Documents/draft-ggf-ogsa-specv1.pdf.
10. H. Jin, “ChinaGrid: Making Grid Computing a Reality”, *Digital Libraries: International Collaboration and Cross-Fertilization - Lecture Notes in Computer Science*, Vol.3334, Springer-Verlag, December 2004, pp.13-24.
11. GridFTP: Universal Data Transfer for the Grid, http://www.globus.org/datagrid/gridftp.html/.
12. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke, “The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Data Sets”, *J. Network and Computer Applications*, pp.187-200, 2001.
13. A. Rajasekar and A. Jagatheesan, “Data grid management systems”, *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, April 2003.
14. L. Guy, P. Kunszt, E. Laure, H. Stockinger, and K.Stockinger, *Replica Management in Data Grids*, Global Grid Forum 5, 2002.
15. PPDG Project: http://www.ppdg.net/.
16. LHC Computing Grid Project: http://lcg.web.cern.ch/LCG/.

pKSS: An Efficient Keyword Search System in DHT Peer-to-Peer Network*

Yin Li, Fanyuan Ma, and Liang Zhang

The Department of Computer Science and Engineering,
Shanghai Jiaotong University, Shanghai, China, 200030
{liyin, fy-ma, zhangliang}@cs.sjtu.edu.cn

Abstract. The state-of-the-art keyword search system for structured P2P systems is built on the distributed inverted index. However, Distributed inverted index by keywords may incur significant bandwidth for executing more complicated search queries such as multiple-attribute queries. In order to reduce query overhead, KSS (Keyword Set Search) by Gnawali partitions the index by a set of keywords. However, a KSS index is considerably larger than a standard inverted index, since there are much more word sets than individual words. And the insert overhead and storage overhead are obviously unacceptable for full-text search on a collection of documents. In this paper, we presents pKSS, a P2P keyword search system that that adopts term ranking approach such as TFIDF and exploits the relationship information between query keywords to improve performance of P2P keyword search. Experimental results clearly demonstrated that the improved keyword search is more efficient than KSS index in insert overhead and storage overhead, and much less than standard inverted index on bandwidth costs for a query.

1 Introduction

In recent years, P2P has emerged as a popular way to share huge volumes of data. The key to the usability of a data-sharing P2P system, and one of the most challenging design aspects, is efficient techniques for search and retrieval of data.

Structured P2P systems, such as Chord [1], Pastry [2], don't support full text search directly. While, as they actually implement distributed hash tables (DHTs) over them, keyword search can easily be implemented by distributing inverted indices among hosts by keyword. Then a query with k keywords can be answered by at most k hosts through the intersection of inverted lists. However, Distributed inverted index by keywords may incur significant bandwidth for executing more complicated search queries such as multiple-attribute queries. This is unacceptably large bandwidth for query in a P2P system because bandwidth available to most nodes in the Internet is rather small.

In order to reduce query overhead, KSS (Keyword set Search)[3] partitions the index by a set of keywords. A KSS index is considerably larger than a standard inverted index, since there are more word sets than there are individual words. And

* Supported by The Science & Technology Committee of Shanghai Municipality Key Project Grant 03dz15027 and 03dz15028.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 112 – 117, 2005.

insert overhead for KSS grows exponentially with the number of the keywords while query overhead is reduced because no intermediate lists are transferred across the network for the join operation. However, the insert overhead and storage overhead of KSS are obviously unacceptable for full-text search on a collection of documents even if KSS makes use of the distance window technology.

Our work aims to design an efficient P2P keyword search system which has the same order of insert and storage overhead as standard distributed inverted index while has the same (or better) performance of keyword search as (or than) KSS. pKSS (P2P keyword search system), presented in this paper, adopts term ranking approach such as TFIDF and exploits the relationship information between query keywords to improve performance of P2P keyword search. In pKSS, instead of publishing keyword pairs as done in KSS, we only publish individual important keywords and associate each publishing keyword with a set of related keywords, and the insert and storage overhead can be greatly reduced when compared with KSS. When doing keyword search, related query keywords are grouped into sets, and search can be done in keyword set like KSS. Therefore the performance of keyword search can also be greatly improved when compared with standard distributed inverted index.

2 Keyword Relationship Discovering

To speed up the keyword search in pKSS, each publishing term is associated with a set of keywords that are usually queried by the user together. To determine the keyword relationship, we take a query log which can be obtained from WWW or FTP search sites as input and map it into a graph which expresses the relationship between keywords. The algorithm has three basic steps:

Step 1 Construct a directed graph *G(A,E)* according to the query log

Step 2 Pruning the graph to $G(A,E)|_{\theta}$ according to a given connectivity threshold θ

Step 3 Output E_{θ}

We discuss each step in turn.

Step1. Construct a directed graph *G(A,E)* according to the query log.
The set of vertices *A in* graph *G(A,E)* corresponds to the search terms used in the user queries. The set of edges *E* corresponds to search terms co-occurrence as observed in the user queries.

E={e|weight(e)>0}. Since the graph *G(A,E)* is a directed graph, $E_{A1->A2}$ and $E_{A2->A1}$ should be distinguished form each other. The weight of a directed edge is defined as follows:

$$\text{weight}(E_{A1->A2}) = \frac{freq(A1 \cap A2)}{freq(A1)} \tag{1}$$

Where A1 and A2 are vertices in set A. The *freq(X)* represents the frequency that search term *X* occurs in users' query. For instance, if a query procedure contains the search terms "p2p" and "search" the frequency of the relevant vertices is added one respectively. The weights on the directed edge (p2p →search) are computed as the normalized frequencies by dividing them with the occurrence frequencies of the "p2p" vertices .The effect of the normalization is to remove the bias for characteristics that appear very often in all users.

Step2. Pruning the graph to *G(A,E)*|$_\theta$ according to a given connectivity threshold θ.
As the connectivity of the resulting graph *G* is usually high, we use a *connectivity threshold*, aiming at reducing the number of edges in the graph. The connectivity threshold represents the minimum weight allowed for the edge's existence. When this threshold is high the graph will be sparse and when the threshold is lower the graph will be dense.

In graph $G(A,E)|_\theta$, the set of vertices *A in* graph $G(A,E)|_\theta$ is same to the set of vertices *A in* graph *G(A,E)* , which corresponds to the search terms used in the user queries. However, the set of edges E_θ corresponds to search terms co-occurrence as observed in the user queries. E_θ can be obtained from following equation.

$$E_\theta = \{e \mid e \in E, weight(e) \geq \theta\} \quad (2)$$

It is obvious that different *connectivity threshold* θ may output different E_θ. The larger *connectivity threshold* θ is, the sparser the graph is. In pKSS, we choose θ*=0.05.*

Step3. Output E_θ.
This step outputs the vertex pairs corresponding to E_θ, which expresses the co-occurrence relationship of query keywords.

3 Keyword Publishing and Search in pKSS

In pKSS, we make two optimizations for keyword search in structured P2P network. Firstly, by adopting TFIDF technique, only important terms that best describe the document are selected as publishing keywords. This can reduce the costs for publishing inverted list of the documents at cost of ignoring some less important documents that may be queried by unimportant keywords which have low TFIDF weight. Secondly, pKSS exploits keyword relationship, the inverted list intersection cost for multiple-attribute queries can be greatly reduced.

When a user shares a document, pKSS first builds the inverted list of the document with each term associated with a weight. The term weight is computed by TFIDF approach. Then the most significant term list is selected as the publishing keywords. Of cause, we can publish every keyword in the inverted list as is done in traditional structured P2P keyword search system[4][5][6]. However, in pKSS, only the first *L* largest weighted terms are published, and if the number of terms in the inverted list are less than *L*, all the terms will be published. Therefore, if *L* is large enough, for example larger than the size of lexicon, all the keywords in inverted list will be published. In pKSS, we let *L*=500 in favor of reducing the insert overhead at the cost of ignoring some less important documents that may be queried by unimportant keywords which have low TFIDF weight.

The index entry to publish in pKSS contains three parts: the keyword itself, the document ID, and a set of keywords that are in the document and related to the publishing keyword. The keyword set in the index entry can be expressed as follows:

$$KS_i = \{K_j \mid (K_i, K_j) \in E_\theta, K_j \in D\} \tag{3}$$

where K_i is the publishing keyword in index entry, D is the document, K_j is the related keyword, KS_i is the related keyword set, and E_θ is the keyword relationship graph introduced in section 2. In pKSS, instead of storing related keyword set in the index entry, we use Bloom Filter to compress the keyword set and store the bloom filter in the index entry. $BF(KS_i)$ represents the bloom filter of the related keyword set KS_i. Thus the index entry of keyword K_i can be represented as follows:

$$IE_i =< K_i, DocID, BF(KS_i) > \tag{4}$$

where IE_i is the index entry of keyword K_i, *DocID* is the document ID.

To publish the index entry to the P2P network, pKSS first compute the hash of the keyword as key, then maps the key to the node in the network using Chord algorithm, and stores the index entry to that node at last. The algorithm of publishing keyword in pKSS works as follows:

Query in pKSS is consisted of a set of keywords. Thus, the query can be expressed as follows:

$$Q = \{k_1, k_2, ..., k_n\} \tag{5}$$

where Q represents the query, $k_i(i=1,2,...n)$ is keyword in the query. To speed up the keyword search process, keywords in the query are grouped into sets of related keywords. Each group has a primary keyword which will be used to accomplish the keyword lookup process by Chord algorithm, while other keywords in the group set are the related keywords of the primary keyword that are used to filter the documents. The grouping method lies in two key points. The first is how to determine the primary keyword of the group set, and the second is how to select the related keywords of the primary keyword. In pKSS, the resolving power of the term is used to determine the primary keyword, and the keyword relationship graph of the query log is used to select related keywords of the primary keyword. The grouping process can be divided by the following steps:

Step1 In the query keyword set Q, select the term k that has maximum IDF value as the primary keyword, create a group set G_k, and remove this term from Q.

Step2 Find all the related keywords of primary keyword k, add them to the group set G_k, and remove these keywords from the query set Q. The related keywords can be selected by the following equation.

$$G_k = \{k_i \mid (k, k_i) \in E\,|_\theta, k_i \in Q\} \tag{6}$$

Step3 If $Q \neq \Phi$, goto step1 to create another group set.

Unlike standard distributed inverted index approach, pKSS performs distributed search based on each group set, not the term only. For each group set G_k, pKSS maps the primary keyword k onto the node in the network by Chord algorithm, then fetches all the document index entries and filters the satisfied documents according to the *BF(KS$_k$)* field. The filtering condition is defined as following equation.

$$BF(G_k) \wedge BF(KS_k) = BF(G_k) \tag{7}$$

where *BF(G$_k$)* is the bloom filter of group set G_k. The filtering condition in equation (7) is in fact to test that every the keyword in set G_k appears in set KS_k. Finally, the intersection of documents fetched according to each group set are the final results that satisfy the query. Thus, compared to the standard inverted list intersection approach, the performance of keyword search in pKSS can be greatly improved by query keywords grouping.

4 Experiments

In this section, we evaluate pKSS by simulation. In order to find the relationship between query keywords, we used the query logs of the FTP search website bingle.pku.edu.cn from Dec 1, 2002 to Dec 31, 2002.

We simulated inserting and querying of a document using pKSS. Next we ran the pKSS algorithm on each text file to create index entries and published them to corresponding virtual peers. We evaluated these algorithms by insert overhead and query overhead.

Insert Overhead is the number of bytes transmitted when a document is inserted in the system. When a user asks the pKSS system to share a file, the system generates index entries which are inserted in the distributed index. Unlike KSS, in which if we generate index entries for a document with *n* keywords for typical keyword-pair scheme the overhead required is bounded by *C(n,2)*, pKSS only generates small index entries which results in a small insert overhead.

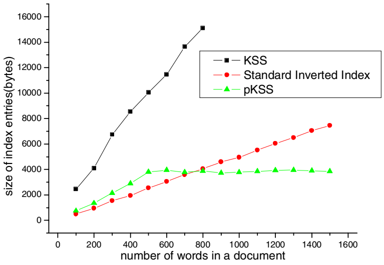


Fig. 1. Insert overhead

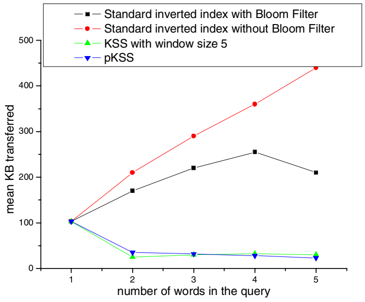


Fig. 2. Query overhead

Fig.1 gives the curves of size of index entries generated vs. number of words in a document using the standard inverted indexing scheme, pKSS with θ=0.05 and KSS with window size of ten. Fig.1 shows that the insert overhead for pKSS is much lower than that for KSS, is a little higher than that of the standard inverted index scheme when the document is small, and is lower than that of the standard inverted index scheme when the document is large.

Query overhead is a measure of bytes transmitted when a user searches for a file in the system. As we know, the overhead to send the intermediate result list in the system from one host to another is the main part of the query overhead.

Fig.2 gives mean data transferred in KB when search using the standard inverted index with Bloom Filter, the standard inverted index without Bloom Filter, KSS with window size of 5, pKSS with θ=0.05, for a range of query words. Fig.2 shows that the query overhead for pKSS is much lower than that of the standard inverted index scheme, with or without bloom filter, and is a little lower than that for KSS when the number of keywords is greater than 3.

5 Conclusions

In this work, we adopt keyword ranking approach such as TFIDF and exploit the relationship between query keywords which can be extracted from users' queries logs, to improve the performance of P2P keyword search system. Experiments results clearly demonstrated that pKSS index is more efficient than KSS index in insert overhead and storage overhead, and more efficient than a standard inverted index in terms of communication costs for query. In a forthcoming paper, the authors will show how the parameter L and θ in pKSS impact the insert and query overhead and the query accuracy.

References

1. I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. Proc. ACM SIGCOMM 2001, Aug. 2001.
2. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. Proc. Middleware 2001, Nov. 2001.
3. Omprakash, D. Gnawali, 2002. A Keyword-set Search System for Peer-to-Peer Networks. MIT's thesis Lib.
4. Reynolds, P., Vahdat, A. Efficient peer-to-peer keyword searching. Technical Report 2002, Duke University, CS Department, Feb. 2002.
5. Jinyang Li, Boon Thau Loo, etc. On the Feasibility of Peer-to-Peer Web Indexing and Search. IPTPS 2003.
6. Shuming Shi, Guangwen Yang, Dingxing Wang, Jin Yu, Shaogang Qu, Ming Chen, Making Peer-to-Peer Keyword Searching Feasible Using Multi-level Partitioning, IPTPS 2004.

A Comparative Study at the Logical Level of Centralised and Distributed Recovery in Clusters

Andrew Maloney and Andrzej Goscinski

School of Information Technology,
Deakin University,
Geelong, Vic 3217, Australia
{asmalone, ang}@deakin.edu.au

Abstract. Cluster systems are becoming more prevalent in today's computer society and users are beginning to request that these systems be reliable. Currently, most clusters have been designed to provide high performance at the cost of providing little to no reliability. To combat this, this report looks at how a recovery facility, based on either a centralised or distributed approach could be implemented into a cluster that is supported by a checkpointing facility. This recovery facility can then recover failed user processes by using checkpoints of the processes that have been taken during failure free execution.

1 Introduction

The advancements in computing are more often than not focused on improving performance rather than improving reliability. Reliability is often neglected because in most cases when reliability is improved, system performance decreases. But it is becoming an increasingly desired feature, especially in such systems as non-dedicated clusters. These systems provide an ideal platform for parallel processing, given their high scalability, availability and low cost to performance ratio [1]. However, given that non-dedicated clusters are composed of a collection of individual computers connected via a network and used by multiple users, reliability is somewhat lacking. Thus, many hours of execution of a parallel application could be lost.

Methods such as *redundancy*, *checkpointing and rollback recovery*, *failure semantics*, and *group failure masking* have been developed to improve reliability, but these methods have not been widely researched in non-dedicated cluster operating systems [2]. It has been shown that checkpointing and rollback recovery is an ideal reliability method for non-dedicated clusters as it provides reliability for users and adapts well to environments where many processes are executing over multiple computers [3]. However, the systems currently developed either do not provide transparent and autonomic recovery for users, based on middleware and are not implemented as a system service, or users have to restart their applications manually after a failure [5][6][7]. The aim therefore of this paper is to detail how the recovery of processes in a cluster can be achieved transparently and autonomically to create seamless recovery in clusters using checkpointing and rollback recovery, and to furthermore show that the outcome will be beneficial to the users of the system. It is also the aim to demonstrate how recovery can be achieved using either a centralised or distributed recovery approach and present their comparative study.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 118 – 124, 2005.

2 Target Platform Architecture

It has been identified that a microkernel based cluster operating system compliments the development of new research services as they can be developed independently of the other system services [2]. The microkernel based operating system uses peer to peer servers to provide the complex functionality of the cluster operating system. Whilst the microkernel only provides the minimal functionality needed, the other system servers provide any additionally needed facilities. Each of these facilities communicates using messages. To incorporate a recovery facility, five existing facilities are used to provide the additional services needed by the recovery facility: the checkpoint facility, remote execution facility, global schedular, process facility, and the self-discovery facility.

The checkpoint facility is responsible for checkpointing applications in the system. When a checkpoint is taken, the process's memory, communication buffers, and process information are copied into a new checkpoint. A coordinated checkpoint facility model has been identified for a cluster operating system by Rough and Goscinski, and will be used as a basis of this checkpointing facility [10]. In their model, checkpoints are taken at periodic intervals and are stored in the volatile storage of other computers within the cluster of computers. The remote execution facility (REX) is responsible for process creation. The facility consists of many Remote Execution Managers; one manager is located on each computer within the cluster. These managers can cooperate together to create new processes within the system [1]. Scheduling of processes within the system is managed by a single centralised Global Schedular. This server combines static allocation and dynamic load balancing components. Using current loads and load trends, the Global Schedular makes decisions on where to create new processes. The process facility is responsible for managing processes within the system. The process facility is made up of many Process Managers and each Process Manager is responsible for the processes on its local computer. Lastly, the responsibility of the self-discovery facility is to provide high-level management of all computing resources on a global cluster-wide basis. The self-discovery facility is a dedicated service, which coordinates the discovery of the installed computing resources and their parameters. The self-discovery facility is also responsible for reporting failures of user processes to the recovery facility.

3 Overview of the Proposed Recovery Facility

The recovery facility uses the checkpoints created by the checkpoint facility to recover failed applications. The recovery facility is composed of many Recovery Managers; one Recovery Manager resides on each of the computers within the system. These Recovery Managers communicate using either single or group communications. Once a failure has occurred in the cluster, the recovery facility is alerted to the failure and begins the recovery process. Only user applications are recovered as an assumption is made that the system servers cannot fail.

Transparency must also be upheld during the recovery of a failed application as the user should not know that it has failed, and does not need to know that it has failed unless the application is unrecoverable. The checkpoint facility described in Section 2 currently only stores checkpoints on the volatile storage of other computers within the system. This feature of the checkpoint facility had to be extended to store the checkpoints on stable storage at regular intervals or when the computers are idle to prevent the checkpoint data being lost when a computer fails.

At-least-*k* delivery semantics are used to store the checkpoints of the processes in the volatile storage of other computers [10]. At-least-*k* semantics guarantee to deliver messages to a user specified number of computers, this value is denoted by *k* (where *k* >= 1). Because of these delivery semantics, each time a new checkpoint of a process is taken, the checkpoint may be stored in the volatile storage of a different computer than that of the computer which was used to store the previous checkpoint of that process.

Once implemented, the recovery facility has many advantages over existing manual recovery methods that exist due to the transparency and autonomy provided, as with manual recovery facilities, the user must have explicit knowledge of the recovery facility, how the recovery facility is used, and how to carry out the recovery operation.

4 The Recovery Facility

Different recovery approaches can be used to control how applications are recovered. The two approaches that exist are *Centralised* and *Distributed*. .

4.1 Centralised Approach for Recovery of Processes

The centralised approach aims at using a Coordinating Recovery Manager in order to dictate to the recovery facility which processes should be started on each computer within the system. As there are many Recovery Managers within the system, one Recovery Manager must become a Coordinating Recovery Manager when a failure occurs in the system. When a failure occurs, an *election* takes place to ensure that there is only one Coordinating Recovery Manager.

The local Checkpoint Manager is queried by the Coordinating Recovery Manager to determine where the checkpoints are stored in the system. To select the computers that are used for the recreation of the processes, the Coordinating Recovery Manager contacts the Global Schedular informing it of the processes that need to be restarted and to request a list of computers that are to be used for the recovered processes. Several parameters are passed to the Global Schedular along with the request. These parameters include the location of the stored checkpoint data in the system so that it may use this information to make a more informed decision on where to recover the processes. The Recovery Managers on these target computers are then contacted by the Coordinating Recovery Manager and informed of the processes that have to be recreated on their respective computers. If a computer within the cluster fails during the recovery process, then the Coordinating Recovery Manager is informed by the self-discovery facility and requests another computer begin the recovery for that process.

The information on where the images are stored is passed on to the remote execution facility. With this information, the remote execution facility attempts to retrieve the checkpoint data from the volatile storage of the other computers. If the image is no longer residing in volatile storage, then the image is retrieved from stable storage. The steps to recover an application when using the centralised approach shown in Figure 1 is as follows: step 1 – A fault is reported to the Local Recovery

Manager: step 2 – Check to see if checkpointing is enabled for the failed process: step 3 – Find out which computers the new processes should be started on: step 4 – Inform the Remote Recovery Managers that a recovery is occurring: step 5a – Tell Process Managers to kill any surviving processes of the application: step 5b – Process Managers kill off any surviving processes: step 6 – Request each REX Manager to create the new processes: step 7 – Get the data from either volatile or stable storage: step 8 – Create the process: step 9 – Coordinating Recovery Manager informed that a process is recovered; and step 10 – Coordinating Recovery Manager informs all that recovery is complete.

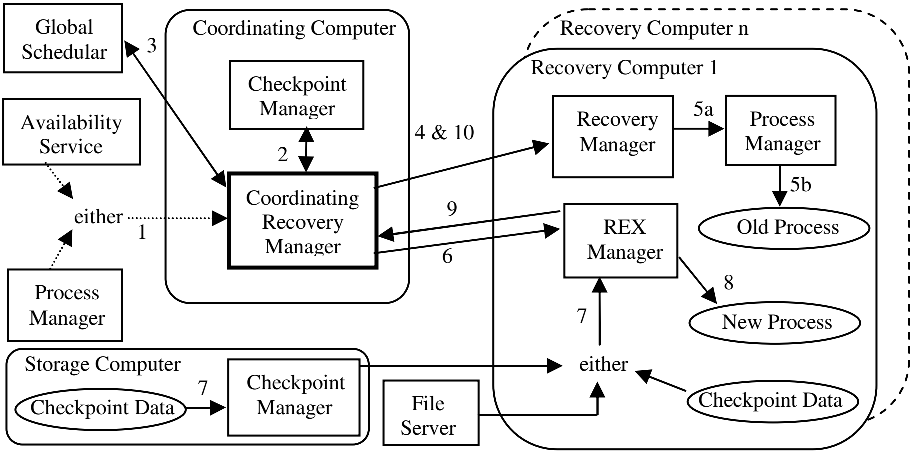


Fig. 1. Operation of Centralised Recovery Mechanism

4.2 Distributed Approach for Recovery of Processes

The distributed approach for recovery of failed applications is quite different to that of the centralised approach. The major difference with the distributed approach is that there is no one Coordinating Recovery Manager. Also, the distributed approach doesn't contact the checkpoint facility, nor does it contact the Global Schedular. Instead, this approach looks at recreating processes whose checkpoints exist on the same computer as the Recovery Manager who is trying to recover the process of the application.

Because the Recovery Manager does not contact the checkpoint facility or the Global Schedular, the recovery process can then get underway faster than in the centralised approach. If a process has been recovered on a heavily loaded computer, then the Global Schedular may choose to migrate the process at a later time once it is recovered. The distributed approach therefore has the advantage that the recovery is completed faster, but may be more expensive later if migration is required to balance the computer loads.

As *k* copies of each checkpoint exist within the system, steps must be taken to ensure that only one of each process is recreated. This is done using *majority voting* with the other Recovery Managers within the system.

If the checkpoint data for a process exists on a local computer, then the Recovery Manager on that computer requests to recover that process. If the Recovery Manager on the local computer has already been informed that another Recovery Manager is attempting to recover that process within the system, then the Local Recovery Manager does not attempt to recover the process.

Once a Recovery Manager has obtained the right to recover a process, it does so immediately. Once the process has been recovered, then the Recovery Manager that was responsible for recovering the process informs the other Recovery Managers that the process has been recovered. Once all Recovery Managers have been informed that each process of the application has been recovered then the recovery is deemed complete.

If during the recovery of processes, a Recovery Manager fails to recover a process, then it sends a *recovery failed* message to the other Recovery Managers within the system. Once this has happened, another Recovery Manager can vote to start to recover the process that was unable to be recovered by the previous Recovery Manager. This also happens if a process is not recovered by another Recovery Manager in a specified length of time. If the recovering of a process fails a predefined amount of times, then it is deemed that the failed application cannot be recovered, and all recovered processes must be terminated. The steps to recover an application when using the distributed approach as shown in Figure 2 is as follows: step 1 – A fault is reported to the Local Recovery Manager: step 2 – Inform the Remote Recovery Managers that a recovery needs to occur: step 3 – Each Recovery Manager sends votes to recover some of the processes: step 4a – Tell Process Managers to kill any surviving processes of the application: step 4b – Process Managers kill off any surviving processes: step 5 – Request each REX Manager to create the new processes: step 6 – Get the data from either volatile or stable storage: step 7 – Create the process; and step 8 – Inform the other Recovery Managers using reliable communications that the process has been created.

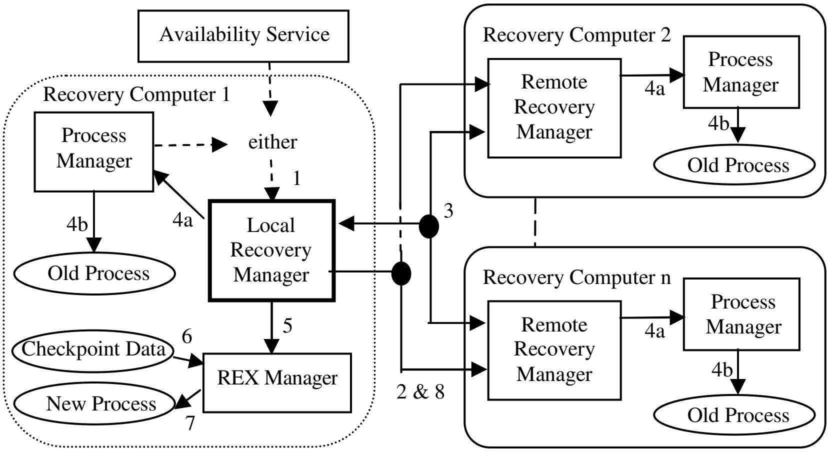


Fig. 2. Operation of Distributed Recovery Mechanism

4.3 Centralised vs. Distributed Recovery

Each of the two approaches has their own strengths and weaknesses. The strengths of the centralised approach are that it uses a Coordinated Recovery Manager to control the steps of the recovery process. The centralised approach also uses the Global Schedular to make an informed decision on where to recreate the processes. Lastly, this approach uses less communication than the distributed approach. The weaknesses however, are that this approach has to deal with selecting a Coordinating Recovery Manager before the recovery can begin, and if the Coordinating Recovery Manager fails during a recovery, then a new Coordinating Recovery Manager needs to be selected. The strengths of the distributed approach are that it eliminates a single point of failure by using Coordinating Recovery Manager. Instead the Recovery Managers request to start a process if the checkpoint data already resides on the local computer; consequently, the checkpoint data does not need to be sent across the network. To ensure that no two Recovery Managers attempt to recover the same process, majority voting is employed by the Recovery Managers. However, this creates a weakness as the use of majority voting greatly increases the amount of traffic transmitted over the network during recovery. Failed processes also may not be recreated on the most suitable computers within the cluster as the computers may be heavily loaded.

Although both approaches are innovative and provide major benefits to the user, the centralised approach is believed to provide a more suitable recovery approach as the proposed advantages appear to be better than that of the distributed approach. The centralised approach uses a Global Schedular in order to select the most appropriate computer to restart the processes. In the distributed approach, the processes could be started on heavily loaded computers and would then have to be migrated to different computers in the cluster. This along with the extra communication due to the use of voting mechanisms increases network traffic significantly.

5 Conclusion

Throughout this document, two logical designs of Recovery Facilities for non-dedicated clusters have been detailed. It has been shown how cluster operating systems can be extended to provide seamless recovery of failed processes through the use of a recovery facility. This recovery facility is a unique and innovative system service in that it can provide transparent and autonomous recovery of processes for the users of the system. This was achieved by using Recovery Managers residing on each of the computers within the system. These managers then use checkpoints that have been taken of the failed processes to recover the failed processes. Furthermore, it has detailed and compared a model for both centralised and distributed recovery.

References

1. Goscinski, A., *Towards A Cluster Operating System That Offers A Single System Image*, In Distributed and Parallel Systems, 2002
2. Maloney, A., Checkpointing and Rollback-Recovery Mechanisms to Provide Fault Tolerance for Parallel Applications, School of Information Technology, Deakin University, 2004, http://www-development.deakin.edu.au/scitech/sit/dsapp/members/ index.php

3. Elnozahy, M., Alvisi, L., Wang, Y. M. and Johnson, D. B., *A Survey of Rollback-Recovery Protocols in Message-Passing Systems*, School of Computer Schience at Carnegie Mellon University, Pittsburgh, PA 15213, 1999
4. Badrinath, R., Morin, C., and Vallée, G., Checkpointing and Recovery of Shared Memory Parallel Applications in a Cluster. In Proc. Intl. Workshop on Distributed Shared Memory on Clusters (DSM 2003), Tokyo, pages 471-477, May 2003.
5. Plank, J. S., Beck, M., Kingsley, G. and Li, K., *Libckpt: Transparent Checkpointing under Unix*, Proceedings of the USENIX Winter 1995 Technical Conference, p. 213-223, 1995
6. Landau, C. R., *The Checkpoint Mechanism in KeyKOS*, from Proceedings of the Second International Workshop on Object Orientation in Operating Systems, September 1992
7. Rough, J. and Goscinski, A, *The development of an efficient checkpointing facility exploiting operating systems services of the GENESIS cluster operating system*, In Future Generation Computer Systems 20, p. 523-538, 2004

Toward Self Discovery for an Autonomic Cluster

Eric Dines and Andrzej Goscinski

School of Information Technology,
Deakin University,
Geelong, Vic 3217, Australia
{esd, ang}@deakin.edu.au

Abstract. Nondedicated clusters are currently at the forefront of the development of high performance computing systems. These clusters are relatively intolerant of hardware failures and cannot manage dynamic cluster membership efficiently. This report presents the logical design of an innovative self discovery service that provides for automated cluster management and resource discovery. The proposed service has an ability to share or recover unused computing resources, and to adapt to transient conditions autonomically, as well as the capability of providing dynamically scalable virtual computers on demand.

1 Introduction

The present generation of parallel processing systems, which have arisen to take advantage of the more accessible computers, tend to be add-ons to existing operating systems (such as Linux and Windows NT) rather than purpose built distributed or cluster operating systems. The most prominent of these add-on systems are PVM [6] and Beowulf [7]. Unfortunately this class of system has several major limitations; a static nature requiring interventionist management techniques [8], a need for dedicated individual component computers in the case of Beowulf [7], an inability to either exploit underutilized desktop workstations as members of a non-dedicated cluster [7] or dynamically manage resource allocation and assignment in real time or derive system knowledge through resource discovery [8].

We have identified five desirable attributes that articulate the requirements of a typical cluster operating system and which overcome these shortcomings:

- ***Automated Cluster Management*** to replace the manual processes currently employed for managing membership and routine administration functions**.**
- ***Resource Sharing*** to take advantage of the casual members of a non-dedicated cluster, such as end user computers.
- ***Resource Discovery*** to determine current and potential members of the cluster, and also their computing resource complement, such as local hard drives, unallocated RAM, or specific installed applications.
- ***Network Performance*** to monitor the underlying performance of the network infrastructure, and optimize scheduling decisions accordingly.
- ***Dynamically Sizable Virtual Computers*** to dynamically reconfigure access to portions of a cluster based on user requirements.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 125 – 131, 2005.

Achieving these attributes requires that the system maintains a dynamic global knowledge of the resources of the non-dedicated cluster, from a fine grained knowledge of the computing resources on a given computer (e.g. free RAM) to knowledge of whole sub-groupings of computers. This knowledge or self-awareness embraces a single system image (SSI) view of the cluster [3] incorporating:

- A dynamic system with computers joining and leaving at unpredictable times;
- A collection of independent computers, often controlled by other users; and
- A collection of logical clusters set up for individual applications.

The aim of this report is to show the outcome of our study into the development of a system service or set of services that are able to provide this functionality, through which the current shortcomings of existing cluster operating systems can be overcome. We propose a self discovery service in an attempt to meet these goals and enable the cluster operating system to self manage autonomically.

2 Target Architecture

The basic system requirements for the deployment of the self discovery service are for a microkernel, client-server (peer to peer) and distributed system architecture. Figure 1 shows a simplified block representation of a microkernel based cluster operating system. At the lowest level, the microkernel forms a hardware abstraction layer that provides only the bare minimum set of services needed to support the marshalling of interrupts, context switching, local IPC and memory management. Operating system functionality and application execution environment are provided by kernel and system servers. Kernel Servers operate in the user context and "use a set of privileged calls to manipulate microkernel data" [2] and provide much of the system functionality. An instance of each of these servers is deployed per machine [5].

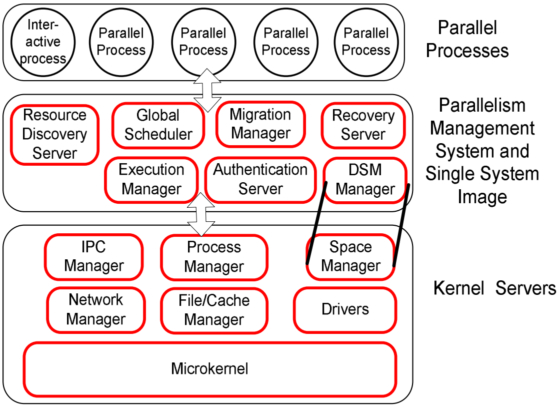


Fig. 1. Microkernel System Architecture

System Servers implement operating system functionality having system wide applicability. The system servers in contrast to the kernel servers can be hosted on any given computer within the cluster, as they are not bound to any specific microkernel.

The major servers that are related to the self-discovery service are as follows:

- ***Execution manger*** creates a process locally or remotely from a file or duplicates a process on one or more computers simultaneously.
- ***Resource discovery manager*** collects system information about its host computer.
- ***Migration manager*** coordinates the relocation of either an active process or a set of processes on one computer to another or a collection of computers.
- ***Process manager*** manages the processes on the local computer. It manipulates the process queues and deals with parent and child process interactions. It cooperates with the execution manager to set up the process' state when the process is created, and the migration manager to transfer a process' state when it is migrated.
- ***IPC Manager*** is responsible for delivering all messages including group communication services to both local and remote process and/or processes.
- ***Global Scheduler*** provides scheduling services in order to allocate/migrate processes to idle and/or lightly computers to share and balance load.
- ***Recovery Server*** oversees the checkpointing of active processes across the system and manages the recovery of these processes should one exit abnormally.

3 The Logical Structure of the Self Discovery Service

The self discovery service is provided by the system discovery server (SDS) and a set of resource discovery servers (RDS). The system discovery server is deployed as a single non-redundant instance for the entire physical cluster system, similarly to the global scheduler (Figure 2) with which it interacts and cooperates closely.

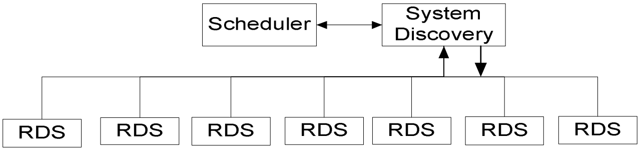


Fig. 2. Architecture of the self discovery server

The five desirable attributes for cluster from section 1 are encapsulated within the context of the three major duties of the self discovery service: managing the physical cluster; managing the virtual cluster; and managing system self-awareness.

3.1 Roles of Servers

The actual roles to be attributed to each of the three components global scheduler, resource discovery server and system discovery servers are as follows.

The **global scheduler** sets the policies for statically allocated work (remote execution manager) and load balancing (migration manager). The global scheduler's view of the cluster is provided by the system discovery server who may alter the presented topology in order to influence scheduling decisions (Figure 3).

The **system discovery server** implements the global scheduler's policies by managing the individual resource discovery servers, collating and analyzing the data provided by the resource discovery server and manipulating the performance indices. The system discovery server organizes functional groups of computers as private virtual clusters (PVC); manages membership to the cluster and loss of a computer/s.

The **resource discovery server** captures computer performance parameters, pre-processes the raw data where appropriate and supplies the collated and processed data to the system discovery server.

3.2 Management of the Cluster

The self discovery service provides a stable fault tolerant platform for the execution of parallel applications within a dynamic computer membership environment.

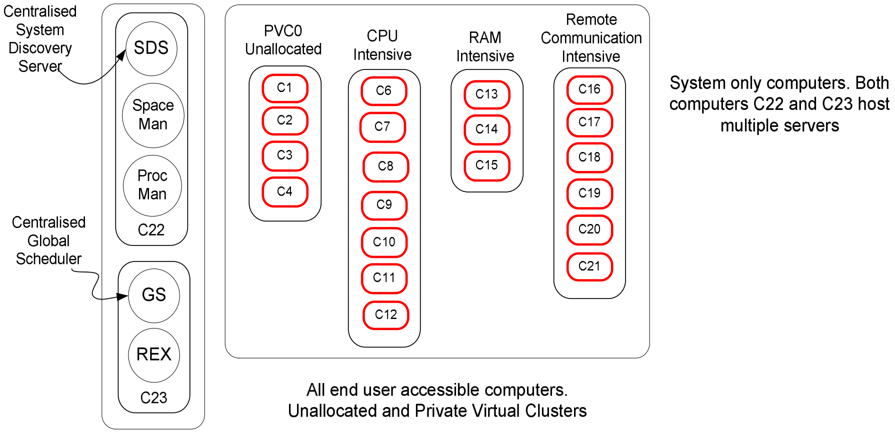

Centralised System Discovery Server
SDS
Space Man
Proc Man
C22
Centralised Global Scheduler
GS
REX
C23
PVC0 Unallocated
C1
C2
C3
C4
CPU Intensive
C6
C7
C8
C9
C10
C11
C12
RAM Intensive
C13
C14
C15
Remote Communication Intensive
C16
C17
C18
C19
C20
C21
System only computers. Both computers C22 and C23 host multiple servers
All end user accessible computers. Unallocated and Private Virtual Clusters

Fig. 3. Private virtual cluster management

Managing the Physical Cluster. The self discovery service has complete oversight of the operation of the cluster system, with the system discovery server managing all computers from the bootstrap to planned or unplanned departure.

Managing the Virtual Clusters. The private virtual cluster (PVC) is a logical aggregation of computers to collectively manage a computing task.

At startup, all computers are in PVC0 (see figure 3). Once the first application has been scheduled for execution, the system discovery server selects one or more computers from PVC0 and allocates them to a new private virtual cluster (PVC1). Processing commences once PVC1 has been successfully initialized.

Similarly configured computers are drawn from the ranked lists (CPU speed, installed RAM and network interface speed, see figure 3) and allocated together in PVC's. The initial groupings are tentative, and act as a forward planning optimization.

Managing System Self-Awareness. The system discovery server maintains a logical map of active computers, with details of current resource capability; updated from periodically received RDS data. The SDS interrogates uncommunicative computers directly and removes them if they do not respond within a timeout period.

3.3 Collection of Data

The attributes of most critical interest to the system discovery server and the resource discovery mechanisms used to capture the required data are presented below.

Static and Dynamic Parameters. There are two general categories of data collected, Static and Dynamic. Static parameters are set at boot-up time, and tend not to change: (i) Processor: clock speed, number of individual CPU's present; (ii) System Memory: installed capacity, unallocated or free RAM; (iii) Hard disk: gross capacity, free disk space, access speed, latency, throughput; and (iv) Installed Software: knowledge of specialised software installations.

Dynamic parameters are not fixed at boot-up time, and tend to fluctuate or change as processing and system load changes: (i) Number or process running; (ii) Available memory; (iii) Free disk space; and (iv) Inter-process communication pattern and volume, especially remote inter-process communication.

Factors Impacting Data Collection. When collecting the system data, the capture must: (i) be conducted in real time, (ii) not adversely affect system performance, (iii) not appreciably impact the network during transmittal.

4 Interaction of Servers

Systems designers have experimented with reporting individual system resource data through the scheduler [Goscinski et al 2002], but at the expense of an overlap of the roles of the individual resource discovery servers and the global scheduler. Introducing the system discovery server simplifies the role and reduces the workload of the global scheduler by removing the overhead of processing the resource-data.

Message 1 (M1) is the regular collection of performance data by the resource discovery servers. M2 shows a dual interaction, where the resource discovery server pushes data (including notification of changed workload) to the system discovery server periodically and the system discovery server requests performance data updates from a resource discovery server on demand. There is also a two way interaction between the global scheduler and system discovery server (M3). The system discovery server, after synthesizing the system's state and global topology from data provided by the individual computers' resource discovery servers (Figure 4), then provides this knowledge to the global scheduler. The global scheduler uses the updated data to schedule the most lightly loaded computer within given PVC.

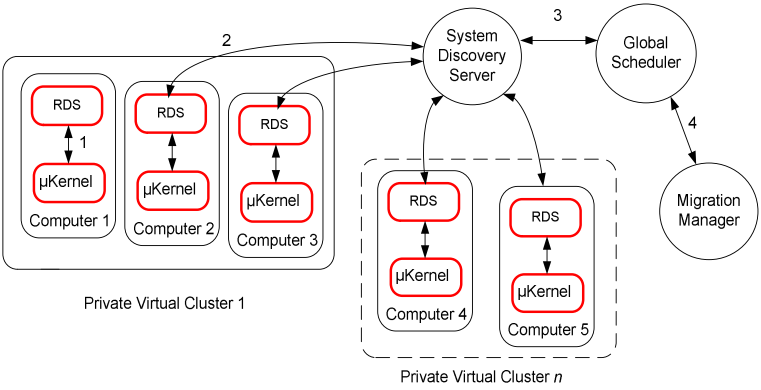


Fig. 4. Server Relationship

The system discovery server does not take part in load balancing or migration, but indirectly influences the cluster operating system's response to changing workloads by dynamically resizing the physical cluster and the private virtual clusters.

The resource discovery server has the ability to act independently when rapid action is required; responding to a user resuming control of their computer or to an urgent directive from the system discovery server.

5 Implementation

The system discovery server has been successfully implemented and is capable of communicating with the newly redeployed resource discovery server and the global scheduler. The resource discovery servers gather the static parameters at system start up and store them in a special area of RAM. Process loading and communication statistics are then collected by the resource discovery server during run time. Some rudimentary pre-processing such as calculating summary totals, and arithmetic means is undertaken at this stage before the processed data is forwarded on to the system discovery server.

The system discovery server builds and maintains a map of the complete system topology to manage the private virtual cluster membership and storage of static and dynamic parameters (section 3.3) within a linked list of data structures.

The global scheduler uses this same map to make its scheduling and load balancing decisions, but solely on the computers within a single private virtual cluster at any time, scheduling each private virtual cluster in turn.

6 Conclusion

The introduction of the self discovery service effectively addresses the inherent problems of existing cluster operating systems, namely their lack of self-regulation, their static character and their need for manual management [8]. The self discovery service has ensured that through improved self-awareness and more numerous points of control, the clusters are able to be much more autonomic and self-configuring. By maintaining a system wide awareness of the state of each member of the cluster, new computers can join or others leave as required. The self discovery service also provides a private virtual cluster construct, enabling the aggregation of similarly configured computers together in dynamic units for ease of management.

References

1. Goscinski A & Zhou W, 1999. Client Server Systems. Wiley Encyclopedia of Electrical and Electronics Engineering, JG Webster (Ed), John Wiley & Sons, Vol. 3, pp. 431-451.
2. Goscinski A “Finding Expressing and Managing Parallelism in programs executing on clusters of workstations”. Computer Communications 22:998-1016, 1999.
3. Goscinski A “Towards and operating system managing parallelism of computers on clusters. Future Generation Computer Systems 17:293-314, 2000.
4. Goscinski A., Fikkers P. and Zhou B. “A Global Scheduling Facility for Clusters Executing Communication Bound Parallel Applications”. School of Computing and Mathematics. Deakin University, 2002.
5. Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R & Sunderam V, “PVM: A Users' Guide and Tutorial for Networked Parallel Computing” MIT Press 1994
6. Merkey P “Beowulf History” http://www.beowulf.org/beowulf/history.html 2003
7. Sterling T and Savarese D “A Coming of Age for Beowulf-class Computing”. Center for Advanced Computing Research. California Institute of Technology June 1999
8. Zaki M & Parthasathy S “Customised dynamic load balancing for a network of work stations” Technical Report, The University of Rochester. New York 1995

Mining Traces of Large Scale Systems

Christophe Cérin and Michel Koskas

[1] Université de Paris XIII, LIPN, UMR CNRS 7030,
F-93430 Villetaneuse - France
cerin@lipn.univ-paris13.fr

[2] Université de Picardie Jules Verne, LaMFA/CNRS UMR 6140,
33, rue St Leu,F-80039 Amiens cedex 1 - France
koskas@laria.u-picardie.fr

Abstract. Large scale distributed computing infrastructure captures the use of high number of nodes, poor communication performance and continously varying resources that are not available at any time. In this paper, we focus on the different tools available for mining traces of the activities of such aforementioned architecture. We propose new techniques for fast management of a frequent itemset mining parallel algorithm. The technique allow us to exhibit statistical results about the activity of more that one hundred PCs connected to the web.

Keywords: Parallel algorithms, global computing platforms, meta-data, data mining application, high performance and distributed databases, trace analysis, data management, resource management.

1 Introduction

Frequent itemset mining (FIM) consists in discovering patterns that appear frequently. In this paper the itemsets are informations about the activity (the CPU/MEMORY loads, the number of IP packets sent or received from/to a dedicated node, timestamp of the measure...) of a set of PCs in a research laboratory. The ultimate goal for that application is to extract information to be pass to a scheduler in order to run jobs with a reasonable knowledge of the "future state" of the global platform.

FIM algorithms are often used in search for other types of patterns (like sequences, rooted trees, boolean formulas, graphs). More than one hundred FIM algorithms were proposed in the literature, the majority claiming to be the most efficient. In any case, it is difficult to appreciate the experimental methodology. For instance it is difficult to have answers to the following questions: what is the part of the execution done in/out-of-core? What is the execution time for generating the 1-itemset (it corresponds in general to a full reading of the database) and the execution time for generating the $k > 1$ itemsets?

Three algorithms play a central role due to their efficiency and the fact that many algorithms are modifications or combinations of these basic methods. These algorithms are APRIORI [1], ECLAT [2] and FP-growth [3].

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 132–138, 2005.

In this paper we introduce challenges, opportunities and technical solutions that we believe to be important for mining the activities of large scale systems. The discussion is conducted with our parallel algorithm for frequent itemset generation [4] in mind. Besides the algorithmic and data-structure issues there is a third factor that quite influences the effectiveness of the different approaches found in the literature. It is the programming technique.

Thus, the organization of the paper is the following. In Section II, we introduce the challenges and those we are concerned about in the paper. Section III is about the data structure we use and section IV is about the problem description and the advantage of our approach. Section V concerns the programming techniques and it shows experimental results. Section VI concludes the paper.

2 Building a FIM Algorithm for Large Scale Systems

Our target architecture is large scale systems. The main properties that we require for large scale systems are:

- *Scalability:* the system must scale to 100000 nodes;
- *Heterogeneity* of nodes across hardware, OS and basic software;
- *Availability:* the owner of a computing resource must be able to decline a policy that will limit the contribution of the resource (the resource will be disconnected in a near future);
- *Fault tolerance:* the architecture must tolerate frequent faults while maintaining performance;
- *Security:* all participating computers should be protected against malicious or erroneous behaviors;
- *Dynamicity:* the system must accommodate to varying configuration; an event may happen at any time;
- *Usability:* the system should be easy to deploy and to use.

In this paper we discuss only the advantage of our algorithm [4] in terms of scalability, dynamicity, fault tolerance and performance at the large. We are also concerning with an implementation.

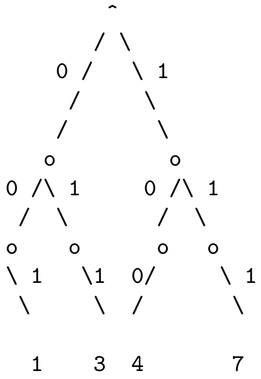
3 The Data Structure

We estimate that the way we represent data will potentially enforce the efficiency of mining algorithms. The parallel algorithm of [4] makes elementary operations on radix-trees data-structures in order to compute the frequent episodes.

Let S be a set of integers written in basis $b = 2$ for instance (it is convenient to chose as basis a power of 2). It is well known that the integers may be represented in a radix tree. A radix tree is a tree which allows to store a set of words over an alphabet A of same length (here the alphabet is the set of digits $0 \ldots b-1$).

Consider the thesaurus of a column in a table. For instance, the lines where a certain item appears are $\{1, 3, 4, 7\}$. A Radix Tree representation of set $\{1, 3, 4, 7\}$ is given on the next page.

Suppose that we have to check if key $5 = 101_2$ is present in the tree on the right side. We descend along the tree until we encounter the prefix 10 after that, since the last bit (1) is not present, we conclude that 5 does not belong to the tree. The previous scheme, explains also how to answer to a SQL like query with an AND clause, for instance this one:



```
SELECT ALL FROM Accident
WHERE
    = Accident KindOfCont 'Car'
    AND
    >= Accident MaxAmount 12,000
GROUP {NULL}
```

It is just necessary to intersect the Radix Tree for the 'Car' item with the Radix Tree for the '$\geq$ 12000' item.

4 Problem Description

The problem of association rule discovery can be formalized [5] as follows. Let $\mathcal{I} = \{i_1, \cdots, i_m\}$ be a set of m distinct *items*. A transaction is any subset of $\mathcal{I}$ and each transaction $\mathcal{T}$ in a database $\mathcal{D}$ of transactions has a unique identifier. A transaction is a p-tuple $< TID, i_1, \cdots, i_k >$ and we call $i_1, \cdots, i_k$ an *itemset* or a *k-itemset*.

An *itemset* is said to have a support of s if $s\%$ of the transactions in $\mathcal{D}$ contains the itemset. An association rule is an expression of the form $A \Rightarrow B$ where $A, B \subset \mathcal{I}$ and $A \bigcap B = \emptyset$. The *confidence* of the association rule is simply the conditional probability that a transaction contains B, knowing that it contains A. It is computed as support$(A \bigcap B)$/support(A).

Given m items, there are potentially 2^m itemsets whose support is above a given support. Enumerating all itemsets is thus not realistic. However, for practical cases, only a small fraction of the whole space of itemsets is above a given support requiring special attention to reduce memory and I/O overheads.

The "Apriori" sequential algorithm forms the core of many variants of association rules discovery algorithms. It uses the fact that a subset of frequent itemset is also frequent, then only candidates found "previously" are used to generate a new candidate set. In [4] we have introduced new techniques for association rules discovering. We have revisited the Apriori algorithm that serves as the main conceptual block for such purpose in showing how to store and generate candidates by the mean of Radix Trees.

From a "large scale point of view", the main properties of the algorithm on p processors is:

- The local database is read once. This step serves in building the 1-itemset, that is to say the radix-trees coding "where" each item appears. The $k > 1$ itemsets are generated by intersections, locally on each node. If we assume that a new itemset can arrive at any-time (a new measure in our application), we should minimize its insertion time. In our case, the cost of inserting one item is a constant time, independent of the number of data since it is based on the tree high which is a constant in our implementation (for instance 40 if we are working with tables with 2^{40} lines). This property is important in the case of the aforementioned property of dynamicity of large scale systems.
- When we exchange information about nodes, only the supports (integers) are exchanged. There is $(p-1)^2$ messages during this steps and the length of each message is proportional to the number of frequent itemsets that are generated (k) multiply by the size of an integer. Thus the volume of information exchanged in any step of the parallel algorithm is exactly $(p-1)^2 \times k \times sizeof(int)$. We note that it is independent of n the number of data in the database. We may assume that in practical cases, this volume is low. The consequence is that in the case of faults that are remedied by checkpointing mechanisms, the checkpoint will contain not too much information. For instance, if we use MPICH-V (a fault-tolerant MPI available on `http://www.lri.fr/~bouteill/MPICH-V/`), the NAS Benchmark BT B on 25 nodes (32MB per process image size) leads to the average time of 68s to perform checkpoint with MPICH-V. The average time to recover from failure with MPICH-CL is 65.8s. The application is much more communication intensive than our frequent itemset algorithm. We are optimist to accomplish a checkpoint in less than 1s on 25 nodes.

However, the number of generated itemsets varies from one iteration to another one. In the context of heterogeneous computing (the candidate and frequent itemset generations are computed on processors whith different CPU speed and with different communication bandwidths) it is more difficult to estimate the time cost of these two steps, hence potential unbalanced work. Techniques to control the load balancing, such as the technique used in [6] in the case of sorting and for a one-communication-step algorithm can not apply. Thus, the problem of controlling load balancing is challenging both in theoretical and programming terms. It is important to mention it at this time.

5 The Programming Technique Part

Let us now comment our implementation choices in the case of our sequential prototype for mining frequent episode. Radix Trees can be implemented with pointers (for the left and right children) when they are loaded into the RAM. We know that pointers do not preserve spacial locality (the next item to be used is "closed" in memory to the current item) and it is not also suited for temporal locality (the current item will be re-used in a near future). To check this fact, we have implemented tree operations (union, intersect) with the STL C++ library and lists and with pointers. We have obtained better experimental results for pointers than for lists (implemented under the STL C++ framework).

But the time completion for union or intersect operation is not good enough for large scale computation. For instance, 600 intersection operations on trees containing 150000 elements each last 58.39 seconds on a Sun bi-opteron v20z system. These 600 operations involve 90M of items.

We decided to shift to bitset abstract data type in order to implement "the line where an item occurs" concept. Remind that Radix Trees have been introduced to store sets of integers. With a bitset, we set to 1 the k-*th* bit if integer k is a member of the set and we set it to 0 otherwise. The STL C++ library offers an interface to bitsets but after some tests with the library and under `g++` release `3.4.1` we have decided to re-implement it, partially in assembler code.

The motivation is to use MMX, SSE-2 or Altivec technologies for 32 bits processors. For such technologies, the processor can address 128 bits registers and we can use them to implement union operation (i.e. "or" operation on two bitsets), intersect operation (i.e. "and" operation on two bitsets. The STL C++ library under `g++` does not use such technologies.

We have also introduced (by hand) prefetching memory instructions. Such optimizations are essential to fully exploit 128 bits registers and to hide memory latencies.

We obtain a gain of at least 30% for our MMX/SSE or Altivec implementations against the STL C++ codes. For instance, the cost of 5000 "and" operations on two bitsets of size 1048576 bytes (representing two sets of 8388608 elements) is 7.75 second on a Duron at 1.9Ghz. It is a very good result comparing to our implementations based on pointers (see above). The effort in coding the new bitset interface is not too important for a great result. We have to use GCC 4.x release. One improvement of GCC 4.x is the possibility to produce SSE2 and Altivec codes in order to use 128 bits registers. Two new compiling options were introduced: `-ftree-vectorize` and `-ftree-loop-linear`.

5.1 Trace Analysis

The trace that we have explored corresponds to a set of 110 stand alone PCs under Windows in a laboratory of researchers, engineers and administration people. The trace records 11 events every 15 min during the day and for a period of 15 days. The trace represents about 50Mb of uncompressed data in size. The name of the table is BigTable. We estimate that if we sample every minute, the file size will be about 11Gb for 15 days and we are optimist.

A Frequent Episode sequential algorithm based on [4] and on bitsets has also been implemented. We have chosen 40 items in the BigTable table of 685673 lines. Our implementation uses `libpcre`[1] for matching patterns. The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5. PCRE has its own native API, as well as a set of wrapper functions that correspond to the POSIX regular expression API.

[1] `http://www.pcre.org/`

Since we have 685673 lines in the flat input table, we have set our bitset size to 131072 bytes. The total memory size is thus about 40MB, that is to say closed to the table size (47MB). So our implementation is an in-core one. Under GCC 3.3.4 (pre 3.3.5 20040809) the execution time on a Duron 1.8Mhz for counting the occurrences of these 40 items and including the setting of bitsets with the line numbers where they appear is 14.58 seconds.

This means that we need 14.58 seconds for generating *1-itemsets* including one pass over the input file. With a support equals to 68567, which represents 10% of the number of lines in the input table, we get 13 *1-itemsets.* The number of *2-itemsets* is 17 (we compute it according to the same support) among potential 78 2-combinations. The number of *3-itemsets* is 6 (we compute it according to the same support) and these is no more frequent itemsets. One of the *3-itemsets* corresponds to the event "a CPU load greater than 90% and the number of running processes is greater than 90 and the available memory is greater than 300MB".

The total number of elements involved in intersect operations is 89342969. The execution time from the end of the *1-itemset* production to the end of the program is less than 1 second (0.91s). This time includes the generation of *2-itemset*, *3-itemset* and *4-itemset.* It is a very good result comparing to the previous incomplete tests based on scripting languages. The number of elements involved in intersect operations is quite impressive and the experiment confirms that the data structure choice is a good one.

Under GCC 4.1 and the following option flags `-O4 -fomit-frame-pointer -fprefetch-loop-arrays -ftree-vectorize -msse2 -ftree-loop-linear` we got the same execution time. In fact, this test cannot allow us to distinguish the performance of the bitset library alone because it uses the same ASM bitset library.

5.2 Impact of the Statistical Results on Placement

The final aim of the experimental study is to discover trends in the behavior of PCs connected on a large scale distributed system. The aim is to place tasks. One of the frequent *3-itemset* that we have generated according to our algorithm is: “CPU load $>$ 90% and Memory Available ‘$> 300K$ and number of processes $>$ 90”. It shows that a high CPU load is frequent. Recommendation to place tasks is difficult with this information. We have also another frequent *3-itemset* saying that “CPU load $<$ 10% and Memory Available ‘$< 20K$ and number of processes $<$ 40”.

It is a little bit surprising and it is due to the choice of the support. More discriminant method should accompany the frequent episode algorithm. For instance, in our result we have 5/6 frequent episodes saying that the load is $<$ 10% and only one frequent episode saying that the load is $> 90'$. We have also 4/6 frequent episodes saying that the memory available is between 20 and 40K. The others occurrences of frequent items in the frequent episode is less or equal to 2.

Moreover, if the application is not a critical one we could spend time on collecting burst events to examine in deep this phenomenon. In this case, the key challenge is to master the disk space to store and/or to factorize massive information with common properties: if the burst occurs for the CPU usage, others informations may not vary a lot.

6 Conclusion

In this paper we have presented how we are currently implementing in the "ACI Masse de donnée Grid Explorer Project" a middleware in charge of controlling common data structures used in order to store activities of participant PCs in a large scale system. Different techniques have been explored for mining the trace of the activity and in order to get performance.

Our Apriori algorithm is based on Radix Tree and/or Bitset data structures. Such data structures have been proved efficient according to a pointer based implementation but bitsets are more promising. We are currently developing a multithreaded version of our bitset library for clusters of SMP. The multithreaded version of the intersect operation of two Radix Trees, for instance, introduces problems with balancing the work among threads. We are investigating such issues.

Concerning the Apriori algorithm, we will implement an out-of-core version in order to deal with large tables and before implementing the parallel version depicted in [4]. Our objective is to capture tables until 2^{40} lines. A compromise between space and efficiency for the Bitset data structures is currently under concern.

Acknowledgements. We also address a special thank to Oleg Lodygensky from LAL laboratory in Orsay - France for his tool that inspect and collect traces.

References

1. R. Srikant and R. Agrawal, "Fast algorithms for mining association rules," in *The International Conference on Very Large Databases (VLDB)*, 1994, pp. 487–499.
2. O. Zaki, Parthasarathy and Li, "New algorithms for fast discovery of association rules," in *In D. Heckerman, H. Mannila, D. Pregibon, R. Uthurusamy and Park editors, Proceedings of the 3rd International Conference on Knowledge Discovery and Data Miing - AAAI Press*, 1997.
3. P. Han and Yin, "Mining frequent patterns without candidate generation," in *In proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, 2000.
4. G. Cérin, Koskas and Le-Mahec, "Efficient data-structures and parallel algorithms for association rules discover," in *3rd International Conference on Parallel Computing Systems (PCS'04), Colima, Mexico*, September 2004.
5. M. J. Zaki, "Parallel and distributed association mining: A survey," *IEEE Concurrency*, vol. Vol. 7, No. 4, October-December 1999, pp. 14-25.
6. M. J. Christophe Cérin, Michel Koskas and H. Fkaier, "Improving parallel execution time of sorting on heterogeneous clusters," in *Proc. 16th International Symposium on Computer Architecture and High Performance Computing (SBAC'04), Foz-do-Iguazu, Brazil*, 2004.

Setup Algorithm of Web Service Composition*

YanPing Yang[1], QingPing Tan[1], and Yong Xiao[2]

[1] Computer College of National University of Defense Technology,
Changsha, Hunan, P.R. China
{yanpingyang, tan}@nudt.edu.cn

[2] PDL Laboratory of National University of Defense Technology,
Changsha, Hunan, P.R. China
yongxiao@nudt.edu.cn

Abstract. A number of web services are now available and it therefore seems natural to reuse existing web services to create composite web services. The pivotal problems of web services composition are how to model the input and output data dependency of candidate web services and how to satisfy that of a service request by composition efficiently. In this paper we present the concept of "invocation layer" based on data dependency between web services invocation and design the algorithms to get the least invocation layers of candidate web services satisfying the given service request.

1 Introduction

A web service is a software system designed to support interoperable machine-to-machine interaction over a network. There might be frequently the case that a web service does not provide a requested service on its own, but delegates parts of the execution to other web services and receives the results from them to perform the whole service. In this case, the involved web services together can be considered as a composite web service.

All-sided development process for composite web services involves solutions to several problems, which, generally speaking, are discovery of useful candidate web services, calculation of their possible composition, and execution of the new generated Web Service. The work presented in this paper is providing concrete approaches to the problem of calculation of web services composition.

We propose the concept of "invocation layer" based on data dependency between web services invocation. We design three algorithms to jointly get the least invocation layers of candidate web services satisfying the given service request. Firstly, we find the relevant web services from the repository. Secondly, we pick out the contributed web services based on dataflow optimization. At last, we use a search algorithm based on A* procedure to find the best composition setup.

The remainder of this paper is organized as follows: Section 2 introduces our motivation and Section 3 describes our algorithms in details. Section 4 proposes to use Bloom Filter to implement the set operations in the algorithms. Finally, conclusions and future plans are given in Section 6.

* The work reported in this paper has been funded by the National Grand Fundamental Research 863 Program of China under Grant No.2003AA001023.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 139 – 148, 2005.

2 Motivation

We represent web services and service request in the standard way [3] as two sets of parameters (inputs and outputs).

Definition 2.1 (Web Service). A *web service ws is 2-tuples $ws=<ws_{in}, ws_{out}>$, where $ws_{in}= \{I_1, I_2,..., I_{|ws_{in}|}\}$ is the set of input parameter, and $ws_{out}= \{O_1,O_2,..., O_{|ws_{out}|}\}$ is the set of output parameters.*

Input and output parameters of web service have the following semantics: In order for the service to be invokable, a value must be known for each of the service input parameters. Upon successful invocation the service will provide a value for each of the output parameters.

A service request can be represented in a similar manner, but its input and output parameter sets have different semantics: The request inputs are the parameters available to the composition (e.g., provided by the user). The request outputs are the parameters that a successful composition must provide. The solution must be able to provide a value for each of the parameters in the problem output. Likewise, we can define formally a composition request r as follows.

Definition 2.2 (Service Request). A *service request r is 2-tuples $r=<r_{in},r_{out}>$, where $r_{in} =\{A_1,A_2,..., A_{|r_{in}|}\}$ is the set of available or existing input parameters and $r_{out} =\{D_1,D_2,..., D_{|r_{out}|}\}$ is the set of desired output parameters.*

For manipulating web service or request descriptions we will make use of the following helper functions:

Definition 2.3 (Function in and out). *The functions are mapped from a web service or service request to its set of input parameters and output parameters respectively. That is, $in(x)=x_{in}$ and $out(x)=x_{out}$, where x is a web service or a service request.*

We assume that both service and request descriptions (x) are well formed in that they cannot have the same parameter both as input and output: $in(x)\cap out(x)=\varnothing$. The rationale behind this assumption is that if a description had an overlap between input and output parameters this would only lead to two equally undesirable cases: either the two parameters would have the same type in which case the output parameter is redundant or they would have different types in which case the service description is inconsistent.

If we can discovery a web service ws satisfying a given service request r, then ws can be invoked using the existing parameters of r and produce the desired parameters of r. We define these conditions as a predication *FullySatisfy*.

Definition 2.4 (Predication *FullySatisfy*). *Let WS be the set of all available web services which can be found from a local file system, resources referenced by URIs or provided by a repository such as UDDI. Let RQ be all service requests. $ws\in WS$ and $r\in RQ$. FullySatisfy is a predicate $FullySatisfy: WS\times RQ\rightarrow Bool$ having the following definition: $FullySatisfy (ws,r) = true$ iff $(in(ws)\subseteq in(r))\wedge (out(ws)\supseteq out(r))$*

In practice, however, it is often impossible that one web service can fully satisfy the given request. Then, one has to combine multiple web services that only partially satisfy the request. Given a request *r* and two web services *x* and *y*, for instance, suppose one can invoke *x* using inputs in *in(r)*, but the output of x does not have what we look for in *out(r)*. Symmetrically, the output of *y* generates what we look for in *out(r)*, but one cannot invoke y directly since it expects inputs not in *in(r)*. Furthermore, using initial inputs of *in(r)* and the outputs of *x*, one can invoke *y* (i.e., in(r)$\cup$out(x)$\supseteq$in(y)). So the request *r* can be satisfied by the *invocation layers* of r→{x}→{y}.We define the conditions above as a predication *LayeredlySatisfy*.

Definition 2.5 (Predication *LayeredlySatisfy*). *Let r be as definition 2.4.* $S_1,\ldots,S_n$, *$(n\geq 1)$ is a sequence of web services set and $S_i \subseteq WS$ ($1\leq i\leq n$). The predication LayeredlySatisfy: $(P(WS))^{Nat} \times RQ \rightarrow Bool$ has the following definition.*

$LayeredlySatisfy((S_1,S_2,\ldots,S_n),r)$ = true iff the following three conditions hold:

(a) $\forall ws\in S_1$ ($in(ws) \subseteq in(r)$)

(b) $\forall i\ 1\leq i \leq n$ $((in(r)\cup \bigcup_{ws\in S_1} out(ws) \cup\ldots\cup \bigcup_{ws\in Si-1} out(ws)) \supseteq \bigcup_{ws\in Si} in(ws))$

(c) $(\bigcup_{ws\in S_1} out(ws)\cup \ldots \cup \bigcup_{ws\in Sn} out(ws)) \supseteq out(r)$

Here, $S_1,S_2,\ldots,S_n$ is called an Invocation Layers Sequence (ILS for short) for r and i(1≤ i≤ n) is called Invocation Layer Number (ILN for short). Especially, n is called the Greatest ILN (GILN for short). According to the definition of predication *LayeredlySatisfy*, we can get out(r) by n layer invocations. Obviously, *FullySatisfy* is special case of *LayeredlySatisfy*.

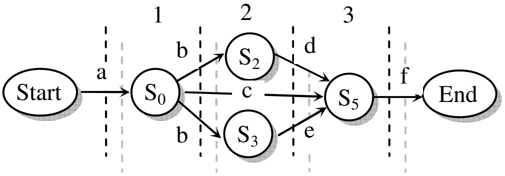


Fig. 1. LayeredlySatisfy(({s_0},{s_2,s_3},{s_5}),r) stands

In Fig.1, there are four web services s_0,s_2,s_3 and s_5 with s_0=<{a},{b,c}, s_2 = < {b}, {d} >, s_3=<{b},{e}>,s_5=<{d,e,c},{f}> and a service request *r* with r=<{a},{f}>. Obviously, *LayeredlySatisfy* $((\{s_0\},\{s_2, s_3\},\{s_5\}),r)$ stands and the *GILN* equals 3.

3 Composition Setup Algorithm

In this section, we introduce the algorithms to get the least invocation layers of candidate web services to satisfy the given service request. Firstly, we find the relevant web services from the repository. Secondly, we pick out the contributed web services based on dataflow optimization. At last, we use a search algorithm based on A* procedure to find the best composition setup.

3.1 Relevant Web Services Finding Algorithm

The first part of our approach is to design an algorithm to find the relevant web services satisfying the predication *LayeredlySatisfy*. The pseudo code of it is shown as follows.

***Algorithm* GetILS** *(Input: web services corpora WS, service request r; Output: invocation layers layer)*

```
  1) $visitedWs \leftarrow \varnothing$
  2) $gottenPara \leftarrow in(r)$
  3) $n \leftarrow 0$
  4) $layer[n] \leftarrow \{start\}$
  5) While $\neg(gottenPara \supseteq out(r))$ do
    5.1) $S \leftarrow \{ws \,/\, ws \in WS, ws \notin visitedWs, in(ws) \subseteq gottenPara\}$
    5.2) if $S = \varnothing$
      5.2.1) then print "Failure!" and return
    5.3) $n \leftarrow n+1$
    5.4) $layer[n] \leftarrow S$
    5.5) $visitedWs \leftarrow visitedWs \cup S$
    5.6)$gottenPara \leftarrow gottenPara \cup (\bigcup_{ws \in S} \mathrm{out(ws)})$
  6) $n \leftarrow n+1$
  7) $layer[n] \leftarrow \{end\}$
8) return
```

Variable visitedWs is a set and used to save the web services that have been visited so far, and variable gottenPara is also a set and used to save the parameters that have been available or generated so far. Array variable layer is used to save the web services of each invocation layer. Constant WS represents a set of all available web services which can be found from a local file system, resources referenced by URIs or provided by a repository such as UDDI. Variable r denotes a given web services composition request. Start and end nodes are virtual services that respectively provide require the data from the problem.

At each iteration, some new web services that can be invoked using gottenPara are found. At some point, if gottenPara $\supseteq$ out(r), then it means that using the parameters gathered so far, one can get the desired output parameters in out(r), thus finding the web services invocation layers with the least GILN satisfying the predication *LayeredlySatisfy*.

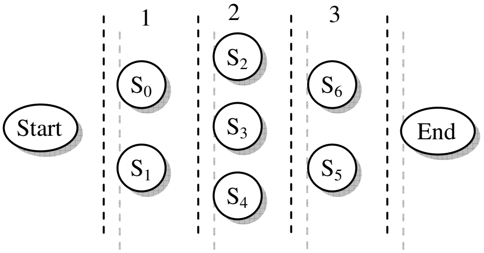


Fig. 2. Invocation layers returned by GetInvocationLayer

For instance, now there is a request r as r=<{a},{f}> and in set WS, a fragment of relevant web services as following: s_0=<{a},{b,c}, s_1=<{a},{g}>, s_2=<{b},{d}>, s_3=<{b},{e}>, s_4=<{g},{h}>, s_5=<{d,e,c},{f}>, s_6=<{a,h},{k}>. Then the algorithm *GetILS* gets the invocation layers as Fig.2.

3.1.1 Analyzing Algorithm *GetILS*

Theorem 3.1 (Termination). *GetILS* will terminate at some point.

Proof. For any given service request r∈ RQ:

1) If r can be satisfied by some composition of several available atomic web services. Since there are only finite number of web services, and each of iteration of while loop adds only "new" set of web services, the condition of gottenPara ⊒out(r) must be satisfied at some point. Then the iteration must end, so the algorithm will terminate.
2) If r can not be satisfied by some composition of several available atomic web services. From condition b) of Definition2.5 for *LayeredlySatisfy*, we can find that the transition between Layer i-1 and i ($S_{i-1} \Rightarrow S_i$) is a partial order relationship, and the greatest lower bound (glb) is in(r) and the least upper bound (lub) is in(r)∪out(r). Meanwhile, the transition relationship between invocation layers is monotonic, and therefore, as Knaster-Tarski Theorem [5] implies, there always exists a fix point, ensuring that after this point, gottenPara will not change. That also means that S will not change, then if sentence of Line 5.2) of the algorithm will stand, causing the algorithm to return. □

Theorem 3.2 (Least GILN) If the input service request can be satisfied by composing existing web services, then *GetILS* can get the ILS $S_1, \ldots, S_n$ satisfying $LayeredlySatisfy((S_1,S_2,...,S_n),r)$ and with the least GILN.

Proof. The former half part of Theorem can be proved by the exit condition of while sentence in Line5. Next, we will proof the latter half part of theorem using counter-evidence. Let $S_1',\ldots,S_m'$ be another ILS of r. That is to say, LayeredlySatify((S_1', S_2',…,S_m'),r) stands and m<n. According to the iteration process of *GetILS*, it will return after the n-th iteration, which is contradicted with the fact that algorithm will return at the m-th iteration. So $S_1, \ldots, S_n$ is with the least GILN. □

3.2 Further Optimization

The *ILS* $S_1,S_2,\ldots,S_n$ returned by *GetILS* may include some web services which have no contributions to the service request. In order to delete these useless web services, we must optimize the ILS further. So, in the second phase of our approach, we perform dataflow analysis to remove unnecessary web services from each invocation layer. The optimization algorithm begins with set of outputs and finds all the web services that generate at least one output in the set of outputs. Next, the inputs to the selected web services are added to the set of outputs and the process is repeated till no more web services are needed. The web services in the resulting invocation layers are the ones contributed to the service request.

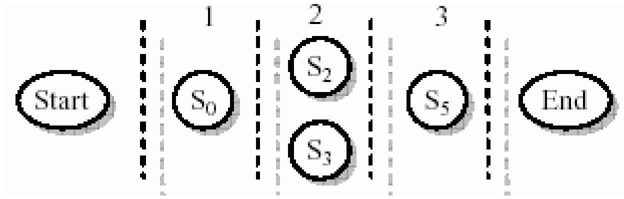


Fig. 3. Invocation layers after optimization

Algorithm ILSOptimization *(input: invocation layers generated by GetILS layer; output: invocation layers after optimization layer)*

```
 1) outputs←out(r)
 2) for i=n to 1 do
 2.1) for each ws∈layer[i] do
         2.1.1) if out(ws)∩ outputs=∅
         2.1.2) then layer[i] ←layer[i] /{ws}
 2.2) for each ws∈layer[i] do
       2.2.1) outputs← outputs ∪ in(ws)
3) return layer
```

After applying algorithm *ILSOptimization* on variable layer (cf.Fig.2) returned by *GetILS*, we get the invocation layers in Fig3.

3.3 Optimal Path Search

We can get an invocation path by selecting the minimal set of web services from each invocation layer returned by *ILSOptimization*. If *LayeredlySatisfy* ((S_1,S_2,…,S_n),r), and at i-th layer, there are m_i web services that can be invoked. When we consider invoke web services in parallel or sequentially, then there are $2^{m_i}-1$ search choices at this layer, leading to $(2^{m_1}-1)\ldots(2^{m_n}-1)$ search paths. So, an effective search algorithm is mandatory. In this paper, we propose to use A* procedure [6].

A* procedure is heuristics-based branch-bound search algorithm, with an estimate of remaining distance, combined with the dynamic-programming principle. The heuristics function of A* algorithm is based on the guesses about distances remaining as well as facts about distances already accumulated. It is comprised into two parts as: u(total path length) = d(already traveled)+u(distance remaining),where d(already traveled) is the known distance already traveled and u(distance remaining) is an estimate of the distance remaining. Since the performance of A* algorithm heavily depends on the quality of the heuristics function, it is important to use the right heuristics to strike a good balance between accuracy and speed.

Definition 3.1 (Heuristics Function). *Given some candidate sets of web services S* $(S \subseteq layer[i])$ *to visit next at Layer i, we design the heuristics function h as h(S)=d(S)+u(S), where d(S) represents the set of available parameters and u(S) represents the set of remaining parameters of out(r). Let output(S) = {s | s is output parameter generated by the visited web services until S in the current search path}. We define d(S) and u(S) as follows:*

d(S) = | in(r) ∪ output (S) | and u(S) = | out(r)/output (S) |

The pseudo code of our search algorithm base on A*search idea is shown as follows. G is the adjacency-list representation of the graph generated by algorithm *GetILS*, whose vertices of layer i are the subsets of variable layer[i] except for ∅ and edges are from one vertex of layer i to each of the next layer i+1 and the root node of G is start.

Algorithm HeuristicsBasedSearch *(Input: service request r, invocation layers layer, heuristics functions d and u; Output: the optimal path π)*

```
1) Initialize OPEN list
2) Initialize CLOSED list
3) Add start node to the OPEN list
4) while the OPEN list is not empty do
4.1) Get node S off the OPEN list with the lowest h(S)
    4.2) Add S to the CLOSED list
    4.3) if $d(S) \supseteq out(r)$
    4.3.1) then return the path from the start node to S according to the function $\pi$
    4.4) for each $S' \in Adj[S]$ do
        4.4.1) $\pi[S'] \leftarrow S$
        4.4.2) $d(S') \leftarrow d(S) \cup ( \bigcup_{ws \in S'} \mathrm{out(ws)})$
        4.4.3) $h(S') \leftarrow d(S') + u(S')$
    4.4.4) if S' is on the OPEN list and the existing one is as good or better
            4.4.4.1) then discard S' and continue
    4.4.5) if S' is on the CLOSED list and the existing one is as good or better
            4.4.5.1) then discard S' and continue
    4.4.6) Remove occurrences of S' from OPEN and CLOSED list
    4.4.7) Add S' to the OPEN list
5) return failure
```

4 Implementation Issues

When implementing the two algorithms above, there are many operations of sets occurring frequently, among which are subset judgment, union, intersection and difference operation. Their implementation efficiency is vital to that of whole algorithm. The key of all these operation is to solve the implementation of membership checking. In this paper, we propose to use Bloom Filter to finish the membership checking operations.

A Bloom Filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. The space efficiency is achieved at the cost of a small probability of false positives, but often this is a convenient trade-off. Therefore, Bloom Filters have received little attention in the theoretical community. In contrast, for practical applications the price of a constant false positive probability may well be worthwhile to reduce the necessary space. It was invented by Burton Bloom in 1970 [4]. Broder in [1] presents a plethora of recent uses of Bloom Filters in a variety of network contexts, with the aim of making these ideas available to a wider community and the hope of inspiring new applications.

A Bloom Filter for representing a set $S=\{x_1,x_2,\ldots,x_n\}$ of n elements is described by an array of m bits, initially all set to 0. A Bloom Filter uses k independent hash functions $h_1,\ldots,h_k$ with range. We make the natural assumption that these hash functions map each item in {1,…,m} the universe to a random number uniform over the range {1,…,m} for mathematical convenience. (In practice, reasonable hash functions appear to behave adequately, e.g. [2].) For each element $x \in S$, the bits $h_i(x)$ are set to 1 for i $(1 \le i \le k)$. A location can be set to 1 multiple times, but only the first change has an effect. Fig.4 gives Bloom Filters example with three hash functions.

To check if an item y is in S, we check whether all hi(y) are set to 1. If not, then clearly y is not a member of S. If all hi(y) are set to 1, we assume that y is in S, although we are wrong with some probability. Hence a Bloom Filter may yield a false positive, where it suggests that an element y is in S even though it is not. For many applications, false positives may be acceptable as long as their probability is sufficiently small.

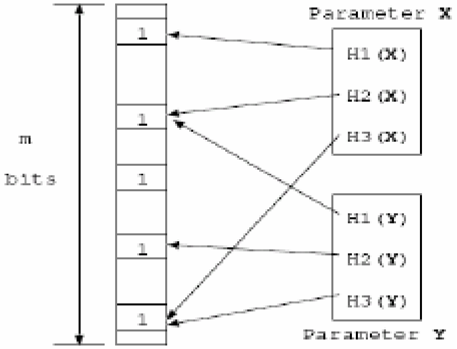


Fig. 4. Bloom Filters with three hash functions

The salient feature of Bloom filters is that the probability of a false positive for an element not in the set, or the false positive rate, can be calculated in a straightforward fashion, given our assumption that hash functions are perfectly random. After all the elements of S are hashed into the Bloom Filter, the probability that a specific bit is still 0 is $(1-(1/m))^{kn}$, hence the probability of a false positive in this situation is $(1-(1-(1/m))^{kn})^k \approx (1-e^{kn/m})^k$, the right hand side is minimized for k=ln2×m/n, in which case it becomes $(1/2)^k = (0.6125)^{m/n}$ .In fact, k must be an integer and in practice we might chose a value less than optimal to reduce computational overhead.

5 Related Works

Service composition is an exciting area which has received a significant amount of interest in the last period. Initial approaches to web service composition [7] used a simple forward chaining technique which results in the discovery of large numbers of services. There is a good body of work which tries to address the service composition problem by using planning techniques based either on theorem proving (e.g., Golog

[9, 10] and SWORD [11]) or on hierarchical task planning (e.g., SHOP-2 [12]). The advantage of this kind of approaches is that complex constructs like loops (Golog) or processes (SHOP-2) can be handled. All these approaches assume that the relevant service descriptions are initially loaded into the reasoning engine and that no discovery is performed during composition.

Recently, Lassila[8] has addressed the problem of interleaving discovery and integration in more detail, which is also our goal in this paper, but he has considered only simple workflows where services have one input and one output.

6 Conclusions

This paper studies how web services are composed to provide more complicated services. We propose the algorithms based on the concept of invocation layer to get the least invocation layers of candidate web services to satisfy the given service request. Thee algorithms have been applied to IntelliFlow system prototype developed at CIT to find web services composition setup.

The idea presented in this paper can be extended in future from different points of view. We are interested in solving the problem when specific costs such as time and money are important. Weighted graphs might be a good option to address the problem for these particular issues. As another extension, empowering the approach to support pre-conditions and post-conditions as part of the request is one of our future plans. This will help in specifying more accurate queries and providing more accurate results. The main idea can also be extended to the composition of general software services or even components. If we can somehow extract the required information (inputs, outputs, input-output dependencies) for each available component, the same approach could be used for other types of software services and components as well. This would be considered as another strength of the proposed method.

One assumption in our paper is that the parameters having same name (properties in the case of DAML-S [18] or strings in the case of WSDL [5]) have same types, which simplifies our composition setup algorithm. We will consider the type-compatible web services composition in next research plan.

References

1. A. Broder and M. Mitzenmacher, Network Applications of Bloom Filters: A Survey. Proc. of the 40th Annual Allerton Conference on Communication, Control, and Computing, pages 636-646, 2002.
2. M.V.Ramakrishna, Practical performance of Bloom Filters and parallel free-text searching, Communications of the ACM, 32(10):1237-1239, October 1989.
3. E. Christensen, F. Curbera, G. Meredith, and S.Weerawarana, 2001, "web services Description Language (WSDL) 1.1, W3C Note": www.w3.org/TR/wsdl
4. B. Bloom. Space/time tradeoffs in hash coding with allowable errors. CACM, 13(7):422-426, 1970.
5. A. Tarski. “A Lattice-Theoretical Fixpoint Theorem and its Applications”. Pacific J. Math., 5:285–309, 1955.

6. S. J. Russell and P. Norvig. “Artificial Intelligence: A Modern Approach (2nd Ed.)”. Prentice-Hall, 2002.
7. S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration, pages 1–7, Edmonton, Alberta, Canada, July 2002.
8. O. Lassila and S. Dixit. Interleaving discovery and composition for simple workflows. In Semantic Web Services, 2004 AAAI Spring Symposium Series, 2004.
9. S. McIlraith, T. Son, and H. Zeng. Mobilizing the semantic web with daml-enabled web services. In Proc. Second International Workshop on the Semantic Web (SemWeb-2001), Hongkong,China, May 2001.
10. S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D.McGuinness, andM.-A.Williams, editors, Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02), pages 482-496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.
11. S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In 11th World Wide Web Conference (Web Engineering Track), 2002.
12. D.Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, October 2003.
13. D.-S. C. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila,D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the Semantic Web. Lecture Notes in Computer Science, 2342, 2002.

Self Healing and Self Configuration in a WSRF Grid Environment

Michael Messig and Andrzej Goscinski

School of Information Technology,Deakin University,
Geelong, Vic 3216, Australia
{messig, ang}@deakin.edu.au

Abstract. The move towards web services in Grid computing requires mechanisms for services to maintain state. This is introduced by the Web Services Resource Framework which provides a basis for web services to access stateful resources. While this allows access to stateful resources, the web services themselves are not stateful. Currently, Grids require a lot of direct involvement of application developers, who are, in general, not computing specialists. The principles of autonomic computing introduce characteristics which are aimed at automatic improvement of computing systems and can be applied to the Grid. This paper addresses the principles of self healing and self configuration in a Grid environment and implements a service using the WSRF.NET framework to investigate the affect and applicability of the Web Services Resource Framework on these principles and improve the WSRF specification.

1 Introduction

The evolution of Grid computing primarily focuses on heterogeneity and interoperability to provide a system which can share resources and services among disparate platforms [1]. Since the Grid's inception, the ability to provide heterogeneous, distributed computing has been a key goal to the acceptance of the Grid. Original implementations of Grid applications and middleware were developed using languages and tools which offered support for multiple architectures, however interoperability was not achieved due to the inflexible design of the underlying system. The introduction of web services for Grid computing improves interoperability by utilising open standards for describing, discovering and interacting with services. Web services by nature however do not provide the ability to maintain state and once a client's request is addressed by the web service, all knowledge of this interaction is lost. Therefore every interaction with a web service by different clients or multiple interactions by the same client have no native means to carry the state of the web service or the state of any resources the web service is using across client interactions. Web services are invoked by the client, therefore when the client calls a method or function provided by the web service, an instance of the web service is created and once the service completes the client's request, the instance of the web service is destroyed.

To offer flexible, interoperable services in a Grid environment, the ability to maintain state is desirable, especially for applications in high performance computing, industry and business where transaction based systems are required. In systems such as Globus

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 149 – 158, 2005.

this was offered by the Open Grid Services Infrastructure and with recent developments in web services a revision of this infrastructure to include new web service standards has resulted in the development of the Web Service Resource Framework (WSRF) [2]. The WSRF provides the ability for web services to maintain stateful information by defining parts of the web service, such as variables, data structures and classes as stateful resources, which are then stored in stateful storage, for example a database, between interactions with clients. The WSRF model allows web services to access stateful resources; however the web services themselves are not stateful [3].

The current move toward autonomic principles for computing is being investigated for their application within the Grid [4] and are addressed in cluster operating systems [5, 6]. The development of the Holos operating system proves that it is possible to provide autonomic principles at the operating system level [5, 6]. The aim of this project is to investigate the ability to apply autonomic principles to Grid computing which is moving towards a service oriented architecture, and more specifically web services as the model for applications within the Grid. In particular the principles of self healing and self configuration of web services in a Grid environment is being investigated. By providing the ability for services in a Grid to recover from problems and reconfigure itself to avoid such problems allows the system to be a dependable, robust and scalable platform which does not require complex maintenance [7]. This will also allow the system as a whole to adapt and dynamically change in response to events and changes within the system.

The aim of this paper, however, is to carry out a preliminary study and examination of the WSRF specification by creating a web service using an implementation of WSRF. Particular attention is paid to the architecture of WSRF, what is provided, the ability for WSRF to contribute to the autonomic principles of self healing and self configuration, and a possible improvement of the WSRF specification to adhere to these principles. The platform used is the WSRF.NET implementation, which is written for the Microsoft .Net environment and extends C# and ASP.Net web services to incorporate stateful resources [8]. An auction service is constructed using the WSRF.NET toolkit which allows a client to bid on an item at auction. The web service exploits WSRF by allowing the current bid on the item to reflect a stateful resource and maintain state across client bids and is used to highlight the possibility of providing self healing and self configuration in a WSRF Grid environment.

The report is structured as follows. In section 2 we examine stateful resources in Grid applications. In section 3 the logical design of the auction service is proposed to highlight the need for state in a WSRF environment. Section 4 discusses the implementation of the auction service using the WSRF.NET platform. Section 5 discusses the experiences with using the framework and provides testing of the service. Finally, section 6 concludes this study and presents future work.

2 Stateful and Stateless Resources

Web services are applications which support standardized, interoperable interaction over a network using well defined interfaces and messaging techniques which exploit XML [9]. The runtime environment responsible for hosting the web service is the application server. The application server is responsible for accepting requests from client applications, invoking the service and if necessary providing a response to the client [10]. Web services by nature are stateless services, that is, they do not natively provide any mechanisms to maintain the state of the resources they are using, or the state of the service itself. The Web Services Resource Framework introduces a standard which provides the ability for web services to access stateful resources. A stateful resource, in terms of the WSRF, is defined as a resource which has a specific set of state data, has a well defined lifecycle and is known to and acted upon by a web service [10]. A resource can be any system component, such as objects, files, databases even printers or groups of other resources [10].

The WSRF's view on stateful resources implies that the web service representing the stateful resource to the client is still regarded as a stateless service and merely delegates responsibility of managing the stateful resource to another component or even the resource itself. By taking the approach where the web service itself remains stateless and the resource maintains its state, the location of the stateful resource must be provided either explicitly by the client or implicitly by a known location or system component known to the web service [10]. The WSRF uses the WS-Addressing standard for this.

The WSRF takes advantage of several standards to provide mechanisms for web services to access stateful resources. The WS-Addressing standard is used to describe a web service in terms of its address, called an End Point Reference (EPR). The WS-Addressing specification also allows additional properties to be described within its XML schema which is utilized by the WSRF which stores information about where the stateful resource is located within the XML schema [11]. The client specifies an EPR when invoking the web service, allowing the web service to locate the stateful resource. This is simply providing the address of a resource to the web service; the WSRF provides the web service with an interface to the resource allowing it to access and manipulate the resource on behalf of the client's request [10].

Whilst the WSRF introduces a standard method for accessing stateful resources, this can be achieved by any web service by using database connectors to access a database, store the data in a file or any other stateful storage mechanism. The WSRF extends the web service model to provide a simple way of accessing the resource while hiding the underlying mechanisms required access the resource and presenting simple functions to the user. The WSRF also provides a standard for describing the data type of the resource or the resource's interface in the web service's interface description document (WSDL) and providing the address of the resource through an EPR.

Autonomic principles are focused on providing system wide mechanisms for increasing reliability, scalability and robustness of systems and are seen as the next step in the evolution of computing [7]. Systems such as Holos address these principles at the operating system level and promote autonomic principles being applied to the system as a whole [6]. To provide support for these principles in a Grid environment, the system must be viewed as a whole, where individual components interact with each other and providing mechanisms such as self healing and self configuration must apply to all components across the entire system. The WSRF implies that WSRF resources are responsible for managing themselves, therefore providing self healing or self configuration properties for these resources is the responsibility of the resource itself. In an autonomic Grid environment, the system as a whole should address these principles rather than individual components.

3 Logical Design of an Auction Service

To examine the design of the WSRF an auction service was developed to accept bids on an item from clients. This service uses the WSRF to maintain the state of the client's bid, allowing multiple clients to bid and outbid each other on items. The auction service must be implemented as a web service and will have a single attribute, the bid value, which is defined as a WSRF Resource. The service provider is responsible for hosting the service, this includes an application server which contains the web service and is responsible for its execution and a WSRF Resource. The location of the actual WSRF resource does not necessarily have to physically be on the same service provider as the auction service however is represented as such for simplicity. The interaction between the client, the auction service and the WSRF Resource is shown in Figure 1.

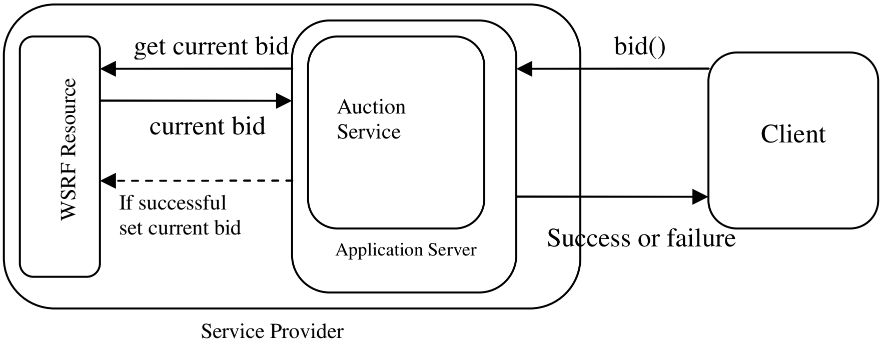


Fig. 1. Logical design of a WSRF Service

The client is able to invoke the auction service's bid method, which is the only publicly accessible method. The web service advertises this method in the WSDL document of the web service and provides an End Point Reference (EPR) to specify the location of the WSRF Resource it requires. When the client invokes the auction service, the auction service connects to the WSRF resource, retrieves the value for the current bid by using the EPR, checks the value and either returns false in the case where the bid is not a valid bid (i.e. bid <= current bid), or true, in which case the client's bid is accepted and the current bid value is set to the client's bid and stored back as a WSRF Resource.

Autonomic principles of self healing and self configuration can be applied to the WSRF web services model. By doing so and applying these principles system wide, reliability of the system is increased. Although the actual WSRF resource used by the auction service may be made reliable due to the underlying storage system used to store the value of the current bid, for example a cluster of distributed databases, the web service itself is still vulnerable to failure. For example, if the client is interacting with the auction service and before the auction service is able to save the state of the WSRF resource the service provider fails, the client's bid is unsuccessful. In a business environment, this could introduce service reliability issues, or legal problems involved in bidding systems. Although this is a simple example, and the auction service is not used to fully justify the incorporation of autonomic principles for web services, more complex services can apply these constructs in more applicable areas, such as high performance computing environments, mission critical applications, business or industry applications. Web services may perform a large number of operations between interactions with clients and if a failure occurs during this time, important information or processing time may be lost.

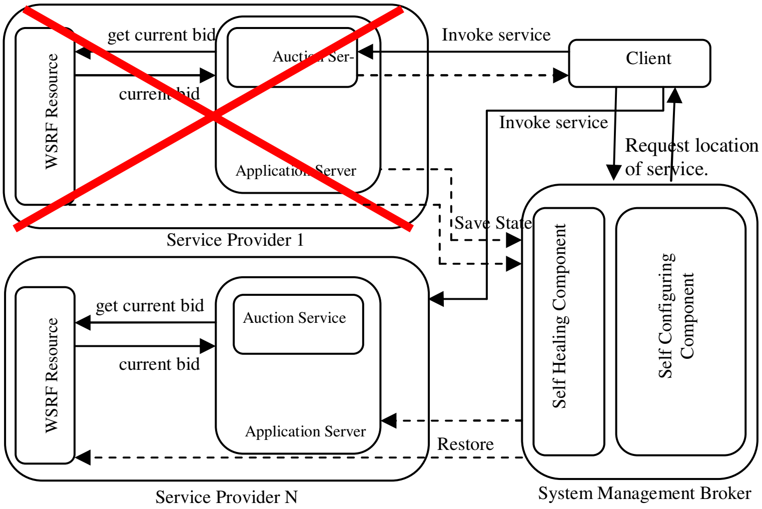


Fig. 2. Self healing and self configuration in a WSRF environment

To support self healing and self configuration in a WSRF environment, the auction service would have to create saved states of the service as well as the resources it is using at specific intervals. Therefore if the service fails it can be resumed from the last saved state and all interactions with resources restored. The information saved about a service must be stored in persistent storage to ensure the information is not susceptible to failure. In terms of providing a facility to support self configuration, this save state can be used to move the service from one machine to another within the Grid, if for example, the machine is being decommissioned, indefinitely fails, or to remove bottlenecks and increase performance of the Grid as a whole. Theoretically it would be possible to save an image of the entire service provider, thus preserving the web service and all of its interactions with other services and resources, however there would be a very large overhead associated with saving, updating and restoring an image of an entire service provider at given intervals. It is possible to introduce a protocol where the service providers notify a broker of their state. Therefore as the state of the web service or the WSRF resource changes, or at specific intervals, the service provider can update the state information with the broker. This provides a snapshot of the service at a given time and will allow a recreation of the service from this information, and only requires saving, updating and restoring a footprint of the service, rather than the entire provider. The frequency of taking a snapshot of a service should be configurable as it is a parameter which changes with each service depending on the service's requirements or quality of service agreements between the service and the client.

As shown in Figure 2, it is possible to reconstruct the state of the auction service by using a broker to manage the state of both the service and the resources which the service is interacting with. Service providers within the system must periodically provide the broker with adequate information about each service's state and interactions with other services and resources allowing any of these services to be reconstructed from this information.

Instead of directly accessing the auction service, the client initially submits a request to the system management broker. The broker then provides the client with the address of the auction service. The client can then interact with the auction service directly, however if the auction service fails and is unreachable, as Service Provider 1 in Figure 2, the client then forwards its request to the system management broker. The system management broker is then responsible for providing the address of the auction service. This address may locate a copy of the original auction service which has been reconstructed from the saved states of the original auction service, or an address which locates the original auction service which has been moved to an alternate trusted location due to some reconfiguration of the system. The system management broker includes mechanisms to discover services which have failed by polling services known to the broker at intervals to determine if they are available and responsive. The broker reconstructs failed services either at the same location as it was previously executing, or at a different location. Doing so renders the service unreachable by clients and therefore the client must make a request to the broker for the service's new location, ensuring that the client will always have the correct and most up to date location of the service.

By providing the ability for services to save their state and not only the state of the resources the service is using, reliability is improved. Services are able to provide a reliable, robust service to clients in environments where a service's interaction with clients is critical, in the case of the auction service, or where services may require large amounts of processing time between interacting with clients and stateful resources. Introducing the principles of self healing and self configuration for Grid environments is possible and the system management broker provides the ability to offer these principles to the Grid as a whole. While the WSRF provides access to stateful resources by web services, the systems responsible for providing the resources are also responsible for their reliability. The Grid should provide system wide support for self healing and self configuration rather than each component in the system. This allows services and resources to be resumed when components within the Grid fail or are moved when the system must be reconfigured to adapt to change. By introducing a system management broker responsible for self healing and self configuration, this is achieved.

4 Implementation

The auction service is implemented using the WSRF.NET implementation of the web services resource framework. WSRF.NET is developed for the Microsoft Visual

Studio .NET environment and uses the IIS application server to execute web services. The WSRF.NET extends the web services model offered by the Microsoft .NET languages by providing classes, methods and attributes for the web service resource framework in the .Net environment [8]. The WSRF.NET implementation uses the Apache Xindice database server to store data related resources, for example, variables, structures and classes which are serializable and converted into XML then stored in the database.

To allow WSRF resources to be used in within Microsoft .Net web services, several attributes are used to identify resources and methods which will be used to create the stateful resources. The auction service is implemented as a web service which accepts a bid from a client application. The bid is compared with the current highest bid (current bid) and if the client's bid is higher, the current bid is updated to reflect the client's bid and the service returns true, otherwise the service returns false.

The single resource which is used for the auction service is the current bid variable which holds the value for the current highest bid. The current bid variable is attributed with the [Resource] attribute to signify that it is a WSRF resource as specified by the WSRF.NET developers guide [12]. The auction service implements several methods which are required by the language and the WSRF.NET implementation. These methods form the constructor for the auction service as well as some initialization methods. The auction service however contains only a single method available through its interface to clients:

```
public bool bid(int clientBid)
```

As previously discussed, the bid method is responsible for returning true or false based on the clients bid. Within the bid method however, the auction service must access the WSRF resource. This is done by using the *get {}* and *set {}* attributes to retrieve and store the values of the resource to and from stateful storage. The interaction with the underlying database is transparent and handled by the WSRF.NET implementation.

To allow the client application to exploit the WSRF.NET auction service, it requires the ability to get resource properties from the auction service, create end point references and access the auction service's bid method via SOAP. The WSRF.NET implementation allows the majority of this functionality to remain hidden from the developer. The client application is written in C# and includes the libraries required by WSRF.NET. The client application implements a single function, itembid() which is responsible for connecting to the web service, placing a bid on behalf of the client and printing the result of the bid. The itembid() function simply creates a proxy which is responsible for the connection to the auction service, calls the auction service's bid function and prints the result of the bid to the screen.

5 Testing

To test the implementation of the auction service and the ability for the WSRF.NET implementation to maintain stateful resources, several auctions were set up for client bids. The testing environment consisted of two machines, one being the service provider and the other executing the client applications. The first test involved starting a bid where only a single client invoked the auction service and executed several bid attempts, while the second test involved two client applications accessing the auction service in succession.

```
Test 1                                   Test 2
  Client 1:                                Client 1:
          Client bid: 25                           Client bid: 25
          Bid Successful                           Bid Successful
          Current Bid is $25                       Current Bid is $25
                                           Client 2:
          Client bid: 75                           Client bid: 10
          Bid Successful                           Bid Unsuccessful
          Current Bid is $75                       Current Bid is $25
                                                   Client bid: 35
          Client bid: 15                           Bid Successful
          Bid Unsuccessful                         Current Bid is $35
          Current Bid is $75               Client 1:
                                                   Client bid: 30
                                                   Bid Unsuccessful
                                                   Current Bid is $35
                                                   Client bid: $40
                                                   Bid Successful
                                                   Current Bid is $40
```

Fig. 3. Testing of the auction service

As can be seen in Figure 3, Test 1 shows the single client performing several bids and thus invoking the auction service several times. The client invokes the auction service with the itembid() method passing values of 25, 75 and 15. Each of these bids is an individual invocation of the auction service, showing the ability for the auction service to maintain the state of the current bid across interactions with the client. The client's bid is successful the first two times the auction service is invoked , this is due to the bid initially beginning at zero when the resource is initiated and as each bid is greater than the previous bid, the result is successful. The final bid however invokes the auction service with a bid that is less than the current bid and therefore is unsuccessful.

To test the WSRF.NET implementation to retain the state of the current bid resource across multiple clients, two clients were used to access the auction service, as shown in Test 2. The auction service is reinitialized to contain a current bid of zero. This test begins with client 1 initiating a bid which is successful as it is greater than zero. Client 2 attempts submit a bid to the auction service with a lower bid than client 1 has previously bid, as the current bid resource has already been updated to reflect the bid of client 1, the bid is unsuccessful. Client 2 then invokes the service with a larger bid, this time the bid is successful. Client 1 imitates this behavior by first attempting to invoke a bid of a smaller value than that of client 2, then a second successful bid which is greater than the bid of client 1.

The tests performed on the auction service show the ability of the WSRF.NET implementation to maintain the state of the current bid resource across client interactions. The WSRF.NET implementation transparently retrieves the current bid value from a database at each invocation of the service and saves the state of the current bid variable to a database at each successful client bid. While this provides the ability to use stateful resources, the web services, application servers and service providers have no means to provide state, or provide any reliability. If the underlying database makes no attempt to provide reliability, the system as a whole does not ensure the reliability of the WSRF resource. Therefore, without introducing any self healing or self configuration mechanisms to support the auction service, reliability cannot be guaranteed.

6 Conclusion and Future Work

Grid computing is moving towards web services as the architecture for applications within the system. By introducing web services, the problem of state is generated and to address this WSRF was developed to deal with the access of stateful resources by web services. The WSRF however, implies that the responsibility of managing stateful resources is that of the resource itself rather than the web service. By taking this approach, the WSRF does not provide a system wide approach to addressing reliability and application developers must be experts in the field of Grid computing to create reliable Grid applications.

The introduction of autonomic principles for Grid computing has addressed the ability for a Grid system to adapt and change as a whole, improving reliability and reducing the complexity of the system. The WSRF does not support the ability to provide reliability to the system as a whole, therefore the introduction of self healing and self configuration must be addressed. The logical design of a self healing and self configuration broker for web services in a WSRF Grid environment outlines the ability to provide reliable web services which are able to be manipulated by the system and adapt to change. The system management broker shows that it is possible to address the issue of stateless web services while providing reliable access to web services and stateful resources. The broker also highlights the ability to improve the current WSRF specification.

The development of an auction service exploiting the WSRF.NET implementation of the Web Services Resource Framework highlights the features of the WSRF specification as well as a need to address the reliability of web services and not simply WSRF resources. The introduction of autonomic principles in a WSRF Grid environment will provide a robust, reliable system which is able to dynamically adapt as the system changes.

Future work on autonomic principles for Grid computing would involve the investigation of new techniques to provide autonomic principles to web services. There are many ways in which self healing and self configuration can be applied holistically to web services and resources within the Grid. Additionally, there are other autonomic principles which may be investigated for their application in a web service environment and their affect on Grid computing.

References

1. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid - Enabling Scalable Virtual Organizations*, in *Grid Computing - Making the Global Infrastructure a Reality*, F. Berman, G.C. Fox, and A.J.G. Hey, Editors. 2001, John Wiley and Sons Ltd. p. 171-197.
2. Czajkowski, K., et al., *From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution*. 2004.
3. Czajkowski, K., et al., *The WS-Resource Framework Version 1.0*. 2004.
4. Messig, M., *Grids and Globus and Where to Now?* 2004, Deakin University: Geelong. TR C4/11.
5. Goscinski, A., J. Silcock, and M. Hobbs. *Building Autonomic Clusters: A Response to IBM's Autonomic Computing Challenge*. in *The 5th International Conference on Parallel Processing and Applied Mathematics*. 2003. Czestochowa, Poland: Springer-Verlag.
6. Goscinski, A., J. Silcock, and M. Hobbs. *Cluster Operating System Support for Parallel Autonomic Computing*. in *18th Annual ACM International Conference on Supercomputing*. 2004. Saint-Malo, France.
7. Horn, P., *Autonomic computing: IBM's Perspective on the State of Information Technology*. 2001.
8. Wasson, G. and M. Humphrey. *Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments*. in *International Conference of Parallel and Distributed Systems*. 2005. Denver, Colorado.
9. Booth, D., et al., eds. *Web Services Architecture W3C Working Draft*. 2003.
10. Foster, I., et al., *Modeling Stateful Resources with Web Services Version 1.1*. 2004.
11. Box, D., et al., *Web Services Addressing (WS-Addressing) W3C Member Submission*. 2004.
12. Morgan, M. and G. Wasson, *WSRF.NET Developer Tutorial*. 2005, University of Virginia.

Study on Life Cycle Model of Dynamic Composed Web Services*

Chen Yanping, Li Zengzhi, Jin Qinxue, and Wang Chuang

Department of Computer Science and Technology,
Xian Jiaotong University, 710049, China
yanping@tom.com, lzz@mail.xjtu.edu.cn

Abstract. Modern requirement of dynamic Web Services rely increasingly on composing concurrent, distributed, mobile, re-configurable and heterogenous services, and substantial progress has already been made towards composed Web Services. In this paper, first, we proposed a life cycle of composed Web services, then designed a model named Service-Cloud model based on the process of forming clouds in nature. Finally, based on Service-Cloud model, we design and implement a prototype.

Keywords: Service management, dynamic Web service composition, life cycle model of composed Web services.

1 Introduction

The real challenge in Web Services composition lies in how to provide a complete solution. This means to develop a tool that supports the entire life cycle of service composition, i.e., discovery, consistency checking and composition in terms of reuse and extendibility. This paper proposes a whole life cycle of composed Web Services, and then provides a Service-Cloud model, which provides a metaphor for composed Web Services to provide. Through the Service-Cloud model, we can not only describe the picture of composed Web Services better but also give the whole life cycle of composed Web Service.

2 Life Cycle of Composed Web Services

W3C provided a life cycle of Web Service, but dynamic composed Web Services are more complex than use pre-existing Web Services directly, and the courses of providing a composed Web Service are also different of providing a pre-existing Web Service. So, here based on the life cycle of Web Services in [2]. We present a whole life cycle of composed Web Services.

* This paper is Supported by National Natural Science Foundation of China (No. 90304006) and Research Fund for Doctoral Program of Higher Education of China (No.2002698018).

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 159 – 164, 2005.

States:

- getReq: the provider agent has accepted a request to provide a service.
- doReq: the provider agent does some process to fulfill the requests.
- done: the provider agent successfully completed the requests and return the results to request agent.
- failed: the provider agent encounter some errors and cannot fulfill the request, and return errors to request agent.

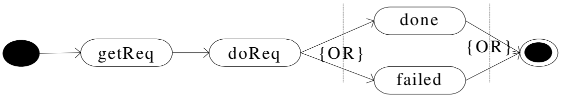


Fig. 1. State transition diagram of request processing of a composed web service

Transitions:

- A composed service starts getReq when it accepts a request.
- A composed service starts execution after it received a request.
- A composed service transitions to either done or failed state depending on the outcome of the doReq stage.
- A composed service exits doReq from either done or failed state.

Substrates transition diagram of doReq is given in figure 2.

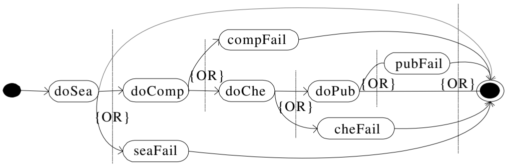


Fig. 2. Substate transition diagram of doReq

States:

- doSea: the provider agent is doing searching to fulfill the requests.
- doComp: the provider agent is doing composition to fulfill the requests.
- doChe: the provider agent is doing checking to meet the requests.
- doPub: the provider agent is doing publication for reuse.
- seaFail: the provider agent encountered a searching error and didn't complete the requested functions, returning a searching error to the request agent.
- compFail: the provider agent encountered a composing error and didn't complete the composition, returning a composing error to the request agent.
- cheFail: the provider agent encountered a checking error and didn't complete the requested functions, returning a checking error to the request agent.
- pubFail: the provider agent encountered a publishing error and didn't complete the publication, returning a publishing error to the request agent.

Transitions:

- A composed service starts execution doSea after it accepts a request.
- A composed service transitions to doComp, compFail, or doSea depending on the outcome of the doSea stage.
- A composed service transitions to either doChe or compFail depending on the outcome of the doComp stage.
- A composed service transitions to either doPub or cheFail depending on the outcome of the doChe stage.
- A composed service exists doReq from doPub, failed, or doSea state which are mutually exclusive.

3 Metaphorizing Composed Web Services into Clouds

There are following reasons we metaphorize the Service-Cloud model into clouds in nature.° In conclusion: the aim of Service-Cloud model is to describe the all phases in a composed Web Services life cycle. This model possesses the ability to rapidly and autonomously adapt even to change situations that were not envisioned during the design time and keeping the running software system constantly available to users, and also makes the creation, reusing, and deployment become even simpler.

Based on the above features of clouds we present a Service-Cloud model. In this Service-Cloud model, the way to compose Web Services is similar to the process of forming clouds on the following five aspects:

- Pre-existing services in Internet corresponds to water in the earth,
- Discovery the needed services from Internet corresponds to the course of water evaporation,
- Web Services in a composed service corresponds to the water drips in a cloud,
- Composition logic of composed service corresponds to the course of service drips augment, and
- The way of decomposition a Web Service into several element services is similar to rains in nature.

4 The Service-Cloud Model

4.1 Types of Services

In our Research, Service-Cloud distinguishes two types of services: service drips and service clouds:

- Service Drip. A service drip is an individual accessible Internet application that provides some functions by itself. An example of a service drip might be a Web flight-verification interface in a travel-mark information system. The details about Service Drip are given in [3,4] in this paper we only give details about Service Cloud.
- Service Cloud. The concept of service cloud is a solution to the problem of dynamic composing a potentially large number of Web Services. At runtime, when a provider

agent receives a request for a new Web Service, then it can composing several pre-existing services to fulfill the requirements.

4.2 Service Cloud

In the Service-Cloud model, the composed service is named as a service cloud. The service cloud is a service container, and it can provide both a whole application and parts of its functions through a standard interface.

4.2.1 Characters

Specially, a service cloud also can be looked as a special service drip when it is being used. So, a service cloud has all the characters of the service drip, and a Service cloud also has the following features besides the characters of a service drip:

- A service cloud can be dynamically composed at run time,
- A service cloud is mainly created for an unanticipated and critical requests,
- A service cloud can be decomposed into several service drips when needed.

4.2.2 Description

A reusable service cloud is a container of certain functions and management, and it includes three main parts: input functions, output functions, and internal composition logic.

Definition 1. Let *Service Cloud* has the form $C(i) = (F_{in}(i), L(i), S(i), F_{out}(i))$, Where: F_{in} is a set of input functions, which include parameters got from user directed, results from other service drips (clouds), or required functions from other service drips (clouds), L is a set of internal composition logic (e.g. state Chart, TLA), the logic ensures the composition, S is the state of this service cloud, and the value of S is one of $\{0,1,2\}$, where:*0:* represents the service is idleness; *1:* represents the service is working correct at present,*2*: represents the service run into a wrong state. This parameter will be changed with the perform instances of included service drips, F_{out} is a set of output functions, which include output to user directed, and results to other clouds.

4.3 Service Cloud Generator

There two kinds of service cloud generators: Forward Service Cloud Generator and Backward Service Cloud Generator.

- Forward service cloud generator (FCG)

Forward service cloud generator is a black box, which inputs are divided into three types: static input parameters, dynamic input parameters, and an composition logic, and FCG outputs are only dynamic parameters. As to user, they need not to understand the details in the generator, and they care only for the outputs. Noted, the inputs and outputs are not only parameter values, but also service drips and service clouds.

- Backward service cloud generator (BCG)

Backward service cloud generator can decompose a complex service into several outputs according to decomposition logic, and the BCG is more complex than FCG. FCG and BCG can describe the reversible relationship between service cloud and service drips. Actually, our researches are emphasis on the designing and implementing the FCG at present because the BCG is more complex than the FCG.

Figure 3 gives the graphical description of the forward and backward cloud generator.

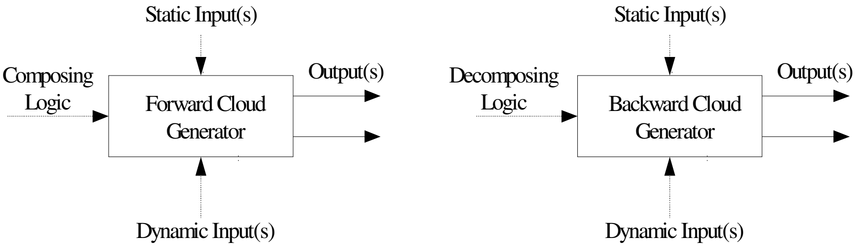


Fig. 3. Graphical description of the forward & backward cloud generator

Definition 2. Let ***Forward Service Cloud Generator FCG*** has the form**:**

$$FCG\colon\ X \to Y$$

$$X \in \left\{ (P\_sin, P\_din, P\_lin) \middle| \ P\_sin \in \bigcup_{i=i1}^{i2} Psin^{i}, P\_din \in \bigcup_{j=j1}^{j2} Pdin^{j}, P\_lin \in \bigcup_{k=k1}^{k2} Plin^{k} \right\}$$

$$Y \in \left\{ p\_dout \middle| \ P\_dout \in \bigcup_{l=l1}^{l2} Pdout^{l} \right\}$$

Where: P_sin is a set of static inputs, $P\_din(dout)$ is a set of dynamic inputs (outputs), P_lin is a set of input logic.

4.4 Composition Logic

Composition logic presents how the services functions can be synchronized and coordinated combined. Composition logic is beyond the conversation logic (which is modeled as a sequence of interactions between two services) and become a sound basis for expressing the business logic that underlies business applications.

Definition 3. The *composition depth* has direct correlation with the composition length of the longest composition route.

As clouds have high-cloud, mid-cloud, and low-cloud, there are also three kinds of composition levels

- High-level composition. There should exist at least one route which composition depth is no less than two.

- Mid-level composition. There should exist at least one route which composition depth is no less than one and less than two.
- Low-level composition. Users use the service directly without composition. The composition depth is zero.

5 Prototype Service-Cloud Based on SMN

In this section, we give a prototype of Service-Cloud based on SMN. This implementation has shown that the ideas behind Service-Cloud fit together, are consistent with one another, and are realizable using existing technologies.

The SMN (Service Management Network) is built on the Internet. The aim of SMN is to accomplish the following key functions of services: apply, create, run, supervise, and edit. SMN is composed of Service Proxies (SPs), Service Controllers (SCs), Service Management System (SMS), Service Create Environment (SCE), and Database etc. SMN offers five main functions: service logic management, service data management, user data management, service performance management and service quantity management, in addition, SMN maintenance a central DB. SMS communicate with Distributed Service Control Point (DSCP) and proxies, this implements the service management. Distributed Service Creation Environment (DSCE) permits manager to create a new service according to certain rules, after been tested, the new service will be imported into SMS and deployed by DSCP. Our fundamental ideas behind the Service-Cloud model are: SMN can realize rapid and auto adaptation even to changes that were not envisioned during design time. In SMN, the Service Create Environment (SCE) is used to create new services, and SCE based on Service-Cloud model prototype architecture is composed of a registry, service cloud generators, and service drips pool.

6 Conclusions

In this paper, we present a whole life cycle of composed Web services, and based on this whole life cycle, we present a Service-Cloud model by metaphorizing discovery, compose, publish, and terminate into evaporate, adhere, augment, and rain separately. We also give a way to plan dynamic compositions in Service-Cloud model. Finally, a prototype based on the concepts of Service-Cloud model is given.

Reference

[1] Jian Yang and Mike. P. Papazoglou, Web Component: A Substrate for Web Service Reuse and Composition, Lecture Notes in Computer Science, Vol. 2348, p21-36, Springer, 2002.

[2] Web Service Management: Service Life cycle. http://www.w3.org/TR/2004/NOTE-wslc-20040211/.

[3] Chen Yanping, etc. Service-Cloud Model of Composed Web Services. In proc. of ICITA 2005. In press.

[4] Chen Yanping, etc. A Whole Life Cycle Model to Dynamic Composed Web Services In proc. of ICMLC 2005. In press.

Fault-Tolerant Dynamic Job Scheduling Policy

J.H. Abawajy

School of Information Technology,
Deakin University, Geelong, VIC., Australia

Abstract. In this paper, we propose a scalable and fault-tolerant job scheduling framework for grid computing. The proposed framework loosely couples a dynamic job scheduling approach with the hybrid replications approach to schedule jobs efficiently while at the same time providing fault-tolerance. The novelty of the proposed framework is that it uses passive replication approach under high system load and active replication approach under low system loads. The switch between these two replication methods is also done dynamically and transparently.

1 Introduction

The Grid [5] offers scientists and engineering communities high performance computational resources in a seamless virtual organization (VO) capable of running the most demanding scientific and engineering applications required by researchers and businesses today. However, a number of major technical hurdles must be overcome before this potential can be realized. One of the main problems to be addressed is that of efficient Grid jobs scheduling. A critical aspect of any distributed processing system is the algorithm that maps jobs to resources. Poor scheduling can leave most of the grid resources sitting idle while one bottleneck application is performed.

A wide variety of scheduling approaches for grid computing are currently available. While they all offer capabilities for resource allocation and distribution, they do not support integrated dynamic scheduling and fault-tolerance processing of Grid applications. Also, all these systems use static scheduling policy whereas we focus here on the dynamic fault-tolerant scheduling approach. With the momentum gaining for grid computing systems and as grids are increasingly used for applications requiring high levels of performance and reliability, the ability to tolerate failures while effectively exploiting the variably sized pools of grid computing resources in an scalable and transparent manner must be an integral part of grid computing systems [7],[19], [6], [2], [16].

In this paper, we propose a fault-tolerant dynamic scheduling policy that loosely couples dynamic job scheduling with job replication scheme such that jobs are efficiently and reliably executed. The novelty of the proposed algorithm is that it employs a hierarchical scheduler as well as hybrid replications approach to schedule jobs efficiently while at the same time providing fault-tolerance to the grid applications. A hierarchical scheduler is used to match a user's job requirements against grid resources at available grid sites, efficiently balancing

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 165–173, 2005.

the system load and provide scalability as well as fault-tolerance. The algorithm uses passive replication approach under high system load and active replication approach under low system loads. The switch between these two replication methods is also done dynamically and transparently.

The rest of the paper is organized as follows. In Section 2, a formal definition of the fault-tolerant scheduling problem is given. This section also establishes the fact that, to a large extent, the problem considered in this paper has not been fully addressed in the literature. Section 3 presents the proposed fault-tolerant scheduling policy. Preliminary performance results of the proposed algorithm is disucssed in Section 4. Finally, the conclusion and future directions are presented in Section 5.

2 Problem Statement and Related Work

2.1 Problem Statement

The fault-tolerant scheduling problem (FTSP) addressed in this paper can be formally stated as shown in Figure 1. Devising a proper schedule to satisfy a set of constraints is fundamental to effective utilization of grids resources, efficient resource sharing, and improved user job response time. However, the problem of scheduling parallel jobs on a set of nodes is NP Complete problem and heuristics are commonly used to solve it.

Given: A set of n jobs, $\mathbf{J}=\{J_1, ..., J_n\}$, where each job, J_i, arrives in a stochastic manner into a system composed of m independent clusters, $\mathbf{S}= \{C_1,...,C_m\}$.

1. Each job, J_i, can be decomposed into t tasks, $\mathbf{T}=\{T_1,...,T_t\}$. Each task T_i executes sequential code and is fully preemptable.
2. Each site, S_j, is composed of **R** shareable (i.e., community-based) resources. Each resource may fail with probability f, $0 \leq f \leq 1$, and be repaired independently.

Objective: Our goal is to design an on-line scheduling policy such that:

1. applications are efficiently and reliably executed to their logical termination;
2. mean response time is minimized; and
3. the scheduler has no knowledge of: (1) the service time of the jobs or the tasks; (2) the job arrival times; (3) how many processors each job needs until the job actually arrives; (4) and the set of processors available for scheduling the jobs.

Fig. 1. Fault-tolerant grid scheduling problem

Access to remote resources was the main motivation for building Grid computing, and it remains the primary goal today. To this end, a variety of successful Grid infrastructures that focuses on simplifying access and usage of Grid computing has been developed over the past few years (e.g., [8]). However, the ability to execute applications whose computational requirements exceed local resources and the reduction of job turnaround time through workload balancing across multiple computing facilities requires efficient Grid job scheduling. Also, as the system increases both in size and complexity, the possibility of a component (e.g., a node, link, scheduler) failure also increases. Thus, the ability to tolerate failures while effectively exploiting the Grid computing resources in a scalable and transparent manner must be an integral part of Grid computing infrastructure. In the following section, we establish the fact that, to a large extent, the problem considered in this paper has not been fully addressed in the literature.

2.2 Related Work

Although job scheduling and fault-tolerance are active areas of research in Grid computing environments, these two areas have largely been and continue to be developed independent of one another each focusing on a different aspects of computing. Research in scheduling has focused on efficiency by exploiting as much parallelism as possible while assuming that the resources are 100% reliable [1],[12]. Also, existing solutions for grid computing systems, to a large extent, are based on requiring static and dynamic application and system resource information, and performance prediction models. This kind of information is not always available and is often difficult to obtain. Moreover, most of the conventional grid-based systems use a static scheduling model (e.g., LSF [20]).

Recently, interest in making Grid computing systems fault tolerant has been receiving attention [3], [7], [19], [18]. For example, several fault detection service architecture have been developed for grid computing systems (e.g., [4], [14], [15]. Similarly, checkpoint-recovery [9] and job replication [18] techniques are popular fault-tolerance approaches on distributed systems. However, as noted in [10], these fault-tolerant approaches typically ignore the issue of processor allocation. This can lead to a significant degradation in response time of the applications [10] and to counter this effect an efficient job scheduling policy is required.

In this paper, we assume that the system components may fail and can be eventually recovered from failure. Also, we assume that both hardware and software failures obey the fail-stop [13] failure mode. As in [9], we assume that faults can occur on-line at any point in time and the total number of faulty processors in a given site may never exceed a known fraction. We also assume that node failures are independent from each other [19]. In addition, we assume that every grid scheduler in the system is reachable from any other grid scheduler unless there is a failure in the network or the node housing the grid scheduler. A scheme to deal with node, scheduler and link failures is discussed in [3].

3 Dynamic Fault-Tolerant Scheduling Policies

The proposed scheduling policy is called *Dynamic Fault-Tolerant Scheduling (DFTS)* policy. In DFTS, the core system architecture is designed around **N**-levels of virtual hierarchy as shown in Figure 2. At the top of the hierarchy, there is a *grid super scheduler ($GS_{N,1}$)* while at the leaf level there is a *local scheduler (LS)* for each site. In between the *grid super scheduler* and the *local schedulers*, there exists a hierarchy of *grid schedulers (GSs)*. The GSs at **level 1** are solely responsible for scheduling jobs whereas the ($GS_{N,1}$) and the GSs above **level 1** are responsible for load balancing.

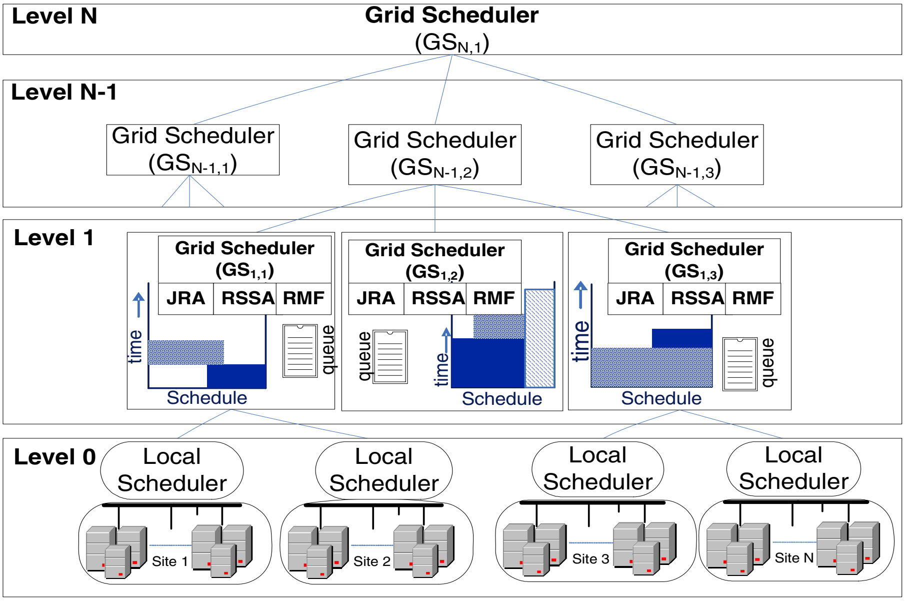


Fig. 2. Basic blocks for a Grid scheduling architecture

Each grid scheduler in the hierarchy is uniquely identified as $GS_{i,j}$ where $0 \leq i \leq N$ denotes the level in the hierarchy and j denotes the grid scheduler id (GID). For example, $GS_{1,1}$ denotes grid scheduler 1 located at level 1. Each $GS_{i,j}$ in the hierarchy also controls a set of sites in the system. For example, sites 1 and 2 in Figure 2 are under the control of $GS_{1,1}$ scheduler. In this case, we say that $GS_{i,j}$ is the parent of LS_1 and LS_2. Similarly, $GS_{N-1,2}$ is the parent of $GS_{1,1}$, $GS_{1,2}$ and $GS_{1,3}$. Finally, when jobs directly submitted to $GS_{1,1}$ are assigned to sites 1 or site 2 or both, we say that the assignment is a *local job placement*.

As shown in Figure 2, the DFTS policy has three main components namely; *Resource Selection and Scheduling Algorithm (RSSA)*, *Replica Management and Failover (RMF)*, and *Job Replication Algorithm (JRA)*. These three components collectively schedule parallel jobs on the appropriate sites, automatically replicate jobs and tasks over several sites and processors, keep track of the number of replicas, instantiate them on-demand and delete the replicas when the primary copies of the jobs and tasks successfully complete execution. DFTS maintains some state information for failure and recovery detections in Application Status Table (*AST*). Also, a fail-over strategy is used when a link or a node failure is detected. A detailed discussion of the fail-over strategy is given in [2], [3]. In the following subsections, we describe these three components in detail.

3.1 Resource Selection and Scheduling Algorithm

The Resource Selection and Scheduling Algorithm (RSSA) is responsible for scheduling grid jobs on sites that match the resource request of the jobs. Without lose of generality, we assume that all incoming jobs are submitted to the $GS_{1,j}$ grid scheduler (i.e., GSs at level 1) where j denotes GID of the the submitting node. We assume the existence of Resource Specification Language (RSL) that provides a common interchange language to describe resources required by the jobs [8]. The code of the job, which has to be executable in the remote resource environment as well as other information such as stdin, stdout, and the name and port used on the remote node, are specified in the job request, written using the RSL.

When a job arrives at a $GS_{1,j}$ for execution, it can either be scheduled to run locally or remotely. The decision to run the job locally or send it to a remote site is made by $GS_{1,j}$ based on the job requirements and the load level of the local sites. For example, if $GS_{1,1}$ receives a job for which the required resources and services is not present within the scope of its control (i.e., site 1 and site 2), it flags the job as possible for remote execution. It then sends a *Request for Execution (RFE)* message on behalf of the job to its parent at the next level of the hierarchy (i.e., $GS_{N-1,2}$). After sending *RFE* message to its parent, $GS_{1,1}$ updates its *base load level* to ensure that jobs with similar requests will not be sent to the appropriate site.

When $GS_{N-1,2}$ receives the *REF* request, it tries to see if the request can be satisfied by any of $GS_{N-1,2}$ children excluding $GS_{1,1}$. This process is recursively followed up the hierarchy until a site with the required services or resources is found or no sites can satisfy the request. If a site that can satisfy the *REF* request is found, the candidate site makes arrangement with $GS_{1,1}$ for the job to be sent over for execution. Upon receiving the message, $GS_{1,1}$ will send the job details to the candidate site.

After dispatching the job request to the candidate site, $GS_{1,1}$ then informs the backup GS scheduler about the assignment and then updates the application status table (AST) to reflect the new assignment.

3.2 Job Replication Algorithm

The replica creation and placement ensures that a job and its constituent task are stored in a number of locations in the hierarchy. Jobs are replicated over sites while tasks are replicated over processors. Specifically, When a job with fault-tolerance requirement arrives into the system, DFTS undertakes the following steps:

1. create a replica of the job;
2. keep the replica and send the original job to a child that is alive and reachable; and
3. update the application status table (*AST*) to reflect where the job replicas are located. This process recursively follows down the cluster tree until we reach the lowest level cluster scheduler (*LCS*) at which point the replica placement process terminates.

3.3 Replica Management and Failover

The DFTS monitors applications at job-level (between non-leaf nodes and their parents) and at task-level (between leaf nodes and their parents). A monitoring message exchanged between a parent and a leaf-level node is called a *report* while that between non-leaf nodes is called a *summary.* A report message contains status information of a particular task running on a particular node and sent every *REPORT-INTERVAL* time units. In contrast, the *summary* message contains a collection of many reports and sent every *SUMMARY-INTERVAL* time periods such that *REPORT-INTERVAL* < *SUMMARY-INTERVAL.*

When a processor completes execution of a task, the report message contains a *FINISH* message. In this case, the receiving scheduler deletes the corresponding replica and informs the backup scheduler to do the same. When the last replica of a given job is deleted, the job is declared as successfully completed. In this case, the cluster scheduler immediately sends a summary message that contains the *COMPLETED* message to the parent scheduler, which deletes the copy of the job and forward the same message to its parent. This process continues recursively until all replicas of the job are deleted.

After each assignment, the children periodically inform their parents the health of the computations as discussed above. If the parent does not receive any such message from a particular child in a given amount of time, then the parent suspects that the child has failed. In this case, it notes this fact in the AST and sends a request for report message to the child. If a reply from the child has not been received within a specific time frame, the child is declared dead. The replica of a job is then scheduled on a health node.

4 Performance Analysis

We used simulation to study the performance of the proposed fault-tolerant scheduling policy. We compared the proposed DFTS scheduling policy with FTSA policy [18] and AHS policy [1].

4.1 Experimental Setup

We used a Grid system composed of eight sites and each site is managed by a grid scheduler. We then create a 4-level hierarchy with the root as the super scheduler, four second-level grid schedulers that act as children of the super scheduler, and 8 local schedulers at the bottom. The workload used is a synthetic matrix multiplication application characterized by *arrival time*, *service demand time* in a dedicated environment, *maximum parallelism*, and *size* in Kbytes. The cumulative service demand is generated using hyper-exponential distribution with mean 14.06 [1] and the maximum parallelism is uniformly distributed over the range of 1 to 64. The default arrival CV is fixed at 1 and the default service time CV is fixed at 3.5 as empirical observations at several supercomputer centers indicated this to be a reasonable value. In all experiments, we configured the system with two replicas as in [18].

We set processor and link time-to-failure to 0.00321 hours and 0.005 hours, respectively. The time-to-repair the link is set to 30 seconds while that of the processors is set to 0.00321 hours. The processor failure and recovery figures are based on the data collected on the experimental assessment of workstation failures in [10]. As in [18], [11], [10], we assume that inter-occurrence times of failures for each processor are independent and identically distributed as exponential random variables with the same failure rate. Also, we assume that the times to failure of workstations and their repair times are mutually independent random variables [10]. As in [1],[17], we used a simple model to capture the communication overhead as follows:

$$T_{comm} = \text{Startup} + \frac{\text{Message size}}{\text{Bandwidth}} \tag{1}$$

In all the experiments discussed here, the communication network latency to be 50μsec with the transfer rate of 100Mbits/sec. These values are typical of modern light-weight messaging layers running on top of gigabit switched LANs [17].

4.2 Validation

A batch strategy is used to compute confidence intervals (at least 30 batch runs were used for the results reported in this paper).

4.3 Results and Discussion

Figures 3 shows the mean response time of the jobs for the three scheduling policies. In the experiments, we assumed that the probability of network failure is zero and the case in which network failure is an issue will be addressed in the extended version of the paper. Also, every 3000 seconds, a non-idle node is randomly selected in each site and made to fail for 6 seconds.

From the data on the graph, we observe that at low system load, FTSA and DFTS are marginally better than AHS policy. This is because of the fact that at this load level, there are many idle processors. This means both FTSA and DFTS can schedule a job on several clusters and at least one of the replicas could finish without being interrupted by a node failure. However, as the load increases, performance of FTSA deteriorates in all three environments while DFTS performs marginally worse (about 4%) than the AHS policy in most instances. This can be explained by the fact that as load increases the number of idle processors decreases. As a result, finding *n* idle clusters for FTSA to

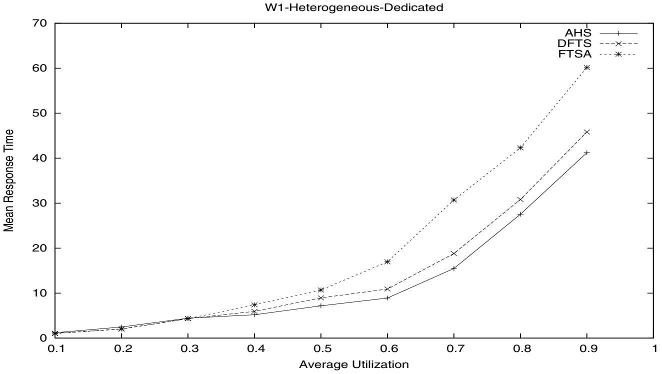


Fig. 3. Performance of the policies in dedicated heterogeneous environments

schedule jobs becomes harder. In contrast, DFTS uses demand-driven approach as in AHS and multiple replicas of a job are only scheduled when there are ample free processors. For the workload type we studied, it seems that the DFTS approach is better than the active replication approach used in FTSA.

5 Conclusion and Future Directions

In this paper, we presented a scalable framework that loosely couples the dynamic job scheduling approach with the hybrid (i.e., passive and active replications) approach to schedule jobs efficiently while at the same time providing fault-tolerance. The main advantage of the proposed approach is that fail-soft behaviour (i.e., graceful degradation) is achieved in a user-transparent manner. Furthermore, being a dynamic algorithm estimations of execution or communication times are not required. An important characteristic of our algorithm is that it makes use of some local knowledge like faulty/intact or busy/idle states of nodes and about the execution location of jobs.

Acknowledgement. I appreciate the help of Maliha Omar without whom this paper would not have been completed. This research is partially funded by Deakin University.

References

1. Jemal H. Abawajy and Sivarama P. Dandamudi. Parallel job scheduling on multicluster computing systems. In *Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'03)*, pages 11–21, 2003.
2. Jemal H. Abawajy and Sivarama P. Dandamudi. A reconfigurable multi-layered grid scheduling infrastructure. In Hamid R. Arabnia and Youngsong Mun, editors, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '03, June 23 - 26, 2003, Las Vegas, Nevada, USA, Volume 1*, pages 138–144. CSREA Press, 2003.
3. Jemal H. Abawajy and Sivarama P. Dandamudi. Fault-tolerant grid resource management infrastructure. *Journal of Neural, Parallel and Scientific Computations*, 12:208–220, 2004.

4. J.H. Abawajy. Fault detection service architecture for grid computing systems. In *Lecture Notes in Computer Science*, volume 3044/2004, pages 107 – 115. Springer-Verlg, 2004.
5. Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.
6. Jrn Gehring and Achim Streit. Robust resource management for metacomputers. In *HPDC '00: Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC'00)*, page 105. IEEE Computer Society, 2000.
7. Soonwook Hwang and Carl Kesselman. Gridworkflow: A flexible failure handling framework for the grid. In *12th International Symposium on High-Performance Distributed Computing (HPDC-12 2003), 22-24 June 2003, Seattle, WA, USA*, pages 126–137. IEEE Computer Society, 2003.
8. I. Foster and C. Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259–278. MORGAN-KAUFMANN, 1998.
9. Leon Juan, Fisher Allan L., and Steenkiste Peter. Fail-safe PVM: A Portable Package for Distributed Programming with Transparent Recovery. Technical report, CMU, Department of Computer Science, Feb 1993.
10. James S. Plank and Wael R. Elwasif. Experimental assessment of workstation failures and their impact on checkpointing systems. In *Symposium on FTC'98*, pages 48–57, 1998.
11. James S. Plank and Michael G. Thomason. Processor allocation and checkpoint interval selection in cluster computing systems. *Journal of Parallel and Distributed Computing*, 61(11):1570–1590, 2001.
12. Anuraag S., Alok S., and Avinash S. A scheduling model for grid computing systems. In *Proceedings of Grid'01*, pages 111–123. IEEE Computer Society, 2001.
13. Fred B. Schneider. Byzantine generals in action: Implementing failstop processors. *ACM Transactions on Computer Systems*, 2(2):145–154, 1984.
14. P. Stelling, I. Foster, C. Kesselman, and G. von Laszewski. C.Lee. A fault detection service for wide area distributed computations. In *Proc. 7th Symposium on High Performance Computing*, pages 268–278, 1998.
15. Brian Tierney, Brian Crowley, Dan Gunter, Mason Holding, Jason Lee, and Mary Thompson. A monitoring sensor management system for grid environments. In *HPDC*, pages 97–104, 2000.
16. Namyoon W., Soonho C., Hyungsoo J., and Park Y. & Park H. Jungwhan M., Heon Y. Y. Mpich-gf: Providing fault tolerance on grid environments. In *Proceedings of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2003.
17. H. Wabnig and G. Haring. Perfromance Prediction of Parallel Systems with Scalable Specifications-Methedology and case Study. *Performance Evaluation Review*, 22(2-4):46–62, 1995.
18. J. B. Weissman. Fault-tolerant wide area parallel computation. In *Proceedings of IDDPS'2000 Workshops*, pages 1214–1225, 2000.
19. Jon B. Weissman. Fault tolerant computing on the grid: What are my options? In *HPDC '99: Proceedings of the The Eighth IEEE International Symposium on High Performance Distributed Computing*, page 26. IEEE Computer Society, 1999.
20. Ming Q. Xu. Effective metacomputing using LSF multicluster. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 100 – 106. IEEE Computer Society, 2001.

An Efficient Dynamic Load-Balancing Algorithm in a Large-Scale Cluster*

Bao-Yin Zhang[1], Ze-Yao Mo[1], Guang-Wen Yang[2], and Wei-Min Zheng[2]

[1] Institute of Applied Physics and Computational Mathematics,
Beijing, 100088, P.R. China
[2] Department of Computer Science and Technology,
Tsinghua University, Beijing, 100084, P.R. China
zby@tsinghua.edu.cn

Abstract. Random stealing is a well-known dynamic load-balancing algorithm. However, for a large-scale cluster, the simple random stealing policy is no longer efficient because an idle node must randomly steal many times to obtain a task from another node. This will not only increase the idle time for all nodes but also produce a heavy network communication overhead. In this paper, we propose a novel dynamic load-balancing algorithm, Transitive Random Stealing (TRS), which can make any idle node obtain a task from another node with much fewer stealing times in a large-scale cluster. A probabilistic model is constructed to analyze the performance of TRS, random stealing and Shis, one of load balance policies in the EARTH system. Finally, by the random baseline technique, an experiment designed to compare TRS with Shis and random stealing for five different load distributions in the Tsinghua EastSun cluster convinces us that TRS is a highly efficient dynamic load-balancing algorithm in a large-scale cluster.

Keywords: Dynamic load balancing, large-scale cluster, transitive random stealing, probabilistic model.

1 Introduction

Large-scale clusters are playing an important role in the supercomputing field. The scale of the clusters is becoming more and more large, which is up to hundreds of or thousands of nodes. In order to achieve scalable performance, it is important to evenly distribute the workload among the processing nodes. Two basic approaches [5] to dynamically schedule task loads can be found in current literature - *random stealing* and *work sharing*.

Random Stealing (RS) attempts to steal a task from a randomly selected node when a node finds its own task queue empty, repeating steal attempts until it succeeds. Random stealing is provably efficient in terms of time, space, and communication for the class of fully strict computations [3,11], and the natural

* This work is supported by Chinese NSF for DYS granted by No. 60425205 and National Postdoctor Science Foundation of China.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 174–183, 2005.

random stealing algorithm is stable [1]. Communication is only initiated when nodes are idle. When the system load is high, no communication is needed, causing the system behave well under high loads.

Previous works [2,7] have shown that simple random stealing policy can balance load efficiently for a fine-grain multithreaded execution model in a parallel computer or in a small-scale cluster with high speed networks. Cilk [2] provides an efficient C-based runtime system for multithreaded parallel programming with a random stealing scheduler on the Connection Machine CM5 MPP, the Intel Paragon MPP, the Sun Sparcstation SMP, and the Cilk-NOW network of workstations. The EARTH runtime system [7] supports several dynamic load balancer policies, whose goal is to design simple balancers that deliver good load distribution with minimum overheads for a fine-grain multithreaded execution model on the high-performance distributed memory machine MANTA [6].

Satin [8,9] presents a system for running divide-and-conquer programs on wide-area systems with an efficient load balancing algorithm, Cluster-aware Random Stealing (CRS). CRS mainly focuses on the performance optimization for wide-area networks with high latency and low bandwidth, yet simple random stealing policy is still used in single cluster systems.

In this paper, we focus on the dynamic load balancing policies in a large-scale cluster. For simple random stealing policy, there exists a problem that an idle node must randomly steal many times to obtain a task from another node. To solve this problem, some policies have been developed. Shis, one of load balance policies based on RS in the EARTH system [4] slightly modifies random stealing by remembering the originating node (history information) from which a task was last received, and sending requests directly to that node (the short-cut path). The authors of the paper [10], present two relatively complicated adaptive location policies which record more history information for global scheduling algorithms.

Here we propose a dynamic load-balancing algorithm, Transitive Random Stealing (TRS), which further improves Shis with a transitive policy. With its transitive policy, TRS can make any idle node obtain a task from another node with much fewer stealing times in a large-scale cluster, so as to reduce the idle time for all nodes and the network communication overhead, and to improve the scalable performance of the system.

In the rest of this paper, we present the transitive random stealing algorithm in the next section. Section 3 constructs a probabilistic model to analyze the performance of TRS, Shis and RS. We evaluate the performance of TRS, Shis and RS by the random baseline technique in the Tsinghua EastSun cluster in Section 4 . Finally, Section 5 concludes our works.

2 Transitive Random Stealing Algorithm

Our design philosophy for dynamic load balancing algorithms is to reduce the idle time for all nodes, rather than balancing work loads equally on all nodes. A node is said to be in the idle state when it has no tasks to execute. Distributing the workload during application execution is achieved by sending the *tokens* to the schedulers on the other nodes. A token contains all the necessary information to create a new *task*. A Task is a piece of code which is to be executed, possibly in parallel with other tasks. Tokens are stored in the task queue on each node.

Here we present our dynamic load-balancing algorithm, Transitive Random Stealing (TRS), which not only remembers the originating node (history information) from which a task was last received and sends requests directly to that node (the short-cut path), but also *forwards* this history information to other nodes which want to steal a task from it (the transitive policy).

Pseudo code for the algorithm is illustrated in Figure 1. (The transId is a variable which remembers the history information, the nodeId of another node. Every node has a local task queue which stores the tokens.)

```
/* The main-loop function for TRS:  */
void transitive random stealing(){
  While(NOT exiting){
    if(idle of node){
      if(local task queue has tokens){
        get a token to execute;
      }else{
        if(transId is blank){
          select a node from other nodes uniformly at random,
          and request for a token from it;
        }else{
          request for a token from the node whose nodeId is transId;
        }
        wait to receive an replying message;
        update its transId with the transId in the replying message;
        if(the replying message includes a token){
          execute the token;
        }
      }
    }else{
      wait for some task running over;
    }
  }
}
/* The function for handling the request:  */
Message handle_request(){
  if(local task queue has tokens){
    return a message with its own nodeId as transId and a token
    from its local task queue;
  }else{
    return a message with its transId and no tokens;
  }
}
```

Fig. 1. Pseudo code of the transitive random stealing algorithm

In TRS, a simple request-reply-update protocol is implemented between the thief and the victim. Whenever an idle node has no tokens in its local task queue, it becomes a thief. The thief selects a victim by its history information or randomly selects a victim if no history information available (transId is blank), then it requests for a token from this victim. If the local task queue of the victim has tokens, the victim replies a message which contains a token from its local task queue and its own nodeId as transId. Otherwise, the victim replies a message which only contains its transId (if no history information, a blank transId is included). When the replying message arrives, the thief updates its transId with the new one (a blank transId is permitted) in the replying message and execute the token if a token is included in the replying message.

The transitive policy is simple and TRS can be easily implemented. But with this simple transitive policy, TRS can make any idle node obtain a task from another node with fewer stealing times in a large-scale cluster. As a result, this will greatly reduce the idle time for all nodes and the network communication overhead, and improve the scalable performance of the system. At the same time, TRS inherits the advantages of simple random stealing policy: communication is only initiated when nodes are idle. When the system load is high, no communication is needed, causing the system behave well under high loads.

As we can see, a few more bytes (transId) is sent in the replying message for TRS than Shis and RS. But the time and bandwidth of the communication are very similar for those messages with little different sizes. In a sense, the key factor which influences the network communication overhead is the times of sending messages.

Note. In some very special conditions, there may be a loop transition of the transId. In order to avoid this case, the implementation of the algorithm can limit the times of transition of the transId. In fact, in the later experiments, we empirically limit the times of transition of transId by $\max\{[\log_2 n - 3], 1\}$, where n is the number of the nodes in the cluster.

3 Probabilistic Model for System States

In this section, we construct a probabilistic model for system states to analyze the performance of TRS, Shis and RS.

The probabilistic model.

1. There are $N + 1$ nodes which are connected by some network topology and can exchange messages with each other.
2. Assume that there are m nodes with nodeId $1, 2, \ldots, m$ which are busy and their task queues have enough tokens. Other nodes with nodeId $m + 1, m + 2, \ldots, N + 1$ are idle and no tokens in their task queues.
3. Assume that all the nodes have no history information in the initial state and no new tasks are spawned dynamically in the whole process.

4. Assume that once one node becomes busy, it will not become idle in the whole process.
5. The evolution of the system can be described by a probabilistic chain X, the chain state X_t after step t is a tuple $(X_t(1), X_t(2), \ldots, X_t(N+1))$ in which $X_t(i)$ represents the probability that the i'th node is busy after step t. Initially, only m node is busy and other nodes are idle, so the start state X_0 is $(\underbrace{1, \ldots, 1}_{m}\underbrace{0, \ldots, 0}_{N-m+1})$.

From the assumption 3 and 4, one idle node only needs to steal one task to become busy in the whole process. The history information, preserved by Shis when the node becomes busy, will not be used again, hence Shis is identical to RS for this probabilistic model.

In the following, we give the transition from the state X_t to the state X_{t+1} for TRS and Shis, then compare their performance.

Transition from the state X_t to the state X_{t+1} for Shis: Each idle node chooses a requested destination uniformly at random from other N nodes (From the assumption above, we know that each idle node has an empty task queue and no initial history information). Every node that receives a request replies a message of no tasks except that one of the nodes with nodeId $1, 2, \ldots, m$ replies a token from its task queue. Every node that receives a token from one of the nodes with nodeId $1, 2, \ldots, m$ becomes busy.

Formally, since the node with nodeId $m+1, m+2, \ldots, N+1$ is symmetric, all $X_t(i)$ are equal for $m+1, m+2, \ldots, N+1$. The probability that an idle node can choose one of the nodes with nodeId $1, 2, \ldots, m$ and becomes busy is m/N, so

$$X_{t+1}(i) = X_t(i) + (1 - X_t(i)) \cdot \frac{m}{N} \tag{1}$$

for $i = m+1, m+2, \ldots, N+1$.

Transition from the state X_t to the state X_{t+1} for TRS: Each idle node chooses a requested destination uniformly at random from other N nodes (From the assumption above, each idle node has an empty task queue and no initial history information). Every node that receives a request replies its transId except that the nodes with nodeId $1, 2, \ldots, m$ reply a token from its task queue and its own nodeId as transId. Every node that receives a token from one of the nodes with nodeId $1, 2, \ldots, m$ becomes busy and records the transId. At the same time, every node, that receives a message only including a non-blank transId, requests for a token from the node with nodeId (non-blank transId), then becomes busy and records the transId (from the assumption above, the non-blank transId must be one of $1, 2, \ldots, m$).

Formally, as the same above, since the node with nodeId $m+1, m+2, \ldots, N+1$ is symmetric, all $X_t(i)$ are equal for $m+1, m+2, \ldots, N+1$. The probability that an idle node can choose one of the nodes with nodeId $1, 2, \ldots, m$ is m/N, the probability that an idle node can obtain a non-blank transId from other nodes except the nodes with nodeId $1, 2, \ldots, m$ is

$$\frac{N-m}{N} \cdot X_t(i).$$

Thus

$$X_{t+1}(i) = X_t(i) + (1 - X_t(i)) \left(\frac{m}{N} + \frac{N-m}{N} \cdot X_t(i) \right) \tag{2}$$

for $i = m+1, m+2, \ldots, N+1$.

We reform the formulas (1) and (2) to the following form

$$X_{t+1}(i) = \frac{(N-m)X_t(i) + m}{N} \quad \text{for Shis and RS,}$$

$$X_{t+1}(i) = \frac{(2 - X_t(i))(N-m)X_t(i) + m}{N} \quad \text{for TRS,}$$

for $i = m+1, m+2, \ldots, N+1$.

Comparing the two recurrence formulas for $X_t(i)$ of Shis and TRS, we easily find that the probability that an idle node becomes busy increases more rapidly by ascending step t for TRS than Shis, because there is an extra factor $(2-X_t(i))$ in the numerator for TRS.

In the following, we compute two examples by the two recurrence formulas to compare the performance of TRS, Shis and RS.

For $N = 128, m = 4$, we compute the probability according to the two recurrence formulas, and have for $i = m+1, m+2, \ldots, N+1$

Step t	1	2	3	4
Shis $X_t(i)$	0.031	0.062	0.091	0.119
TRS $X_t(i)$	0.031	0.091	0.199	0.379
Step t	5	6	7	8
Shis $X_t(i)$	0.147	0.173	0.199	0.224
TRS $X_t(i)$	0.626	0.865	0.982	0.999

For $N = 256, m = 4$, we compute the probability according to the two recurrence relations, and have for $i = m+1, m+2, \ldots, N+1$

Step t	1	2	3	4
Shis $X_t(i)$	0.016	0.031	0.046	0.061
TRS $X_t(i)$	0.016	0.046	0.104	0.210
Step t	5	6	7	8
Shis $X_t(i)$	0.076	0.090	0.104	0.118
TRS $X_t(i)$	0.386	0.629	0.865	0.982

As we can see, the probabilities that an idle node obtains a task from another node rapidly increase for TRS by ascending step t. This means that TRS can make an idle node obtain a task from another node with much fewer stealing times than Shis and RS. At the same time, comparing the cases between $N = 128$ and $N = 256$, it shows that the larger the scale of the cluster is, the more efficient TRS is than Shis and RS.

4 Performance Evaluation Based on Random Baseline Technique

In this section, using the random baseline technique, we experimentally compare TRS with Shis and RS for five different load distributions in the Tsinghua EastSun cluster which has 32 nodes (4×Xeon III 700s, Fast Ethernet, Redhat 8.1). Here we implement each of the three algorithm in an MPI application in which a process simulates a node. The processes implement two threads except the process with rank 0, one thread for dealing the main loop, the other for handling the request. The process with rank 0, by the random baseline technique, implements a task generator which distributes the same load distributions to the other processes for the three algorithms respectively.

In order to stress to test the performance of these algorithms on different load distributions, we make use of the task generator randomly generating five different load distributions instead of scheduling some real parallel programs. The task generator generates three types of load distributions uniformly distributed on all nodes, half of all nodes and 1/8 of all nodes, two types of binomial distributions, $\mathrm{Bi}(n, 1/3)$ and $\mathrm{Bi}(n, 1/8)$, where n is the number of the nodes. From the knowledge of Statistics, the binomial distribution $\mathrm{Bi}(n, p)$ approaches the Poisson distribution, when the number n is large and the probability p is small. All of the five types of load distributions distribute $5n$ tasks to the nodes of 10 times in the runtime. We assume that every task has the same executing time.

We compare the performance of the three algorithms by counting the total times of stealing from other nodes for each algorithm (the total times includes the times of stealing nothing from other nodes). The experiments are implemented in the Jcluster environment [12], a high performance Java parallel environment which provides the MPI-like message passing interface. Figure 2,3,4,5,6 illustrate the results for the five type of load distributions.

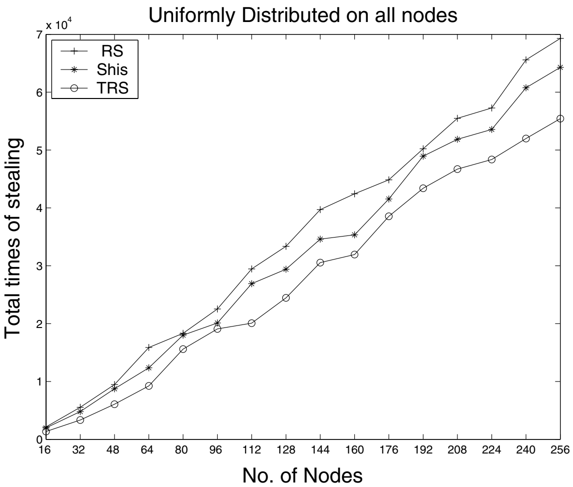


Fig. 2. Task load uniformly distributed on all nodes

For the task load distribution uniformly distributed on all nodes, the difference of the performance for the three algorithms is not so distinct in the small-scale cluster. However, along with the increase of the size of the nodes, TRS behaves a good performance. For the other four task load distributions, several ten thousands of or several hundred thousands of stealing times are economized for TRS than Shis and RS in the large-scale clusters. This greatly reduces the idle time for all nodes and the network communication overhead, so as to improve the scalable performance of the system. These experimental results convince us that TRS is a highly efficient dynamic load balancing algorithm in a large-scale cluster.

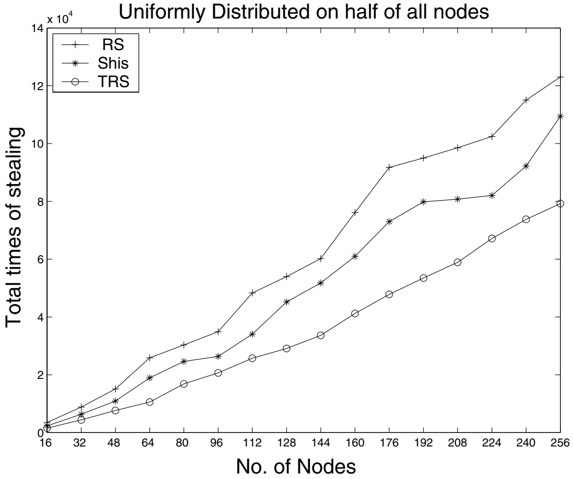


Fig. 3. Task load uniformly distributed on half of all nodes

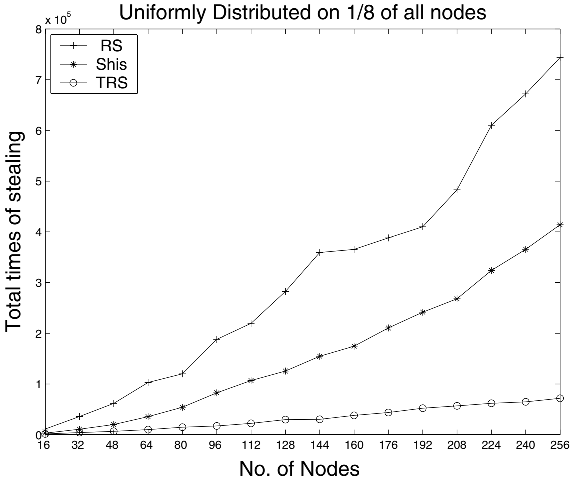


Fig. 4. Task load uniformly distributed on 1/8 of all nodes

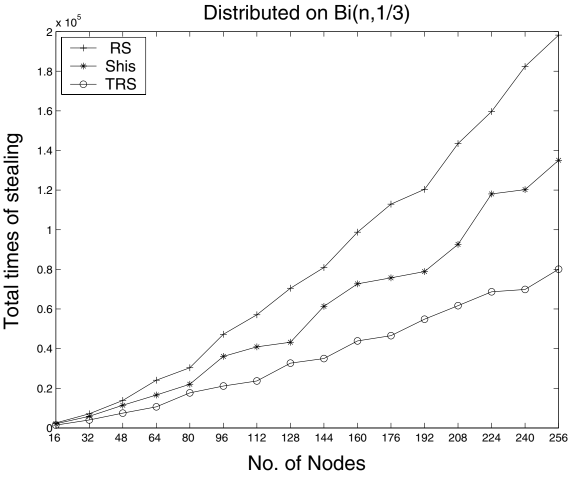


Fig. 5. Task load distributed on Bi$(n, 1/3)$

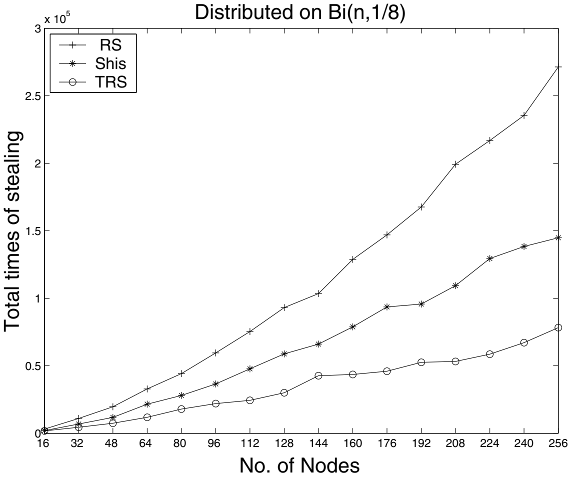


Fig. 6. Task load distributed on Bi$(n, 1/8)$

5 Conclusions

In this paper, we propose the Transitive Random Stealing algorithm (TRS) which provides an efficient dynamic load balancing policy, the transitive policy. With this policy, TRS can make any idle node obtain a task from another node with much fewer stealing times in a large-scale cluster. Consequently, this will greatly reduce the idle time for all nodes and the network communication overhead, so as to improve the scalable performance of the system. Both analytical and experimental results convince us that TRS is a highly efficient dynamic load balancing algorithm in a large-scale cluster.

References

1. P. Berenbrink, T. Friedetzky, L.A. Goldberg, "The Natural Work-Stealing Algorithm is Stable", SIAM Journal on Computing, Vol. 32(5), 2003, pp. 1260-1279.
2. R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, and Y. Zhou, "Cilk: An efficient multithreaded runtime system", Proceedings of the 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95, Santa Barbara, California, July 1995, pp. 207-216.
3. R.D. Blumofe, and C.E. Leiserson, "Scheduling Multithreaded Computations by Work Stealing", Proceedings of the 35th Annual IEEE conference on Foundations of Computer Science (FOCS'94), Santa Fe, New Mexico, November 20-22, 1994.
4. H. Cai, Olivier Maquelin, Prasad Kakulavarapu, and G.R. Gao, "Design and Evaluation of Dynamic Load Balancing Schemes under a Fine-grain Multithreaded Execution Model", Proceedings of the Multithreaded Execution Architecture and Compilation Workshop, Orlando, Florida, January 1999. Delaware, May 1999.
5. Eager, D.L., Lazowska, E.D., and Zahorjan, J., "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", Performance Evaluation, Vol. 6, 1986, pp. 53-68.
6. W.K. Giloi, U. Bruning, and W. Schroderpreikschat, "MANTA: Prototype of a distributed memory architecture with maximized sustained performance", Proceedings of Eurornicm PDP96 Workshop, 1996.
7. Herbert H.J. Hum, Olivier Maquelin, Kevin B. Theobald, Xinmin Tian, Xinan Tang, Guang R. Gao, Phil Cupryk, Nasser Elmasri, Lau-rie J. Hendren, Alberto Jimenez, Shoba Krishnan, Andres Marquez, Shamir Merali, Shashank S. Nemawarkar, Prakash Panangaden, Xun Xue, and Yingchun Zhu. "A design study of the EARTH multiprocessor", Proceedings of the IFIP WG 10.3 Working Conference on Parallel Architectures and Compilation Techniques, PACT '95 (Lubomir Bic, Wim Bohm, Paraskevas Evripidou, and Jean-Luc Gaudiot, eds.), Limassol, Cyprus, ACM Press, June 27-29, 1995, pp. 59-68.
8. Rob V. van Nieuwpoort, T. Kielmann, and Henri E. Bal, "Satin: Efficient Parallel Divide and Conquer in Java", Proceedings of Euro-Par 2000, Munich, Germany, August 29-September 1, 2000, pp. 690-699.
9. Rob V. van Nieuwpoort, Thilo Kielmann, and Henri E. Bal, "Efficient Load Balancing for Wide-area Divide-and-Conquer Applications", Proceedings of Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01), Snowbird, UT, June 18-19, 2001.
10. N.G. Shivaratri, and P. Krueger, "Two Adaptive Location Policies for Global Scheduling Algorithms", IEEE International Conference on Distributed Computing Systems (ICDCS), 1990.
11. I.C. Wu, and H. Kung, "Communication Complexity for Parallel Divide and Conquer", 32nd Annual Symposium on Foundations of Computer Science (FOCS'91), San Juan, Puerto Rico, Oct. 1991, pp. 151-162.
12. B.Y. Zhang, "A Java paralle environment", available at http://vip.6to23.com/jcluster/

Job Scheduling Policy for High Throughput Grid Computing

J.H. Abawajy

School of Information Technology,
Deakin University, Geelong, VIC, 3217, Australia

Abstract. The growing computational power requirements of grand challenge applications has promoted the need for merging high throughput computing and grid computing principles to harness computational resources distributed across multiple organisations. This paper identifies the issues in resource management and scheduling in the emerging high throughput grid computing context. We also survey and study the performance of several space-sharing and time-sharing opportunistic scheduling policies that have been developed for high throughput computing.

Keywords: Grid computing, high throughput computing, resource management, job scheduling, opportunistic scheduling.

1 Introduction

Grid computing [6] is emerging as a new paradigm for Internet-based parallel and distributed computing. Until recently, the focus of high throughput computing (HTC) [11] have been to provide convenient access to a pool of remote machines within a single administrative domain for execution of batch jobs while fully preserving the rights of their owners [3].

However, the demand for more computing resources coupled with advances in Grid middleware technologies have mandated the marriage of HTC and grid technologies resulting in High Throughput Grid Computing (HTGC). Condor-G [7] and Nimrod/G [5] are examples of resource management and scheduling systems built using the Globus toolkit services [10]. The result is very beneficial for the end user, who is now enabled to utilize large collections of resources that span across multiple domains as if they all belonged to the personal domain of the user [12].

The focus of this paper is on the job management and scheduling problem for high throughput grid computing platforms. The motivation for this work is that the distributed systems that solve large-scale problems will always involve aggregating and scheduling many resources. Also, the number of jobs to be executed in high throughput grid computing nearly always outnumbers the available resources [12]. A wide variety of scheduling approaches for grid computing are currently available [2]. As the main goal of grid scheduling is to find an optimal or near optimal schedule to allocate jobs to computational resources for execution to achieve a high performance, they are not suitable for high throughput platforms, which we are interested in.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 184–192, 2005.

Therefore, an effective and efficient resource management and job scheduling mechanisms that decide how to allocate resources to jobs in a fair manner is a key requirement for the success of high throughput grid computing. In this paper, we survey several space-sharing and time-sharing opportunistic scheduling policies that have been developed for high throughput computing. Using simulation, we study the performance of these policies. Our results demonstrate that timesharing scheduling policies can be used in an opportunistic setting to improve both mean job slowdowns and mean response times with little or no throughput reduction.

The remainder of this paper is organized as follows. In Section 3, we present the system model used in this paper. The scheduling problem and related works are discussed in Section 3. A detailed description of five opportunistic scheduling policies are discussed in Section 4. The performance evaluation models are discussed in detail in Section 5. The simulation results and performance comparisons of the five scheduling policies is presented in Section 6. concluding remarks and future directions are discussed in Sections 7.

2 High Throughput Grid Computing

Figure 1 shows the high throughput grid computing architecture used in this paper. The core of the system includes a set of independent distributed storage and computing resources, job and resource management services, a broker, a grid middleware infrastructure, local resource management and scheduling. These services collectively allow users to execute large-scale applications over many resources in the grid.

The system has N independent and autonomously administered sites (i.e, $Site_1$, $Site_2$,..., $Site_N$). Each site, $Site_i$, has one or more clusters. Each cluster is composed of high performance commodity hardware, software, and networking designed to provide the most economic computing power for a large number of users at a single site. Access to the resources is abstracted via a common interface. Different individuals or organizations own each one of them and they have their own access policy, cost, and mechanism. An example of common usage policy states that external jobs are only run in a cluster or workstation when resources are not in use by the local users [4].

The resource owners manage and control resources using their favorite resource management and scheduling system such as PBS [9], Condor [4] and LSF [13]. These systems are collectively referred to as local resource management and scheduling in Figure 1. These software are completely under the control of the local site administrators. The resources, depending upon a site policy and capabilities, can be run and configured to operate in a variety of different ways.

The information management service keeps track of resource specific information such as machine availability. It also perform resource status detecting and recruiting. It can also interact with the Grid Information Service (GIS) to receive resource specific information (such as hardware and software capabilities).

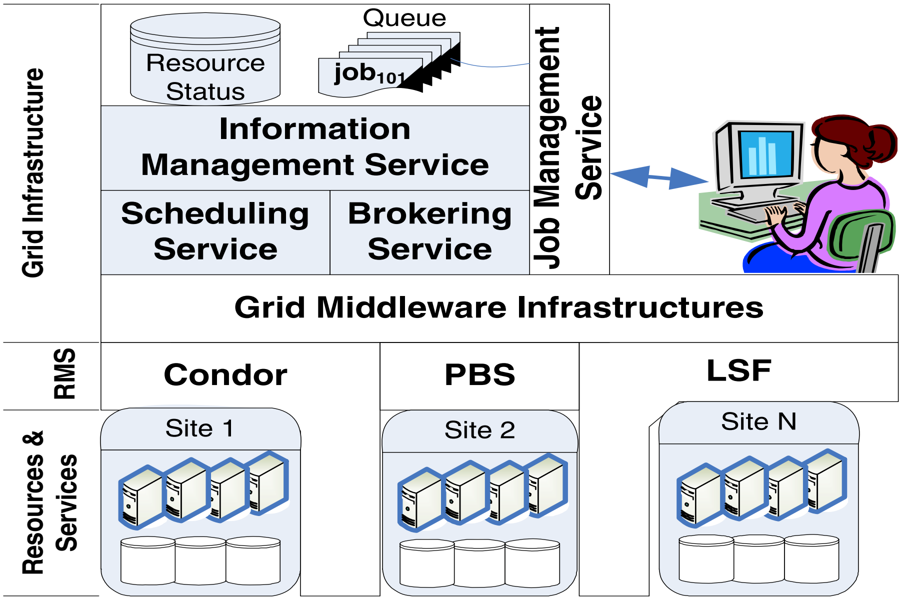


Fig. 1. High throughput grid computing architecture

The job management service (JMS) provides services such as accepting job submissions from users, registers them to the job wait queue, forwarding jobs to resources, monitoring of jobs and prioritizing jobs in queues using a range of techniques. Users interact with the system via the client components. The user generates a request and submits it to the job management service, which in turn invokes the scheduling broker service.

The main task of the broker service is to to find a suitable allocation of resources to the applications by utilizing information from job parameters, resource status, file locations, and system state. Once an application schedule has been chosen, the job scheduling service map it onto the selected resource configuration.

The broker also interacts with grid middleware infrastructures that sit between the users environment and the actual resources. The purpose of the grid middleware infrastructures is to expand the reach of a user to any sort of batch system such as Condor [4] and LSF [13]. The aim is to achieve the same objectives as existing grid computing (e.g., [10]), which is to present users and developers with a simple, uniform, interface to distributed, heterogeneous computing resources.

3 Scheduling Problem

The scheduling problem can be formulated as follows:

Given: A set $J = \{j_1, j_2, ..., j_m\}$ of applications in ready queue and a set $R = \{r_1, r_2, ..., r_n\}$ of available grid resources.

Objectives: The goal of the scheduler is to construct a mapping from elements of J onto elements of R with the goal of optimizing throughput.

The scheduling problem is well known to be intractable and heuristics are commonly used to find a scheduling algorithm that is guided by an objective function that it tries to optimize. Regardless, the solution for the scheduling problem in high throughput grid computing must encompasses the following phases:

1. Brokering - this enables the selection of appropriate resources to each requests.
2. Selection - as high throughput computing allows each user to submit many jobs at the same time, the scheduler must ensure that each user receives a fair allocation of service. Therefore, a good selection algorithm is required to implement a fair allocation of service among the users.
3. Placement - the placement decision is based upon a policy governing the usage of the resources. In high throughput computing, the owners of the resources will rightfully retain ultimate control over their own machines. In addition, the owners may change scheduling policies according to local decisions. For this reason, an opportunistic placement of batch jobs on idle machines for execution is used.
4. Rescheduling - high throughput grid computing resources are shared and their availability and load varies from time to time. Based on the usage policy, an executing job can be suspended and must be re-assigned to the next available resource.

In the rest of the paper, we focus on the job placement component of the scheduling problem. Although HTC is quite an active area of research and opportunistic scheduling policies are in common use in real installations, very little work addressing opportunistic scheduling of batch jobs exist other than ensuring that each batch user receives a fair allocation of service. Recently, a number of opportunistic batch job scheduling approaches have been proposed [8] [3] [1]. The following section gives details of five opportunistic batch job placement approaches.

4 Opportunistic Job Scheduling Policies

Opportunistic scheduling policies for high thrughput computing can be generally divded into space-sharing and time-sharing approaches. In time-sharing policies, processors are temporally shared by jobs. In space-sharing policies, however, processors are exclusively allocated to a single job until its completion. In this section, we review some of the existing opportunistic policies.

4.1 Space Sharing Policies

First Come First Served Scheduling Policy. - The FCFS policy is the most commonly used opportunistic scheduling policy in HTC environments (e.g., Condor [4]). In the FCFS scheduling policy, all new jobs are added to the job wait queue in the order of their arrival. At scheduling point, a job at the head of the queue is assigned to the idle processors where it executes until completion or until it is evicted by the arrival of the workstation owner process. When a job is evicted from the workstation, it is checkpointed and placed at the head of the job wait queue where it waits for re-scheduling.

FCFS is very easy to implement and incurs very little scheduling overheads. Moreover, allocating resource to jobs in the order that the jobs arrive is fair and predictable, but suffers from severe performance degradation, as large jobs may block the execution of the small jobs.

Job Rotate Scheduling Policy. - The job rotate (JR) scheduling policy [8] is essentially the same as as the FCFS policy described above with the exception that when an executing job is evicted, it is placed at the end of the job wait queue. The JR algorithm offers low overhead like FCFS and has shown to be better able to serve short jobs in preference to long jobs regardless of arrival order [8]. It is also easy to implement given existing opportunistic scheduling mechanisms. The problem with this policy is that its based on the frequency of the workstation owner activity. Note that in the absence of eviction, this policy reduces to the FCFS policy.

Multilevel Opportunistic Feedback Policy. - In the Multilevel Opportunistic Feedback (MQF) [1] policy, jobs that have arrived to the system for execution are classified into new jobs and evicted jobs where new and evicted mean that the new job has not received any service since arrival while evicted refers to the fact that the job has received services already. To represent these two classes of jobs, MQF keeps two queues one for holding new jobs that has arrived to the system but not yet scheduled and another queue for holding evicted jobs. We refer to these queues as New and Evicted respectively.

The New queue entries are sorted in the order of the arrival while entries in the Evicted queue are sorted based on the size of the CPU consumption of the jobs from the arrival point in descending order. The job at the head of the Eviction queue is with the smallest CPU consumption while the job at the tail of the queue is with the largest CPU consumption. Scheduling is done such that the job at the head of the New queue is always scheduled first and only if the New is empty then the job at the head of the Evicted queue is assigned to the workstations.

4.2 Time Sharing Policies

Global Round Robin Scheduling Policy. - The Global Round Robin [8] policy uses a central global batch queue, hence the name global round robin, where ready batch jobs are held and scheduling is done round robin on this queue. All new jobs are added to the job wait queue in the order of their arrival. There is a fixed quantum length per job and at each scheduling point the job at the head of the global queue is assigned to an idle workstation where it executes for one-time quanta. When a job completes its quantum of service on a processor, the job is preempted and placed at the tail of the global queue. Then the job at the head of the global queue is scheduled. This process is repeated until all the jobs in the global queue complete execution. When a job is evicted from the workstation before completion, it is checkpointed and placed at the tail of the job wait queue where it waits for re-scheduling.

Proportional Local Round Robin Scheduling Policy. - The Proportional Local Round Robin policy [3] combines the best features of both JR policy [8] and the global RR [8] policy while avoiding their shortcomings. Moreover, it can be used in both dedicated and shared environments, as it is not dependent on the frequency of the evictions.

The proportional local round robin policy is similar to the global RR [8] policy in that there is a central queue for holding unscheduled jobs. Also, all new jobs are added to the job wait queue in the order of their arrival. Similarly, evicted jobs are added at the tail of the user's batch queue. However, proportional local round robin policy differs from global RR [8] policy in that the policy associates with each processor a local ready queue, a quantum length and a multiprogramming level (MPL). The MPL parameter controls the number of jobs that can be actively executing in the workstation at any given time. Moreover, jobs from the user's batch queue are assigned to each idle workstation in groups of equal in number to workstations multiprogramming level.

Each processor applies the RR policy only on the jobs that are in their local queues and preempted jobs are inserted back in the processor's local ready queue and not to the central queue. In addition, we assign variable quantum length to the jobs in the local ready queue based on the proportion of CPU consumption since the arrival of the job to the system. The quantum size, , of the job j is computed as follows:

$$q_j = \frac{T_{max}}{max(1, T^j_{usage}} \times q_{local} \tag{1}$$

where q_{local} is the default workstation quantum size, T_{max} is the maximum CPU time used by resident batch job and is the CPU time so far consumed by job j.

Note that whenever the number of jobs in a workstation falls beyond the multiprogramming level, the workstation can be assigned another job from the user's batch queue. This allows overlapping the execution and the communication processes as such increasing the utilization of the workstation while decreasing scheduling overheads as opposed to the global RR [8] policy.

5 Performance Analysis

We used a discrete event simulation to evaluate the performance of the five scheduling policies. We used the same system and workload models as in [8] [3] which fairly represents the actual activities of the high throughput system. The workload essentially consists of a mix of small and large jobs while the computing resources consist of a mixture of interactive workstations and clusters controlled by batch schedulers (i.e., Condor). If a local job arrives while grid job is executing, the grid job will be preempted and will be moved to another available resource in the pool.

We used the mean slowdown time as chief metrics to compare the performance of the scheduling policies discussed in this paper. We define the mean slowdown time as follows:

$$\text{Slow down} = \frac{\sum_{j=1}^{N} SD(j)}{N} \tag{2}$$

wher $SD(j)$ is the slowdown of job j and defined as the response time of a job j (i.e., the difference between completion and submission times of the job) divided by the service demand of job j as follows:

$$SD(j) = \frac{T_{finish}(j) - T_{submit}(j)}{T_{service}} \tag{3}$$

where $T_{finish}(j)$ completion and submission times of the job, $T_{submit}(j)$ is the submission time of the job and $T_{service}$ is the service demand of job.

In all experiments performed in this paper, a batch strategy is used to compute confidence intervals (at least 31 batches used, each batch contains 3000 jobs). At 90% confidence level, this strategy produced between 4.5% - 10.2% confidence intervals for the five policies. For the sake of clarity, we have not included the confidence interval information on the plots. However, wherever possible, we verified the results in this paper with previously published results. The next section discusses the results of the simulation.

6 Results and Discussion

In this section, we report the preliminary set of results obtained with the aim of testing the effectiveness of the proposed scheduling strategy. Due to space limitations, only a subset of the results is presented.

Figure 2 shows the mean slowdown (vertical axis) as the function of the batch size (horizontal axis) for the five policies. In the figure, "CG" refers to FCFS policy, "Local" refers to Proportional Local Round Robin Scheduling Policy, "Global" refers to Global Round Robin Scheduling Policy, "JR" to the job rotate policy, and "MQF" refers to the multilevel feedback policy.

From the data on the graph we observe that as the batch size increases, the mean slowdown of all the policies also increases. Note that there is no difference between the two policies with respect to the MRT but the slowdown under the MOP policy is much better especially as the batch size increases. The poor performance of the RR policy can be explained by the fact that as the batch size increases the queue length also increases and hence the wait times for short jobs is much greater than the other policies.

The result also demonstrates that timesharing scheduling policies can be used in an opportunistic setting to improve both mean job slowdowns and mean response times with little or no throughput reduction. Also we observed that timesharing scheduling policies performs better than the exiting scheduling policies. Furthermore, this improved slowdown can be achieved without a significant loss of throughput results in a more interactive nature of the system thus increasing its appeal.

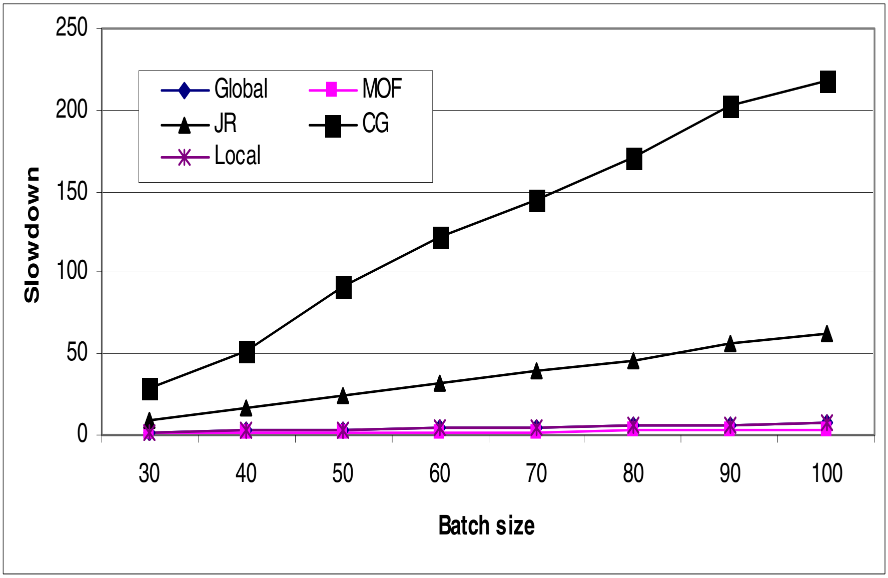


Fig. 2. An Architecture of the job scheduling policy

7 Conclusion and Future Directions

The purpose of high throughput grid computing is to enable community of users (e.g., scientific and engineering) to solve large-scale problems on a pool of shared resources by offering high throughput computational resources [11] in a seamless virtual organization (VO). However, high throughput grid computing is evolving and will ultimately require the support of efficient scheduling strategies. This paper have identified the issues in resource management and scheduling in the emerging high throughput grid computing context. It also surveyed and studied the performance of several space-sharing and time-sharing opportunistic scheduling policies that have been developed for high throughput computing.

Although we have looked at one facet of the scheduling problem of the high throughput grid computing, solutions that encompasses all four aspects (i.e., Brokering, Selection, Placement and Rescheduling) are needed for effectively utilizing the resources while optimizing throughput. We are currently working to achieve this goal.

Acknowledgement. I appreciate the help of Maliha Omar without whom this paper would not have been completed. This research is partially funded by Deakin University.

References

1. J. H. Abawajy. An Opportunistic Job Scheduling Policy for High Throughput Computing Environments. In *Proceedings of the 11th IEEE International (ICPAD'02)*, pages 336–343, 2002.
2. J. H. Abawajy. Survey of Grid Resource Scheduling Approaches. Technical report, Deakin University, 2005.
3. Jemal. H. Abawajy. Preeemptive job scheduling policy for distributively-owned workstation clusters. *Parallel Processing Letters*, 14(2):255–270, 2004.
4. J. Basney and M. Livny. Managing network resources in condor. In *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 298–299, San Francisco, California, August 2000.
5. R. Buyya, Abramson D., and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings of the HPC ASIA2000, the 4th International Conference on High Performance Computing in Asia-Pacific Region*, San Francisco, California, 2000.
6. Ian Foster. The grid: A new infrastructure for 21st century science. *Physics Today*, 55(2):42–47, 2002.
7. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.
8. G. D. Ghare and S. T. Leutenegger. Improving Small Job Response Time for Opportunistic Scheduling. In *Proceedings of the 8th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS'00)*, pages 557–564, August 2000.
9. Robert L. Henderson. Job scheduling under the portable batch system. In *Job Scheduling Strategies for Parallel Processing*, pages 279–294. Springer-Verlag, 1995.
10. I. Foster and C. Kesselman. Globus: A Toolkit-Based Grid Architecture. In *The Grid: Blueprint for a Future Computing Infrastructure*, pages 259–278. Morgan Kaufmann, 1998.
11. Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
12. Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.
13. Ming Q. Xu. Effective metacomputing using LSF multicluster. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 100 – 106. IEEE Computer Society, 2001.

High Performance Task Scheduling Algorithm for Heterogeneous Computing System

E. Ilavarasan, P. Thambidurai, and R. Mahilmannan

Department of Computer Science & Engineering and Information Technology,
Pondicherry Engineering College, Pondicherry – 605014, India
eilavarasan@yahoo.com

Abstract. A key issue in obtaining high performance from a parallel program represented by a Directed A-cyclic Graph (DAG) is to efficiently mapping it into the target system. The problem is generally addressed in terms of task scheduling, where the tasks are the schedulable units of a program. The task scheduling problems have been shown to be NP-complete in general as well as several restricted cases. In order to be of practical use for large applications, scheduling algorithms must guarantee high performance by minimizing the schedule length and scheduling time. In this paper we propose a new task-scheduling algorithm namely, High Performance task Scheduling (HPS) algorithm for heterogeneous computing system with complexity O (v + e) (p+ log v), which provides optimal results for applications represented by DAGs. The performance of the algorithm is illustrated by comparing the schedule length, speedup, efficiency and the scheduling time with existing algorithms reported in this paper. The comparison study based on both randomly generated graphs and graphs of some real applications shows that HPS algorithm substantially outperforms existing algorithms.

1 Introduction

Heterogeneous Computing (HC) system is a suite of distributed processors interconnected by high-speed networks, thereby promising high speed processing of computationally intensive applications with diverse computing needs. A well-known strategy behind efficient execution of a huge application on HC system is to partition it into multiple independent tasks and schedule such tasks over a set of available processors. A task-partitioning algorithm takes care of efficiently dividing an application into tasks of appropriate grain size and an abstract model of such a partitioned application can be represented by a Directed A-cyclic Graph (DAG). This paper deals with DAG structured parallel applications. Each task of a DAG corresponds to a sequence of operations and a directed edge represents the precedence constraints between the tasks. Each task can be executed on a processor and the directed edge shows transfer of relevant data from one processor to another. Task scheduling can be performed at compile-time or at run-time. When the characteristics of an application, which includes execution times of tasks on different processors, the data size of the communication between tasks, and the task dependencies, are known a priori, it is represented with a static model. The objective function of this problem is to map the tasks on the processors and order their execution so that task precedence requirements

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 193 – 203, 2005.

are satisfied and a minimum overall completion time is obtained. The problem of scheduling of tasks with required precedence relationship, in the most general case, has been proven to be NP-complete [1] [2] and optimal solutions can be found only after an exhaustive search. The motivation behind our work is to develop a new task-scheduling algorithm to deliver high performance in terms of both performance metrics (schedule length ratio, speedup, efficiency) and a cost metric (scheduling time). We have improved the work done in [5] [6] and proposed a new task scheduling algorithm.

The rest of the paper is organized as follows: In the next section, we define the task scheduling problems. In Section 3 we present the related works, Section 4 introduces HPS algorithm and Section 5 provides performance analysis and discussions. Finally Section 6 concludes the paper with some final remarks.

2 Task Scheduling Problems

A scheduling system model consists of an application, a target computing system and criteria for scheduling. An *application* program is represented by a Directed Acyclic Graph (DAG), $G=(V, <, E)$, where $V=\{v_i, i=1...n)$ is the set of *n* tasks. < represents a partial order on V. For any two tasks v_i, $v_k \in V$, the existence of the partial order $v_i < v_k$ means that v_k cannot be scheduled until task v_i has been completed, hence v_i is a predecessor of v_k and v_k is a successor of v_i. The tasks executions of a given application are assumed to be non-preemptive. E is the set of directed edges. Data is a $n \ x \ n$ matrix of communication data, where $data_{i,k}$ is the amount of data required to be transmitted from task v_i to task v_k. In a given task graph, a task without any parent is called an *entry task* and a task without any child is called *exit task*. Without loss of generality, it is assumed that there is one *entry task* to the DAG and one *exit task* from the DAG. In an actual implementation, we can create a pseudoentry task and pseudoexit task with zero computation time and communication time.

Heterogeneous computing system consists of a set P = {p_j : j =0,…, m-1} of *m* independent different types of processors fully interconnected by a high-speed arbitrary network. The bandwidth (data transfer rate) of the links between different processors in a heterogeneous system may be different depending on the kind of the network. The data transfer rate is represented by an $m \ x \ m$ matrix, $R_{m \ x \ m}$. The data transfer rate for each link is assumed to be 1.0 and hence communication cost and amount of data to be transferred will be the same. W is a $n \ x \ m$ computation cost matrix in which each w_{ij} gives the Estimated Computation Time *(ECT)* to complete task v_i on processor p_j where 0<=i<n and 1<=j<=m. The *ECT* value of a task may be different on different processor depending on the processors computational capability. The communication cost between two processors p_x and processor p_y, depends on the channel initialization at both sender processor p_x and receiver processor p_y in addition to the communication time on the channel. This is a dominant factor and can be assumed to be independent of the source and destination processors. The channel initialization time is assumed to be negligible. The communication cost of the *edge(i,k),* which is for transferring data from task v_i (scheduled on processors p_x) to task v_k (scheduled on processor p_y) is defined by

$$C_{i,k} = data_{\ i,k} / R_{\ x,y} \tag{1}$$

Otherwise, $C_{i,k}$ = 0 when both the tasks v_i and v_k are scheduled on the same processor. A task graph with 10 tasks, and its computation cost matrix given in [6] are shown in Fig.1 and Table 1.

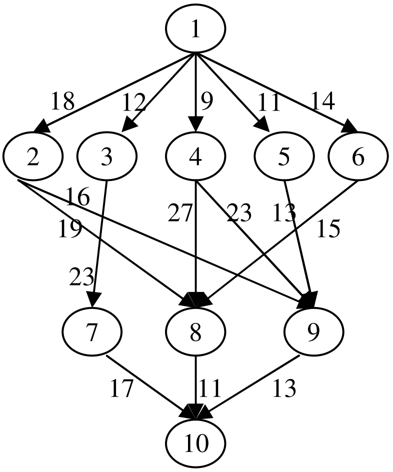


Fig. 1. Task graph with 10 tasks given in [6]

Table 1. Computation cost matrix given in [6]

Task	P_1	P_2	P_3
1	14	16	9
2	13	19	18
3	11	13	19
4	13	8	17
5	12	13	10
6	13	16	9
7	7	15	11
8	5	11	14
9	18	12	20
10	21	7	16

Let $ST(v_i,p_j)$ and $EFT(v_i,p_j)$ are the Earliest Start Time and Earliest Finish Time of task v_i on p_j, respectively. For the entry task $\mathrm{v}_{\mathrm{entry}}$, $EST(v_{entry}, p_j) = 0$, and for the other tasks in the graph, the *EST* and *EFT* values are computed recursively, starting from the entry task, as shown in Eq. (2) and (3). In order to compute the *EFT* of a task v_i, all immediate predecessor tasks of v_i must have been scheduled.

$$EST(v_i,p_j) = \max \{\mathrm{avail}[j], \max(\mathrm{AFT}(\mathrm{v_t}+\mathrm{C_{t,i}}))\} \text{ , Where } \mathrm{v_t} \in \mathrm{pred}(\mathrm{v_i}) \quad (2)$$

$$EFT(v_{i,},p_j)=\mathrm{W_{ij}}+\mathrm{EST}(\mathrm{v_i},\mathrm{p_j}) \quad (3)$$

Where $pred(v_i)$ is the set of immediate predecessor tasks of task v_i and *avail[j]* is the earliest time at which processor p_j is ready for task execution. If v_k is the last assigned task on processor p_j, then *avail[j]* is the time that processor p_j completed the execution of the task v_k and it is ready to execute another task when we have a non insertion-based scheduling policy. The inner max block in the *EST* equation returns the ready time, i.e., the time when all the data needed by v_i has arrived at processor p_j. After a task v_t is scheduled on a processor p_j, the earliest start time and the earliest finish time of v_t on processor p_j is equal to the actual start time $AST(v_t)$ and the actual finish time $AFT(v_t)$ of task v_t, respectively. After all tasks in a graph are scheduled, the schedule length (i.e. the overall completion time) will be the actual finish time of the exit task v_{exit}. Finally the schedule length is defined as

$$Schedule\ Length = \max\{AFT(v_{exit})\} \quad (4)$$

The objective function of the task-scheduling problem is to schedule the tasks of an application to processors such that its schedule length is minimized.

3 Related Works

Efficient application scheduling is critical for achieving high performance in heterogeneous computing system, because of its key importance on performance, the scheduling problem has been extensively studied and various heuristics have been proposed in the literature [3-12]. These heuristics are classified into a variety of schemes such as priority-based [4,5,6], cluster-based [7], guided random search based [8] and task duplication based schemes [9,10,11].

Priority-based schemes [5,6,7] assume a priority for each task that is utilized to assign the tasks to the different processors. Priorities based scheduling algorithms, such as Mapping Heuristics (MH) [4], Levelized Min Time (LMT) [5], Heterogeneous Earliest Finish Time (HEFT) [6] and Critical-Path-On a Processor (CPOP) [6] have been proposed in the literature for heterogeneous systems. The complexity of MH, LMT, HEFT, and CPOP algorithms is O (v^2 x p), O (v^2 x p^2), O (v^2 x p) and O (v^2 x p) respectively. HEFT and CPOP algorithms are proved to be improvement over MH and LMT algorithms in terms of average Schedule Length Ratio (SLR), speedup, and run time. We have chosen the recently proposed algorithms [5,6] for improvement.

4 High Performance Task Scheduling (HPS) Algorithm

In this section we present the proposed HPS algorithm. The framework of the HPS algorithm is shown in Fig. 2. The algorithm consists of three phases, namely, level sorting, task prioritization, and processor selection. The detailed explanation of the HPS algorithm is given below:

In the level-sorting phase, the given DAG is traversed in a top-down fashion to sort task at each level in order to group the tasks that are independent of each other. As a result, tasks in the same level can be executed in parallel. Given a DAG $G = (V, E)$, level 0 contains entry tasks. Level *i* consist of all tasks v_k such that, for all *edges* (v_j, v_k), task v_j is in a level less than *i* and there exists at least one *edge*(v_j,v_k) such that v_j is in level *i-1*. The last level comprises of some of exit tasks.

In the task prioritization phase of the algorithm, priority is computed and assigned to each task. The attributes used to calculate the priority of a task are *Down Link Cost (DLC)*, *Up Link Co st (ULC)* and *Link Cost (LC*) of the task. The *DLC* of a task is the maximum communication cost among all the immediate predecessors of the task. The *DLC* for all task at *level 0* is 0; for all other tasks at *level l*, the *DLC* is computed by using Eq. (5)

$$DLC(v_j)=Max\{C_{i,j}\}, \text{ where } i=1 \text{ to } x, \text{ and 'x' is the number of immediate parents } v_j \quad (5)$$

The *ULC* of a task is the maximum communication cost among all the immediate successors of the task. The *ULC* for *exit task* is 0; for all other tasks at *level l*, it is computed by using Eq. (6)

$$ULC(v_j)=Max\{C_{j,k}\} \text{ where } k=1 \text{ to } x, \text{ and 'x' is the number of immediate child's } v_j \quad (6)$$

```
Read the DAG, associated attributes values, and the number of processor P;
 For each level $L_i$ do
 Begin
    Initialize the priority queue with entry tasks in level ($L_i$);
    For all tasks $v_k$ in the queue do
      Begin
         $LC(v_k) = max \{LC(v_j)\} + ULC(v_k) + DLC(v_k),$
                        Where $v_j$  $pred(v_k)$;
         Update the tasks in priority queue based on LC;
      End;
    While there are unscheduled tasks in the queue do
      Begin
         Select the highest priority task, $v_k$ from the queue for scheduling;
         For each processor $p_k$ in the processor set P do
           Begin
              Compute EFT ($v_k$,$p_k$) value using insertion based
                  Scheduling policy;
              Assign the task $v_k$ to the processor $p_k$, which minimizes the EFT;
           End;
      End;
   End
```

Fig. 2. HPS Algorithm

The *LC* of a task is the sum of *DLC, ULC* and maximum *LC* of all its immediate predecessor tasks. The *LC* of a task is calculated by using Eq. (7)

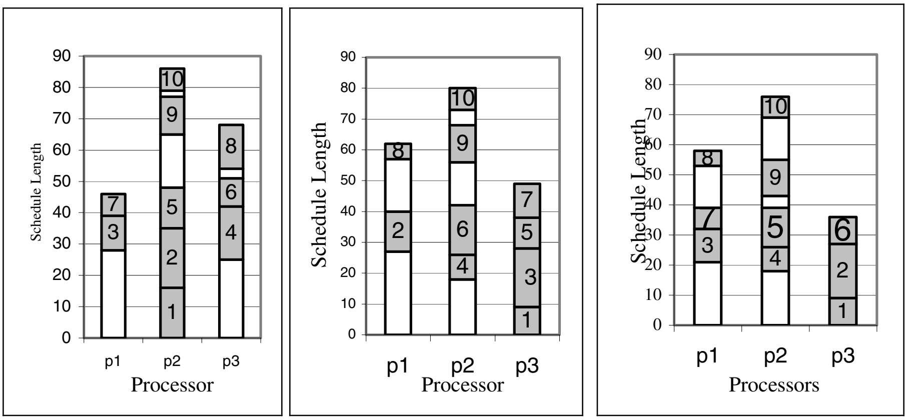
$$LC(v_j)=\left\{\begin{array}{l} ULC(v_j)\ For\ entry\ task \\ \max\{LC(v_i)\}+ULC(v_j)+DLC(v_j)\ For\ all\ other\ tasks \\ v_i \varepsilon pred(v_j) \\ \end{array}\right\} \tag{7}$$

Priority is assigned to all task at each *level i*, based on its *LC* value. At each level, the task with highest *LC* value receives the highest priority followed by task with next highest *LC* value and so on in the same level. While assigning priority if two tasks are having same *LC*, priority will be given according to the order in queue. For example, for the task graph in Fig. 1, the LC value for task 1 is 18 and for task 2, it is {max (18)+18+19}=55. For task 8, LC value is {max (55,54,47)+27+11}= 93. Similarly LC value is calculated for all the tasks of the graph given in Fig. 1.

In the processor selection phase, the processor, which gives minimum *EFT* for a task is selected for executing that task. It has an insertion-based policy, which considers the possible insertion of a task in an earliest idle time slot between two already scheduled tasks on a processor. At each level, the earliest start time and earliest finish time of each task on every processor is computed using Eq. (2) and (3). Calculation of *EST* and *EFT* value for the task graph in Fig. 1 is illustrated below: For example, for the task 8, $EST(8, P_1)$ = max {39, max (46,53,51)} = 53, $EFT(8, P_1)$

= 5+53=58, *EST(8, P_2)* = max{39, max(46, 51, 39)}=51, *EFT(8, P_2)* = 15+51=66, *EST(8, P_3)* = max{36, max(36, 53, 36)}=53 and *EFT(8, P_3)* =14+53=67. Similarly EST and EFT value for all task of the graph given in Fig. 1 is calculated.

The tasks are selected for execution based on their priority value. Task with highest priority is selected and scheduled on its favorite processor for execution followed by the next highest priority task in that level. Similarly all the tasks in all the levels are scheduled on to the suitable processors. The processors selected for executing the tasks of task graph in Fig. 1 is as follows: For example, task 1 is the entry task; hence its data arrival time is 0 and P_3 gives the minimum *EFT* for task 1. Hence processor P_3 is selected for executing task 1. For task 2, the data arrival time from its predecessor (task 1 in P_3) is 9 and the *EFT* of this task on P_1, P_2 and P_3 are 40, 36, and 27. Since P_3 gives minimum *EFT* for task 2, it is selected for executing task 2. Similarly the processor best suited to execute every task in the graph given in Fig. 1 is determined.



(a) CPOP Algorithm (b) HEFT Algorithm (c) HPS Algorithm

Fig. 3. The schedule length generated by CPOP, HEFT and HPS algorithms

The time complexity of HPS algorithm is equal to *O (v + e) (p + log v)* where *v* is the number of tasks, *e* number of edges and *p* number of processors. For implementation, we used breadth first search for level sorting which takes *O (v + e)* time complexity. A binary heap was used to implement the priority queue, which has time complexity of *O (log v)*. Each task in the priority queue is checked with all the p processors in order to select a processor that gives the earliest finish time. Hence the complexity of the algorithm is *O* (*v* + *e*) (*p*+*log v*). As an illustration, Fig. 3 presents the schedules obtained by the CPOP, HEFT and HPS algorithms for the sample DAG of Fig. 1. The schedule length, which is equal to 76, is shorter than the schedule lengths of the related work; specifically, the schedule lengths of HEFT, CPOP and LMT Algorithms are 80, 86, and 91 respectively.

5 Performance Analyses and Discussion

In this section, we present the comparative evaluation of proposed HPS algorithm and the existing algorithms for heterogeneous systems such as LMT, HEFT and CPOP for

DAGs with various characteristics by simulation. For this purpose, we consider two sets of graphs as the workload for testing the algorithms: randomly generated task graphs and the graphs that represent some of numerical real world problems.

5.1 Randomly Generated Application Graphs

A random task graph generator has been developed, which allows the user to generate a variety of test DAGs with various characteristics that depends on several input parameters and they are *number of tasks in the graph (v), out degree ( ), in degree ( ), shape parameter of a graph ( )* and Communication to Computation Ratio (CCR) and Range percentage of computation cost (η). By varying value we can generate different shape of the task graph. The height of the graph is randomly generated from a uniform distribution with a mean value equal to $\sqrt{v}/\alpha$ and the width for each level is randomly selected from a uniform distribution with mean value equal to $\sqrt{v}*\alpha$. A dense graph (shorter graph with high parallelism) and a longer graph (low parallelism) can be generated by selecting >>1.0 and <<1.0 respectively. CCR is the ratio of the average communication cost to the average computation cost. The computation intensive applications may be modeled by assuming CCR = 0.1, whereas data intensive applications may be modeled assuming CCR = 10.0. Range percentage of computation costs on processors, (η). It is basically the heterogeneity factor for processors speeds. A high percentage value causes a significant difference in a task's computation cost among the processors and a low percentage indicates that the expected execution time of a task is almost equal on any given processor in the system. The average computation cost of each task v_i in the graph, i.e., W_i, is randomly selected from a uniform distribution with range [0 , 2*Wdag], where Wdag is the average computation cost of the given graph, which is set randomly in the algorithm. Then, the computation cost of each task v_i on each processor p_j in the system is randomly set from the following range:

$$W_i*(1-\eta/2)<=W_{i,j}<=W_i*(1+\eta/2) \tag{8}$$

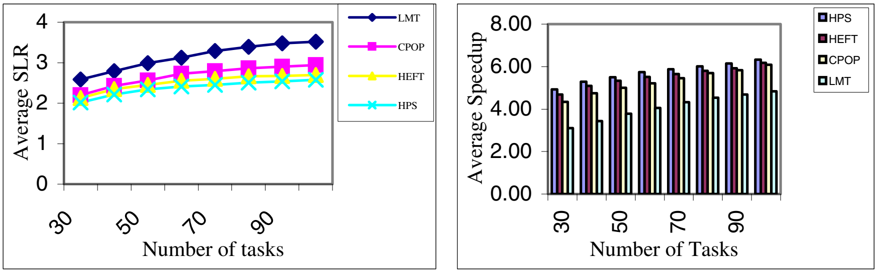
For experiments, we set the following range of values for the parameters. v = {30,40,50,60,70,80,90,100}, = {0.5,1.0,2.0}, = {1,2,3,4,5}, = {1,2,3,4,5}, CCR= {0.1,0.5,1.0,5.0,10.0} and η={0.1,0.5,1.0}.

5.2 Experimental Results

The experimental results are organized in two major test suites.

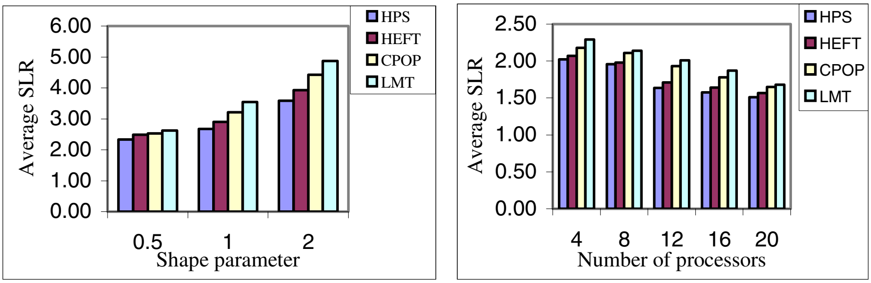
Test Suite 1: In this test suite, we evaluated the quality of schedules generated by the algorithms with respect to the graph characteristics values given in section 5.1. We have generated around 620 random task graphs with different characteristics and scheduled these graphs on to a HC system consists of 15 processors. The average SLR and speedup generated by each of the algorithm are plotted and are shown in Fig. 4a and Fig. 4b. Each data point in the reported graph is the average of the data obtained in 30 experiments. The average SLR value based ranking (starting with minimum ending with maximum) of the algorithms is {HPS, HEFT, CPOP, and LMT} and the Speedup value based ranking (starting with maximum and ending with minimum) of the algorithms is {HPS, HEFT, CPOP, and LMT}. The average SLR value of the HPS algorithm on all generated graphs is better than the HEFT algorithm by 6 percent, the CPOP algorithm by 13 percent and the LMT algorithm by 33 percent.

The performance of the algorithm is also evaluated with respect to the graph structure, by varying the α value from 0.5, 1.0 and 2.0 and it is shown in Fig. 5a. The simulation studies confirm that HPS algorithm substantially outperforms reported algorithms. Further, we evaluated the efficiency of the algorithms by scheduling task graphs consists of fixed numbers of tasks (120) on to HC system consists of varying number of processors (4,8,12,16,20). For this experiment, we have used 100 numbers of randomly generated task graphs. The results obtained by this experiment are shown in Fig.5b. As expected the average SLR is reduced while increasing the number of processors and at the same time HPS outperforms LMT, CPOP and HEFT algorithms.



(a) Average SLR (b) Average speedup

Fig. 4. Performance of the algorithms for random generated task graph

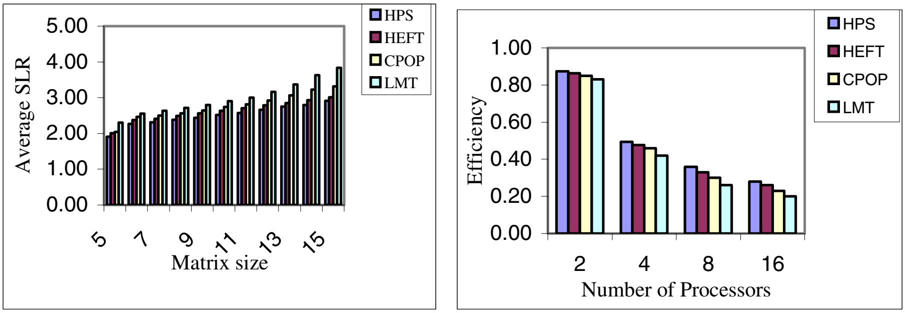


(a) Average SLR for varying (**α**) (b) For varying number of processors

Fig. 5. Performance of the algorithms for shape parameter and varying processors

Test Suite 2: In this test suite, we are considering application graphs of three real world problems such as Gauss Elimination algorithm, Fast Fourier Transformation and molecular dynamics code given in [6][12]. For the experiment of Gauss elimination applications, heterogeneous computing systems with five processors and CCR and ECT value given in section 5.1 are used. Since the structure of the application is known, the parameters such as number of tasks, in degree and out degree are not needed. A new parameter matrix size (*m*) is used in place on number of tasks (*v*). The total number of task in a Gaussian elimination graph is equal to $(m^2+m-2)/2$. We evaluated the performance of the algorithms at various matrix sizes from 5 to 15 with an increment of one. The smallest size graph in this experiment has 14 tasks and the largest one has 119 tasks. The simulation results are given in Fig. 6a and Fig. 6b for various matrix sizes shows that HPS outperforms other reported algorithms. For FFT related experiment the graph characteristic such as CCR, ECT value given in section 5.1 is used. Since the structure of the application is known, other parameters such as number of tasks, in degree and out degree are not needed. The number of data points in FFT is another parameter in our experiments, which varies from 2 to 32 incrementing powers of 2. Fig.7a and Fig.7b presents the average SLR values for FFT graphs at various sizes of input points.

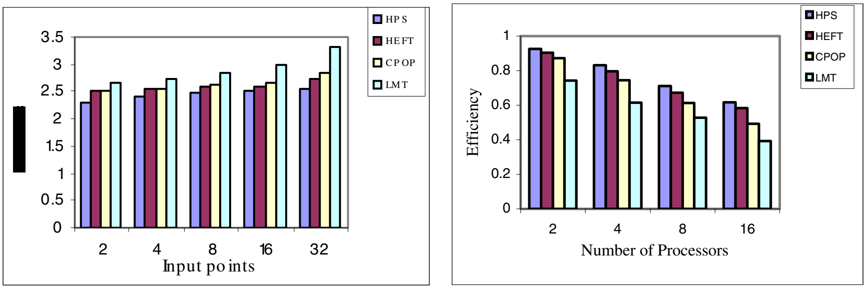
The combined column shows the percentage of graphs in which the algorithm on the left gives a better, equal or worse performance than all other algorithms combined. The ranking of the algorithms based on the occurrences of best results is HPS, HEFT, CPOP and LMT.



(a) Average SLR

(b) Efficiency

Fig. 6. Average SLR and efficiency comparison for Gaussian Elimination Graphs



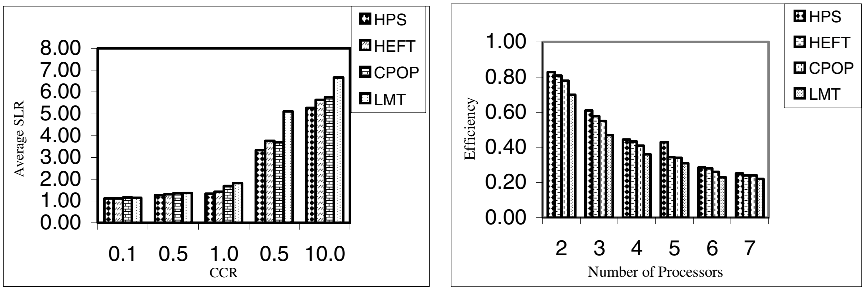
(a) Average SLR

(b) Efficiency

Fig. 7. Average SLR and efficiency comparison for FFT applications

The task graph of the molecular dynamics code given in [6][12] is also part of our experiment since it has an irregular task graph. Since the number of task is fixed in the application and the structure of the application is known, the graph characteristics CCR and ECT values given in section 5.1 are alone used. Fig. 8a and Fig. 8b shows the performance of the algorithms (Average SLR and Efficiency) with respect to five different CCR values when the number of processor is equal to seven. The simulation results shows that HPS algorithm substantially outperforms HEFT, CPOP and LMT algorithms.

With respect to the experiments conducted for the above study, we have counted the number of times that each scheduling algorithm in the experiments produced better, worse or equal schedule length than every other algorithm. Each cell in Table 2 indicates the comparison results of the algorithm on the left with the algorithm on the top.



(a) Average SLR (b) Efficiency

Fig. 8. Average SLR and efficiency comparison for Molecular dynamics code

Table 2. Pair-Wise Comparison of the Scheduling Algorithms

Algorithm		HPS	HEFT	CPOP	LMT	COMBINED
HPS	Better	*	441	498	607	71%
	Equal		198	140	72	19%
	Worse		81	82	41	10%
HEFT	Better	81	*	556	657	61%
	Equal	198		38	27	12%
	Worse	441		126	36	27%
CPOP	Better	82	126	*	634	39%
	Equal	140	38		40	10%
	Worse	498	556		46	51%
LMT	Better	41	36	46		7%
	Equal	72	27	40	*	9%
	Worse	607	657	634		84%

6 Conclusion

The HPS algorithm proposed here has been proven to be optimal for DAGs by reducing the schedule length with low complexity. The performance of this algorithm has been observed experimentally by using large set of randomly generated task graphs with various characteristics and application graphs of several real world problems such as Gaussian Elimination, Fast Fourier Transformation and Molecular dynamics code. The simulation result confirms that HPS algorithm substantially better than the existing algorithms such as LMT, CPOP and HEFT in terms of performance matrices (average schedule length ratio, speedup, efficiency, frequency of best results) and scheduling time. The complexity of HPS algorithm is $O\ (v + e)\ (p{+}log\ v)$, which is less when compared with other scheduling algorithms reported in this paper. We have planned to extend this algorithm for arbitrary-connected networks and also for the dynamic networks.

References

[1] R.L. Graham, L.E. Lawler, J.K. Lenstra, and A.H. Kan, “Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey”, *Annals of Discrete Mathematics*, pp. 287-326, 1979.

[2] T. Cassavant and J.A. Kuhl, “Taxonomy of Scheduling in General Purpose Distributed Memory Systems”, *IEEE Trans. Software Engineering*, vol.14, no. 2, pp. 141-154, 1988.

[3] C.C. Hui and S.T. Chanson, “Allocating Task Interaction Graphs to Processors in Heterogeneous Networks”, *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 9, pp. 908-926, Sept. 1997.

[4] H.EI-Rewini and T.G.Lewis, “Scheduling Parallel Program Tasks onto Arbitrary Target Machines,”, *Journal of parallel and Distributed Computing*, vol.9, pp.138-153, 1990.

[5] M.Iverson, F.Ozguner and G.Follen, “Parallelizing Existing Applications in a Distributed Heterogeneous Environments”, *Proc. Heterogeneous Computing Workshop*, pp.93-100, 1995

[6] H. Topcuglou, S. Hariri and M.Y. Wu, “Performance Effective and Low-Complexity Task Scheduling for Heterogeneous Computing”, IEEE Trans. on Parallel and Distributed Systems, vol. 13, No.3, Feb’ 2002.

[7] M. Kafil and I. Ahmed, “Optimal Task Assignment in Heterogeneous Distributed Computing Systems,” *IEEE Concurrency*, vol. 6, no. 3, pp. 42-51, July- Sept. 1998.

[8] M.K. Dhodhi, I.Ahmad, A. Yatama, “An Integrated Technique for Task Matching and Scheduling onto Distributed Heterogeneous Computing Systems”, *Journal of parallel and distributed computing* “, 62,pp.1338-1361, 2002.

[9] Atakan Dogan and Fusun Ozguner, “LDBS: A Duplication Based Scheduling Algorithm for Heterogeneous Computing Systems”, *Proc. Int’l. Conf. Parallel Processing (ICPP’02).*

[10] Sanjeev Basker and Prashanth C.SaiRanga, “Scheduling Directed A-cyclic Task Graphs On Heterogeneous Network of Workstations to Minimize Schedule Length”, *Proc. ICPPW*, 2003.

[11] Rashmi Bajaj and D.P. Agrawal, “Improving Scheduling of Tasks in a Heterogeneous Environments,”, *IEEE Trans. on Parallel and Distributed Systems*, vol. 15, No.2, Feb’ 2004.

[12] S.J. Kim and J.C. Browne, “A General Approach to a Mapping of Parallel Computation upon Multiprocessors Architectures,” *Proc. int’l Conf. parallel processing*, vol. 2, pp. 1-8,1988.

Execution Environments and Benchmarks for the Study of Applications' Scheduling on Clusters

Adam K.L. Wong and Andrzej M. Goscinski

School of Information Technology, Deakin University,
Geelong, Vic 3216, Australia
{aklwong, ang}@deakin.edu.au

Abstract. In this paper, we have demonstrated how the existing programming environments, tools and middleware could be used for the study of execution performance of parallel and sequential applications on a non-dedicated cluster. A set of parallel and sequential benchmark applications selected for and used in the experiments were characterized, and experiment requirements shown.

1 Introduction

Many parallel applications can be executed on very cost-effective parallel computer systems, non-dedicated clusters, which are already owned by many universities, and business and industry institutions. To do this, they can be used of course as dedicated clusters during weekends and at nights. Although individual PCs of such clusters are normally used by their owner users to run sequential applications (local jobs), the cluster as a whole or its subsets could also be employed by users to run parallel applications (cluster jobs) even during working hours. The reason is that PCs in their working environments are on average idle for much more than 50% of time [2, 5, 19]. Therefore, a cluster has the potential of supporting the concurrent execution of a mixture of parallel and sequential applications or a set of parallel applications, which could lead to the improvement of the overall execution performance of applications.

When multiple parallel applications need to share a cluster, both space- and time-sharing scheduling approaches can be used. Static space-sharing [8, 23] is a simple approach that involves finding enough idle computers in a cluster and mapping processes of a parallel application onto these computers. Although owner-users would be protected from any possible performance loss, utilization of cluster computers is usually far from optimal because of fragmentation. For instance, a parallel application needs to wait until enough idle computers are available before it can be started; otherwise, it has to sacrifice the level of its parallelism. Even worse, a sacrificed parallel application cannot utilize the excessive computational power from any occupied but lightly loaded computer where a sequential application is running.

Time-sharing is intrinsically supported in a cluster via local scheduling. In this case, the local scheduler is responsible for time sharing of the CPU among all the processes which have been allocated to that computer. Processes from a parallel application can be placed into some or all of the computers in the cluster depending on the required parallelism. However, processes belonging to the same parallel application would not be guaranteed to execute at the same time across the computers

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 204 – 213, 2005.

in the cluster. Previous studies [1, 3, 21, 24] have found that if the parallel application is communication intensive, this uncoordinated scheduling of processes would lead to a great loss of performance in the cluster since a process stalls when it communicates with a non-scheduled process. [21] and [17] have presented the results of co-scheduling of multiple parallel applications on a cluster using local scheduling but the results are quite different. [21] shows that co-scheduling of parallel applications on a cluster worsens their execution performance. That result is difficult to assess as the experiment is not described and applications used in the experiment are not defined.

Our research shows that concurrent execution of parallel and sequential applications and concurrent execution of multiple parallel applications on a non-dedicated cluster improves the execution performance of parallel applications, and makes the execution performance of sequential applications only slightly worse [22, 12]. We carried out the study by using the well known and widely used benchmarks: NAS Parallel Benchmarks [15] and BYTE's Unix Sequential Benchmarks [7]. However, [22, 12] mainly address execution aspects, experiments carried out, experimental results achieved and their interpretation. The execution environment and benchmark preparation was only addressed to satisfy the aims of those papers. Because these two elements have formed a wide and general platform of benchmark-based study of high-performance computing on clusters and could be useful to other researchers, we decided to describe them in detail.

The aim of this paper is to show (i) how the existing programming environments, tools and middleware could be used for the study of execution performance of parallel and sequential applications and multiple parallel applications executing concurrently on a non-dedicated cluster; and (ii) the way how the publicly available and frequently used benchmarks should and could be adapted to carry out such performance study.

2 The Performance Study Results Achieved

The results of executing NAS parallel applications with BYTE-based sequential applications of various workloads on a non-dedicated cluster have demonstrated that parallel applications benefit from having its processes migrating from heavily loaded computers to lightly loaded computers which are executing sequential applications [12]. Such a dynamic load-balancing based scheduling of a mixture of parallel and sequential applications works particularly well for both the owner-users and the cluster-user of a non-dedicated cluster when the workload of the computers generated by their corresponding owner-users is low (I/O-Bound sequential workload) and the number of such computers is large. By sharing the computers of owner-users, which are normally not accessible in a dedicated cluster, parallel applications can gain extra processing power to perform `CPU-hungry' computations. On the other hand, owner-users of their computers could suffer from a slight degradation of the execution performance, which tends to be insignificant when the sequential workload of the computers move towards I/O-bound applications and the number of owner-users is large in the cluster. Although the relative slowdown generated in each case may be noticeable to an owner-user, we think that it would be acceptable to most of the users, whose computers are sitting idle for more than 50% of time during working hours.

Contrary to the results obtained by other researchers, we have found that even if a parallel application is communication intensive, there is no performance loss of the parallel application due to uncoordinated communications and synchronizations of processes. Our study of the scheduling of a parallel application executing concurrently with sequential applications does not confirm the results reported in [3, 21, 24], which recommend synchronized scheduling of parallel and sequential applications. We have demonstrated that there is no need for synchronization and that its lack does not decrease the execution performance. Concurrent execution of a parallel application and sequential applications on a cluster did not make the execution performance of a parallel application worse, especially when a slow but commonly found network such as 100 Mbits Ethernet is used.

We have also demonstrated that co-scheduling of parallel applications on cluster does not lead to deterioration of the execution performance as it was shown in [21]. Although we cannot fairly judge [21]'s results due to the incomplete experimental detail provided, we are strongly convinced that their poor performance on co-scheduling for multiple parallel applications could be caused by factors such as the network type in use as well as the physical memories available in the computers.

3 The Execution Environment of the Scheduling Study

In this sections, we describe how to employ a dynamic load balancing system and the implementation of MPI [9] to construct a cluster that can support the execution and scheduling of a mixture of a parallel application and sequential applications.

3.1 The OpenMosix Dynamic Load-Balancing System

Dynamic load balancing is an efficient method of scheduling processes on a cluster as it can provide a unified way to utilize both space- and time-sharing for scheduling processes of sequential and parallel applications. By taking advantage of a process migration facility, allocation of parallel processes to computers of a cluster can be changed dynamically according to the actual workload on each of the computers [11].

The openMosix system [16] is a Linux kernel extension which can turn a network of ordinary computers into an openMosix cluster. It is in fact an open source version of Mosix [6] developed by Barak et al. as a part of the Mosix Distributed Operating System project. In a nutshell, the openMosix/Mosix technology consists of the PPM (Preemptive Process Migration) mechanism and a set of algorithms for adaptive resource sharing. The PPM can migrate any process, at any time, to any available computer in the cluster based on the information provided by resource sharing algorithms or triggered by the users.

There are two resource sharing algorithms used in openMosix/Mosix: dynamic load-balancing and memory ushering. The former can reduce the load difference between pairs of computers by migrating processes from a heavily loaded computer to a lightly loaded one. The memory ushering algorithm is triggered when a computer suffers heavily from paging due to running out of free memory to hold processes. It then overrides the dynamic load-balancing algorithm and attempts to migrate a process to a computer that has sufficient free memory.

3.2 LAM/MPI

Currently, most existing clusters are managed by centralized operating systems such as Unix/Linux and Windows. This implies that executing parallel applications on a cluster requires some support from a run-time environment so that the parallel applications can utilize distributed computers. One of the most important supports is provided by the IPC mechanism that allows processes of a parallel application to communicate. LAM/MPI [4] was selected for our project because it could be used with the openMosix system without too much difficulty.

To take advantage of the higher communication speed between processes on the same SMP computer, LAM provides three different client-to-client transport layers: tcp, usysv and sysv [13]. The tcp transport uses TCP sockets for all interprocess communication. The usysv and sysv transports are multi-protocol, i.e., processes located on the same node communicate via shared memory and processes on different nodes communicate via TCP sockets. When integrating the LAM/MPI into the openMosix, care must be taken to select the tcp transport layer in LAM/MPI. If either the usysv or sysv transport layer is selected, automatic process migration from openMosix would be prevented. We used in our project LAM/MPI-6.5.9 [13] that requires a compile-time selection of such a transport layer communication protocol.

3.3 The Global Scheduling System

Since the openMosix package exists as a kernel patch of the Linux operating system, an openMosix cluster can be constructed by installing a copy of the openMosix-enabled Linux kernel in each of the computers in the selected cluster. Once an openMosix cluster is set up, the executions of computer applications can be started on any computer of the cluster and the distribution and balance of workloads will be done automatically. The current version of the openMosix-enabled Linux kernel used in our project is openMosix-2.4.20 [16]. We have set openMosix to make load balancing decisions based only on the workload of each of the computers.

One way to execute an MPI parallel application on our openMosix cluster is to place all the processes of a parallel application on one computer initially and allow the openMosix system to migrate processes from that computer to other computers in order to balance the workload of the cluster [6]. The actual processor-to-processor communication mechanism for the distributed processes is relied on the IPC subsystem of openMosix rather than the LAM/MPI daemon[1]. The drawback of this approach of executing parallel applications on our openMosix cluster is in the overhead caused by the single MPI daemon in the cluster. Since communications of a migrated process to any other processes are handled by its handler (created by openMosix) located on the machine where it is started, a single MPI daemon for the cluster implies that all handlers are concentrated on one computer, which becomes a communication bottleneck of the processes.

Alternatively, an MPI parallel application can be executed by providing a MPI daemon for each of the computers of the cluster. In such a way, processes from an MPI application are initially placed to different computers and therefore the handlers

[1] The MPI daemons are responsible for process initialization and handling of communications among processes of a parallel application.

created by openMosix for handling inter-process communications would be spread evenly across different computers of the cluster. However, adopting this approach to execute MPI applications in our openMosix cluster imposes a restriction that the identity of the computers being used must be known prior to the execution.

In summary, the system exploits two level scheduling. The higher level scheduling, which offers global scheduling, exploits load balancing that is provided by the openMosix systems. This level only schedules processes of a parallel application taking into consideration load of each individual PC of the cluster. The lower level scheduling offers local scheduling for processes running on a given PC of the cluster.

4 The Selection and Preparation of Application Benchmarks

This section addresses the issues of the selection and the preparation of the benchmark applications for our experiments

4.1 Selection of the Parallel Application

The behaviour and scheduling requirements led us to the specification of program attributes that must be present because they influence the execution performance of a parallel application. These attributes form a basis of the selection of benchmarks for our experiments. They are as follows.

- *Computation attributes*: In general, the problem size of a parallel program is directly proportional to its execution time. It can be broadly classified into a computation bound program and a communication bound program.
- *Communication attributes*: Different parallel programs have different communication features. The common communication features in parallel programs are communication volume and communication pattern.
- *Memory attributes*: The main memory of a program required for its execution affects the scheduling behaviour, as it could lead to memory swapping.
- *Topology attributes*: The topology in process-to-processor mapping of a parallel program defines the size (number of processes) and the structure (the connections of processes) of the program.

Table 1. A classification of the selected parallel applications

Program	**Selection Attributes**			
	Computation	**Comm. Volume**	**Comm. Pattern**	**Topology**
MPI-Povray	Comp. Bound	Low	Point-to-point	Any
PTSP	Comm. Bound	Medium	Point-to-point	Any
EP	Comp. Bound	Negligible	Negligible	Any
LU	Comm. Bound	Medium	Point-to-point	Power-of-2
BT	Comm. Bound	High	Collective	Square-of-n
MG	Comm. Bound	High	Collective	Power-of-2

To evaluate the impact of concurrent execution of parallel and sequential applications on their performance, we carried out an analysis of the NAS programs as well as other parallel applications to identify those that posses the attributes addressed above. We have found that EP, LU, BT and MG, of the NAS Parallel Benchmarks [15], MPI-Porvay [20] and Parallelized TSP [14], and the sequential benchmark: BYTE's Unix Bench [7] can represent real world parallel applications with a broad range of program attributes in terms of computation, communication and topology. We have carefully confined the problem size of the selected programs such that their requirement on main memory during execution would be satisfied and that memory swapping would not occur. The features of these programs are summarized (also using the results presented in [10, 15, 18] for NPB) in Table 1.

4.2 Sequential Applications and the Construction of Workload Benchmarks

To achieve the aim of our research, there was a need to identify and determine the influence of sequential applications with different workloads (ranging from CPU-bound to I/O-bound) executing concurrently with a parallel application on a cluster. The sequential applications must meet the following requirements:

- *Controllability:* The amount of workload of the sequential application must be easily and precisely adjusted.
- *Repeatability:* The exactly same amount of workload of the sequential application can be repeated for different experiments.
- *Durability*: The execution time of the sequential application must be comparable (usually fairly long) with the parallel application when they run concurrently.

Controllability, repeatability and durability can be achieved by using sequential benchmarks. Following our study of sequential benchmarks, we have selected the BYTE's Unix Benchmark Suite [7].

The BYTE's Unix Benchmark Suite consists of a set of sequential applications, which were designed to test the performance of a single-processor computer system in the dimensions such as arithmetic operations, memory operations, disk operations, system calls as well as system loading. Each application contained in the benchmark suite is classified as either I/O-bound or CPU-bound as shown in Table 2.

Table 2. Classification of the BYTE Unix Benchmark applications

Category	**Program**
CPU-bound	dhry2reg, whetstone-double, pipe, spawn, shell, syscall, arithoh, short, int, long, float, double, C, dc, Hanoi
IO-bound	execl, fstime, fsbuffer, fsdisk, context1

To benchmark a computer system using the BYTE Unix Benchmark, users are required to specify a fixed time period for the execution of a sequential application. Performance of the computer system is calculated based on the amount of work that the sequential application has completed within the declared time period [7]. Therefore, an assumption made in each measurement is that the sequential application being tested must fully occupy the CPU of a computer.

Our earlier performance studies with the BYTE suite has shown that the use of a fixed time period to measure system performance would produce incorrect results if time sharing of a single computer by multiple applications is in place. This occurs because running other applications concurrently with a sequential application from the benchmark suite on a computer leads to the reduction of the execution time allocated to that sequential application. Therefore, this execution model of the benchmark suite did not allow us to carry out our scheduling experiments by mixing and running a parallel application and sequential applications from the benchmark suite directly on a computer cluster. Consequently, we modified the execution model of the benchmark suite to a workload based model, in the sense that each of the sequential applications would finish an amount of work in the time period of execution of a parallel application. Having the applications from the benchmark suite changed into workload based, a set of sequential benchmarks can be constructed by choosing different applications from the BYTE suite and packing them together into groups according to the particular workloads required.

The workload composition (WC) is made up of a number of CPU-bound and I/O-bound applications that can be varied to generate different workloads with varying number of CPU-bound and I/O-bound applications, i.e., WC = {Seq_1, Seq_2,…,Seq_n}, where Seq_i is any sequential application from the BYTE's Unix Benchmark suite with i = 1..n and n is any positive integer. We constructed three sets of sequential benchmarks with different workload compositions, Seq_{IO}, Seq_{IB} and Seq_{CPU}. Seq_{IO}, Seq_{IB} and Seq_{CPU} represent I/O-bound, In-Between and CPU-bound sequential benchmarks with a workload of 20%, 50% and 80% CPU utilization, respectively as listed in Table 3. The constructed workloads model demanding users and represent heavy utilization of cluster computers.

Table 3. Workload Compositions

Workloads	**Components**
Seq_{IO}	fstime, idle-burst, fsbuffer, idle-burst, fsdisk, idle-burst
Seq_{IB}	execl, context1, spawn, fsdisk, whetstone-double, C, fstime, syscall, hanoi, dc
Seq_{CPU}	dhry2reg, whetstone-double, pipe, spawn, shell, syscall, arithoh, int, double, C, execl, context1, hanoi, short, float

5 Performance Study of Applications on Our openMosix Cluster

Our openMosix cluster consists of 16 Pentium II (350MHz) computers, each with 383 Mbytes of main memory. The computers are connected together by a 100Mbit/s Fast Ethernet network. The following subsections address the different issues related to the performance study of parallel applications in clusters.

The Influence of the Dynamic Load-Balancing System. When the openMosix dynamic load-balancing system is employed to schedule processes of parallel applications, execution overhead induced could primarily come from two sources: process migration and load-monitoring of computers. Because the former depends on the workload and the number of computers used as well as the actual number of migrations performed, it is difficult to be measured. The latter can be measured easily at any given cluster. This overhead is the additional time needed to run the openMosix software without any process migration occurring and is purely caused by the load-monitoring unit of the openMosix software. Figure 2 shows the results of two executions (with and without openMosix) of the MPI-Povray application on different number of computers and the results show that such overhead induced is insignificant.

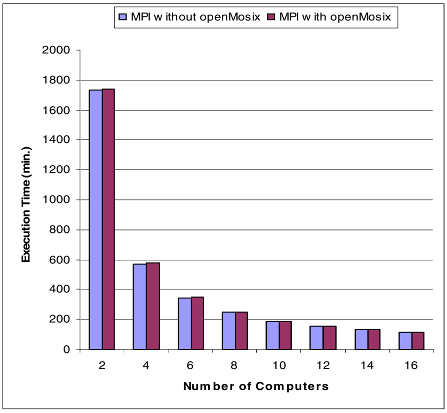


Fig.2. Executions of MPI-Povray on our openMosix Cluster

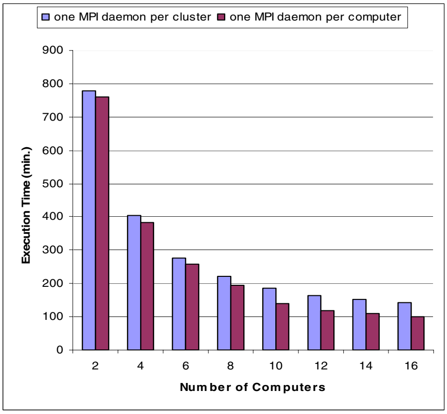


Fig.3. Executions of Parallelized TSP on our openMosix Cluster

The Influence of MPI Daemons. There are two different ways of executing a parallel program on our openMosix: one MPI daemon per cluster and one MPI daemon per computer. Figure 3 shows the results of the execution of Parallelized TSP on different number of computers, with a problem size of 21 cities, in the two different ways. It can be observed that the overhead induced in the one daemon per cluster system increases as the number of computers increases.

The Influence of Program Topology. Some parallel programs are topology specific; this means that their executions are restricted to some well defined number of processors and specific process-to-processor mappings. For the selected NPB programs, the MG and LU programs can only be compiled for running with a power-of-2 number of processes, the BT program for running with a square number of processes, and EP for running with any number of processes [15].

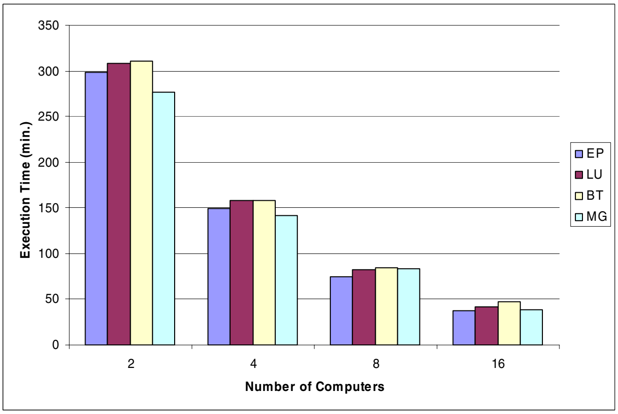


Fig. 4. Executions of NPB programs on our openMosix Cluster

For this reason, we have performed an experiment by compiling the EP, LU, BT and MG programs for running with 16 processes and let each of the programs run on a power-of-2 number of processors. The result as shown in Figure 4 has confirmed the topology specific nature of the programs as their speedups are proportional to a power-of-2 of number of computers used. We could not satisfy fully the topology requirement for the BT program. However, we assumed that the loss of performance for executing BT with a slightly different process-to-processor mapping (in the cases of 2 and 8 computers used) will not distort too much the experiment outcomes.

6 Summary and Conclusions

In this paper, we have presented how to construct an applications execution and scheduling environment for clusters by integrating existing parallel programming environments, tools and middleware. A prototype of such execution and scheduling environment was built: the openMosix cluster. It consists of a network of computers running the Linux operating system. MPI was used to support communication of processes of parallel applications and the openMosix load balancing system was used to schedule processes of sequential and parallel applications across computers in the cluster based on the actual workload in each of the computers. This system has been demonstrated to be flexible, low-cost and well suited for the performance study of benchmarks' scheduling on clusters. .

We have demonstrated that the way the publicly available and frequently used programming benchmarks (both sequential and parallel) could be adapted to carry out such study. For parallel applications, we have identified, selected and characterized the following programs: MPI-Povray, Parallelized TSP and the NAS programs. For sequential applications, we have identified and selected the BYTE's Unix Benchmark Suite, from which constructed a set of workload-benchmarks. These workload benchmarks can represent sequential applications of various workloads ranging from IO-bounded to CPU-bounded. We have also presented the influence of a dynamic load-balancing system, the number of MPI daemons used as well as the topology of parallel programs on the execution performance of parallel applications.

Reference

1. R.H. Arpaci, A.C. Dusseau, A.M. Vahdat, L.T. Liu, T.E. Anderson and D.A. Patterson. The Interaction of Parallel and Sequential Workloads on a Network of Workstations. In *Proceedings of 1995 ACM Joint International Conference on Measurement and Modeling of Computing Systems*, pages 267-278, May 1995.
2. A. Acharya, G. Edjlali and J. Saltz. The Utility of Exploiting Idle Workstations for Parallel Computation. In *Proceedings of 1997 ACM Sigmetrics International Conference on Measurement and Modeling of Computer Systems*, pages 225-236, May 1997.
3. C. Anglano. A Comparative Evaluation of Implicit Coscheduling Strategies for Networks of Workstations. In *Proceedings of 9th International Symposium on High Performance Distributed Computing (HPDC9)*, pages 221-228, August 2000.
4. G. Burns, R. Daoud and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379-386, University of Toronto, 1994.
5. W. Becker. Dynamic Balancing Complex Workload in Workstation Networks - Challenge, Concepts and Experience. In *Proceedings High Performance Computing and Networking (HPCN) Europe Lecture Notes on Computer Science (LNCS)*, pages 407-412, 1995.

6. A. Barak, S. Guday and R.G. Wheeler. The MOSIX Distributed Operating System, Load Balancing for UNIX. Springer-Verlag, 1993.
7. BYTE's UnixBench. The BYTE's Unix Benchmark Suite. URL: http://www.tux.org/pub/tux/niemi/unixbench, June 2004.
8. P.J. Chuang and N.F. Tzeng. A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers. *IEEE Transactions on Computers*, 41(4):467-479, 1992.
9. The MPI Forum. MPI: a message passing interface. In *Proceedings of the 1993 Conference on Supercomputing*, pages 878-883, 1993.
10. A. Faraj and X. Yuan. Communication Characteristics in the NAS Parallel Benchmarks. In *Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002)*, Nov. 2002.
11. A.M. Goscinski. Distributed Operating Systems, The Logical Design. Addison-Wesley, Sydney, 1991.
12. A.M. Goscinski and A.K.L. Wong. Performance Evaluation of the Concurrent Execution of NAS Parallel Benchmarks with BYTE Sequential Benchmarks on a Cluster. In *Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS 2005)*, Fukuoka, Japan, July, 2005.
13. The LAM/MPI Homepage. URL: http://www.lam-mpi.org, lasted access: June 2004.
14. T.H. Lai and S. Sahni. Anomalies in Parallel Branch-and-Bound Algorithms. Communications of the ACM, 27(6):594-602, Jane 1984.
15. NAS Parallel Benchmarks. URL: http://www.nas.nasa.gov/Software/NPB/, Nov. 2004.
16. The openMosix Homepage. URL: http://openmosix.sourceforge.net, June 2004.
17. P. Strazdins and J. Uhlmann. Local Scheduling out-performs Gang Scheduling on a Beowulf Cluster. Technical Report TR-CS-04-01, ANU, Canberra, 2004.
18. J. Subhlok, S. Venkataramaiah and A. Singh. Characterizing NAS Benchmark Performance on Shared Heterogeneous Networks. In *11th International Heterogeneous Computing Workshop*, April 2002.
19. F. Tandiary, S.C. Kothari, A. Dixit and E. W. Anderson. Batrun: Utilizing Idle Workstations for Large-scale Computing. *IEEE Parallel and Distributed Technology*, 4(2):41-48, 1996.
20. L. Verrall. MPI-Povray: Distributed Povray Using MPI Message Passing. URL: http://www.verrall.demon.co.uk/mpipov.
21. F.C. Wong, A.C. Arpaci-Dusseau and D.E. Culler. Building MPI for Multi-Programming Systems using Implicit Information. In *Proceedings of the 6th European PVM/MPI User's Group Meeting,* pages 215-222, 1999.
22. A.K.L. Wong and A.M. Goscinski. Scheduling of a Parallel Computation-Bound Application and Sequential Applications Executing Concurrently on a Cluster – A Case Study. In *Proceedings of the 2nd International Symposium on Parallel and Distributed Processing and Applications (ISPA04)*, Dec. 2004.
23. Y. Zhu. Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers. *Journal of Parallel and Distributed Computing*, 16(4):328-337, 1992.
24. B.B. Zhou, X. Qu and R.P. Brent. Effective Scheduling in a Mixed Parallel and Sequential Computing Environment. In *Proceedings of the 6th Euromicro Workshop of Parallel and Distributed Processing*, pages 32-37, Jan. 1998.

Data Distribution Strategies for Domain Decomposition Applications in Grid Environments

Beatriz Otero, José M. Cela, Rosa M. Badia, and Jesús Labarta

Dpt. d'Arquitectura de Computadors,
Universitat Politècnica de Catalunya, Campus Nord,
C/ Jordi Girona, 1-3, Mòdul D6, 109, 08034, Barcelona-Spain
{botero, cela, rosab, jesus}@ac.upc.es

Abstract. In this paper, we evaluate message-passing applications in Grid environments using domain decomposition technique. We compare two domain decomposition strategies: a balanced and unbalanced one. The balanced strategy is commonly strategy used in homogenous computing environment. This strategy presents some problems related with the larger communication latency in Grid environments. We propose an unbalanced domain decomposition strategy in order to overlap communication latency with useful computation. This idea consists in assigning less workload to processors responsible for sending updates outside the host. We compare the results obtained with the classical balanced strategy. We show that the unbalanced distribution pattern improves the execution times of domain decomposition applications in Grid environments. We considered two kinds of meshes, which define the most typical cases. We show that the expected execution time can be reduced up to 53%. We also analyze the influence of the communication patterns on execution times using the Dimemas simulator.

1 Introduction

Domain decomposition is used for efficient parallel execution of mesh-based applications. These applications use techniques such as finite element and finite difference, which are widely used in many disciplines such as engineering, structural mechanics and fluid dynamics. Mesh-based applications use a meshing procedure for discretizing the problem domain. Implementing a mesh-based application on a Grid environment involves partitioning the mesh into sub-domains that are assigned to individual processors in the Grid environment. In order to obtain optimal performance a desirable partitioning method should take into consideration several features: traffic in the network, latency and bandwidth between processors inside the host, latency and bandwidth between hosts, etc.

We consider distributed applications that perform matrix-vector product operations. These applications solve problems that arise from the discretization of partial differential equations on meshes, such as explicit finite element analysis of sheet stamping or car crash problems. These applications require high computational capabilities [1]. Typically, the models are simplified to the extent that they can be computed on presently available machines; usually many important effects are left out because the computational power is not adequate to include them.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 214 – 224, 2005.

Previous work makes reference to the relationship between architecture and domain decomposition algorithms [2]. There are studies on latency, bandwidth and optimum workload to take full advantage of the available resources [3], [4]. There are also analyses about the behavior of MPI applications in Grid environments [5], [6]. In all these cases, the workload is the same for all the processors. In [7], Li et al. provides a survey of the existing solutions and new efforts in load balancing to address the new challenges in Grid computing. In this paper, we evaluate message-passing applications in Grid environment using domain decomposition technique. The objective of this study is to improve the execution time of the distributed applications in Grid environments by overlapping remote communications and useful computation. In order to achieve this, we propose a new data distribution pattern in which the workload is different depending on the processor. We use the Dimemas tool [8] to simulate the behavior of the distributed applications in Grid environments.

This work is organized as follows. Section 2 describes the tool used to simulate the Grid environment and defines the Grid topologies considered. Section 3 deals with the studied distributed applications and the workload assignment patterns. Section 4 shows the results obtained in the environments specified for the three different data distribution patterns. The conclusions of the work are presented in Section 5.

2 Grid Environment

We use a performance simulator called Dimemas. Dimemas is a tool developed by CEPBA[1] for simulating parallel environments [5], [6], [8]. DIMEMAS simulator considers a simple model for point to point communications. This model decomposes the communication time in five components:

1. Latency time is a fix time to start the communication.
2. Resource contention time is dependent of the global load in the local host [10].
3. The transfer time is dependent of the message size. We model this time with a bandwidth parameter.
4. The WAN contention time is dependent of the global traffic in the WAN [9].
5. The flight time represents the time invested on the transmission of the message to the destination, not consuming CPU latency [10]. It depends on the distance between hosts. We consider hosts distributed at same distances, since our environment is homogeneous.

We consider an ideal environment where resource contention time is negligible: there are an infinite number of buses for the interconnection network and as many links as different remote communication has the host with others hosts. For the WAN contention time, we use a lineal model to estimate the traffic in the external network. We have considered the traffic function with 1% influence from internal traffic and 99% influence from external traffic. Thus, we model the communications with just three parameters: latency, bandwidth and flight time. These parameters are set according to what is commonly found in present networks. We have studied different works

[1] European Center for Parallelism of Barcelona, www.cepba.upc.edu.

to determine these parameters [9], [11]. Table 1 shows the values of these parameters for the internal and external host communications. The internal column defines the latency and bandwidth between processors inside a host. The external column defines the latency and bandwidth values between hosts. The communications inside a host are fast (latency 25 μs, bandwidth 100 Mbps), and the communications between hosts are slow (latency of 10 ms and 100 ms, bandwidth of 64 Kbps, 300 Kbps and 2 Mbps, flight time of 1ms and 100 ms).

Table 1. Latency, bandwidth and flight time values

Parameters	Internal	External
Latency	25 μs	10 ms and 100 ms
Bandwidth	100 Mbps	64 Kbps, 300 Kbps and 2Mbps
Flight time	-	1 ms and 100 ms

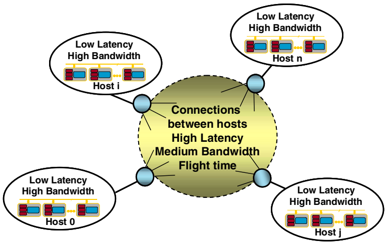


Fig. 1. General Topology: n hosts with m processors per host

We model a Grid environment using a set of hosts. Each host is a network of Symmetric Multiprocessors (SMP). The Grid environment is formed by a set of connected hosts. Each host has a direct full-duplex connection with any other host. We do this because we think that some of the most interesting Grids for scientist involve nodes that are themselves high-performance parallel machines or clusters. We consider different topologies in this study: two, four and eight hosts. Figure 1 shows the general topology of the host connections.

3 Data Distribution

This work involves the use of distributed applications that solve sparse linear systems using iterative methods. These problems arise from the discretization of partial differential equations, especially when explicit methods are used. These algorithms are parallelized using domain decomposition of the data distribution. Each parallel process is associated with a particular domain.

A matrix-vector product operation is carried out in each iteration of the iterative method. The matrix-vector product is performed using a domain decomposition algorithm, i.e., as a set of independent computations and a final set of communications. The communications in a domain decomposition algorithm are associated with the domain boundaries. Each process must exchange the boundary values with all its neighbours. Then, each process has as many communication exchanges as neighbour domains [12], [13]. For each communication exchange, the size of the message is the length of the common boundary between the two domains. We use METIS to perform the domain decomposition of the initial mesh [14], [15], [16].

Balanced distribution pattern. This is the usual strategy for domain decomposition algorithms. It generates as many domains as processors in the Grid. The computational load is perfectly balanced between domains. This balanced strategy is suitable in homogeneous parallel computing, where all communications have the same cost.

Unbalanced distribution pattern. Our proposal is to create some domains with a negligible computational load. Those domains are devoted only to manage the slow communications. In order to do this, we divide the domain decomposition in two phases. First, balanced domain decomposition is done between the number of hosts. This guarantees that the computational load is balanced between hosts. Second, unbalanced domain decomposition is done inside a host. The second decomposition involves splitting the boundary nodes of the host sub-graph. We create as many special domains as remote communications. Note that these domains contain only boundary nodes, so they have negligible computational load. We call these special domains *B-domains* (boundary domains). The remainder host sub-graph is decomposed in (*nproc-b)* domains, where *nproc* is the number of processors in the host and *b* stands for the number of *B-domains*. We call these domains *C-domains* (computational domains). As a first approximation we assign one CPU to each domain. The CPUs assigned to *B-domains* remain inactive most of the time. We use this policy in order to obtain the worst case for our decomposition algorithm. This inefficiency could be solved assigning all the *B-domains* in a host to the same CPU. Figure 2 shows an example of a finite element mesh with 256 degrees of freedom (dofs) with the boundary nodes for each balanced partition. We consider a Grid with 4 hosts and 8 processors per host. We do an initial decomposition in four balanced domains. Figure 3 shows the balanced domain decomposition. We consider two unbalanced decomposition of the same mesh. First, we create a sub-domain with the layer of boundary nodes for each initial domain (*singleB-domain*), which contains seven computational domains (Figure 4). Second, we create some domains (*multipleB-domain*) for each initial partition using the layer of boundary nodes. Then, the remainder mesh is decomposed in five balanced domains (Figure 5).

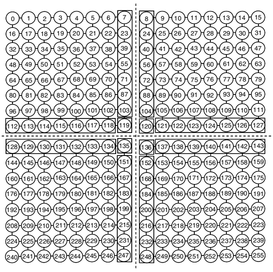


Fig. 2 Boundary nodes per host

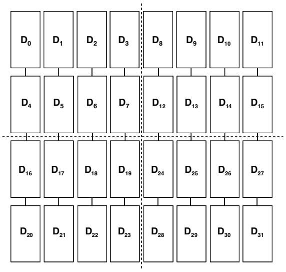

D0 D1 D2 D3 D8 D9 D10 D11
D4 D5 D6 D7 D12 D13 D14 D15
D16 D17 D18 D19 D24 D25 D26 D27
D20 D21 D22 D23 D28 D29 D30 D31

Fig. 3 Balanced distribution

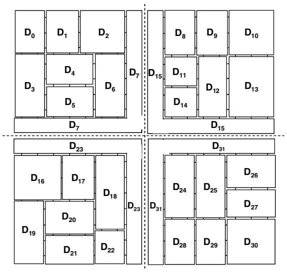


Fig. 4. *SingleB-domain* distribution

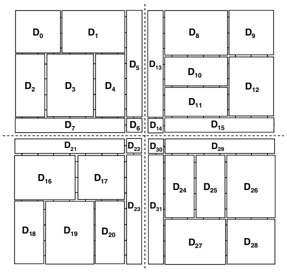
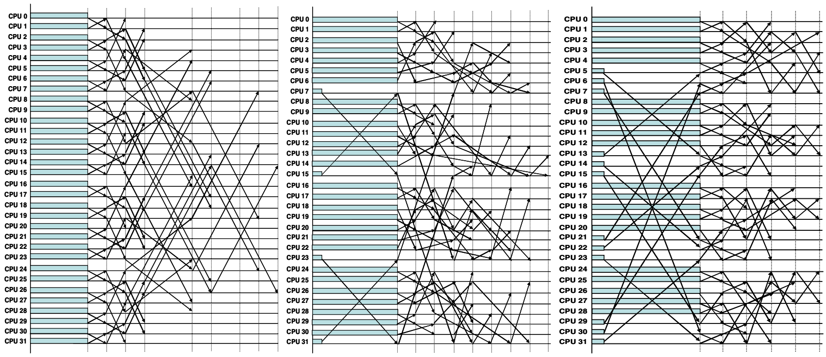


Fig. 5. *MultipleB-domain* distribution



(a) (b) (c)

Fig. 6. Communication diagram for a computational iteration: (a) Balanced distribution; (b) Unbalanced distribution (*singleB-domain*); (c) Unbalanced distribution (*multipleB-domain*).

We must remark that the communication pattern of the balanced and the unbalanced domain decomposition may be different, since the number of neighbours of each domain may also be different. Figure 6 illustrates the communication pattern of the balanced/unbalanced distributions for this example. The arrows in the diagram represent processors interchanging data. The beginning of the arrow identifies the sender. The end of the arrow identifies the receiver. Short arrows represent local communications inside a host, whereas long arrows represent remote communications between hosts. In Figure 6.a, all the processors are busy and the remote communications are done at the end of each iteration. In Figures 6.b and 6.c, the remote communication takes place overlapped with the computation. In Figure 6.b, the remote communication is overlapped only with the first remote computation. In Figure 6.c, all remote communications in the same host are overlapped.

4 Results

In this section we show the results obtained using Dimemas. We simulate a 128 processors machine using the following Grid environment. The number of hosts is 2, 4 or 8; the number of CPUs/host is 4, 8, 16, 32 or 64; thus, we have from 8 to 128 total

CPUs. The simulations were done considering lineal network traffic models. We consider three significant parameters to analyze the execution time behaviour: the communication latency between hosts, the bandwidth in the external network and the flight time.

As data set, we consider a finite element mesh with 1,000,000 dofs. This size is usual for car crash or sheet stamping models. We consider two kinds of meshes, which define most of the typical cases. The first one, called stick mesh, can be completely decomposed in strips, so there are, at most, two neighbors per domain. The second one, called box mesh, cannot be decomposed in strips, so the number of neighbors per domain could be greater than two. The size of the stick mesh is 10^4x10x10 nodes. The size of the box mesh is 10^2x10^2x10^2 nodes.

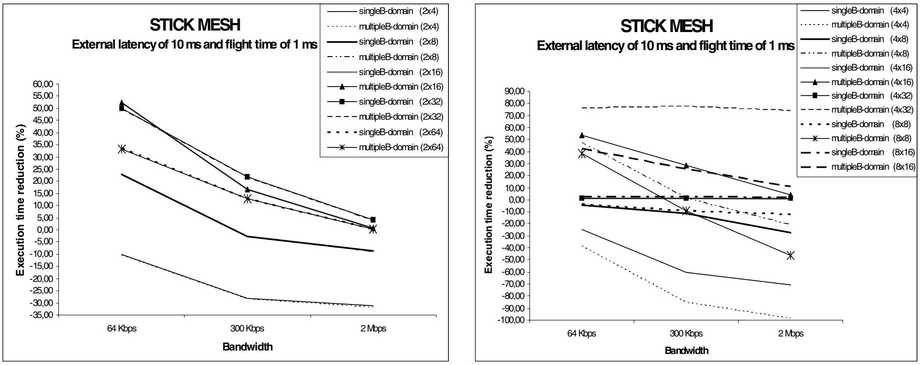


Fig. 7.a. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 1 ms

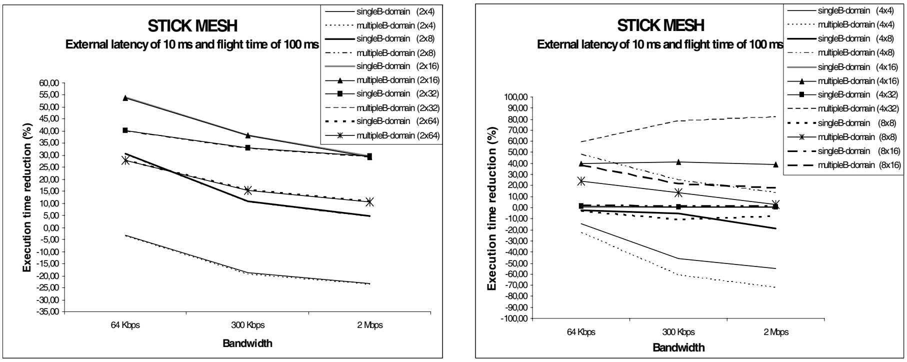


Fig. 7.b. Execution time reduction for the stick mesh with external latency of 10 ms and flight time of 100 ms

Figures 7.a and 7.b show the time reduction percentages for each Grid configuration in stick mesh as a function of the bandwidth. The unbalanced decomposition reduces the execution time expected for the balanced distribution in most cases. For a Grid with 2 hosts and 4 processors per host, the predicted execution time of the balanced distribution is better than other distributions because the number of remote communications is two. In this case, the *multipleB-domain* unbalanced distribution has only one or two processors per host computation.

The results are similar when we consider that the external latency is equal to 100 ms (Figures 8.a and 8.b). Therefore, the value of this parameter has not significant impact on the results for this topology. In the other cases, the benefit of the unbalanced distributions ranges from 1% to 53% of time reduction. The execution time reduction increases until 82 % for other topologies and configurations. For 4 and 8 hosts, the *singleB-domain* unbalanced distribution has similar behavior than the balanced distribution, since the remote communications cannot be overlapped and they have to be done sequentially. In this case, the topologies having few processors per computation are not appropriate. The unbalanced distribution reduces the execution time up to 32 %.

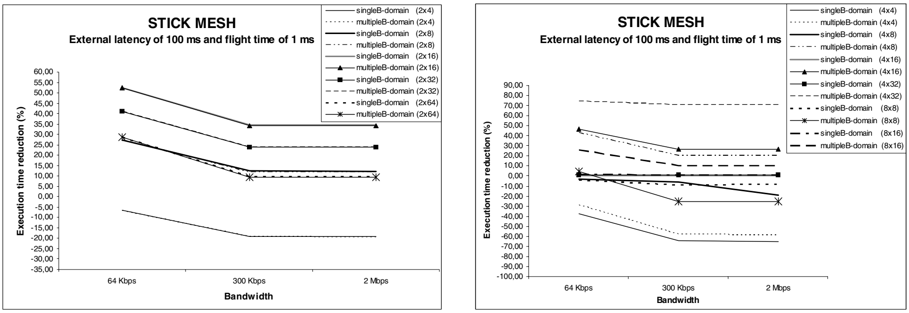


Fig. 8.a. Execution time reduction for the stick mesh with external latency of 100 ms and flight time of 1 ms

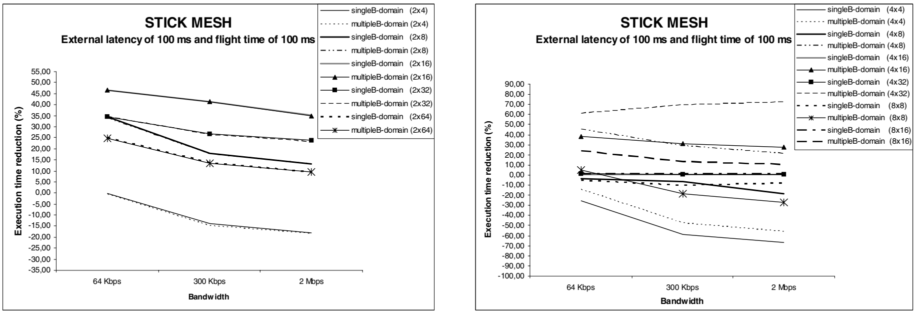


Fig. 8.b. Execution time reduction for the stick mesh with external latency and flight time of 100 ms

Figures 9.a and 9.b show the reduction of the expected execution time obtained for each Grid configuration varying the flight time, the external latency and the bandwidth in a box mesh. For the 2 hosts configuration in a box mesh, the behaviour for *singleB-domain* and *multipleB-domain* unbalanced distribution is similar, since the number of remote communications is the same. Variations of the flight time and the external latency improve the results up to 85%. Figure 9.b shows the reduction on the expected execution time obtained for 4 and 8 hosts. The influence of the external latency on the application performance in a box mesh increases the percentage of reduction of the execution time up to 4%. We suppose that the distance between hosts is the same. However, if we consider hosts distributed at different distances, we obtain similar benefits for the different distributions.

The number of remote and local communications varies depending on the partition and the dimensions of the data meshes. Table 2 shows the maximum number of communications for a computational iteration. The number of remote communications is higher for a box mesh than for a stick mesh. Thus, the box mesh suffers from higher overhead.

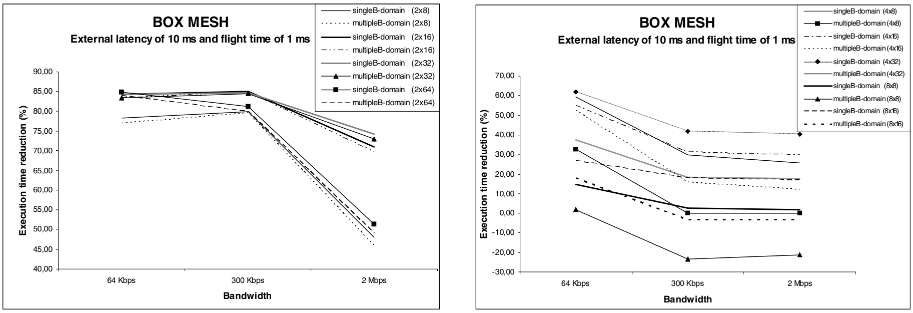


Fig. 9.a. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 1 ms

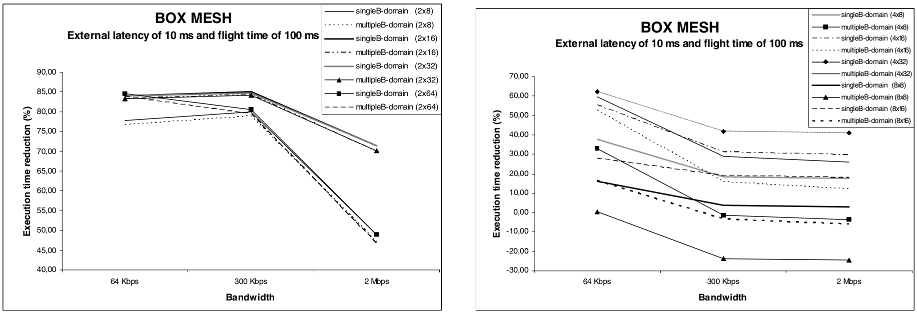


Fig. 9.b. Execution time reduction for the box mesh with external latency of 10 ms and flight time of 100 ms

We propose using unbalanced distribution patterns to reduce the number of remote communications required. Our approach shows to be very effective, especially for box meshes. We observe that the *multipleB-domain* with unbalanced distribution is not sensible to the latency increase until the latency is larger than the computational time. However, the execution time for the balanced distribution increases with the latency.

The *multipleB-domain* unbalanced distribution creates as many special domains per host as external communications. Then, the scalability of the unbalanced distribution will be moderated, because a processor is devoted just to manage communications for every special domain. The optimum domain decomposition is problem dependent, but a simple model could be built to approximate the optimum. We propose to assign all *B-domains* in each host to a single CPU, which concurrently manages the communications.

Table 2. Maximum number of communications for a computational iteration

			STICK MESH			
	BALANCED		***singleB-domain***		***multipleB-domain***	
Host xCPUs	**Remote / Local Communication**		**Remote / Local Communication**		**Remote / Local Communication**	
2x4	1	1	1	1	1	1
2x8	1	1	1	1	1	1
2x16	1	1	1	1	1	1
2x32	1	1	1	1	1	1
2x64	1	1	1	1	1	1
4x4	1	1	2	2	1	3
4x8	1	1	2	2	1	3
4x16	1	1	2	2	1	3
4x32	1	1	2	2	1	3
8x8	1	1	2	2	1	3
8x16	1	1	2	2	1	3
			BOX MESH			
2x4	2	3	1	3	1	3
2x8	4	5	1	6	1	6
2x16	5	8	1	7	1	8
2x32	6	7	1	15	1	14
2x64	7	8	1	25	1	24
4x8	7	5	3	6	4	6
4x16	10	9	3	11	4	9
4x32	9	8	3	22	4	14
8x8	13	5	6	7	13	7
8x16	13	4	6	13	13	11

It is also important to look at the MPI implementation [17]. The ability to overlap communications and computation depends on this implementation. A multithread MPI implementation could overlap communication and computation, but problems with context switching between threads and interferences between processes could appear.

In a single thread MPI implementation we can use non-blocking `send/receive` with a `wait_all` routine. However, we have observed some problems with this approach. The problems are associated with the internal order in no blocking MPI routines for sending and receiving actions. In our experiments, this could be solved programming explicitly the proper order of the communications. But the problem remains for a general case. We conclude that it is very important to have no blocking MPI primitives that actually exploit the full duplex channel capability. As future work, we will consider other MPI implementations that optimize the collective operations [18], [19].

5 Conclusions

In this paper, we presented an unbalanced domain decomposition strategy for solving problems that arise from discretization of partial equations on meshes. Applying the unbalanced distribution in different platforms is simple, because the data partition is easy to obtain. We compare the results obtained with the classical balanced strategy used. We show that the unbalanced distribution pattern improves the execution time of domain decomposition applications in Grid environments. We considered two kinds of meshes, which define the most typical cases. We show that the expected execution time can be reduced up to 53%.

The unbalanced distribution pattern reduces the number of remote communications required. Our approach proves to be very effective, especially for box meshes. However, the unbalanced distribution can be inappropriate if the total number of processors is less than the total number of remote communications. The optimal case is when the number of processors making calculation in a host is twice the number of processors managing remote communications. Otherwise, if the number of processors making calculations is small, then the unbalanced distribution will be less efficient than the balanced distribution.

References

1. G. Allen, T. Goodale, M. Russell, E. Seidel and J. Sahlf. “Classifying and enabling grid applications”. Concurrency and Computation: Practice and Experience; 00: 1-7, 2000.
2. W. D. Gropp and D. E. Keyes. “Complexity of Parallel Implementation of Domain Decomposition Techniques for Elliptic Partial Differential Equations”. SIAM Journal on Scientific and Statistical Computing, Vol. 9, nº 2, 312-326, 1988.
3. D. K. Kaushik, D. E. Keyes and B. F. Smith. “On the Interaction of Architecture and Algorithm in the Domain-based Parallelization of an Unstructured Grid Incompressible Flow Code”. 10th International Conference on Domain Decomposition Methods, 311-319. Wiley, 1997.
4. W. Gropp, D. Kaushik, D. Keyes and B. Smith. “Latency, Bandwidth, and Concurrent Issue Limitations in High-Performance CFD”. Computational Fluid and Solid Mechanics, 839-841.MA, 2001.
5. R. M. Badia, J. Labarta, J. Giménez, F. Escale. “DIMEMAS: Predicting MPI Applications Behavior in Grid Environments”. Workshop on Grid Applications and Programming Tools (GGF8), 2003.
6. R. M. Badia, F. Escale, E. Gabriel, J. Gimenez, R. Keller, J. Labarta, M. S. Müller. “Performance Prediction in a Grid Environment”. 1st European across Grid Conference, SC, Spain, 2003.
7. Y. Li and Z. Lan. “A Survey of Load Balancing in Grid Computing”. Computational and Information Science, 1st International Symposium, CIS 2004. LNCS 3314, 280-285, Shanghai, China.
8. Dimemas, Internet, 2002, http://www.cepba.upc.es/dimemas/
9. R. M. Badía, J. Gimenez, J. Labarta, F. Escalé and R. Keller. “DAMIEN: Distributed Applications and Middleware for Industrial Use of European Networks. D5.2/CEPBA”. IST-2000-25406.
10. R. M. Badía, F. Escalé, J. Gimenez and J. Labarta. “DAMIEN: Distributed Applications and Middleware for Industrial Use of European Networks. D5.3/CEPBA”. IST-2000-25406.
11. B. Otero y J. M. Cela. “Latencia y ancho de banda para simular ambientes Grid”. TR UPC-DAC-2004-33, UPC. España, 2004. http://www.ac.upc.es/recerca/reports/DAC/2004/index,ca.html
12. D. E. Keyes. “Domain Decomposition Methods in the Mainstream of Computational Science”. 14th International Conference on Domain Decomposition Methods, Mexico City, 79-93, 2003.
13. X. C. Cai. “Some Domain Decomposition Algorithms for Nonselfadjoint Elliptic and Parabolic Partial Differential Equations”. TR 461, Courant Institute, NY, 1989.

14. K. George and K. Vipin, “Multilevel Algorithms for Multi-Constraint Graph Partitioning”. University of Minnesota, Department of Computer Science. Minneapolis. TR 98-019, 1998.
15. K. George and K. Vipin, “Multilevel k-way Partitioning Scheme for Irregular Graphs”. University of Minnesota, Department of Computer Science. Minneapolis. TR 95-064, 1998.
16. Metis, Internet, http://www.cs.umn.edu/~metis
17. Message Passing Interface Forum, MPI-2: Extensions to the Message Passing Interface, July, 1997.
18. N. Karonis, B. Toonen, and I. Foster. “Mpich-g2: A Grid-enabled Implementation of the Message Passing Interface”. Journal of Parallel and Distributed Computing, 2003.
19. I. Foster and N. T. Karonis. “A Grid-enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems”. Supercomputing. IEEE, November, 1998. http://www.supercomp.org/sc98

Inter-round Scheduling for Divisible Workload Applications

DongWoo Lee and R.S. Ramakrishna

Department of Information and Communication,
Gwangju Institute of Science and Technology, Republic of Korea
{leepro, rsr}@gist.ac.kr

Abstract. Most common jobs of Grid computing are arbitrarily divisible. Divisible load theory(DLT) provides the mathematical machinery for time-optimal processing. With multiple round load distributions, idle processor periods can be harnessed for useful computation. Optimized rounds for the purpose can be planned in advance. The Grid is dynamic in nature. The above theory does not fully account for this. Any realistic scheduling strategy based on DLT has to take this fact into account. Existing multiple rounds algorithms do not involve time-varying effects due to environmental changes. This is a situation that leads to processing delays, such as Disk I/O contention. The proposed inter-round scheduling algorithm takes this into consideration. It involves time-varying resource performance degradation and results in resonable performance.

1 Introduction

Resource scheduling is of vital importance for optimal Grid performance. Commonly encountered Grid jobs are arbitrarily divisible. The segmented tasks are allocated to dispersed resources and processed. Divisible Load Theory[1] has been considered to be a mathematically tractable framework to schedule arbitrary divisible workloads in parallel machines.

In this paper we employ divisible load scheduling in a multiple round framework that adapts to changing environments brought about by dynamic resources. A multi-round scheduling has been developed with a view to harness idle processor cycles, a drawback of single round algorithms. The strategy is to allow small chunks of workload in the initial rounds. This ensures that all the processors are busy. Load distribution is undertaken in parallel with the undergoing computation. This results in overlapping of communication and computation.

Uniform Multi-Round Scheduling(UMR)[4] uses uniform chunk size in each round. This leads to a near-optimal number of rounds, on both homogeneous and heterogeneous platforms. But, this approach produces unexpected results in the Grid in which resources are not dedicated to a specific task. As it uses system parameters determined before load distribution, the actual makespan could vary due to system dynamics. A load on each system component varies in time due to the presence of background jobs. This increases the makespan of the system. UMR does not resize the chunk to be assigned to a processor when a new round

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 225–231, 2005.

begins as it is not adaptive in nature. Invariably, it follows the policy of working with uniform chunk size. The chunk size is predetermined at the beginning of load distribution.

We propose UMIO(Uniform Multi-Round Adaptive Disk I/O) Inter-round scheduling that exploits current monitoring information about processor loads to adjust the chunk size in response to changing system status. In UMR, every processor in a round gets identically sized load. But, each processor expends a different execution time according to its status. Some processors are slower or faster than others. The time to process some given chunk can vary according to the load on the procesor. In Inter-round scheduling, we allow uniform chuck size, but it is based on the current processor I/O load as I/O contention creates a bottleneck that slows down a processor. We can also use other system parameters to get a more precise model. When a new round begins, a processor's current I/O load affects the load distribution in the next round. This Inter-Round scheduling tries to minimize the gap between the theoretical makespan and the real one arising from changing system dynamics brought about by shared resource contentions.

The rest of the paper is organized as follows: In section 2 we explain several divisible workload scheduling algorithms. Section 3 presents the proposed UMIO algorithm which is evaluated by simulation in Section 4. Section 5 concludes the paper.

2 Related Work

In single round divisible load scheduling[1,2] on homogeneous platforms, the total load is divided into unform chunks. In heterogeneous environments, the load assigned to a processor is determined by its computational speed. Casanova and Yang[3] proposed a simple model for large scale Grid computing. The model does not consider overheads on shared resources.

Multi-Round Scheduling[1] was developed to reduce the time for which a processor idly waits to receive its load—a situation that occurs in single round scheduling. By repeatedly transmitting the load to each processor over multiple rounds, the load size can be scaled down. This is in stark contrast to the single round case. The next processor can receive the load immediately after the first one and then work on it. The MI(Multiple Installment) Algorithm, which sends the load in more than one installment to reduce communication overheads was proposed by Bharadwaj et al[1]. Yang and Casanova[6] proposed a more realistic affine overhead model for MI and transferring problem of outputs. The number of rounds must be specified manually in this case. The UMR(Uniform Mutil-Round Algorithm) was proposed by Yang and Casanova[4]. As it allows uniform load size, it is possible to find the optimum number of rounds that minimizes computational makespan. In [5], a robust method for controlling the size of load in a progressive manner is mentioned. It has no device to control its adaptatibility between rounds. In the proposed UMIO, we incorporate the functionality to adapt to resource changes.

3 UMIO(Uniform Multi-round Adaptive Disk I/O) Algorithm

As for the UMR which works with the same load size in each round, the makespan could change due to varying system load. To solve this problem, we propose the UMIO Algorithm that can accomodate changes in inter-round resource status. Figure 1 shows how multi-round scheduling distributes chunks of workloads over serveral rounds.

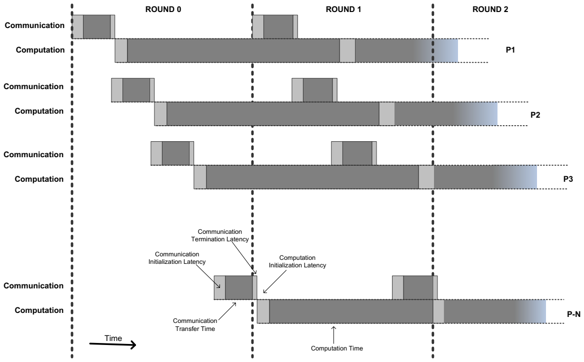


Fig. 1. Uniform Multi-round Scheduling Diagram(UMR[4])

The parameters are related as follows:

$$TC_i = nLat_i + \alpha_i z_i T_{cm} + tLat_i \tag{1}$$

$$TP_i = cLat_i + \alpha_i w_i T_{cp} \tag{2}$$

Here, TC_i is the time taken to transfer chunk size α_i to processors P_i; TP_i is the time taken to compute chunk size α_i at processors P_i; $nLat_i$, $tLat_i$ and $cLat_i$ are the initial and final communication latencies while $cLat_i$ is the computational initial latency. Futher, w_i is the ratio of the time taken by P_i to compute a given load, to the time taken by a standard processor to compute the same load; z_i is the ratio of the time taken by a link to P_i to transfer a given load, to the time taken by a standard link to transfer the same load. Also, T_{cp} is the time taken to process a unit load by the standard processor and T_{cm} is the time taken to transfer a unit load using a standard communication channel.

3.1 Disk I/O Contention Model

Contention parameters are added to the ealier UMR model in accordance with the following relationships:

$$TP_i{}' = cLat_i{}' + \alpha_i w_i{}' T_{cp} \tag{3}$$

$$w_i{}' = B_i{}^{ioCF_{cp}} w_i \tag{4}$$

$$cLat_i{}' = B_i{}^{ioCF_{init}} cLat_i \tag{5}$$

Here, $ioCF_{init}$ is the initial background I/O contention factor and $ioCF_{cp}$ is the background I/O contention factor of computation. These two contention factors are measured by a local resource monitoring system such as SWS(Storage Weather Service)[7,8]. Further, B_i is the number of tasks competing for I/O at P_i. By replacing the UMR's TC_i model with new factors, we can use UMR to obtain near-optimal number of rounds[4]. We find the approximate number of rounds through constrained minimization. The objective of this is to minimize $Ex(M, \alpha_0)$[4], the makespan of the application. M is the number of rounds and α_0 is the initial chunk size in round 0. By computing the initial chunk size and M, the chunk size of the next round can be obtained by

$$Ex(M, \alpha_0) = \frac{W}{N} + McLat + \frac{1}{2}N(nLat + \alpha_0 z_i T_{cm}) + tLat \tag{6}$$

Here, W is the total workload and N is the number of processors. M is computed by the Lagrange multiplier method the same as in [4]. Once we have near-optimal number of rounds, M^*, we can get the initial chunk size, α_0.

3.2 Inter-round Scheduling of UMIO

UMIO begins with predetermined M and α_0. Before distributing the load in the next round, a new chunk size is calculated as described above. Figure 2 shows inter-round scheduling. Between rounds, a processor's status changes owing to contentions as described in the I/O contention model. Therefore, rescheduling based on the current status of each processor is needed.

In this paper, we assume that all the processors are affected to the same degree of contention from concurrently running applications. This is reasonable because local resources in the Grid are clustered homogeneous machines involving the same type of hardware and software. If an application is deployed to a local cluster, it suffers from I/O contention to the same extent over all the resources. In such a situation, the UMIO algorithm works with the same chunk size in a round.

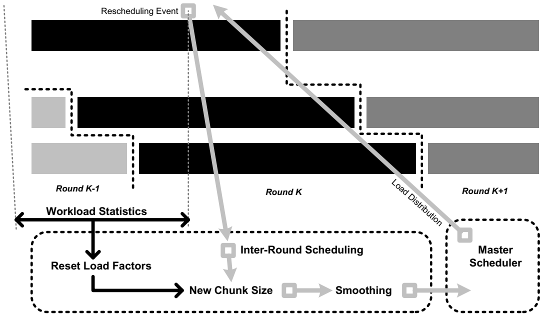


Fig. 2. Inter-Round Scheduling Procedure

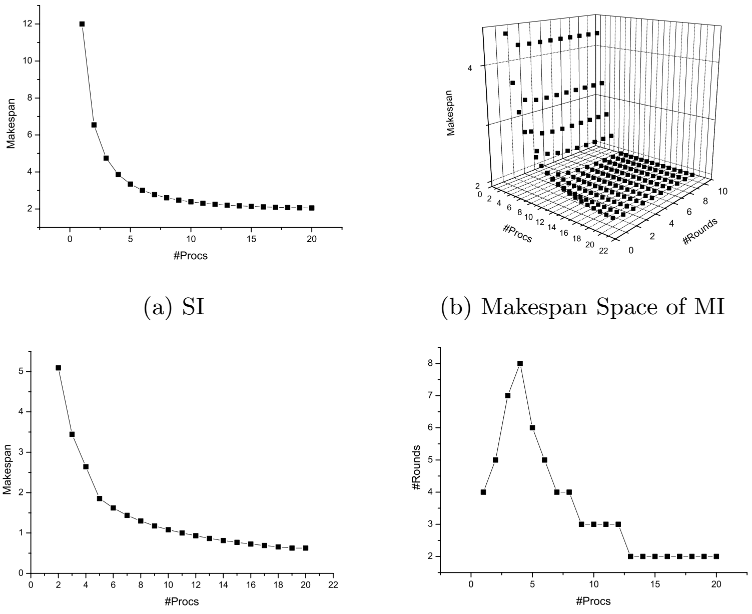
During the first round following initial scheduling, the contention factors of all the processors are collected by the master scheduler in parallel. These indicate the current processing power of the processors. These factors are incorporated into the UMIO model and the updated M and α_0 are computed. The scheduler then computes the chunk size for the next round, i.e., $K+1$. If the new M is not equal to the initial M and the next round $K+1$ is greater than the initial M, the scheduler takes the last round's chunk size for the new schedule because the last chunk size is the largest. UMR and UMIO employ a small chunk size in the initial round with a view to start all the processors immediately. The *smoothing* operation of Figure 2 is to select proper chunk size by comparing the initial and the new M.

4 Experiments and Results

We evaluate UMIO with SI(Single Installment)[2], MI(Multiple Installment)[1] and UMR[4]. SI and MI do not involve an affine cost model. But, UMR and UMIO follow an affine cost model including overhead costs described in Section 3. Figure 3 shows the makespan of SI, MI and UMR. We set w=10,z=5, T_{cm}=1, T_{cp}=1, $cLat$=0.1, $tLat$=0.1 and $cLat$=0.1. The number of processors ranged from 2 to 20. The number of rounds ranged from 2 to 10 for MI. SI leveled at 10 processors having a makespan of 2.4. We searched the makespan space of MI. It shows similar results as did SI when the number of processors increases at a certain fixed number of rounds(*#Rounds*). Also, increasing the number of rounds results in enhanced makespan, but not as much as is obtainable by increasing the number of processors. MI leveled at 9 processors and 3 rounds which has a makespan of 2.0. The advantages of multi-round scheduling can be discerned by comparison between SI and MI.

Figures 3 (c) and (d) show UMR's makespan and its optimized number of rounds verses the number of processors. UMR outperforms MI with 5 processors when the makespan is 1.85. The number of rounds grows up to 8 as the number of processors reaches 4. After that, the number of rounds decreases to 2 because there are enough processors to process the workload. The flat makespan of UMR is about 0.62. In the case of UMR, we assume that there is no background job to share the computing power, i.e., each resource uses 100% of the computing power.

Figure 4 plots the dependence of UMIO's makespan and the number of rounds against the number of processors. The result is similar to that of UMR but the makespan has stepwise changes because of inter-round rescheduling. We allow a performance degradation of 5% per round to simulate time-varying resource changes. In each experiment in a certain simulation(*#Procs*), each processor has a different computing power in a round. The makespan of UMIO flattened at 1.0 even though it suffered from computing power degradations due to contention. The round variation looks similar to that of UMR. But there is a difference between UMR and UMIO due to the time-varying processor performance changes as can be seen in Figure 4 (b).



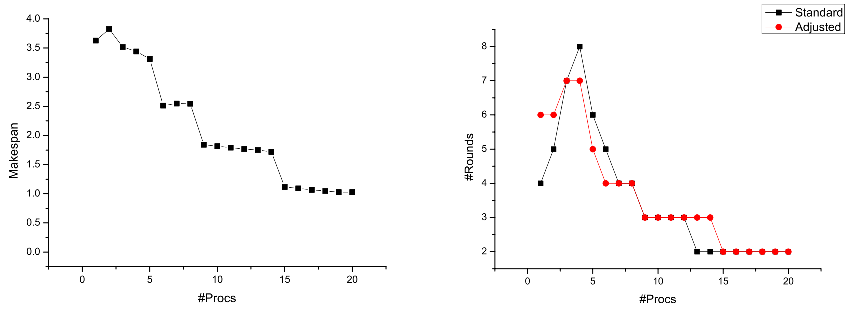
(a) SI

(b) Makespan Space of MI

(c) UMR

(d) Round Variation of UMR

Fig. 3. Makespan of SI, MI and UMR



(a)

(b)

Fig. 4. Makespan: (a) UMIO (b) UMR(standard) vs UMIO(adjusted)

Figure 5 shows makespans of SI, MI, UMR and UMIO. Because SI and MI have no affine cost model, the actual graph shifts upward by as much as the overhead cost. UMIO outperforms SI and MI after 10 processors. But, UMIO outperform SI and MI over the entire range. Because UMIO has an affine cost model and the Disk I/O contention model, we enforce 5%/Round degradations. Also, UMIO is close to UMR as many processors are involved even though each processor suffers from contentions in every round.

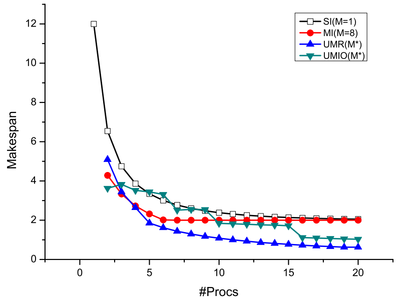


Fig. 5. Comparison SI, MI(best case, M=8), UMR and UMIO

5 Conclusions

In this paper, we have made two contributions: 1) the Disk I/O contention model for multi-round divisible load scheduling—a step toward a realistic DLT model in the Grid, and 2) Inter-round scheduling for adapting to changes in resource performance status. In addition to the benefit that accrues from multi-round scheduling and UMR's optimal number of rounds for a certain environment, one can also impart adaptability to the changes in environment to schedule divisible load applications in the Grid.

References

1. Veeravalli Bharadwaj, Debasish Ghose, Venkataraman Mani and Thomas G. Robertazzi, Scheduling Divisible Loads In Parallel And Distributed Systems, IEEE Press, 1996.
2. Bharadwaj Veeravalli, Xiaolin Li, Chi Chung Ko. On the influence of start-up costs in scheduling divisible loads on bus networks, IEEE Trans. on Parallel and Dist. Systems Vol. 11, No. 12, Dec 2000
3. Henri Casanova and Yang Yang, Algorithms and Software to Schedule and Deploy Independent Tasks in Grid Environment, Feb. 2003
4. Yang Yang and Henri Casanova, A Multi-Round Algorithm for Scheduling Divisible Workload Applications: Analysis and Experimental Evaluation, Department of Computer Science and Engineering, UCSD, Technical Report CS2002-0721, September 26, 2002.
5. Yang Yang and Henri Casanova, RUMR: Robust Scheduling for Divisible Workloads, HPDC2003, June 2003.
6. Yang Yang and Henri Casanova, Extensions to the Multi-installmemnt Algorithm: Affine Costs and Output Data Transfers, July, 2003.
7. DongWoo Lee, R.S.Ramakrishna, Disk I/O Performance Forecast using Basic Prediction Techniques for Grid Computing, Lecture Note in Computer Science, Springer-Verg, LNCS 2763, pp. 257-269, 2003.
8. DongWoo Lee, R.S.Ramakrishna, Storage Weather Service: Storage Performance Forecast, Cluster Computing and the Grid(CCGrid), 2003, Tokyo, Japan.

Scheduling Divisible Workloads Using the Adaptive Time Factoring Algorithm*

Tiago Ferreto and César De Rose

Catholic University of Rio Grande do Sul (PUCRS),
Faculty of Informatics, Porto Alegre, Brazil
{ferreto, derose}@inf.pucrs.br

Abstract. In the past years a vast amount of work has been done in order to improve the basic scheduling algorithms for master/slave computations. One of the main results from this is that the workload of the tasks may be adapted during the execution, using either a fixed increment or decrement (*e.g.* based on an arithmetical or geometrical ratio) or a more sophisticated function to adapt the workload. Currently, the most efficient solutions are all based on some kind of evaluation of the slaves' capacities done exclusively by the master. We propose in this paper the Adaptive Time Factoring scheduling algorithm, which uses a different approach distributing the scheduling between slaves and master. The master computes, using the Factoring algorithm, a time slice to be used by each slave for processing, and the slave predicts the correct workload size it should receive in order to accomplish this time slice. The prediction is based on a performance model located on each slave which is refined during the execution of the application in order to provide better predictions. We evaluated the proposed algorithm using a synthetic testbed and compared the obtained results with other scheduling algorithms.

1 Introduction

Load balancing has been an ongoing issue for decades. Algorithms based on list-scheduling which manage a list of ready to execute tasks that are sent to slave processors are mainly used because of their suitability to dynamically evolving computations, and also because they cope with heterogeneous resources, since when one processor has finished his work it simply gets more work from the list. This is a simply way to automatic compensate for the differences in the performance of the slaves.

A vast amount of work has been done in order to improve the basic algorithms for master/slave computations. One of the main features concerning load balancing that resulted from this is that the workload of the tasks may be adapted during the execution, using either a fixed increment or decrement (*e.g.* based on an arithmetical or geometrical ratio) or a more sophisticated function to adapt the workload. We present a briefly review of some of these techniques in Section 2.

Yet the solutions presented are all based on some evaluation by the master of the slaves' capacities and of the tasks workload. This implies a significant overhead since the master has to maintain some kind of information about its slaves. We present in

* This research was done in cooperation with HP-Brazil.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 232–239, 2005.

this paper the Adaptive Time Factoring scheduling algorithm, which uses a different approach distributing the scheduling between slaves and master. The master computes, using the Factoring algorithm, a time slice to be used by each slave for processing, and the slave predicts the correct workload size it should receive in order to accomplish this time slice. The prediction is based on a performance model located on each slave which is refined during the execution of the application in order to provide better predictions.

In this paper we review in Section 2 some scheduling algorithms used for master/slave applications with a brief state of the art for each one. Section 3 presents our algorithm and the way each slave can evaluate its capacities. In order to validate our algorithm we devised a synthetic small testbed and Section 4 shows the measurement results that we have obtained using the algorithm proposed in comparison to other algorithms. At last we draw some conclusions about our contribution.

2 Related Work

We present below some classic self-scheduling algorithms proposed in the literature. Self-scheduling [1] represents a large class of dynamic centralized loop scheduling methods. These methods divide the total workload based on a specific distribution, providing a natural load balancing to the application during its execution. We present also some adaptive algorithms that add extensions to the classic self-scheduling algorithms in order to support heterogeneity and adaptability. They consider the load variation in the system environment and adjust the size of the chunks delivered to each processor dynamically. This class of algorithms presents a good performance on dynamic and heterogeneous environments based on its ability to adapt itself to the changes in the environment during the execution of an application.

The Pure Self-scheduling or Workqueue scheduling algorithm divides equally the workload in several chunks. A processor obtains a new chunk whenever it becomes idle. Due to the scheduling overhead and communication latency incurred in each scheduling operation, the overall finishing time may be greater than optimal.

The Fixed-size Chunking scheduling algorithm [2] proposes that each processor receives chunks with size K each time it becomes idle. Although it is hard to determine the best K value in realistic applications due to the high number of dependable variables, the authors give an approximation for an acceptable fixed chunk-size K (using *Pth* order statistics to model the last P chunks).

The Guided Self-scheduling algorithm [3], schedules large chunks initially, implying reduced communication/scheduling overheads in the beginning, but at the last steps too many small chunks are assigned generating more overhead [1]. Each time a processor requests for more work, the algorithm assigns to it a chunk of size equal to the size of the remaining workload divided by the total number of processors being used for the computation.

Factoring [4] was specifically designed to handle iterations with execution-time variance. With factoring, iterations are scheduled in batches of P equal-sized chunks. The total size of the chunk per batch is a fixed ratio (α) of the remaining workload, *i.e.* *Remaining_Workload* / $\alpha *$ *Number_Of_Processors*.

Weighted Factoring Self Scheduling [5] is an improved loop scheduling algorithm addressing load imbalance in a heterogeneous environment. In this algorithm, processors are dynamically assigned decreasing size chunks of iterations in proportion to their processing speeds.

Adaptive Weighted Factoring [6,7] is an adaptive algorithm based on probabilistic analysis, being able to accommodate load imbalances caused by predictable and unpredictable phenomena. In the Adaptive Weighted Factoring, the weight values are adapted after each iteration in the computation. The newly computed weights are not only based on the performance of particular processors during the previous iteration step, but also on their cumulative performance during all the previous iterations.

Adaptive Factoring [8,9] allows a relaxation of some of the theoretical assumptions imposed by models used in earlier methods, therefore making this technique more robust to any load variations present in the environment and improving the performance of applications characterized by highly irregular behavior. In this algorithm, the weights are dynamically assigned to processors at run time by closely following the rate of change in processor speed. The model used for this method allows the dynamic computation of new weights for each processor, when a new chunk is allocated.

In all algorithms shown above, the information needed to evaluate the best processor to run the remaining workload is centralized at the master process, which is responsible for the decision regarding increasing or decreasing the chunk that is executed by each slave process. We propose in the following Section another approach, where the evaluation of the chunk size to be assigned to each slave is done by the slave itself.

3 Adaptive Time Factoring Scheduling Algorithm

The Adaptive Time Factoring (ATF) scheduling algorithm is, like others algorithms (e.g. Weighted Factoring [5], Adaptive Weighted Factoring [6,7]), based on the decreasing scheme proposed by Factoring [4]. However, instead of decreasing, for each round, the number of tasks to be processed by each slave, it decreases the time slice that each slave should use. Each slave predicts the best chunk size it should process based on the time slice given using a performance model.

The main features of the algorithm are the utilization of time instead of chunk sizes as a scheduling metric, and the distribution of the performance model structure between the slaves. The utilization of time instead of chunk sizes facilitates handling heterogeneous slaves due to better scheduling abstraction, *i.e.* the scheduler can assure that scheduling the same amount of time for different slaves will result in approximately the same completion time benefiting overall performance. The adoption of a distributed performance model managed by the slaves instead of a centralized one at the serevr provides better scalability support avoiding a centralized bottleneck and faster adaptation of the model due to performance variable fluctuations. Based on this data distribution, the algorithm scheduling decision is also distributed between master and slaves. In contrast to other algorithms, the slave also participates at the scheduling decision calculating the chunk size to be processed using its local performance model.

The main goal of the algorithm is to minimize execution time of applications in heterogeneous and dynamic environments. It addresses particularly applications using the master/slave model containing divisible workloads, *i.e.* the total amount of work to be processed can be divided in equal-size chunks.

During the execution of the algorithm, each slave builds an internal performance model which contains the slave's execution and communication time demands to process chunks of the application workload. It enables the prediction of the chunk size to be processed by the slave in order to fully use the time slice given by the master. Detailed information related to the performance model and the prediction method used in the algorithm is presented in Section 3.1.

The Adaptive Time Factoring scheduling algorithm is based on a distributed scheduling method. Each slave computes a chunk sent by the master and, based on its internal performance model, predicts the best chunk size to be computed at the next iteration considering the time slice given by the master. The master distributes time slices in decreasing chunks between the slaves. The decreasing method used is based on the Factoring [4] algorithm with a fixed value $\alpha = 2$.

In order to obtain efficient slave predictions for scheduling, it's necessary to build and refine the performance model in each slave before using the predictions. Due to this requirement, the Adaptive Time Factoring scheduling algorithm is divided in two distinct phases: setup phase and adaptive phase.

The setup phase is used to build the local performance model on each slave and to refine it in order to produce predictions with minimum error. It's like an initial benchmarking of each slave using the application workload. At the beginning, the master sends to each slave a chunk with minimum size and waits for the results. After receiving the results from a slave, it sends another chunk to the slave duplicating its size by a factor of two. It continues this process until it receives a signal from the slave indicating to start with the adaptive phase. This signal is sent when the slave already has an efficient performance model capable of producing good predictions, *i.e.* the model provides minimum error comparing predicted and measured execution times.

The adaptive phase turns over an increasing size mechanism to a decreasing one. However, instead of decreasing the chunk size, it decreases the time slice used by each slave to predict the more appropriate chunk size to be processed. Since the beginning of the algorithm, each time the master sends a chunk to each slave, it includes in the message a time slice for the next round. This time slice is used by the slave to predict the chunk size that it can be execute at the next round. The slave returns a message with the results of the chunk processing, the execution time that it took to process the chunk and the chunk size predicted. This chunk size is only considered at the algorithm after the slave sends the signal to the master in order to start the adaptive phase. The time slice for the round is computed as:

$$timeSlice_{i+1}^{root} = (workload_{i+1} * avgExecTime)/\alpha * nSlaves$$

where $timeSlice_{i+1}^{root}$ is the time slice computed for the next round $(i + 1)$, $workload_{i+1}$ is an estimation of the remaining workload at the next round $(i + 1)$, $avgExecTime$ is the average execution time for a chunk with minimum size, $alpha$ is a parameter of the Factoring algorithm which is fixed to 2 and $nSlaves$ is the total number of slaves. The average execution time is computed each time the master receives a new result from some slave using chunk size and execution time values. Since the time slice sent is related to the next round, it's necessary to use an estimation of the remaining workload at the next round. It is compute as:

$$workload_{i+1} = workload_i - \frac{nSlaves * timeSlice_i}{avgExecTime}$$

The estimation is based on the subtraction, on the current workload, of the average chunk size that can be processed by all slaves during the current time slice. In order to minimize the gap between slaves completion times, the following rule is used: the first slave in a round computes its time slice as presented above (root time slice), and all the others compute their time slices as:

$$timeSlice_{i+1} = timeSlice^{root}_{i+1} - \delta^j$$

where $timeSlice^{root}_{i+1}$ is the first time slice computed at the beginning of the round, and δ^j is the time taken to set this new time slice since the first time slice of this round has been computed.

The master algorithm for the adaptive phase is presented in Figure 1. It keeps in a loop sending chunks and receiving the results to available slaves until the workload is empty. Before sending the chunk to a slave, it computes the time slice, which depends if the slave is the first to compute this value or not, as described before. It assigns the new chunk size with the slave's prediction size previously returned with the last result, and sends to the slave the chunk and time slice.

At reception, the master receives the result of chunk processing, execution time took to process the chunk and the predicted chunk size for the next round. The average execution time is computed using chunk size and execution time parameters.

Algorithm 1. Adaptive Time Factoring algorithm

```
while workload is not empty do
  for each available slave do
    if beginning of round $i$ then
      compute $timeSlice^{root}_{i+1}$
    else
      compute $timeSlice_{i+1}$
    end if
    chunk $\Leftarrow predictedSize$
    send $chunk$ and $timeSlice_{i+1}$
  end for
  receive $result$, $execTime$ and $predictedSize$
  compute $avgExecTime$
end while
```

Due to the existence of a performance model on each slave, any change in the machines (*e.g.* machine turned down, start of a concurrent application) results in an adaptation of the best chunk size to be processed by the slave. It is important to emphasize that different slaves can be in distinct phases of the algorithm at the same time, *i.e.* some of the machines can be executing at the setup phase and others at the adaptive phase. This condition happens when more slaves are included during the execution.

3.1 Local Prediction of the Computational Load

In order to estimate the most suited workload, a slave needs a performance model for the execution of chunks of size $chunkSize_i$. The model may include various data such as the execution time, memory utilization, etc, used to process a given chunk. In this preliminary version of our prototype we only take into account the execution time.

Given some N values $chunkSize_1, chunkSize_2, \ldots chunkSize_n$ and the slaves's data t (*e.g.* the execution time) the slave has to estimate $t(chunkSize)$. In a multi-parameter model we could use algorithms such as the Singular Value Decomposition [10], one of the most robust for data modeling. It would fit the function t as a linear combination of standard base functions (*e.g.* $x \rightarrow e^x$, $\surd$, polynomials, ...).

Nevertheless in the case where t only depends on the processor's speed, an affine model of the time required *vs.* the chunk size to run is most realistic and used by other algorithms [11]. The modeling problem is therefore a basic linear interpolation problem of the measured running time $t_j, j = 1 \ldots n$ *vs.* the chunk size $chunkSize_j$. Besides the estimated coefficients a, b of the affine approximation $t = a + b \times chunkSize$, the correlation coefficient is used to determine the correction of the interpolation and thus decide if a larger chunk should be sent in the initial phase.

The interpolation algorithm is very fast and thus does not prejudice the execution of the application. Moreover, it is trivial for a slave to determine the adapted chunk size, given the execution time t it has to run and the affine model (a, b). Note that in the case of a more complex, non-linear model, it would have to use a more time-consuming algorithm such as a gradient or dichotomic search to solve the $t = f(chunkSize)$ equation.

4 Evaluation

In order to evaluate the performance of the Adaptive Time Factoring scheduling algorithm (ATF) we devised a simple master/slave application and executed it in a heterogeneous cluster. This application consists in w multiplications of two matrices of size $n \times n$. The minimum chunk size is the multiplication of two matrices ($w = 1$). With this application we are able to easily vary the size and the number of chunks to be processed, generating different conditions to evaluate the behavior of our algorithm.

We executed the application in a cluster with 16 machines connected through a Fast-Ethernet network. The testbed consists of four different types of nodes divided in classes, from A to D (4 nodes per machine class). To give an idea of the performance of each machine class Table 1 presents their execution times for the computation of a chunk for three different matrix sizes.

We compared our Adaptive Time Factoring scheduling algorithm (ATF) to the classical Workqueue algorithm (WQ), and to two factoring algorithms, the non-adaptive Factoring Algorithm (FAC), and the Adaptive Weighted Factoring (AWF) with $\alpha = 2$.

Table 1. Execution time for one chunk (one matrix multiplication - $w = 1$)

n	execution time (seconds)			
	Class A	**Class B**	**Class C**	**Class D**
300	0.80	1.00	1.34	1.44
500	5.18	6.89	9.35	9.52
700	14.82	19.74	27.03	27.22

Table 2. Comparison of the execution times of the algorithms (in seconds)

t	n											
	300				**500**				**700**			
	WQ	FAC	AWF	ATF	WQ	FAC	AWF	ATF	WQ	FAC	AWF	ATF
1000	61.74	62.86	61.7	62.15	486.22	483.98	483.94	483.04	1391.57	1385.12	1378.60	1380.34
1500	94.45	93.79	91.38	92.80	727.03	726.86	724.55	725.84	2081.26	2080.94	2068.97	2072.44
2000	122.69	124.63	121.54	124.77	964.97	965.68	963.32	964.99	2767.57	2769.48	2754.77	2756.48
2500	152.76	155.70	158.16	155.05	1208.22	1207.80	1205.15	1206.89	3456.94	3452.83	3438.00	3444.37
3000	190.16	187.5	183.98	181.9	1449.34	1448.31	1442.63	1445.86	4151.01	4144.19	4126.22	4134.71

We used three different matrix sizes (n): 300, 500 and 700, and five number of multiplications for the workloads w: 1000, 1500, 2000, 2500 and 3000. The obtained results are presented in Table 2.

In most cases ATF outperforms WQ and FAC, particularly in bigger matrices. This is expected because of the heterogeneity of the testbed. Adaptive algorithms can adapt the number of chunks scheduled to a node depending on their performance thus making better use of the resources.

ATF has similar results to AWF in all cases (difference around 1% in execution times). We think this is a very promising result considering that AWF is one of the latest algorithms introduced and is also known for having the best results for the kind of measurements we are performing ([6,7]). Besides, we believe that the benefits of adaptability and distributed scheduling presented in ATF, in order to improve scalability and performance, couldn't be explored in our experiments, since the measurements have been done in a small heterogeneous cluster. We believe that using more machines, ATF will eventually overcome AWF.

5 Conclusions

In this paper we presented the Adaptive Time Factoring (ATF) scheduling algorithm. It is based on a distributed scheduling method, in which each slave computes a chunk sent by the master and, based on its internal performance model, predicts the best chunk size to be computed at the next iteration considering the time slice given by the master. The master distributes time slices between the slaves using a decreasing method based on the Factoring algorithm.

We presented experimental measurements with ATF in a heterogeneous platform and compared it to other algorithms. The results show that ATF outperforms Workqueue and Factoring in most cases, particularly in bigger matrices. ATF showed also similar results to Adaptive Weighted Factoring in all cases (difference around 1% in execution times). We think this is a very promising result considering that AWF is known for having the best results for the kind of measurements we performed. We also believe that ATF, due to its distributed scheduling mechanism, will eventually overcome AWF in a testbed with more machines.

References

1. Chronopoulos, A.T., Andoine, R., Benche, M., Grosu, D.: A class of loop self-scheduling for heterogeneous clusters. In: Proceedings of CLUSTER'2001. (2001)
2. Kruskal, C.P., Weiss, A.: Allocating independent subtasks on parallel processors. IEEE Transactions on Software Engineering **11** (1985) 1001 – 1016
3. Polychronopoulos, C.D., Kuck, D.J.: Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. IEEE Transactions on Computers **36** (1987) 1425–1439
4. Hummel, S.F., Schonberg, E., Flynn, L.E.: Factoring: A method for scheduling parallel loops. Communications of the ACM **35** (1992) 90–101
5. Hummel, S.F., Schmidt, J.P., Uma, R.N., Wein, J.: Load-sharing in heterogeneous systems via weighted factoring. In: Proceedings of the 8th Symposium on Parallel Algorithms and Architectures. (1997)
6. Banicescu, I., Velusamy, V.: Performance of scheduling scientific applications with adaptive weighted factoring. In: Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2001) - Heterogeneous Computing Workshop, San Francisco (2001)
7. Banicescu, I., amd Johnny Devaprasad, V.V.: On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. Cluster Computing: The Journal of Networks, Software Tools and Applications **6** (2003) 213–226
8. Banicescu, I., Liu, Z.: Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes. In: Proceedings of the High Performance Computing Symposium (HPC 2000)ac, Washington (2000) 122–129
9. Banicescu, I., Velusamy, V.: Load balancing highly irregular computations with the adaptive factoring. In: Proceedings of the IEEE - International Parallel and Distributed Processing Symposium (IPDPS 2002) - Heterogeneous Computing Workshop, Fort Lauderdale (2002)
10. Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T.: Numerical Recipes in C: The Art of Scientific Computing. 2nd edn. Cambridge University Press (1993)
11. Beaumont, O., Legrand, A., Robert, Y.: Scheduling divisible workloads on heterogeneous platforms. Parallel Computing **29** (2003) 1121–1152

Adaptive Policy Triggering for Load Balancing[*]

Dan Feng and Lingfang Zeng

Key Laboratory of Data Storage System, Ministry of Education,
School of Computer, Huazhong University of Science and Technology, Wuhan, China
dfeng@hust.edu.cn, jonseng@263.net

Abstract. Load balancing is an attractive problem in storage system. With the fast growth of high-speed network technology and novel storage architecture, smarter storage device becomes an effective way to solve the problem. In this paper, we present an effective object migration & replication policy in our object storage system (OSS), denoted adaptive policy triggering. This policy migrates/replicates object from congested object storage controllers to relatively uncongested object storage controllers according to the information of three facets in the OSS, including metadata server, object storage controller and storage object itself. First, clients interact with the metadata server (MS). So those global policies are triggered by the MS. Second, the OSC has better knowledge of its own load than MS. It is reasonable that local policy is triggered by the OSC. Third, the storage object is encapsulated with data, attributes and methods. These attributes can be set or got when objects are accessed. And object attribute values are good as policy threshold. Furthermore, object methods also can be triggered according to some policies.

1 Introduction

Load-balancing strategies may be best when a large number of users with high-speed connections to the servers access a relatively small quantity of rarely updated information. In that case, load balancing is beneficial primarily in that it provides a measure of redundancy (e.g. replication, snapshot etc.). Load balancing is also good when most of the hits on the server are on a small group of pages, which negates using hyperlinks to disperse content to multiple servers.

However, server load balancing has been excessively studied. Such load balancing policy improves network performance by distributing traffic efficiently so that individual servers are not overwhelmed by sudden fluctuations in activity. Server load balancing is a guide to this critical component of high availability, clustering, and fault tolerance, all of which provide the infrastructure for reliable Internet sites and large corporate networks. Those technologies often solve problems, but not always, and, specially, not for storage system.

Moreover, scarce research aim at helping users reduce server hardware and lessen bandwidth needs in large data centers. In fact, load balancing and failover are crucial

[*] This work was supported by the National Basic Research Program of China (973 Program) under Grant No. 2004CB318201, Huo Yingdong Education Foundation under Grant No.91068, the National Science Foundation of China under Grant No.60273074 and No.60303032.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 240 – 245, 2005.

features in any storage system. Nowadays, storage industry wants an alternative that people can keep servers costs low and keep the amount of work they have to do as small as possible. So, intelligent storage device is taken into account.

The OSS (object storage system) is the next wave of storage technology and devices [1]. Above mentioned researches identify weak points in devoting their mind to the servers (or application servers) and overlooking storage nodes or using traditional storage interface (file or block) in storage nodes, but are trying to gear toward load balancing. It may be a new approach for us to design smart storage system with the help of the OSS. This paper just discusses the intelligence of object storage device and provides a load balancing policy based on adaptive policy triggering.

2 Object Storage System Overview

2.1 The OSS Structure

In the OSS, objects are primitive, logical units of storage that can be directly accessed on an object storage controller (OSC). The OSS built from the OSCs is shown in Figure 1. A metadata server (MS) provides the information necessary to directly access objects, along with other information about data including its attributes, security keys, and permissions (authentication). The OSCs export object-based interface, and the access/storage unit is object. It operates in a mode in which data is organized and accessed as objects rather than as an ordered sequence of sectors. Clients contact with MS and get the information about objects. The OSCs receive and process those requests with some policies. In our previous work [2], smart object storage controller is introduced.

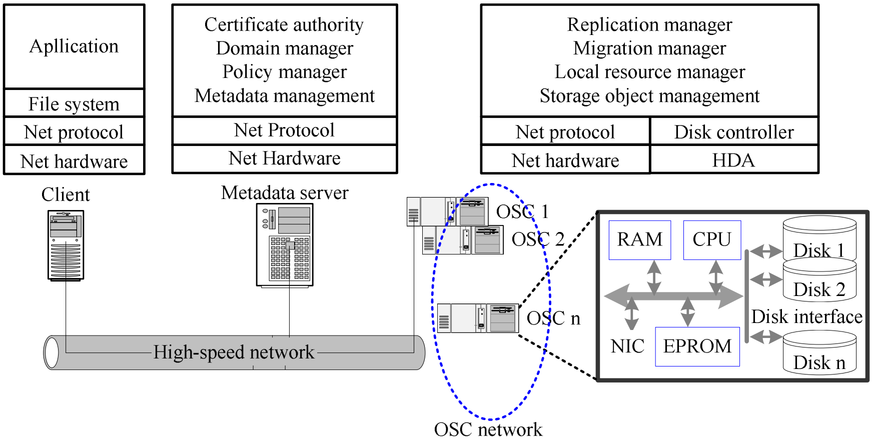


Fig. 1. Object storage system architecture

2.2 Object Attribute and Method

The object is the fundamental unit of data storage in the OSS. Storage object [1], [3], [4], [5] is a logical collection of bytes in the OSC. An object on the OSC consists of an ordered set of sectors associated with an object ID (OID). Data is referenced by the OID and an offset into the object. Conceptually similar to a file, it is allocated and placed on the media by the OSC itself, while the operating system manages its files and metadata in these object constructs, instead of managing sectors of data.

For convenient managing, other object attribute can be extended to three types, such as public attribute, privacy attribute, share attribute. For example, public attribute is opaque to the storage device and are used by applications or MS to store higher-level information about the object, such as OID, object name, object type (e.g. file, device, database table) and storage map. Also, in the OSS, the OSC is a special device object and comprises some attributes, such as the OSC's initial capacity, remaining capacity and IP address.

In the OSS, method may be a user-defined modular operation on stream data, and it is applied to per-object basis. It takes input data, performs operation on the data, and then passes data to its output. Each object in the OSS has associated with reading and writing streams. Clients can insert any modular methods into read and write streams. When clients read/write objects, the objects enter the stream at one end, progress call those register methods, and the methods are executed when the data passing through them. The OSS provides flexible methods by supporting arbitrary data stream operations. Thus, clients can upload any kind of methods, and register any kind of method for any objects.

3 Adaptive Policy Triggering

A "policy" can be thought of as a coherent set of rules to administer, manage, and control access to network resources [5]. With object attribute scalability, abundance clues are obtained to guide or direct in the solution of self-managing by the OSS.

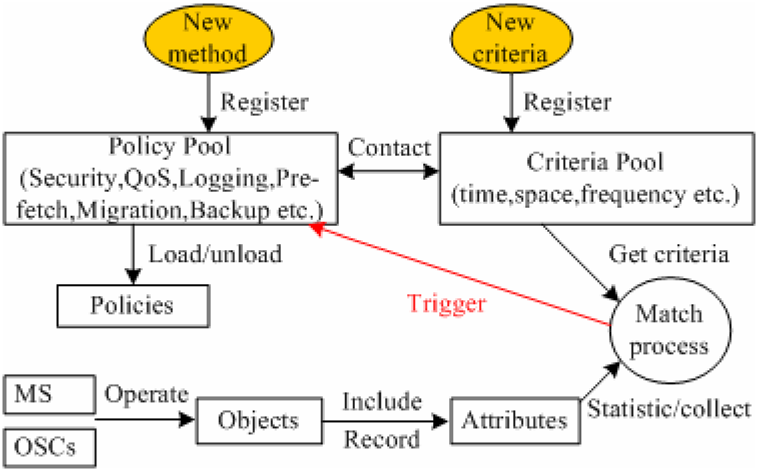


Fig. 2. Adaptive policy triggering based on OSS

Figure 2 shows our design for adaptive policy triggering. It is a simple version from our previous work [8]. In our design, criteria and policies are separated. It is because that one policy may correspond to several criteria, and one criterion may adapt to different policies. The criteria pool is filled with general values regarded to be useful by one or more of the management policies. Most popular criteria such as time, frequency of access, capacity and size are initially registered to the pool. A new policy registered into the policy pool has to contact corresponding criteria in case they are not already registered. So the OSS can provide a solution for dynamic loading or unloading policy.

The adaptive policy triggering ensures that the registered criteria are updated on performance of storage system state. Those policies themselves are just descriptions of how to implement system management function and specify system states and how to response to them. These may include any proposed storage system management policy. When clients or system operate objects, the OSS records correlative object attribute values. For the adaptive policy triggering, those correlative policies are triggered and therefore have the largest effect on the storage system load. The adaptive policy triggers the policy depending on the match process between object attribute values and criteria from the criteria pool.

4 Load Balancing Based on Adaptive Policy Triggering

As distributed systems span the globe, placing objects near the point where the objects are accessed is becoming important to improve service performance and to reduce network load. With the adaptive policy triggering, the OSS will have the ability to transparently migrate or replicate objects among different OSCs based on, for example, QoS issues.

4.1 Triggered by Metadata Server

MS authenticates client request and authorizes client access data with capability. In fact, a map recorded information between OID and the OSCs is got from the MS. After authentication and authorization, client directly access data with map and capability got from the MS. At the same time, all MSs may act as a role of router. They routing management information and copy (or move) some object metadata. Management message can be routed efficiently for metadata discovery and system management.

As a resource manager, the MS record concerned information of all OSCs, such as total capacity, used space, available space, optional striping requirement among the OSCs, total I/O bandwidth and IP address. The MS monitor data requests in the OSS. According to the object access pattern, object may be migrated, replicated or stripped among the OSCs to achieve good locality, load balancing and high scalability. Load balancing among the MSs can also be achieved based on this same object access monitoring.

For instance, whenever additional space is required, there would be a central authority to which MS in the network could turn to find additional space or to find all the OSCs available to it. This could be the basis for operating systems being more dynamic and flexible as to what hardware they are operating with at any point. It need not be the peripheral set that was present at system generation time or even power up time.

4.2 Triggered by Object Storage Controller

Local intelligence is achieved in the OSCs and is a basis of the whole object storage system. The OSC has better knowledge of its own load than MS. If I/O load of one OSC reach the local threshold, the OSC may initiate replication of popular objects to enforce load balancing or may migrate some hot spot object.

Moreover, with the OSCs understanding quite precisely which objects are in use and which are not, the cache space can be more effectively utilized. It should also make scalability more linear by increasing storage management capability at the same rate as the number of the OSCs increases. The OSCs would take over space management, eliminating any increase in OS overhead.

At the same time, each OSC has some additional processing power to apply some other tasks. They could contribute to breaking the task of data management into many simple, small functions performed concurrently. If the OSCs knew enough about what work was going on, they could make sure that an export operation only took place when an object was in a consistent state. Support load balancing by having the OSCs be as knowledgeable as possible about their own conditions and informing the appropriate service of those conditions or acting in response to those conditions as guided by policy assignments. The OSC could allocate an object to whatever zone is most appropriate given the users interest in cost versus performance.

4.3 Triggered by Storage Object

Object attributes may describe how object data are stored or accessed. So object can initiate load balancing policy itself. Object is composed of data, attributes and methods. Some policy about object management is listed in object attribute set. This is similar to the inode attributes inside a traditional file system. But object method may lib-like program and can be dynamicly loaded/unloaded. (Of course, those methods must be registered firstly.)

For instance, an object attribute could be set for an object object when the object was closed after an updating, the OSC could automatically keep the old version of the object while giving the new one a separate OID. Similarly, an object attribute might be set to indicate that an object should be exported after it was updated. Object attributes describe characteristics of the data. Most of object attributes are used by the OSC to manage the storage object. Which include object ID, block pointers, logical length etc. In the OSS, the OSC is a special kind of object (device object) whose attributes are used by applications and MS to store device information. Some structure information is opaque to OSC and includes higher-level information about system management. For example, like HP AutoRAID [7], a few OSCs may constitute a RAID. And some object attributes contain information about its environment, group and user access control information etc.

5 Conclusions and Future Work

Differing data requirements, system complexities, and cost constraints mean that storage system load balancing needs vary widely from servers to devices. The adaptive policy triggering is provided to offering a novel loading balancing solutions to meet this diversity of need. And the adaptive policy triggering makes it possible for storage system to take advantage of object storage architecture to enable intelligent storage.

This paper first outlines object storage system and object characteristic. Next, the paper explores details of the object storage technology, and then it places those details in the context of development of the load balancing, which includes replication and migration. Our current design constitutes only narrow application. We will perform other application using the adaptive policy triggering, such as backup, caching and logging. At the same time, large-scale applications will enforce us to minimize computational overheads and to require the use of more efficient data structures and powerful machine learning algorithm.

References

1. Intel Corporation, “Object-Based Storage: The Next Wave of Storage Technology and Devices”, January 2004, accessible from http://www.intel.com/labs/storage/osd/
2. Ling-Fang Zeng, Dan Feng, Ling-jun Qin, “SOSS: Smart Object-based Storage System”, the Third International Conference on Machine Learning and Cybernetics, Shanghai, pp.3263-3266, 26-29 August 2004.
3. IBM Storage Tank, March 2004, accessible from http://www.haifa.il.ibm.com/storage.html
4. White paper: Object Storage Architecture, January 2004, accessible from http://www.panasas.com/activescaleos.html
5. P. J. Braam, “The Lustre storage architecture”. Technical report, Cluster File Systems, Inc., January 2004, accessible from http://www.lustre.org/docs/lustre.pdf
6. A. Westerinen, J. Schnizlein, J. Strassner etc. “Terminology for Policy-Based Management”, RFC 3198, November 2001.
7. HP AutoRAID: Setting New Standards for Fault-Tolerant Storage. March, 2004, accessible from http://www.interex.org/pubcontent/interact/sept95/11spot/spot.html
8. Dan Feng, Lingfang Zeng, Fang Wang, Lingjun Qin, Qun Liu, “Adaptive Policy Trigger Mechanism for OBSS”, The International Conference on Advanced Information Networking and Applications (AINA-2005),Tamkang University, Taipei, Taiwan, March 28 - 30, 2005.

Parallel Algorithms for Fault-Tolerant Mobile Agent Execution

Jin Yang[1], Jiannong Cao[1], Weigang Wu[1], and Cheng-Zhong Xu[2]

[1] Internet and Mobile Computing Lab,
Department of Computing, Hong Kong Polytechnic University,
Hung Hom, Kowloon Hong Kong
[2] Department of of Electrical and Computer Engg,
Wayne State University, Detroit, Michigan 48202, USA

Abstract. Redundancy is a basic technique for achieving fault tolerance, but the overhead introduced by redundancy may degrade system's performance. In this paper, we propose efficient replication based algorithms for fault-tolerant mobile agent execution, which allows for parallel processing in the agent execution so as to reduce the overheads caused by redundancy. We also investigate the heartbeat based failure detector approach and modify it for use in our proposed algorithms. Performance evaluation has been performed to compare the proposed algorithms with the existing algorithm. Both analytic and simulation results show that our new algorithms can significantly improve system's performance.

1 Introduction

A Mobile Agent (MA) is a program that can migrate from host to host in a network of heterogeneous computer systems to execute the tasks specified by its owner. The migration path can be fixed according to a predefined itinerary or dynamically decided using a self-initiated itinerary. A mobile agent works autonomously and communicates with other agents and host systems. During the migration, the agent carries its code and some kind of execution state. On each host of the network, a MA platform is responsible to execute the mobile agent's operations, provides a safe execution environment, and offers services for MAs residing on this host. A MA system is the set of all MA platforms of the same type together with the MAs running on these platforms as part of an agent-based application. Many applications of mobile agent have been reported including Electronic Commerce[11], Information Retrieval[12], Network Management[13,14] and Mobile Computing[15].

However, before we implement mobile agent-based applications, some important issues such as fault tolerance must be addressed. Many fault-tolerance schemes have been proposed, and one of the most popular solutions is the replication based scheme [5,6,7,8,9]. The basic idea of replication based scheme is to maintain some replicas for the working MA. If the working MA failed and is detected by a replica, then the replica will create a new working MA to continue the task. On the other hand, if the working MA detects that the replica failed, it will generate a new replica to replace the failed one. So the working MA and the replica will guard each other. Replication based schemes has its shortcomings. The first is the overheads, which may slowdown the system execution dramatically. The second problem is the failure detection. Heartbeats

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 246 – 256, 2005.

algorithm is a well known failure detection technique, which requires the peers to keep on exchanging the heartbeat messages. But for mobile agent applications, no message can be delivered during a MA's migration. So the traditional heartbeats algorithm does not work properly and modification is needed.

In this paper, we address these two problems. We propose replication based algorithms for fault-tolerant mobile agent execution. Parallel processing is introduced in our proposed algorithms. System's overheads are reduced and the performance is improved through parallel processing. We also modify the traditional heartbeats algorithm with handover procedures for failure detection in fault tolerant mobile agent executions. To the best of our knowledge, this work is the first study of the implementation of failure detector in mobile agent environment.

The rest of the paper is organized as follows: section 2 describes related works and the motivations of our research. Section 3 describes our proposed mobile agent fault tolerant execution algorithms and the failure detection mechanisms in detail. Section 4 presents the analysis of the algorithms' performance, and validates the analysis results through the simulation. The performance is compared with the well known rear-guard algorithm. Finally, section 5 concludes this paper.

2 Related Works and Our Works' Motivations

Most of the replication-based MA execution algorithms in literature are based on the same rear-guard model. A working agent is followed by one or several replicas, called the rear guard agents [5]. If the working agent failed, the rear guard agent will continue the job for the failed agent. Later works made improvement on and reported implementations of this model. In [6], the authors presented a "sliding window" mechanism. Before each migration of a MA, a specific number of backups of this MA are duplicated in order to avoid the collapse or disappearance of this MA. In fact, the backups of the agent just play the role of the rear guard agents. The size of the window is adjustable and determines the number of backups used. In [7], "surrogate of agent" is used, which is just another name of rear guard agent. A MA will leave a surrogate on each host it visited. Once a surrogate finds out that the MA failed, it will recreate an agent to continue the job. A mobile shadow scheme is proposed in [8], which employs a pair of replica mobile agents, the master and the shadow. In [9] a pipelined model is proposed, in which a witness agent is behind a working agent. In fact, both the shadow and the witness agent act as a rear guard agent.

The rear guard agent only guards the failure status of the working agent, and keeps consistency with the working agent in order to continue the work of a failed working agent. In order to improve the system performance, we can let the replicated MA undertake tasks that can be done concurrently with the working agent. In [1], the authors make the use of two reverse MAs to execute in parallel by reverse itinerary to gain higher system execution speed. In [2], two MAs executing in reverse itinerary to speed up the execution and improve fault tolerance. But these works focus on achieving load balance and sensor networks' performance respectively. The fault tolerance execution of MA is not their main concern thus not addressed.

A problem common to all these works is that they did not mention how to detect failures. Failure detector [3, 4] is the mechanism necessary for detecting the failure of an executing entity in the system. The heartbeat-style failure detectors have been widely implemented in real systems. Paper [3] also described how to configure a failure detector to satisfy the predefined QoS. In [4], authors proposed how to make estimations about the arriving time of heartbeat messages. However, the conventional heartbeat-style failure detectors have several problems for mobile agent systems. First, a MA cannot deliver the heartbeat message during its migration. We call this period the dumb period. Second, there is the possibility of false detection. In [3], the authors proposed a set of quantitative measures for false detection, which include: mistake recurrence time, and mistake duration. If a replica receives a false detection from the failure detector, it may regenerate a new working MA to replace the "failed" one, but the fact is the working MA is not failed. So it may cause duplicate execution.

In summary, although rear-guard algorithm provides fault tolerance for MA system, it is not efficient. Also, conventional heartbeat-style failure detectors are costly and the false detections will cause extra troubles. All these will affect the system performance. But the fact is many applications for data retrieval applications such as network management need fast data collection. Data submitted late usually is not useful, and even harmful to the system. So fault-tolerance algorithms should be efficient. We will describe our proposed efficient replication based MA fault tolerance algorithms in section 3.

3 Replication Based MA Fault-Tolerant Algorithms

The main idea of improving the efficiency of replication based mobile agent algorithms is to introduce parallel processing among the replicas. According to whether the MA's itinerary is predefined or not, we propose two algorithms, namely Reverse MAs Algorithm (RMAA) and Alternate MAs Algorithm (AMAA).

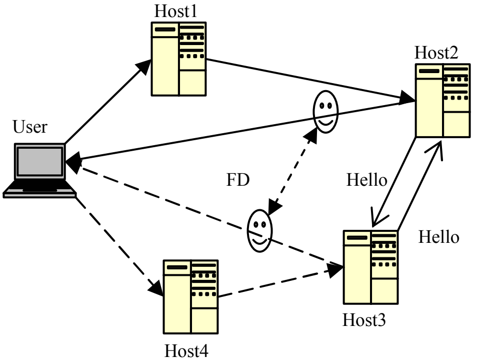


Fig. 1. RMAA execution process

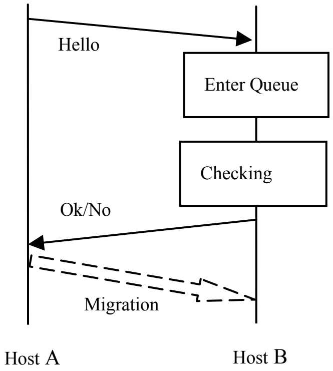


Fig. 2. Landing procedure

3.1 RMAA

RMAA is well suited for MA applications with a predefined itinerary and no requirement on the host visiting sequence. One typical example is the information retrieval applications. In RMAA, the original predefined itinerary is the *forward* *itinerary* and the *reverse itinerary* is an itinerary that reverses the sequence of hosts in the forward itinerary. There are two MAs in RMAA. One is called *Forward MA* (FMA) which will visit hosts according to the forward itinerary, and another is called *Reverse MA* (RMA) which will visit hosts according to the reverse itinerary.

Fig. 1 illustrates the RMAA scheme. The pair of MAs is dispatched by the user's mobile agent system at the same time. They execute concurrently along their own itineraries until they reach two neighboring hosts (e.g., Host 2 and Host 3), which indicates that all the hosts on the itinerary have been visited. The two MAs will then return the MA platform on user's host. In order to prevent both MAs failures due to the failure of the host, the two MAs are not allowed to land on the same host. For this purpose, a landing procedure is needed (Fig. 2).

The two MAs send the coordination message "Hello" before migration to the next host. The "Hello" message is put into a queue on the MA platform of the next host, which ensures that the host only accepts one MA with the earlier "Hello" in the pair MAs. A MA can migrate to the host only if it has received an "Ok" message as response from the host. If two MAs send the "Hello" message to the host simultaneously, the host will receive both of them. But in the queue, one will precede another one. For the sender of the later "Hello" message, MA platform will reply it with a "No" message. When the MA receives a "No" message, it knows that another MA is on the neighboring host. So this MA will go back user host. The MA which got the "OK" message will return user host too after it finishes its execution.

Same with the rear-guard algorithm, we assume that the FMA and the RMA will not fail at the same time. During the execution of the pair of MAs in RMAA, one MA may fail during its execution or migration. The failure detector will detect the failure and inform another MA, and the living MA will generate a new MA to replace the failed MA (Fig. 3). For this purpose, FMA and RMA should keep each other's computing results (this is the same with rear-guard algorithm). A distinguish advantage of the RMAA algorithm is that it can handle the itinerary partition due to links failure. In Fig. 4, the itinerary is partitioned into two separated sections. It is obviously that the pair of MAs can finish their tasks if they will not fail.

The RMAA scheme can be easily extended to accommodate n ($n \geq 1$) pairs of MAs to speed up the execution in large-scale networks. The original itinerary can be separated into n sections, and on each section RMAA is executed.

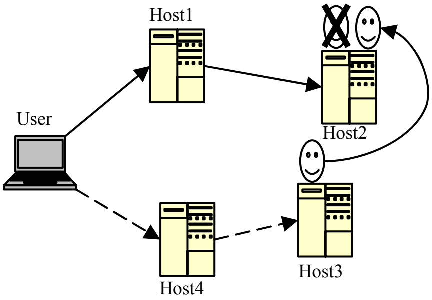


Fig. 3. The failure handling of RMAA

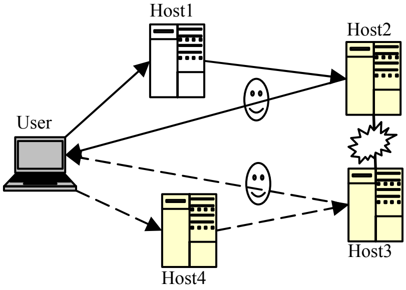


Fig. 4. Itinerary partition

RMAA can be implemented on system level in a way transparent to the application programmer. What the programmer needs to do is just to provide the MA's task and itinerary to RMAA. RMAA will create FMA and RMA to finish the users' task. The algorithm of pseudocode format for executing RMAA is illustrated in the following box.

```
//RMAA is a class which implements all the functions of RMAA algorithm. User just needs to create //a
RMAA object and provides the Task and Itinerary to the RMAA object.
1. RMAA rmaa = new RMAA (Itinerary, Task); //RMAA creates 2 members: a FMA and a RMA;
2. rmaa.Launch(); //FMA and RMA are launched;
//FMA and RMA execute the same code in parallel. We only describe FMA's execution.
3. if (rmaa.FMA.tryMigration() = = OK) //will not encounter RMA
    {rmaa.FMA.migration(); //migrate to next host
     result = rmaa.FMA.Task.start();
     rmaa.FMA.synchronize(result); //synchronize the computing result for failure handling.
     goto 3; //Finish the execution on current host, then try to go to next host.
   }else  //will encounter RMA if migrate to the next host. So FMA returns home.
    rmaa.FMA.returnHome();

// Pseudocode for the MA failure handling. Suppose ma gets a message from failure detector.
if (msg = ma.getmessage() = = MA_Failure) //get asynchronous message from failure detector
   ma1 = ma.clone();  //this ma will clone a new ma according to the failed ma's infomation.
   ma1.migration(msg.host, failureMA_id); //the cloned ma migrates to the host.
//After the cloned ma lands on the host, it will check the reported ma is really failed or not.
   if (ma1.check(failureMA_id) = = ReallyFailed) //if the reported ma really failed, its job will be
       ma1.resumeFailedma();                //continued.
```

3.2 AMAA

A predefined itinerary is necessary for RMAA. But one of the fundamental features for mobile agent is autonomy, which allows a MA to determine the next host dynamically without a predefined itinerary. RMAA is not applicable under such a context while the rear-guard algorithm can still work. But the rear-guard algorithm is not efficient and we seek a faster algorithm.

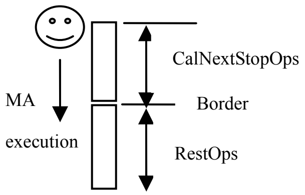


Fig. 5 A MA's operations

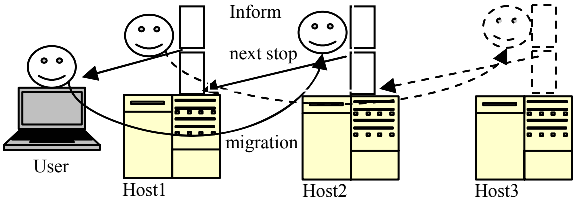


Fig. 6 AMAA execution process

For a mobile agent application without predefined itinerary, an agent needs to compute the next stop before every migration. Accordingly, we divide a MA's operations into two sections (Fig. 5): *CalNextStopOps* contains all the necessary operations which have to be done in order to get the next stop; *RestOps* includes the rest operations (the de-registration operations at lest). The border between these two sections can be different for different applications. Some applications can determine the next stop in the first few steps; some get it at last.

AMAA involves two MAs. One MA which is on the head is called *Leading MA* (LMA); the other MA which is behind the LMA is called *Slave MA* (SMA). The two MAs should arrange their operations in two sections as described above. Fig. 6 shows the execution process of AMAA. The MA platform on user host launches two MAs. One lands on the first stop and becomes LMA. The other who is waiting on the user side becomes SMA. When the LMA got the result of the next stop, it sends a message to the SMA which is still waiting on the user side. SMA migrates to the next stop and becomes the new LMA and starts its execution (former LMA becomes SMA now). When LMA determines the next stop, it sends a message to SMA. Now the SMA may or may not finish the *RestOps*. When SMA finishes the *RestOps*, it will migrate to the next stop. The process will continue until the task is finished.

Same with the rear-guard algorithm and RMAA, Failure detector will inform the failures of MAs, and the living MA will generate a new MA to replace the failed MA. Different from RMAA, AMAA can not handle the itinerary partition.

AMAA can be extended to involve *n* (*n* ≥ *2*) MAs easily. Among the *n* MAs, One acts as the LMA. The rest *n-1* MAs form a sequence of SMAs. When the LMA gets the next stop, it informs the last SMA. The last SMA migrates to the next stop and becomes the LMA. Previous LMA becomes the first SMA in the sequence of SMAs.

Same with RMAA, AMAA can also be implemented at the system level. Users need not provide the itinerary, but the task is required to separate into two sections as we described. The following box illustrates the pseudocode for AMAA.

```
//AMAA is a class which implements all the functions of AMAA algorithm. User just needs to create
//a AMAA object and provides the Task to the RMAA object.
1. AMAA ma[ ] = new AMAA (Task); //AMAA creates 2 members: an LMA and a SMA;
2. NextHost = FirstHost; ma[0].end = false; ma[1].end = false; //Initiation;
3. ma[0].goto 4; ma[1].goto 9; //ma[0] is current LMA and ma[1] is current SMA.
//ma[0] and ma[1] share the same code from 4 to 9. In the following, "ma" can be ma[0] or ma[1].
4. ma.migration(NextHost);
5. NextHost = ma.Task. CalNextStopOps();
6. if (NextHost ≠ NULL)
     ma.informSMA(NextHost); //After the current SMA get this message, it will migrate
   else              // to next host and becomes the new LMA. This ma becomes the new RMA.
     {ma.informSMA(NULL); //No next host, so inform SMA to return home.
      end = true; //This mark will make LMA return home
     }
7. result = ma.Task. RestOps(); //Finish the rest operations.
8. ma.synchronize(result); //synchronize the computing result
9. if (end = = ture) //No next host.
     ma.returnHome();
   else if (ma.getNextHost() ≠ NULL) //SMA get the next host which is sent by LMA
       {goto 3;              //SMA will migrate to the next host.
       } else  //No next host
         ma.returnHome(); //it is time to go home.
// Pseudocode for the MA failure handling in AMAA is the same with RMAA.
```

3.3 Failure Detection Mechanisms for MA Applications

The function of failure detection is a fundamental requirement for replication based fault tolerance algorithm. As we introduced in section 1, heartbeat-style failure detectors are widely used. But a big problem for a heartbeat-style failure detector is false detection. For MA applications, another problem is the dumb period (section 2).

For the problem of dumb period, a handover procedure is needed. A simple solution is that before a MA starts migration, the "migration" tag is piggybacked in heartbeat message. When the failure detector monitoring the MA receives the message with the "migration" tag, it will stop the failure detection for this MA and wait until it receives the new heartbeat message (at this time, the MA lands on a new host). The problem for this scheme is that, if the MA is lost during migration, failure detector can not detect it. An enhanced scheme is based on MA's reliable migration. When a MA starts a migration, it sends a replica to the next host and waits until the replica landing on the next host. During the migration process, the waiting MA can keep sending heartbeat messages to the failure detector. After the replica lands on the new host, it informs the waiting MA and the waiting MA will hand over the task of heartbeat message exchanging to the replica. Through this scheme, the dumb period problem can be solved and the failure detector can keep on the monitoring task.

False detection is an inherent problem for heartbeat-style failure detectors. What we can do is to add a checking procedure. When a new MA is generated to replace the failed MA, the new MA should check the failed MA's status on the host of the failed MA. We assume we can check the real status locally. If the new MA finds out that it is a false detection, it will kill itself to avoid the duplicate execution.

4 Performance Analyses and Evaluations

In this section, we first make an analytic analysis on the execution time for the different fault tolerant MA execution algorithms, and then describe our simulation study.

4.1 Analysis on Execution Time

In the following discussing, we assume the execution time T for a MA is the same on each host. N is the number of hosts. The time for a MA migrating from the current host to the next stop is T_m. T_{Task_exe} is the total execution time of each algorithm. For rear-guard algorithm, when the working MA starts a migration, it will inform the rear guard MA to keep following. We assume the time needed for this operation is T_{inform}. In RMAA, $T_{landing}$ is the time needed by each landing procedure. In AMAA, like the rear-guard algorithm, T_{inform} is the time needed by the operation of LMA informing the next stop to SMA, and according to Fig. 5, we assume the time taken for each *CalNextStopOps* is $T_{CalNextStopOps}$; the time taken for each *RestOps* is $T_{RestOps}$. It is obviously that $T = T_{CalNextStopOps}+T_{RestOps}$. For simplicity, we do not consider the cost of heartbeat messages and synchronization messages, because they are needed by all of our discussed algorithms.

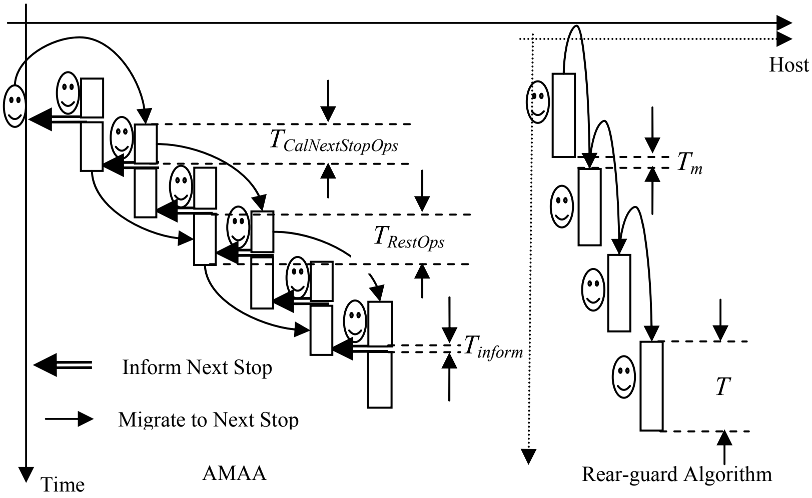


Fig. 7. Execution time comparisons

For the rear-guard algorithm, the whole task is finished by the single working MA and no parallel processing is involved (Fig. 7). So we can get: $T_{Task\_exe} = N(T+T_m+T_{inform})$. For RMAA, the FMA and RMA execute in parallel, so ideally the execution time is: $T_{Task\_exe} = N(T+T_m+T_{landing})/2$. AMAA allows partial parallelism in MA executions, and its execution time depends on how much job is done in parallel. From Fig. 7, we can figure out how to compute the execution time for AMAA.

$$T_{Task\_exe} = \begin{cases} N(T_m+ T_{CalNextStopOps} + T_{inform})+ T_{RestOps} & T_{RestOps} < T_m+ T_{CalNextStopOps} + T_{inform} \\ (N+1)(T_m+ T + T_{inform}) & T_{RestOps} \geq T_m+ T_{CalNextStopOps} + T_{inform} \end{cases}$$

The parameter $T_{RestOps}$ determines the degree of parallism that can be achieved. If we can increase the $T_{RestOps}$, the execution time of AMAA will be shortened. However, the reduction in the execution time is bounded that the total time will be no less than $(N+1)(T_m+T+T_{inform})/2$, if $T_{RestOps}$ is greater than $T_m+T_{CalNextStopOps}+T_{inform}$. We define this $T_{RestOps}$ as the AMAA critical value. For AMAA involving n MAs, it is easy to see that the task execution time will be $(N+1)(T_m+T+T_{inform})/n$, $(2\leq n\leq N$, $T_{RestOps} \geq T_m+T_{CalNextStopOps}+T_{inform})$.

Table 1 summarizes the execution modes and execution time for all the algorithms discussed in this paper. Note that for AMAA in the table, we assume that $T_{RestOps}$ is set as the critical value.

Table 1. Execution mode/time comparisons

	Itinerary	Execution mode	Theoretical Execution Time (*N* hosts)	
			2MAs	*n*MAs (*n*>2)
Rear-Guard	Self-initiate	Non-parallel	$N(T+T+ T_{inform})$	$N(T+T+ T_{inform})$
RMMA	Predefined	Full- parallel	$N(T+ T_m+ T_{landing})/2$	$N(T+ T_m+ T_{landing})/\mathrm{n}$
AMAA	Self-initiate	Partial- parallel	$(N+1)(T_m +T+ T_{inform})/2$	$(N+1)(T_m +T+ T_{inform})/n$

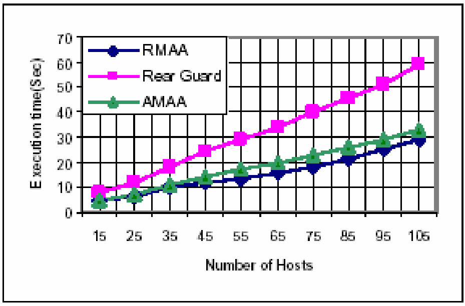
We can see that RMAA can provide the fastest execution speed because T_{inform} is almost the same with $T_{landing}$ and they are some time intervals comparing with T_M and T. But RMAA needs a predefined itinerary and it also requires the system allow a random hosts accessing sequence. These requirements make RMAA inflexible. AMAA has the same degree of flexibility as the rear-guard algorithm, but its execution time can only be shortened if the next stop can be calculated quickly (then $T_{RestOps}$ will becomes bigger). If AMAA can only get the next stop at the last step of its operation ($T_{RestOps} = 0$), the execution time will be the same with the rear-guard algorithm: $T_{Task\_exe} = N(T_m+T_{CalNextStopOps}+T_{inform})+T_{RestOps} = N(T_m+T+T_{inform})$. But normally the $T_{RestOps}$ will not be zero, because a MA has to perform some routing operations on a host at last, such as deregistration, release resources, etc. So we can always gain the partial parallelism so as to shorten the execution time. The results in Table 1 are just the theoretical values. In practice, the real execution time will be longer due to various overheads. We will compare the execution time in the simulation study to be described in the next subsection.

4.2 Simulation Results

In order to compare the execution time in realistic environment, we performed simulations of the rear-guard algorithm, RMAA and AMAA on the Naplet MA platform [16].

The simulations are carried out on a PC with Pentium 4 CPU (2.5GHz), 256MB RAM. The software environment is: Window XP, Java version 1.4, and Naplet MA platform. Five Naplet MA platforms are installed on the PC and we simulate the MA traveling 15, 25, 35, 45, 55, 65, 75, 85, 95, 105 nodes respectively using different fault tolerant algorithms. The number of MA failures is set to be 1/20 of the total number of hosts that have been visited and the failures are uniformly distributed along its itinerary. The exchange frequency of the heartbeat messages is 5 messages per second. Enhanced handover scheme is adopted in the simulation. For AMAA, we set the $T_{RestOps}$ to be its critical value, which means a MA will send out the next stop message in the mediate of its execution.

From the simulation results in Fig. 8, we can see that RMAA only takes about half of the rear-guard algorithm's execution time. AMAA also takes near half of the execution time of rear-guard algorithm. For the number of messages exchanged, Fig. 9 shows that it increases in direct proportion to the execution time. That is because the heartbeat messages take up the most part of the exchanged messages during MA's execution. Longer execution time will cause more heartbeat messages exchanging.



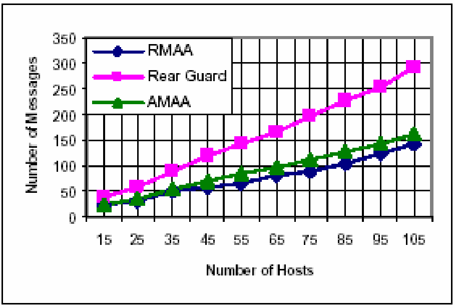


Fig. 8. Execution time comparisons

Fig. 9. Exchanged messages

5 Conclusions and Future Works

In this paper, we described the two efficient replication-based mobile agent execution algorithms. The algorithms allow for parallel processing and provide fault tolerance. Analytic analysis and simulation results show that the proposed algorithms can improve system's execution speed dramatically. The shorter execution time can help MA bypass host failures with greater probability and reduce the number of heartbeat messages exchanged. The overhead caused by heartbeat-style failure detector is an important issue in designing a high-performance fault tolerant MA system. In our future work, we will investigate this issue and attempt to design alternative approach to failure detection, e.g., using the watch-dog technique with remote notifications.

Acknowledgement

This work is supported in part by the University Grant Council of Hong Kong under the CERG Grant PolyU 5075/02E and China National 973 Program Grant 2002CB312002.

References

1. Jiannong Cao, Yudong Sun, Xianbin Wang, Sajal K. Das. Scalable load balancing on distributed web servers using mobile agents. In Journal of Parallel and Distributed Computing. 63(2003) 996-1005. May, 2003.
2. Hairong Qi; Yingyue Xu; Xiaoling Wang; Mobile-agent-based collaborative signal and information processing in sensor networks Proceedings of the IEEE Volume 91, Issue 8, Aug. 2003 Page(s):1172 - 118
3. Wei Chen; Sam Toueg; Aguilera, M.K.; On the quality of service of failure detectors Computers, IEEE Transactions on Volume 51, Issue 5, May 2002 Page(s):561 - 580
4. R. C. Nunes and I. Jansch-Pôrto. Qos of timeout-based self-tuned failure detectors: the effects of the communication delay predictor and the safety margin. In Proceedings of the International Conference on Dependable Systems and Networks (DSN'04), page 753, Florence, Italy, June 2004.

5. Johansen, D.; van Renesse, R.; Schneider, F.B.; Operating system support for mobile agents Hot Topics in Operating Systems, 1995. (HotOS-V), Proceedings., Fifth Workshop on , 4-5 May 1995Pages:42 – 45
6. Tao Shu; Cao Yang; Yin Jianhua; Xu Ning; A mobile agent based approach for network management. Communication Technology Proceedings, 2000. WCC - ICCT 2000. International Conference on , Volume: 1 , 21-25 Aug. 2000 Pages:547 - 554 vol.1
7. Komiya, T.; Ohsida, H.; Takizawa, M.; Mobile agent model for distributed systems Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on , 2-5 July 2002 Pages:131 – 136
8. Pears, S.; Jie Xu; Boldyreff, C.; Mobile agent fault tolerance for information retrieval applications: an exception handling approachAutonomous Decentralized Systems, 2003. ISADS 2003. The Sixth International Symposium on , 9-11 April 2003 Pages:115 – 122
9. Pleisch, S.; Schiper, A.; Fault-tolerant mobile agent execution Computers, IEEE Transactions on , Volume: 52 , Issue: 2 , Feb. 2003 Pages:209 – 222
10. M.J. Fischer, N.A. Lynch, and M.S. Paterson, Impossibility of Distributed Consensus with One Faulty Process Proc. Second ACM SIGACT-SIGMOD Symp. Principles of Database Systems, pp. 1-7, Mar. 1983.
11. Maes, R.H. Guttman, and A.G. Moukas, Agents that Buy and Sell Comm. ACM, vol. 42, no. 3, pp. 81-91, Mar. 1999.
12. W. Theilmann and K. Rothermel, Optimizing the Dissemination of Mobile Agents for Distributed Information Filtering IEEE Concurrency, pp. 53-61, Apr. 2000.
13. A. Bieszczad, B. Pagurek, and T. White, Mobile Agents for Network Management, IEEE Comm. Surveys, Sept. 1998.
14. T. Gschwind, M. Feridun, and S. Pleisch, ADK—Building Mobile Agents for Network and Systems Management from Reusable Components Proc. First Int'l Conf. Agent Systems and Applications/Mobile Agents (ASAMA '99), Oct. 1999.
15. K. Takashio, G. Soeda, and H. Tokuda, Mobile Agent framework for Follow-Me Applications in Ubiquitous Computing Environment Proc. Int'l Workshop Smart Appliances and Wearable Computing (IWSAWC '01), pp. 202-207, Apr. 2001.
16. Naplet: A flexible and reliable mobile agent system for network-centric pervasive computing. http://www.ece.eng.wayne.edu/~czxu/software/naplet.html

Design and Multithreading Implementation of the Wave-Front Algorithm for Constructing Voronoi Diagrams

Grace J. Hwang, Joseph M. Arul, Eric Lin, and Chung-Yun Hung

Department of Computer Science and Information Engineering,
Fu Jen Catholic University, Taipei, Taiwan
{jihwang, arul, seed}@csie.fju.edu.tw

Abstract. The Voronoi diagram is one of the most fundamental data structures in computational geometry, which is concerned with the design and analysis of algorithms for geometrical problems. In this paper, a parallel algorithm for constructing the Voronoi diagram on CREW (Concurrent Read and Exclusive Write) model is proposed. This is an improved algorithm based on Preilowski and Mumbeck's work. In their algorithm, they apply the Neighbor-Point-Theorem and present a parallel approach to check neighbor points. In this article, we propose an improved approach, Wave-Front algorithm, which is a quite different way to check neighbor points. The algorithm is then implemented in both sequential and multithreaded models. Since the Wave-Front algorithm has inherently concurrent tasks that can be executed simultaneously, multithreaded version was executed to observe the performance. Computational results indicate the effectiveness of the threaded model.

1 Introduction

The Voronoi diagram is one of the most popular geometrical structures in computational geometry [1], which is a branch of computer science concerned with designing efficient algorithms for solving geometrical problems. It partitions a plane with n given points into n convex polygons such that each of which consists of the points closer to one given point than to any others. The Voronoi diagram is an important problem in many applications; including placement and motion planning, mesh generation and proximity problems.

There are several parallel algorithms existing for computing the Voronoi diagram of n planar points on the CREW PRAM model [2]. For instance, Chow [3] uses inversion and computes the convex hull of points in three dimensions. The algorithm runs in $O(\log^3 n)$ time and uses O(n) processors. Preilowski and Mumbeck [4] present a time-optimal algorithm that employs the Neighbor-Point-Theorem to compute Voronoi polygon for each point. Their algorithm runs in $O(\log n)$ time only but uses $O(n^3)$ processors. Aggarwal et al. [5] parallelize a sequential divide-and-conquer algorithm and run in $O(\log^2 n)$ time using O(n) processors. Similar to Chow's result, Evan and Stojmenovic [6] present an $O(\log^3 n)$ algorithm using O(n) processors. Cole et al. [7] also apply the divide-and-conquer approach to construct Voronoi diagram. They present two algorithms, the first one runs in $O(\log n \log\log n)$ time using $O(n \log n / \log\log n)$ processors, the other one runs in $O(\log^2 n)$ time using $O(n / \log n)$ processors. Some other related and recent works such as those Amto et al. [8]

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 257 – 266, 2005.

reduce three-dimensional convex hulls to two-dimensional Voronoi diagrams and Blelloch et al. [9] implement a practical parallel algorithm for the Delaunay triangulation on general distributions.

The goal of this research is to present the proposed Wave-Front algorithm based on Preilowski and Mumbeck's work and describe a parallel implementation using multithreaded model. A number of techniques to further exploit Thread Level Parallelism (TLP) have been researched. Some products including Intel's hyper-threading have been announced [10]. The threaded model can be applied with great success to a wide range of programming, such as large scale, computationally intensive programs and client server applications. From a software or architecture perspective, user programs can schedule threads to logical processor as they would on multiple processors. Multithreading is specifically to take advantage of multitasking environment. In Wave-Front algorithm, multitasking can easily be achieved and is visible from the behavior of the program. Hence, by using multithread, more tasks could be completed to maximize the running efficiency of the program.

The reminder of this paper is organized as follows: Section 2 describes the neighbor-point theorem and multithreading. Section 3 presents the Wave-Front algorithm. Section 4 shows the experimental results. Both of the sequential and multithreaded versions of the Wave-Front algorithm are implemented. The computational results show that the performance of threading version is quite effective. Finally, we conclude with some discussion in Section 5.

2 Neighbor-Point Theorem and Multithreading

In this section, we briefly describe the neighbor-point theorem and multithreading to help understand the Wave-Front algorithm. In later sections, we would describe its design and implementation.

2.1 Neighbor-Point Theorem

The capability of the neighbor-point theorem is to determine all the neighbor points for some point p from a given set of points, the Voronoi polygon for p is therefore obtained. In the following paragraph, we first introduce some definitions and then describe the neighbor-point theorem [4].
Assume, S, is a finite set of points. Then:

1. Let Seg(x_i, x_j) be the segment of a line from x_i to x_j.
2. Let PB(x_i, x_j) be the perpendicular bisector of Seg(x_i, x_j).
3. Let L(x_i, x_j) be the straight line through x_i and x_j.
4. L and R are the subsets of S and they lie left and right of L(x_i, x_j) going along the direction from x_i to x_j.
5. Let $S_{\text{left}} := \{s \mid s$ is the intersection-point of PB(x_i, z) and PB(x_i, x_j) for z in $L\}$.
6. Let $S_{\text{right}} := \{ s \mid s$ is the intersection-point of PB(x_i, z) and PB(x_i, x_j) for z in $R\}$.
7. Define the following order "<" on the points of S_{left} and S_{right}:

   If $p \neq q$ in ($S_{\text{right}} \cup S_{\text{left}}$) then $p < q$ if and only if p lies left of q on PB(x_i, x_j) going along the direction from x_i to x_j.

After the definitions of related terms as given above, the Neighbor-Point Theorem can be described as follows (see Fig. 1).

Neighbor-point Theorem:
Let x_i and x_j in S, $i \neq j$, then
x_j is a neighbor point of x_i if and only if max(S_{left}) < min(S_{right}).

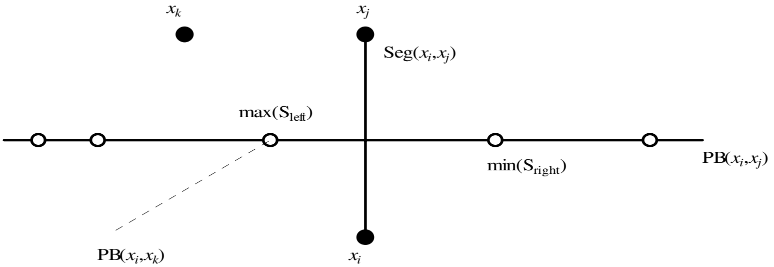


Fig. 1. An illustration of neighbor-point theorem

The neighbor-point theorem can be applied to construct Voronoi diagrams for both sequential and parallel algorithms. We describe the parallel algorithm originally proposed in [4]:

CREW-PRAM Algorithm:

```
1. For all $i$ = 1 to $n$ {
2.   For all $j$ = 1 to $n$ ($j \neq i$) {
3.     Check whether $x_j$ is a neighbor-point for $x_i$ using the Neighbor-point Theorem.
4.     }.
5. Compute the Voronoi polygon for $x_i$ by sorting the pieces got in step 1.
6. }
```

In step 3 of the algorithm, it needs n-2 processors except two for x_i and x_j and the running time is $O(\log n)$. The algorithm also needs $O(n^2)$ processors for steps 1 and 2 when $O(n^2)$ pairs of points exist. Thus, in total, the algorithm needs $O(n^3)$ processors for steps 1 to 3. In step 5, it needs $O(n)$ processors and $O(\log n)$ running time, since the Voronoi diagram is a planar graph and the number of edges is bounded by $O(n)$. Therefore, the running time of this algorithm is $O(\log n)$ with $O(n^3)$ processors. The reason why it uses so many processors, $O(n^3)$, comes from the approach of checking neighbor points. It examines every other point with p to find its neighbor points. However, the neighbor points are usually not far away from p and the checking process could be improved. This leads to our motivation to modify the algorithm.

2.2 Multithreaded Model

Traditional computer programming causes all events to occur in series, unless the programmer takes other measures to allow them to happen concurrently. Behren et al. [11] have shown that the weaknesses of threads are artifacts of specific threading implementations and not inherent to the threading paradigm. Thus, multithreaded implementation in a right way in a particular application can certainly improve performance. Using multithreaded model, the program can be executed asynchronously if more than one activity happen at a time [12-13]. There is little advantage to being asynchronous unless you can have more than one activity going at a time. Even though threaded model can be applied to a wide range of programming problems such as computationally intensive programs, high performance application programs, real time application programs and geometric programs such as Voronoi diagram, unless there are inherently concurrent tasks, one cannot accomplish better performance [14]. This program is inherently concurrent. Hence, adding multithread, improves performance greatly. In the following section we will explain how multithreading is used in the proposed Wave-Front version of the algorithm.

3 The Wave-Front Algorithm

We first present the Wave-Front algorithm in this section and then introduce how multithread is implemented to this algorithm.

3.1 Design of the Wave-Front Algorithm

As we mentioned in the previous section, the neighbor points are not far away in general and the checking may be limited to the area close to the specific point *p*. Therefore, we propose an idea using an *h*×*h* table and take turns to scan the points in the cells (see Fig. 2).

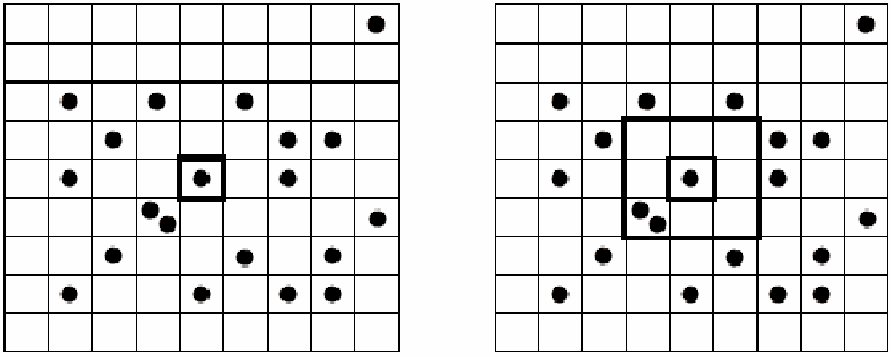


Fig. 2. An illustration of the Wave-Front Algorithm. The left of the *h*x*h* table is corresponding to *k* (= 0) iteration and the right one is corresponding to *k* (= 1) iteration. The iteration is repeated until the Voronoi polygon for point *p* is found.

To determine the neighbor points of some specific point *p*, we start to check the points in the allocated cell of *p* and then all of the cells next to the allocated cell, and then all of the cells next to the cells just scanned, the process is repeated until all of the neighbors are found. The order of scan on cells is like the propagation of wavefront. If we imagine that a stone is thrown into the allocated cell of *p*, and then the points in the cells are checked only when the wave front just arrives at the cells. This is why the improved algorithm is referred to the name of “Wave-Front”. In the implementation, the scan is in the order of the iteration k (see Fig. 2 for the first two iterations). The iteration is terminated if the Voronoi edges are closed and hence the Voronoi polygon for point p is found, otherwise the process is continued to next iteration.

We now describe the Wave-Front algorithm as follows:

Wave-Front Algorithm:
sArray[h][h] : an h×h array that stores a given set of n points according to their x and y-coordinates. Note that any cell in the table, *sArray*[r][s], may have more than one point or may be empty. *sArray*[r][s] stores all of the points in the cell and is represented by a linked list.

```
1. For all $i = 1$ to $n$ {
2.     Let $k = 0$, where $k$ represents the $k$th iteration of the wave-front.
3.     Let $r$, $s$ be the index such that $x_i$ is stored in $sArray[r][s]$.
4.     Check all the points in $sArray[r][s]$ whether each of them is a neighbor-point of
       $x_i$ using the Neighhor point Theorem.
5.     Record the Voronoi edges found in step 4.
6.     While (the Voronoi edges are not closed.) {
7.             Check all the points in $sArray[r][s]$ whether each of them is a neighbor
               point for $x_i$ using the Neighbor-point -theorem, where $r$ and $s$ is in the
               range of the $k$th iteration of the wave-front scan.
8.             Record the Voronoi edges found in step 6.
9.             }
10.    }
```

It is noted that the running time in worst case is still $O(n^3)$. However, if the data size n is large and the points of the set are uniformly distributed, then the efficiency of this algorithm seems to improve greatly. This is because the points we have to check for a point p are highly around p with a limited area. If n is large, the performance would be usually better.

3.2 Multithreading Implementation of the Wave-Front Algorithm

The above-mentioned algorithm uses “wave-front” to figure out the neighbor-points of a set of selected local points around a given point. In the kth iteration, it checks the points in the corresponding cells. The multithreaded version of the algorithm spawns T threads according to k. If T = 4, then four threads are spawned and simultaneously check the points corresponding to cells. Thus, by using T threads, similar tasks could be accomplished at the same time. Each thread can simultaneously search according to the kth iteration in which the threads are implemented. In this application, each thread can complete its work without waiting for other threads to complete execution. Thus, there is no waiting time involved in any thread. Another approach would be to spawn as many threads as possible to each cell as they scan. In the later approach, more threads would be spawned. More threads with a limited amount of task for each thread would mean more overhead due to spawning and deleting of large number of threads. In this research, we have tried both the approach to observe the running time.

The second approach does not do well as expected due to a larger number of threads and the overhead involved in performing similar tasks. The results will be presented in the next section.

4 Experimental Results

We first implemented the original approach to observe the performance and the Wave-Front algorithm in the sequential environment. Since the Wave-Front algorithm keeps a table of *h*×*h* cells, we have to consider the influence of the number of cells in the table. Hence, we try to find out which density (points per cell) would bring the best performance. After our experiments with different sizes of *n*, we found that the best performance occurred when the density is 0.1. The density could neither be too large nor too small. When the density becomes larger, there are more points in a cell. This increases the number of unnecessary checking. When the density is getting smaller and smaller, the unnecessary testing would be reduced. However, the performance would sharply decrease if the density were too small. Since the time taken to process the empty cells would be increasing, and the computational time is becoming longer.

Table 1 shows the running time for different sizes of *n*, when the density is fixed to 0.1. We can find that the ratio of the time taken by the Wave-Front and the original one is decreasing when *n* is increasing. This illustrates the efficiency of the sequential version of the Wave-Front algorithm.

Table 1. Comparisons of original and Wave-Front algorithms in sequential environment

Points	100	200	400	600	800	1000	2000	3000	4000
Original	0.15	0.77	3.62	8.92	16.85	27.52	122.2	290.42	542.44
Wave-Front	0.1	0.4	1.59	3.48	5.96	9.18	35.64	78.92	136.41
Wave-Front / Original	67%	52%	44%	39%	35%	33%	29%	27%	25%

The efficiency of Wave-Front could be observed from Figure 3 clearly. There is a sharp increase for the original algorithm when the data size *n* is greater than 1000. However, the curve does not increase rapidly using Wave-Front algorithm. This verifies that the number of testing for some specific point *p* is always *n* -1 in the original algorithm and it is almost a constant in practice for the Wave-Front.

We have also implemented the Wave-Front algorithm in the threaded model. The first implementation spawns fixed T threads for every T iterations. For instance, two threads are spawned simultaneously for T=2 (see Fig. 2, $k = 0$ is the innermost square, $k = 1$ is the next innermost one, and so on). One thread is spawned for $k = 0$, and the other one for $k = 1$. After these two iterations are done, then two threads are spawned for k=2 and k = 3. This process is continued until the Voronoi polygon of *p* is found.

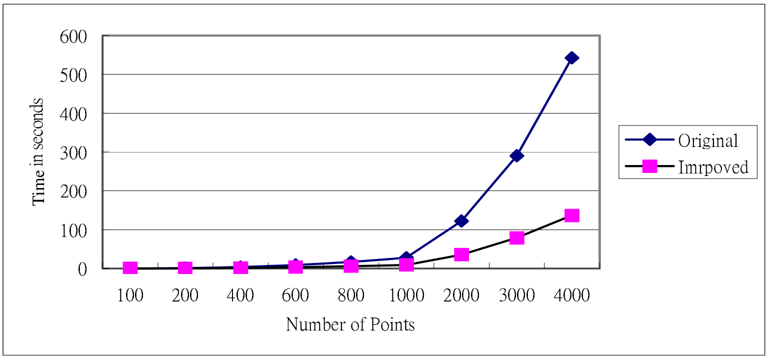


Fig. 3. Comparison of original vs. Wave-Front algorithm

Table 2 presents the results when we use different numbers of threads for a fixed size table 600*600, and Table 3 shows the results for density = 0.1 (hence the table size *h*×*h* is adjusted by data size *n*). From the results of tables 2 and 3, we can find the overall performance of table 3 is superior to those of table 2. This verifies again the results of the adjustable size table at an optimal density = 0.1 are usually better than those of a fixed size table. From table 3, we can observe that the overall performance is getting better when we increase the number of threads to T = 8. However, it seems getting worse for T = 16 and 32. Most of the neighbor points are around inner *k*. Even when T threads are spawned according to the number of *k*, the outer square for *k* may execute without finding any neighbor points. Thus, it only increases the running time. Hence we find the running time of T = 16 and T = 32 may not do better than T = 8.

Table 2. Comparisons of running time for the Wave–Front algorithm. Spawning each of T threads for each square *k* for fixed size table (*h*×*h*=600*600)

Points	100	200	400	600	800	1000	2000	3000	4000
No threads	1.47	2.53	4.5	6.62	9.19	12.26	36.92	79.98	136.36
2 threads	0.05	0.07	0.17	0.24	0.26	0.31	0.53	0.63	0.99
4 threads	0.09	0.1	0.16	0.27	0.3	0.34	0.42	0.59	0.97
8 threads	0.14	0.14	0.30	0.42	0.50	0.67	1.05	1.33	1.94
16 threads	0.37	0.65	0.85	1.0	1.19	1.43	2.18	2.74	3.99
32 threads	0.53	0.81	1.48	1.45	1.64	2.12	2.77	3.77	4.93

Table 4 presents the data where each thread is spawned for each cell. Thus T threads are spawned for T cells. It is repeated until we find voronoi polygon. It is a fine-grain approach which may involve more overhead due to many simultaneous threads being spawned in an application. It is clear that even when we spawn four threads it does not perform well as compared to Table 3. Since each cell is of a small area and not much computation can be accomplished simultaneously using several threads and so it does not perform well. It shows that the threads must have data intensity to perform a task. This table also reveals another aspect of spawning many threads, which defeats our purpose of having more threads and not accomplishing several simultaneous tasks. Hence, the last row which presents the data using T=32 does not perform better than the first row, using single thread. Besides, for a data size 4000 using single threaded approach it gives 136.41 seconds. While, spawning T=32 threads using one thread for each cell it gives 140.43 seconds. It clearly shows that spawning more threads do not necessarily improve performance, but rather slow down the performance. This program is a good example for a multithreaded version where threads can be implemented without interaction between them and it depends how the threads are implemented as opposed to say that the multithread not necessarily speedup in a single processor environment. Threads are a powerful tool to improve the performance of a program if it is implemented by the programmer at the right way taking advantage of the simultaneous tasks that can be accomplished in a data intensive program.

Table 3. Comparisons of running time for the Wave – Front algorithm. Spawning each of T threads for each square *k* with density = 0.1.

Points	100	200	400	600	800	1000	2000	3000	4000
No threads	0.1	0.4	1.59	3.48	5.96	9.18	35.64	78.92	136.41
2 threads	0.01	0.01	0.05	0.1	0.12	0.18	0.56	1.02	1.91
4 threads	0.003	0.01	0.02	0.09	0.1	0.2	0.63	1.15	1.89
8 threads	0.006	0.02	0.06	0.04	0.15	0.2	0.63	1.5	2.51
16 hreads	0.02	0.03	0.16	0.21	0.27	0.34	1.03	2.18	3.11
32 hreads	0.01	0.07	0.16	0.25	0.35	0.44	1.19	2.34	3.25

Table 4. Comparisons of running times for the Wave – Front algorithm. Spawning T threads for each cell with density = 0.1.

Points	100	200	400	600	800	1000	2000	3000	4000
No threads	0.1	0.4	1.59	3.48	5.96	9.18	35.64	78.92	136.41
2 threads	0.1	0.38	0.89	1.75	2.96	3.66	9.78	17.62	25.5
4 threads	0.09	0.43	1.05	1.72	2.69	3.56	9.95	17.5	26.33
8 threads	0.33	1.17	2.64	4.85	6.99	9.46	26.38	46.4	69.38
16 threads	0.46	1.69	4.46	7.95	11.17	15.07	41.82	75.39	107.73
32 threads	0.57	2.14	5.46	9.48	14.62	18.87	53.97	95.34	140.43

5 Conclusion

In this paper, we present the Wave-Front algorithm for computing Voronoi diagrams based on the previous work of Preilowski and Mumbeck's. Notice that this amended version of the algorithm is possible and useful because of two reasons. The first one is the special property of Neighbor-Point-Theorem, in which it is unnecessary to check all points in order to find out whether the Voronoi edges are closed for a specific point. Second, there is a property for the Voronoi diagram that the average number of edges of a Voronoi polygon is no more than six. Moreover, if the points are distributed uniformly, it would take less number of iterations before the Voronoi polygon is found.

According to the computational results, we can find out that the performance of the Wave-Front algorithm is greatly improved than that of the original one, though they both have the same time complexity of $O(n^3)$ in the worst case. The reason is that the Wave-Front algorithm only checks a few points around a specific area in most cases, unlike the original algorithm which needs to take $O(n)$ time. When the number of points increases, the time saved by the Wave-Front algorithm gets larger and larger.

Besides the sequential implementations for the original and Wave-Front algorithms, we have also implemented the multithreading model for Wave-Front algorithm. The results show that the threaded implementation further improve sequential version to a great extent. The challenge is not just to use threads to improve performance, but rather how to implement in a data intensive geometrical program. In the future, we plan to implement this algorithm in an OpenMP environment where we can really test the parallel version of this algorithm under CREW model. A lot more interesting research can be done in the area of multithreaded performance measurement especially on geometric data intensive programs.

Acknowledgement. This project was partially funded by the National Science Council of Taiwan. [NSC 92-2213-E-030-005]. The authors wish to acknowledge the support of NSC.

References

1. F. Aurenhammer, "Voronoi Diagrams- A Survey of a Fundamental Geometric Data Structure," *ACM Computing Surveys,* Vol. 23, No. 3, Sep. (1991).
2. S. G. Akl, K. A. Lyons, Parallel Computational Geometry, *Prentice-Hall*, Inc, USA, (1993).
3. L. Chow, Parallel Algorithms for Geometric Problems, Ph.D. thesis, University of Illinois at Urbana-Champaign, (1980).
4. W. Preilowski, W. Mumbeck, "A Time-optimal Parallel Algorithm for the Computing of Voronoi Diagrams," *Lecture notes in Computer Science, No. 344*, (1988), pp. 424-433.
5. A. Aggarwal, B. Chazelle, L. J. Guibas, C. O'unlaing, and C. K. Yap, "Parallel Computational Geometry," *Algorithmica, Vol. 3,* (1988), pp. 293-327.
6. David J. Evans and I. Stojmenovic, "On Parallel Computation of Voronoi Diagram," *Parallel Computing, Vol. 12*, (1989), pp. 121-125.
7. R. Cole, M. T. Goodrich, and C. O'Dunlaing, "Merging Free Trees in Parallel for Efficient Voronoi Diagram Construction, in Automata, Languages and Programming," M. S. Paterson (Editor), *Lecture Notes in Computer Science, No. 443, Springer-Verlag*, Berlin, (1990), pp. 432-445.

8. N. Amato, M. Goodrich, and E. Ramos, “Parallel Algorithms for Higher-Dimensional Convex Hulls,” In *Proc. Annu. 35th IEEE Symp. on Foundation of Computer Science (Focs 94)*, (1994), pp. 683-694.
9. G. Blelloch, J. Hardwick, G. Miller, and D. Talmor, “Design and Implementation of a practical parallel Delaunay Algorithm,” *Algorithmica, , Vol.4,* (1999), pp.243-269.
10. Intel Corporation, Hyper Threading Technology. http://developer.intel.com/technology/hyperthread/2001/
11. R. V. Behren, J. Condit and E. Brewer, “Why Events Are a Bad Idea,” *Proc. Of HotOS IX: The 9th Workshop on Hot Topics in Operating Systems*, May (2003), pp19-24.
12. M. J. Bedy, S. Carr, X. Huang and C. Shene, “The Design and Construction of a User-level Kernel for Teaching Multithreaded Programming,” 29th ASEE/IEEE frontiers in Education, Vol. II (1999), pp. (12b3-1)-(12b3-6)
13. S.E.Choi and E.C.Lewis, “A Study of Common Pitfalls in Simple Multithreaded Programs,” *In Proc. of the 31st ACM SIGCSE Technical Symposium on Computer Science Education* Mar. (2000).
14. M. Ji, E. W. Felton and K. Li, “Performance Measurements for Multithreaded Programs,” Proc. of 1998 SIGMETRICS conference, June (1998).

A Proposal of Parallel Strategy for Global Wavelet-Based Registration of Remote-Sensing Images*

Haifang Zhou[1], Yu Tang[2], Xuejun Yang[1], and Hengzhu Liu[1]

[1] School of Computer, National University of Defense Technology, Changsha, China
[2] School of Electronic Technology, National University of Defense Technology, China
fang_mini@hotmail.com, haifang_zhou@163.com

Abstract. With the increasing importance of multiple multiplatform remote sensing missions, digital image registration has been applied into many fields, and specially plays a very important role in remotely sensed data processing. Firstly a brief introduction of existing parallel methods of wavelet-based global registration is given. And then the communication optimization for GP method is described. The optimized algorithm is named *Group-Optimized-Parallel* (GOP for short). To find out the reason of occasionally lower efficiency of GOP than other methods, a more careful analysis is presented in theory and proved in experiments. Moreover, we give a quantitative criterion, called *Remainder Items*, to choose the best solution in different input conditions.

1 Introduction

Image registration is defined as the process that determines the most accurate match between two or more images acquired at the same or at different times by different or identical sensors. Digital image registration has been applied into many fields, and specially plays a very important role in remotely sensed data processing. Because of the growing of data amount and requirement for intensive computation to process these data, parallel and automated image georegistration has become a highly desirable technique.

In our earlier work, we have shown the status of research on automatic registration of remote sensing images, but also classified and analyzed the existing serial or parallel registration algorithms from the point of a novel view [1]. For global image registration method is more suitable for automatic processing than the CP-based approach and much recent research [2-7] has focused on the use of wavelets for global image registration, a more elaborate description of development of parallel wavelet-based global image registration is given in [8] and a first evaluation of these automatic parallel methods is done in theory and experiments. In [8], we classified previous parallel methods into three types as Parameter-Parallel (PP), Image-Parallel (IP) and Hybrid-Parallel (HP), and also proposed a new parallel strategy, *Group-Parallel* (GP), based

* This work is partially supported by the National 863 High Technology Plan of China under the grant No. 2002AA1Z201 and 2002AA104510, and the Grid Project sponsored by China ministry of education under the grant No. CG2003-GA00103.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 267 – 276, 2005.

on the analyses of disadvantages of old methods. But after the further study, we found that the communication mode of GP can be optimized, and GP strategy dose not always get better performance than other methods.

In this paper, we firstly give a brief introduction of existing parallel methods of wavelet-based global registration. And then we describe the communication optimization for GP method that proposed in [8] originally. The optimized algorithm is named *Group-Optimized-Parallel* (GOP for short). To find out the reason of occasionally lower efficiency of GP/GOP than other methods, a more careful analysis is presented in theory and proved in experiments. Moreover, we give a quantitative criterion, called *Remainder Items*, to choose the best solution in different input conditions.

2 Overview of Previous Methods and GP Strategy

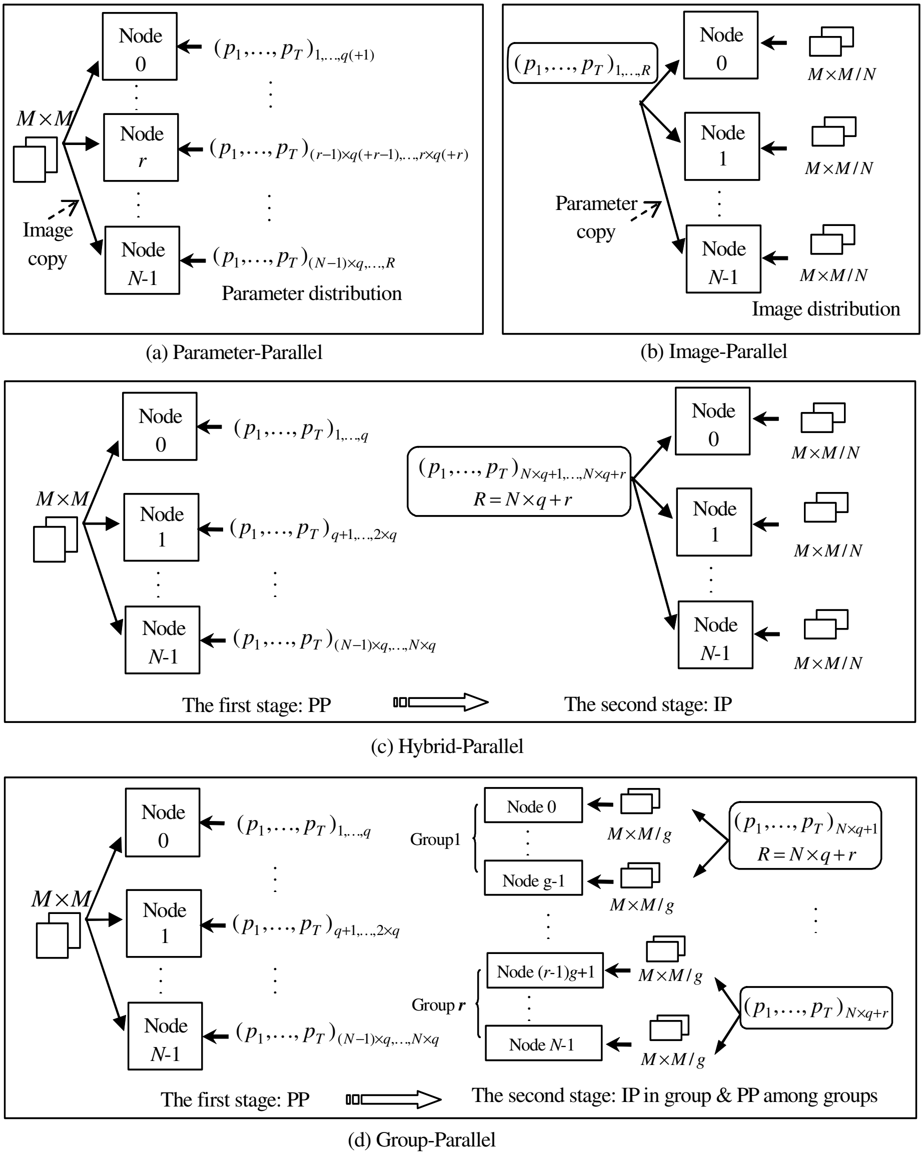
2.1 Brief Introduction of Wavelet-Based Global Image Registration

We assume that any *input image* is being registered relative to a known *reference image*. According to [9], image registration can be viewed as the combination of four components: 1) Feature space, the set of characteristics used to perform the matching and which are extracted from reference and input data; 2) Search space, the class of potential transformations that establish the correspondence between input image and reference image; 3) Search strategy, which is used to choose which transformations have to be computed and evaluated; 4) Similarity metric, which evaluates the match between input image and transformed reference image for a given transformation chosen in the search space.

As to the wavelet-based global image registration, wavelet coefficients form the feature space; and only rigid transformations are considered as search space in most application; the search strategy follows the multi-resolution approach provided by the wavelet decomposition. In our experiments, the search space is composed of 2-D rotations and translations; and cross correlation measure is used as similarity metric. So far, the process of global image registration based on wavelets can be described as following: after performing the wavelet decomposition of both reference and input images, at the each level of decomposition, the wavelet-compressed version of reference image is transformed using different combinations of rotation and translations; for each transformation cross correlation between the input image and the transformed reference image is computed; the transformation corresponding to the maximum of cross correlation is the best transformation at current level, and becoming the center of a next level search scope that is reduced and refined. The iterative search starts from the smallest wavelet image towards the larger size images, and the final registration transformation is found at the full resolution image. Please see [3] for more details on this serial process.

2.2 Existing Parallel Methods and GP Strategy

For clear and consistent presentation, some concepts and notations are given firstly.



(a) Parameter-Parallel

(b) Image-Parallel

(c) Hybrid-Parallel

(d) Group-Parallel

Fig. 1. Four parallel strategies for wavelet-based global image registration

- *M*: the image size, short for $M \times M$.
- *N*: the number of processors/nodes in a parallel computer system, where $N \geq 1$.
- *Mapping parameter*: the parameter that describes one type of transformation. E.g., the mapping parameter of rotation is $\theta \in [\theta_1, \theta_2]$.
- *Solution*: the mapping function that is used to match input and reference images, which is denoted by $(p_1, p_2, \ldots, p_k)$, where $p_i (i = 1, \ldots, k)$ is different *mapping*

parameter, and $p_i \in \Omega_i$ (Ω_i is the range of p_i). i.e., a solution is composed of one or more mapping parameters.

– *S*: search space composed of all solutions, $S = \{(p_1, p_2, \ldots, p_k) \mid p_1 \in \Omega_1, p_2 \in \Omega_2, \ldots, p_k \in \Omega_k\}$.

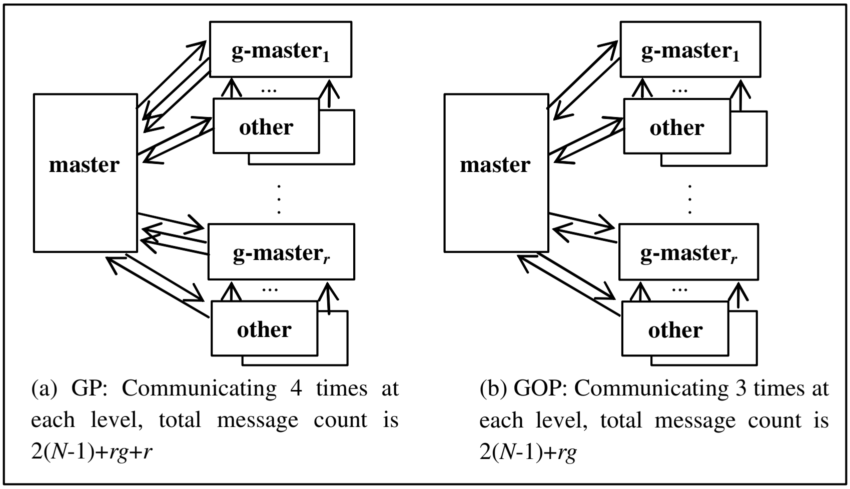
– *R*: the number of solutions in search space, i.e.,$|S| = R$.

From the description of section 2.1, wavelet-based global registration searches best solution in search space of each level of decomposition. In [8], the previous parallel algorithms [3][10-11] were formulated based on the relationship between the number of solutions (*R*) and how they are distributed over a number of processors (*N*), and classified into three types: 1) Parameter-Parallel (PP); 2) Image-Parallel (IP); 3) Hybrid-Parallel (HP). In our earlier experiments, HP excels PP and IP in the most cases, but we also find that the performance of HP sometime decreases with the increasing of *N*. This often happens when $r \neq 0$ and *M* is not very large. After in-depth analysis, the reason is found that the overhead introduced by IP for *r* remaining correlations counteracts the benefit from load balance. That is to say, the IP stage in HP mode is efficient only when the computation to communication ratio is high enough. To achieve this goal, we should make the sub-image dealt with by each processor as large as possible but amount of correlations as small as possible during the IP stage. Based on this analysis, a new parallel strategy, *Group-parallel* (GP), is proposed in [8]. Figure 1 shows the different idea of these four strategies, where $q = \lfloor R/N \rfloor$ and $r = R\%N$ (% denotes that *R* leaves *r* modulo *N*), and *g* denotes the number of processors in each group. And for more details, please refer to [8].

3 Optimization for GP

In GP (see figure 1(d)), at each level of decomposition, there are 4 communication processes happened: the first communication is happened in the PP stage. The global-master node (Node0 for example) must gather the local results of PP stage from other nodes. The second and third communication is both happened in the second stage. Each group-master (every group has a group-master node, e.g. Node0 for group 1, and Node $((r-1)g+1)$ for group *r*) should gather local IP results from other nodes within its group and send the global IP result that is computed by use of gathered local IP results to the global-master node. This is a two-level reduction operation. The last communication is a broadcast operation, in which the global-master scatters the final result globally for next level of search.

After in-depth analysis, we can find that the first and the third communication is independent from each other, so the first communication can be delayed and merged with the third one. Although the data amount to transfer is not decreased, the message number is reduced. Figure 2 shows the reduction of the number of message after the communication merging. We named the optimized GP as *Group-Optimized-Parallel* (GOP for short).



(a) GP: Communicating 4 times at each level, total message count is 2(*N*-1)+*rg*+*r*

(b) GOP: Communicating 3 times at each level, total message count is 2(*N*-1)+*rg*

Fig. 2. The number of message at each wavelet level before and after optimization

4 Performance Analysis of Four Strategies

In [8], we have analyzed the complexity of PP, IP, HP and GP respectively, where we use the LogGP as a computation model for analyzing these parallel algorithms on distributed memory machines. The complexity of algorithm is evaluated in terms of two measures: the computation time T_{comp} and the communication time T_{comm}. For communication time, let t_α denotes startup time for a message, and t_β stands for transfer time per word. In addition, let *n* denote levels of wavelet decomposition, and a solution includes *T* mapping parameters denoted by $(p_1, p_2, \ldots, p_T)$. There are *R* solutions in the search space at each level of decomposition. Original image size is *M* and the parallel system has *N* processors/nodes.

In this section, to calculate parallel speedup and efficiency, we firstly give the complexity formula of serial algorithm of automatic wavelet-based registration as equation 1:

$$T_s = \sum_{i=0}^{n} \frac{1}{4^i} \mathrm{O}(R \cdot M^2) + \mathrm{O}(n \cdot R) = \mathrm{O}(R \cdot M^2 + n \cdot R) \tag{1}$$

In [8], complexity analysis has ignored the effect of communication times. If considering this effect, we can get the following complexity formulas of PP (equation 2), IP (equation 3), HP (equation 4), GP (equation 5, 6) and GOP (equation 7, 8).

$$T_{N-PP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 2n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N + \frac{(N-r) \cdot M^2}{N}) \tag{2}$$

$$T_{N-IP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 2n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot t_\beta \cdot R \cdot N + n \cdot R \cdot N + \frac{(N - M^2 \% N) \cdot R}{N}) \tag{3}$$

If $R\%N = 0$, $T_{N-HP} = T_{N-GP} = T_{N-GOP} = T_{N-PP}$. If $R\%N \neq 0$, there are:

$$T_{N-HP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 3n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot t_\beta \cdot r \cdot N + n \cdot r \cdot N + \frac{(N - M^2\%N) \cdot r}{N}) \tag{4}$$

$$T_{N-GP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 4n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N) \quad (N\%r = 0) \tag{5}$$

$$T_{N-GP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 4n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N + \frac{r^2 \cdot M^2}{N \cdot (N-r)}) \quad (N\%r \neq 0) \tag{6}$$

$$T_{N-GOP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 3n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N) \quad (N\%r = 0) \tag{7}$$

$$T_{N-GOP} = \mathrm{O}(\frac{R \cdot M^2}{N} + 3n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N + \frac{r^2 \cdot M^2}{N \cdot (N-r)}) \quad (N\%r \neq 0) \tag{8}$$

Comparing equation 7and 8 with equation 5 and 6, it is proved in theory that GOP excels GP for its less communication cost. Because GOP has same computation process as GP, we only discuss GOP in the remainder of this paper.

Based on equation 1, 2 and 7, we can deduce the speedup and efficiency formulas of the parallel algorithms PP and GOP, seeing equation 9-12. These formulas show that the speedup of PP and GOP will both rise with the scale of system and data set. But if value of *M* and *N* became very large, the scalability of PP would be restricted for load imbalance. Contrarily, GOP has perfect theoretical analysis conclusion, i.e., keeping *N* unchanged, speedup S_{N-GP} approximates to *N* and efficiency E_{N-GP} approximates to 1 with the increasing of problem scale ($M \to \infty$). This conclusion shows that parallel performance and scalability of GOP are better than PP. Moreover, based on iso-efficiency model [12], we can deduce iso-efficiency function of GOP (equation 13) from equation 11 and 12, where $f_{E-GOP}(N)$ is a linear function of *N*.

By comparing equation 3, 4 and 7, we find that the items including M^2 are same as each other, so we will not enumerate the deduction process of speedup, efficiency formula and iso-efficiency function of IP and HP here. They will get the similar results as equation 11, 12 and 13.

$$\begin{aligned} S_{N-PP} &= \frac{\mathrm{O}(R \cdot M^2 + n \cdot R)}{\mathrm{O}(\frac{R \cdot M^2}{N} + n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N + \frac{(N - R\%N) \cdot M^2}{N})} \\ &= \mathrm{O}(\frac{(R + n \cdot R/M^2) \cdot N}{R + \frac{n \cdot t_\alpha \cdot N}{M^2} + \frac{n \cdot t_\beta \cdot T \cdot N^2}{M^2} + \frac{n \cdot N^2}{M^2} + N - R\%N}) \\ &\approx \mathrm{O}(\frac{R \cdot N}{R + N - R\%N}) \quad M \to \infty \end{aligned} \tag{9}$$

$$E_{N-PP} = \frac{S_{N-PP}}{N} = \mathrm{O}(\frac{R}{(R - R\%N)/N + 1}) \qquad M \to \infty \tag{10}$$

$$\begin{aligned} S_{N-GOP} &= \frac{\mathrm{O}(R \cdot M^2 + n \cdot R)}{\mathrm{O}(\frac{R \cdot M^2}{N} + n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N)} \\ &= \mathrm{O}(\frac{(R + n \cdot R/M^2) \cdot N}{R + \frac{n \cdot t_\alpha \cdot N}{M^2} + \frac{n \cdot t_\beta \cdot T \cdot N^2}{M^2} + \frac{n \cdot N^2}{M^2}}) \\ &\approx \mathrm{O}(N) \quad M \to \infty \end{aligned} \tag{11}$$

$$E_{N-GOP} = \frac{S_{N-GOP}}{N} = \mathrm{O}(N) \qquad M \to \infty \tag{12}$$

$$f_{E-GOP}(N) = \frac{E_N}{1 - E_N} \cdot (n \cdot t_\alpha + n \cdot t_\beta \cdot T \cdot N + n \cdot N) \tag{13}$$

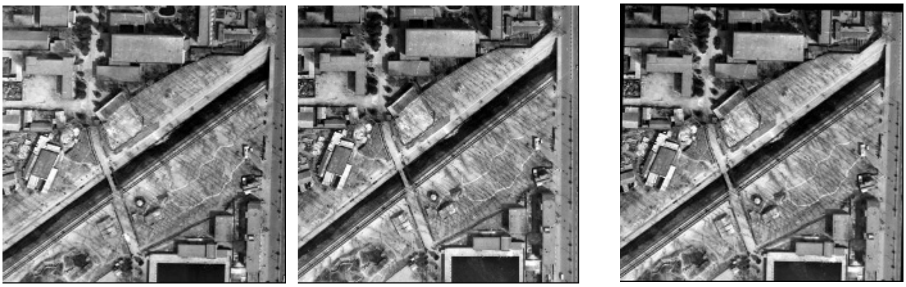
Though the changing trend of speedup and efficiency of the above four strategies are similar, their parallel execution time are different. To find a quantitative criterion for choosing a best solution in different application, we need to analyze execution time in detail. Because the first three items of equation 2, 3, 4 and 7 are same in the order of magnitudes, we only compare the *Remainder Items*, which are called RIs. These remainder items (RIs) of the four strategies are denoted as $t_{pp}, t_{ip}, t_{hp}, t_{gop}$. The rules of selecting best strategy in different situations based on RIs are presented as follows:

- When $R < N$. We can know from the foregoing analysis that DOP of PP is *R*, and salability of PP is limited. So we should select one of the other three parallel strategies (their parallel execution time is equal to T_{IP} in this situation).
- When $R > N$ and $R\%N = 0$. The RIs of PP, HP and GOP are equal, i.e., $t_{pp} = t_{hp} = t_{gop} = n \cdot N$. But the RI of IP is $t_{ip} = n \cdot t_\beta \cdot R \cdot N + n \cdot R \cdot N$. Apparently, complexity of IP is higher than the others. Hence, we should select one strategy from PP, HP or GP in this situation.
- When $R > N, R\%N \neq 0$ and $N\%r = 0$ ($r = R\%N$). Then we can deduce that $t_{pp} = n \cdot N + (N - r) \cdot M^2/N$, $t_{ip} = n \cdot t_\beta \cdot R \cdot N + n \cdot R \cdot N + \frac{(N - M^2\%N) \cdot R}{N}$, $t_{hp} = n \cdot t_\beta \cdot r \cdot N + n \cdot r \cdot N + \frac{(N - M^2\%N) \cdot r}{N}$, and $t_{gop} = n \cdot N$. Because $t_{ip} > t_{pp}(t_{hp}) > t_{gop}$, GOP is the optimum strategy in this situation.

- When $R > N, R\%N \neq 0$ and $N\%r \neq 0$ ($r = R\%N$). The computing formulas of t_{pp}, t_{ip}, t_{hp} are as same as ones in the above condition of $N\%r = 0$, but the formula of t_{gop} is changed. We deduce its formula based on equation 8, which is $t_{gop} = n \cdot N + \frac{r^2 \cdot M^2}{N \cdot (N - r)}$. Apparently, complexity of t_{ip} is highest, but comparison of the other three strategies depends on value of some parameters. For example, if $n = 3$, $r = 1$, $N = 8$ and $M = 256$, then $t_{gop} - t_{pp} = \frac{256^2}{8 \times 7} - \frac{7 \times 256^2}{8} < 0$, so GOP is better than PP; if $n = 3$, $r = 7$, $N = 8$ and $M = 256$ then $t_{gop} - t_{pp} = \frac{7^2 \times 256^2}{8} - \frac{256^2}{8} > 0$, so PP is better than GOP. That is to say, in this situation, we should select the best strategy by computing RIs according to some parameters. That is why GOP strategy dose not always get better performance than other methods.

5 Experiments and Conclusion

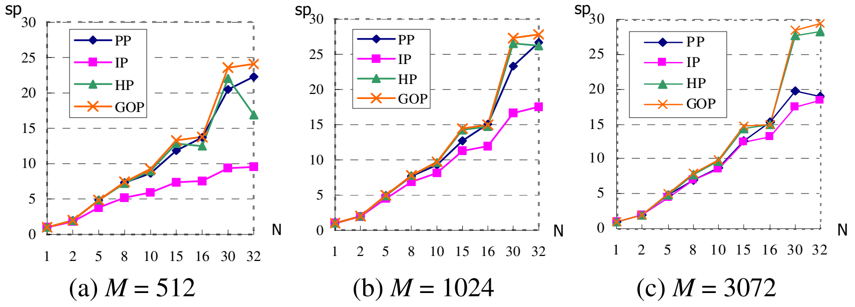
For comparison, we also implement GOP algorithm on a same computer YH used in [8]. YH has 32 processors with 1GB local storage for each processor. Speed of YH CPU is valued as 1.66 gigaflops/sec. Topology of network is fat tree, and point-to-point bandwidth is 1.2Gb/s. Various remotely sensed images with different size (*M*=256/512/1024/2048/3072) are used for testing.



(a) Reference image (b)Input image (c)Output image

Fig. 3. Registration result of a pair of test images

Figure 3 shows the registration result of a pair of test images, in which Figure 3(c) is the output of the transformed input image to match the reference image. Figure 4 gives the comparison of speedups achieved by four parallel algorithms with different datasets (image size *M* changes from 512 to 3072) on our parallel platforms YH.



(a) $M = 512$ (b) $M = 1024$ (c) $M = 3072$

Fig. 4. Speedups of four algorithms achieved on YH with different datasets

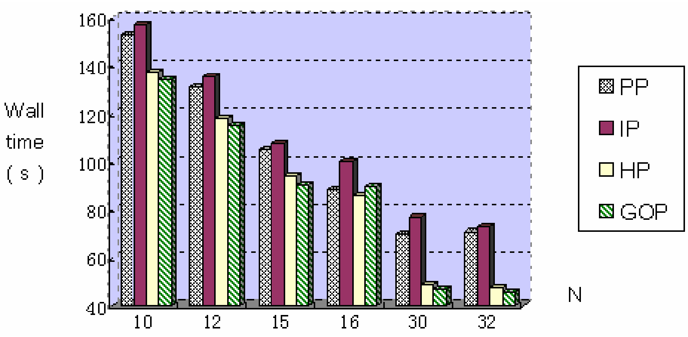


Fig. 5. Execution time of four algorithms achieved on YH with one test dataset (M=3072)

From figure 4, we can conclude that GOP is better than PP in execution time and speedup. Furthermore, the optimized effect of GOP is better than HP in most situations, especially when $N \% r = 0$. The performance of HP is fluctuant sometimes for its RI (t_{ip}) is affected by several factors. These conclusions are in accordance with foregoing theoretical analysis. When value of M is large, the effect of unbalanced load becomes more obvious with the increasing of N. In figure 4, when N>*16*, the performance of PP is worse than that of HP and GOP; and when N is up to 32, the speedup even begins to fall. Therefore, performance can be improved by balancing load, and the improving scope will increase with the increasing of M.

The validity of RIs analyzed above is proved in figure 5. For example, when $N = 12$, then $r = 5$, the RI of PP is $t_{pp} = (7 \times M^2)/12$, the RI of GOP is $t_{gop} = (25 \times M^2)/(12 \times 7)$, so $t_{pp} > t_{gop}$, accordingly, execution time of PP is longer than GOP in figure 5. But when $N = 16$, then $r = 13$, $t_{pp} = (3 \times M^2)/16$, $t_{gop} = 13^2 \times M^2/16 \times 3$, so $t_{pp} < t_{gop}$, accordingly, execution time of GOP is longer than PP in figure 5. Hence, we can exactly select an optimum parallel strategy by calculating RI.

Future work will include the study of combination of global registration and CPs-based methods with emphasis on both speed and accuracy. Automatic registration of remotely sensed data is a very complex problem, and as stated in [3], we feel that only a future system that integrates multiple automated registration techniques will be able to address such a task for multiple types of remote sensing data.

References

1. Zhou, H., Liu, G., Zheng, M., Yang, X.: A research on serial and parallel strategies of the automatic image registration for remote sensing. Journal of national university of defense technology. Vol. 26(2). (2004)56-61. (In Chinese, cited by EI)
2. Moigne, J. Le., Xia, W., El-Ghazawi, T.: Towards an intercomparison of automated registration algorithms for multiple source remote sensing data. In the Proceeding of the first image registration workshop, NASA/GSFC. 1997.11.
3. Moigne, J. Le., Campbell, W. J., Cromp, R. F.: An automated image registration technique based on the correlation of wavelet features. IEEE Transaction on Geoscience and Remote Sensing. 40(8) (2002)1849-1864.
4. Thévenaz, P., Ruttimann, U. E., Unser, M.: A pyramid approach to sub-pixel registration based on intensity. IEEE Transaction on Image Processing. 7(1) (1998) 27–41.
5. Pinzon, J., Ustin, S., Castaneda, C., Pierce, J.: Image registration by non-linear wavelet compression and singular value decomposition. In Proceedings of IRW, NASA/GSFC. (1997)1-6.
6. Chettri, S., Campbell, W. J., Moigne, J. Le. : A scale space feature based registration technique for fusion of satellite imagery. In proceedings of ImageRegistration Workshop (IRW97), NASA/GSFC, (1997) 29-34.
7. Fonseca, L., Manjunath, B. S., Kenney, C.: Scope and applications of translation invariant wavelets to image registration. In proceedings of Image Registration Workshop (IRW97), NASA/GSFC. (1997) 13-28
8. Zhou, H., Yang, X., Liu, H., Tang, Y.: First evaluation of parallel methods of automatic global image registration based on wavelets. In proceedings of the 2005 international conference on parallel processing. IEEE Computer Society. Norway, Oslo. 2005. 6
9. Brown, L.: A survey of image registration techniques. ACM Computing Surveys. 24(4) (1992) 325–375.
10. El-Ghazawi, T., Chalermwat, P.: Wavelet-based image registration on parallel computers. In SuperComputing'97: High Performance Networking and Computing: Proceedings ACM/IEEE. 1997.11.
11. Chalermwat, P.: High performance automatic image registration for remote sensing. [Ph.D. Thesis]. George Mason University. Fairfax, Virginia. 1999.
12. Grama, A. Y., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. IEEE Parallel & Distributed Technology. 1(3) (1993) 12-21.

Performance Analysis of a Parallel Sort Merge Join on Cluster Architectures

Erich Schikuta

Research Lab on Computational Technologies and Applications,
Institute of Knowledge and Business Engineering,
University of Vienna,
Rathausstraße 19/9, A-1010 Vienna, Austria
erich.schikuta@univie.ac.at

Abstract. We developed a concise but comprehensive analytical model for the well-known sort merge Join algorithm on cost effective cluster architectures.

We try to concentrate on a limited number of characteristic parameters to keep the analytical model clear and focused. We believe that a meaningful model can be built upon only three characteristic parameter sets, describing main memory size, the I/O bandwidth and the disk bandwidth. We justify our approach by a practical implementation and a comparison of the theoretical to real performance values.

1 Introduction

Today clusters of workstations are the focus of many high performance applications searching for viable and affordable platforms replacing expensive supercomputer architectures. A cluster system is a parallel or distributed processing system consisting of interconnected stand-alone workstations working together as a single, integrated computing resource [1]. We believe that a cluster system is a suitable environment for parallel database systems. There is an urgent need for novel database architectures due to new stimulating application domains with huge data sets to administer, search and analyze.

Due to its inherent expressive power the most important operation in a relational database system is the join. It allows to combine information of different relations according to a user specified condition, which makes it the most demanding operation of the relational algebra. Thus the join is obviously the central point of research for performance engineering in database systems.

In the past a number of paper appeared covering this topic, like [2], [3], [4], which proposed and analyzed parallel database algorithms for parallel database machines. [5] presents an adaptive, load-balancing parallel join algorithm implemented on a cluster of workstations. The algorithm efficiently adapts to use additional join processors when necessary, while maintaining a balanced load. [6] develops a parallel hash-based join algorithm using shared-memory multiprocessor computers as nodes of a networked cluster. [7] uses a hash-based join

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 277–286, 2005.

algorithm to compare the designed cluster-system with commercial parallel systems.

In this paper we will present an analysis and evaluation of the Sort Merge Join. This work is part of a running project for a comprehensive analysis of parallel join operations. We did a similar research for all important parallel join operations (e.g. Hybrid Hash Join [8]). A focus on analyzing hardware characteristics of the underlying system is beyond the scope of this paper. So we are interested in the specifics of the algorithms and not of the machines.

2 Parallel Database Operations

2.1 Declustering

Declustering is the general method in a parallel database system to increase the bandwidth of the I/O operations by reading and writing the multiple disks in parallel. This denotes the distribution of the tuples (or records, i.e. the basic data unit) of a relation among two or more disk drives, according to one or more attributes and/or a distribution criteria. Three basic declustering schemes can be distinguished, range declustering (contiguous parts of data are stored on the same disk), round-robin declustering (tuples are spread consecutively on the disk), and hash-declustering (location of a tuple is defined by a hash function).

The disjoint property of the declustered sets can be exploited by parallel algorithms based on the SPMD (single program, multiple data) paradigm. This means that multiple processors execute the same program, but each on a different set of tuples. A realistic assumption of our model is that the relations of the database system are too large to fit into the main memory of the processing units. Consequently all operations have to be done externally and the I/O costs are the dominant factor for the system performance.

2.2 The Join Operation

The join operation ‘merges’ two relations R and S via two attributes (or attribute sets) A or B (respective relations R and S) responding to a certain join condition. The join attributes have to have the same domain. Two different types of join operators are distinguished, the equi-join and the theta-join. In the following only the equi-join is discussed. Three different approaches for join algorithms are distinguished, sort merge, nested loop, and hash based join. Basically the parallel versions of these approaches can be realized on a conventional client-server scheme. The server stores both relations to join and distributes the declustered tuples among the available clients. The clients perform the specific join algorithm on their sub relations and send the sub results back to the server. The server collects the result tuples and stores the result relation.

3 Analytical Model

3.1 Model Parameters

In the following (see Table 1) we specify several parameters and a few derived terms, which describe the characteristics of the model environment and build the basis for the derived cost functions.

Table 1. Parameters of the cost model

m	number of tuples of relation R (inner relation)
n	number of tuples of relation S (outer relation)
p	number of processors
n_t_m	number of tuples per message
b	bucket size (tuples per bucket)
s	selectivity factor (percentage of the product of m and n giving the result size
l_f	loop_factor (number of loops necessary to build hash buckets due to number of open file limitations)
read	read one tuple from disk
write	write one tuple to disk
receive	receive one message
send	send one message
find_target	find the right target client
hash	store a tuple into a main memory hash table
probe	probe a main memory hash table with a tuple
fill	fill a tuple into main memory
compare	compare the keys of two tuples in main memory and build a result tuple if keys match.

The specific values of the basic parameters and the derived functions used in the theoretical model to calculate "real" numbers were profiled by a specific test program on the real hardware. The values are given in section 4.

For the declustering of the data among the clients the server reads the two input relations, described by equation (1),

$$server\_read = (m + n) * read \tag{1}$$

determines the respective target client by a declustering function with p as one of its parameters (2)

$$server\_compute = (m + n) * find\_target \tag{2}$$

and sends the tuples (packed in messages) to the target client (3).

$$server\_send = (\frac{m}{n\_t\_m} + \frac{n}{n\_t\_m}) * send \tag{3}$$

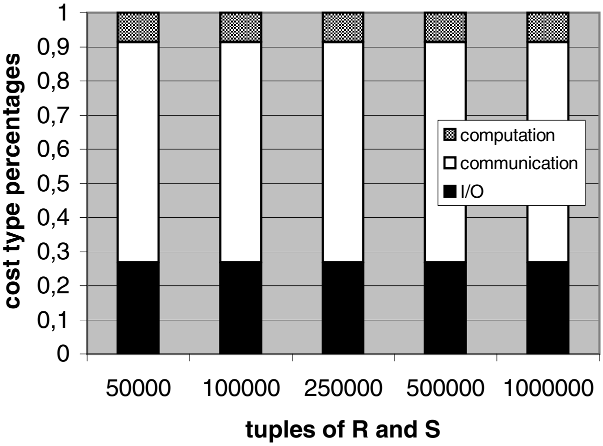


Fig. 1. Percentage of server-side cost types

After sending the messages the server is in an idle-state. It waits for the results of the clients (4) and writes it to disk (5).

$$server\_receive = \frac{m * n * s}{n\_t\_m} * receive \tag{4}$$

$$server\_write = m * n * s * write \tag{5}$$

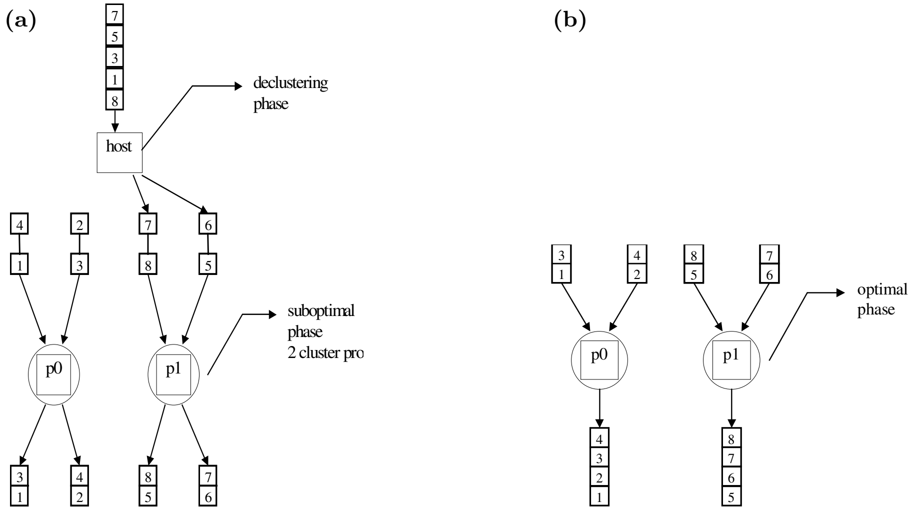
The total cost of the server is defined in (7) by the sum of (1) to (5).

$$\begin{aligned} server\_cost = {} & server\_read + server\_compute + \\ & + server\_send + server\_receive + server\_write \end{aligned} \tag{6}$$

The work of the server, as I/O-costs (read,write), message-costs (send, receive) and computational costs, is obviously independent of the number of processors used. Figure 1 shows this situation graphically by splitting the total server execution costs into the parts on I/O costs (read and write operations), communications costs (send and receive operations) and pure computational costs.

3.2 Sort Merge Join

The parallel version of the sort merge join algorithm is a straight forward adaption of the traditional single processor version of the algorithm. The inner relation R is first partitioned using a *split table* (range declustering). A function is applied to the join attribute of each tuple to determine the appropriate disk site. As the tuples arrive at a client they are gathered in buckets. The buckets are sorted in ascending order of the join attribute and written to two temporary files of equal size. Every client uses binary sort merge to sort its part of



(Supotimal phase) **(Optimal phase)**

Fig. 2. Phases of sort merge join

relation R. Parallel binary merge sort is described in [9]. An analysis and evaluation of parallel binary merge sort is given in [10]. In our algorithm we use only the *suboptimal* and *optimal* part of the sort algorithm. The suboptimal phase (see (a) Figure 2) merges pairs of longer and longer runs (i.e. ordered sequences of pages). In every step the length of the runs is twice as large as in the preceding run. At the beginning each processor reads two sorted pages, merges them into a run of 2 pages and writes it back to the disk. This is repeated, until all buckets are read and merged into 2-buckets-runs. If all buckets are merged, the *suboptimal phase* continues with merging two 2-page-runs to a sorted 4-page-run. This continues until all 2-page-runs are merged. The phase ends, when the two temporary files are sorted. At the end of the *suboptimal phase* 2 sorted temporary files exist on each node. During the following *optimal phase* each processor merges the 2 temporary files (see (b) in Figure 2).

In a second phase the outer relation S is partitioned using the same *split table*. Every client receives its tuples of relation S, builds and sorts buckets and sorts the temporary files of relation S using binary sort merge. In a third phase the sorted temporary files of R and S are merged and result tuples are built.

The work for the clients start with receiving the tuples of the inner and outer relation from the server. Every client gets only $\frac{m}{p}$ tuples of the inner relation R and $\frac{n}{p}$ tuples of the outer relation S. The costs for receiving are described by [7] and for writing to the local disk by [8].

$$client\_receive = \frac{\frac{m}{p}}{t\_m} * receive + \frac{\frac{n}{p}}{t\_m} * receive \tag{7}$$

$$build\_temp\_files = \frac{m}{p} * write + \frac{n}{p} * write \tag{8}$$

In the following step every bucket has to be sorted, which is

$$sort\_bucket = (\frac{\frac{m}{p}}{b} + \frac{\frac{n}{p}}{b}) * b^2 * compare \tag{9}$$

After sorting the buckets are written to the local disk. analogous to [8].

In the suboptimal phase and the optimal phase of binary merge we have to sort the two input relation.

$$sort\_R = (\frac{m}{p} * log\frac{m}{p}) * (read + compare + write) \tag{10}$$

$$sort\_S = (\frac{n}{p} * log\frac{n}{p}) * (read + compare + write) \tag{11}$$

Next the sorted input relations have to be merged and result tuples have to be built,

$$merge = (\frac{m}{p} + \frac{n}{p}) * (read + compare) \tag{12}$$

At last the algorithm sends back the join results. Finally the client has to write the result tuples back to the server [13].

$$send\_result = (\frac{m}{p} * \frac{n}{p}) * \frac{s}{t\_m} * send \tag{13}$$

The complete costs of the client are given in [15]. The total cost for the sort merge join [15] is the sum of the cost of the server and [15].

$$\begin{aligned} client\_sort\_cost = {} & client\_receive + build\_temp + \\ & + sort\_bucket + sort\_R + sort\_S + merge + send\_result \end{aligned} \tag{14}$$

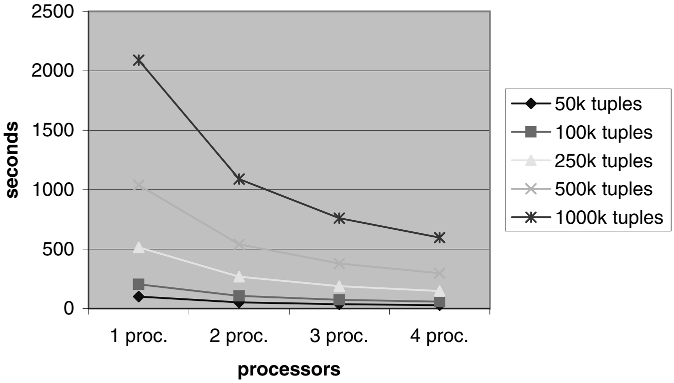


Fig. 3. Theoretical speedup sort merge join

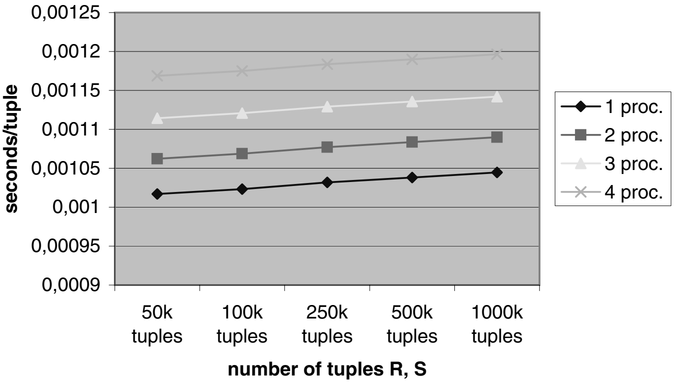


Fig. 4. Theoretical cost per tuple sort merge join

$$sort\_merge\_cost = server\_cost + client\_sort\_cost \tag{15}$$

Figure 3 shows the theoretical speedup behavior of the sort merge join using different number of processors and input tuples.

The costs per tuple change only slightly when the number of input tuple increases. The costs per tuple increase because the main parts of sort merge are from the order of $O((m + n) * log(m + n))$. The results can be seen in Figure 4 for 1 to 4 clients.

4 Model Justification

To justify the presented model we evaluated and compared it to a practical performance analysis were we implemented the algorithm.

Test-bed for our analysis was an off-the-shelf "el-cheapo" PC cluster consisting of 5 single processor nodes (computational units) running the Linux operating system. The algorithms were realized with the C language and PVM as communication library. The values of the parameters used in our tests are given in Table 2. We used a test module to determine the values of the basic parameters and the derived functions (measured in seconds) of our cost model

Table 2. Specific values of the basic parameters

m	50k,100k,250k,500k,1000k
n	50k,100k,250k,500k,1000k
p	1,2,3,4
n_t_m	100
b	1000
s	1/m
l_f	10

Table 3. Values for derived functions of the cost model

read	0,0000105 seconds
write	0,00001 seconds
receive	0,0025 seconds
send	0,0025 seconds
find_target	0,000005 seconds
Hash	0,00001 seconds
probe	0,00001 seconds
fill	0,0000008 seconds
compare	0,0000008 seconds

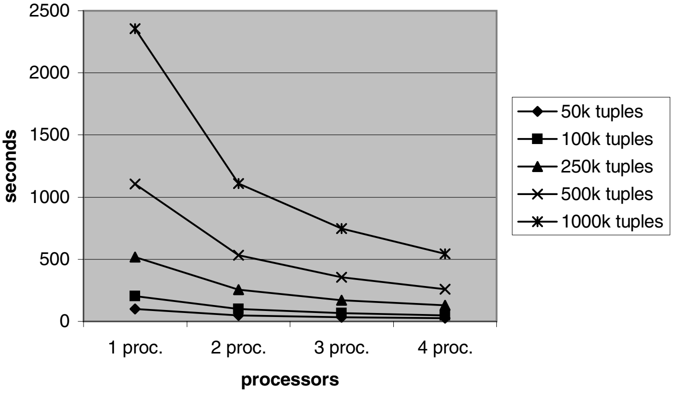


Fig. 5. Real speedup Sort Merge Join

(see Table 3). Figure 5 gives the real execution times for the Sort Merge Join. All given values are the averages of at least 20 runs.

The real values correspond to the theoretical values, besides the specific result for the one-processor case, amazingly well. The asymptotic runtime behavior for increasing workloads and processing nodes (speed-up) of the model and the reality is about same. Not only the trend of the data is the same, but also the real execution values match the ones calculated by the model. The difference was only about 10 percent, which is due to the simplified model ignoring operating system specifics. Summing up this result shows that the simplified approach described in the previous section models the reality very well and justifies it as a basis to analyze the algorithms behavior on a cluster architecture thoroughly.

5 Conclusion and Lessons Learned

The results of the analytic model justified by the practical implementation leads to the following lesson, which can be the basis for future data intensive applications on cluster architectures:

- *Disk IO*, and not network bandwidth, *is the limiting factor* for distributed data intensive IO operations.
- It can be expected that the cumulated *network bandwidth* of a typical cluster *is larger than the IO bandwidth.* This situation is based on the current technology and also supported by the actual technology trend that the characteristics of network hardware develops faster than the disk hardware.
- Corollary: *Clusters are a viable platform for data intensive applications.*
- To build up an *analytical model* of cluster systems for data intensive operations it is sufficient to concentrate on the characteristics of *main memory, IO bandwidth and disk bandwidth.*
- It is possible to *model the execution time* behavior of data intensive operations on clusters *accurately*, which allows the building of query analyzer for parallel/distributed database systems.
- Summing up: *Clusters can be a suitable platform for parallel/distributed database systems.*

Acknowledgements

I would like to express my thanks to Peter Kirkovits, who helped in designing the algorithms and the programming of the test suite. The work described in this paper was partly supported by the Special Research Program SFB F011 AURORA of the Austrian Science Fund.

References

1. Baker, M., Buyya, R.: Cluster Computing at a Glance. In: High Performance Cluster Computing. Prentice Hall (1999) 3–47
2. Pirahesh, H., Mohan, C., Cheng, J., Liu, T., Selinger, P.: Parallelism in relational database systems: Architectural issues and design approaches. In: Proc. Of the IEEE Conf. On Distributed and Parallel Database Systems, IEEE Computer Society Press (1990)
3. Stonebraker, M., Aoki, P., Devine, R., Litwin, W., Olson, M.: Mariposa: A new architecture for distributed data. In: Proc. Of the Int. Conf. On Data Engineering, IEEE Computer Society Press (1994)
4. Moreno, E.: Hash join algorithms on smps clusters: Effects of netcaches on its scalability and performance. Journal of Information Science and Engineering **18** (2002)
5. Amin, M.B., Schneider, D.A., Singh, V.: An adaptive, load balancing parallel join algorithm. In: Sixth International Conference on Management of Data (COMAD'94), Bangalore, India (1994)
6. Jiang, Y., Makinouchi, A.: A parallel hash-based join algorithm for a networked cluster of multiprocessor nodes. In: Proceedings of the COMPSAC '97 - 21st International Computer Software and Applications Conference. (1997)
7. Tamura, T., Oguchi, M., Kitsuregawa, M.: Parallel database processing on a 100 node PC cluster. In: Proc. of the Supercomputing 97, IEEE Computer Society Press (1997)

8. Schikuta, E., Kirkovits, P.: Cluster based hybrid hash join: Analysis and evaluation. In: Proc. IEEE International Conference on Cluster Computing, Chicago, IEEE Computer Society Press (2002)
9. Bitton, D., Boral, H., Dewitt, D., Wilkinson, W.: Parallel algorithms for the execution of relational operations. ACM Trans. Database Systems **8** (1983) 324–353
10. Schikuta, E., Kirkovits, P.: Analysis and evaluation of sorting for parallel database systems. In: Proc. Euromicro 96, Workshop on Parallel and Distributed Processing, Braga, Portugal, IEEE Computer Society Press (1996) 258–265

Parallel Clustering on the Star Graph

M. Fazeli, H. Sarbazi-Azad, R. Farivar

Sharif University of Technology and IPM School of Computer Science
{m_fazeli, azad, r_farivar}@ce.sharif.edu,
azad@ipm.ir

Abstract. In this paper, a parallel algorithm for data clustering is presented on a multi-computer with star topology. This algorithm is fast and requires a small amount of memory per processing element, which makes it even suitable for SIMD implementation. The proposed parallel algorithm completes in $O(K+S^2 - T^2)$ steps for a clustering problem of N data patterns with M features per pattern and K clusters, where $N.M = S!$, $K.M = T!$, and $M=R!$, on a s-star interconnection network.

1 Introduction

Feature vector is a basic notion of pattern recognition. A feature vector v is a set of measurements ($v_1, v_2, \ldots, v_M$) which map the important properties of a collection of data into a Euclidean space of dimension M [6]. Clustering algorithm partitions a set of feature vectors into cluster groups. It is a valuable tool in exploratory pattern analysis, and helps making hypotheses about the structure of data. It is important in syntactic pattern recognition, image segmentations, registration, and many other applications. There have been many methods proposed for clustering feature vectors [6], [7], [8], [9], [10], [11].

One popular clustering technique is the *squared-error* algorithm. This clustering algorithm is as follows [5]. Let N represent the number of patterns which are to be partitioned and let M represent the number of features per pattern. Let F [0…*N-1,* 0..*M-1*] be the feature matrix such that the F [*i,j*] denotes the value of *j*-th feature in the *i*-th pattern.

Let S_1, S_2, … , and S_k be K clusters. Each pattern belongs to exactly one of the clusters. Let $C[i]$ represent the cluster to which pattern i belongs. Thus, we can define S_k as

$$s_k = \{i \mid C[i] = k, 0 \le k \le K-1\}$$

Let $|s_k|$ be the cardinality or size of the pattern s_k. The center of cluster k is a 1×M vector defined as

$$center[k,j] = \frac{1}{|s_k|} \sum_{i \in s_k} F[i,j], \quad 0 \le j \le M;$$

The squared distance $d2$ between pattern i and cluster k is given by

$$d2[i,k] = \sum (F[i,j] - center[k,j])^2$$

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 287 – 292, 2005.

The squared error for the k-th cluster is defined as

$$E2[k] = \sum_{i \in s_k} d2[i,k], \quad 0 \le k < K;$$

And the squared error for clustering is given by

$$ERROR\ [K] = \sum_{k=0}^{K-1} E2[k].$$

In the clustering problem, we are required to partition the N input patterns such that the squared error for the clustering becomes minimum. In practice, this is done by trying out several different values of K. For each K, the clusters are constructed using an iterative refinement technique in which we begin with an initial set of K clusters, and move each pattern to a cluster with which it has the minimum squared distance and re-compute cluster centers. The last two steps are iterated until no pattern is moved further from its current cluster. The final clustering obtained in this way, however, is not guaranteed to be globally minimum. In fact, different initial clustering can result in different final clusters.

This paper[1] proposes a parallel algorithm for pattern clustering on the star graph with a run time of $O(K)$ and a memory usage of $O(1)$. The algorithm combines several communication techniques in a novel method to perform pattern clustering on a *NM*-node star graph. This algorithm relies on window broadcasting communication at some stages during computation, as will be discussed later. It also uses a special kind of processor ordering introduced in [1] in order to assign the data to the PE's in the initialization phase.

2 Routing and Data Communication in the Star Graph

In this section, some useful definitions and notation are introduced. A routing algorithm, called *Send*, will be also introduced. This algorithm is used in the last phase of our algorithm.

Definition 1. Let $S_{n-1}(i)$ be the sub graph of S_n in which the last symbol of all its node addresses are equal to i.

A $S_{n-1}(i)$ is an (n-1)-star defined on symbols {1,2,…,n}-{i}. Thus, S_n can be decomposed into n sub-(n-1)-stars, $S_{n-1}(i)$, $1 \le i \le n$. For example, a $S_3(4)$ would contain four 3-stars, namely $S_3(1)$, $S_3(2)$, $S_3(3)$, and $S_3(4)$.

Definition 2. Let m_1 and m_2 be two distinct symbols from {1,2,…,n}. We use notation m_1*m_2 to represent a permutation of {1,2,…,n} whose first and last symbols are m_1 and m_2, respectively, with * representing any permutation of the n-2 symbols in {1,2,…,n}-{m_1, m_2}. Similarly, m_1* is a permutation of n symbols whose first symbol is m_1, and *m_2 is a permutation of n symbols whose last symbol is m_2 [3].

Definition 3. Two or more nodes from distinct S_{k-1}'s are *corresponding* if they have the same index in their respective S_{k-1}'s according to the processor ordering scheme which is introduced in Section 2.2. for example the nodes with addresses 2341, 1342, 1243, 1234 are the corresponding nodes in a sample S_4.

[1] The detailed explanation of the proposed parallel algorithm is available in [2].

In our parallel algorithm, a useful function called *Send* is used to transmit the contents of the nodes of a $S_{k-1}(i)$ to the corresponding nodes of another $S_{k-1}(j)$. Since the host network is S_n, the last N-K symbols of the upper level S_k (in which the $S_{k-1}(i)$ and $S_{k-1}(j)$ are embedded) are the same. This function gets four values as inputs: i and j as the k-th symbols of two S_{k-1}'s , k as the dimension of the upper level sub graph in which $S_{k-1}(i)$ transmits its nodes contents to the nodes in $S_{k-1}(j)$, and n is the dimension of the host network S_n. Notation $\delta_{k,n}$ in the *send* function represents that the last n-k symbols are the same. The pseudo code of the *send* function is present in [2].

***Rule* 1.** Every node value in a particular $S_{k-1}(i)$ is sent to its *corresponding* node in $S_{k-1}(j)$ using *Send* function, if i and j are in a descending order in the symbol set (i.e. j is less than i and greater than the other remaining symbols).

Let $S=X_1\, X_2\, \ldots\, X_{i-1}\, X_i\, \delta_{k,n}$, $X_i \in \{1,2,3, \ldots ,n\}$ be the source node in the particular S_{k-1}, and the *Send* function be used to transmit the node contents of $S_{k-1}(X_i)$ to $S_{k-1}(X_{i-1})$. The routing steps are as follows:

Step1: $X_1\, X_2 \ldots X_{i-1}\, X_i\, \delta_{k,n} \longrightarrow X_i\, X_2 \ldots X_{i-1}\, X_1\, \delta_{k,n}$
Step2: $X_i\, X_2 \ldots X_{i-1}\, X_1\, \delta_{k,n} \longrightarrow X_{i-1}\, X_2 \ldots X_i\, X_1\, \delta_{k,n}$
Step3: $X_{i-1}\, X_2 \ldots X_i\, X_1\, \delta_{k,n} \longrightarrow X_1\, X_2 \ldots X_i\, X_{i-1}\, \delta_{k,n}$

In this rule, for the sake of clarity, we suppose that $X_i > X_{i-1} > X_{i-2} \ldots > X_1$. According to our processor ordering scheme the node $X_1\, X_2\, \ldots\, X_{i-1}\, X_i\, \delta_{k,n}$ has the greatest index in the $S_{k-1}(X_i)$ and the node $X_1\, X_2 \ldots X_i\, X_{i-1}\, \delta_{k,n}$ has also the greatest index in the $S_{k-1}(X_{i-1})$, therefore the node $X_1\, X_2 \ldots X_i\, X_{i-1}\, \delta_{k,n}$ which is selected as a destination node is the *Corresponding* node of the source node.

***Rule* 2.** Two consecutive neighboring nodes S_1 and S_2 in $S_k(i)$ send data to consecutive neighboring nodes D_1 and D_2 in $S_k(j)$, if i and j are in the descending order in the symbol set.

Suppose that node $X_1\, X_2\, \ldots\, \alpha\, \ldots\, X_{i-1}\, Z_i\, \delta_{k,n}$ and node $Y_1\, Y_2\, \ldots\, \alpha\, \ldots\, Y_{i-1}\, Z_i\, \delta_{k,n}$ in $S_{k-1}(Z_i)$ are two consecutive neighbors, and α and Z_i are in descending order in the symbol set such that $\{T\} < \{\alpha , Z_i\} < \{S\}$ where $\{T\} \cup \{S\} = Symbol\ Set - \{\alpha , Z_i\}$. After sending the contents of nodes in $S_{k-1}(Z_i)$ to nodes in $S_{k-1}(Y_i)$, we have the following steps for sending data from node $X_1\, X_2 \ldots \alpha \ldots X_{i-1}\, Z_i\, \delta_{k,n}$,

Step1: $X_1\, X_2 \ldots \alpha \ldots X_{i-1}\, Z_i\, \delta_{k,n} \longrightarrow Z_i\, X_2 \ldots \alpha \ldots X_{i-1}\, X_1\, \delta_{k,n}$
Step2: $Z_i\, X_2 \ldots \alpha \ldots X_{i-1}\, X_1\, \delta_{k,n} \longrightarrow \alpha\, X_2 \ldots Z_i \ldots X_{i-1}\, X_1\, \delta_{k,n}$
Step3: $\alpha\, X_2 \ldots Z_i \ldots X_{i-1}\, X_1\, \delta_{k,n} \longrightarrow X_1\, X_2 \ldots Z_i \ldots X_{i-1}\, \alpha\, \delta_{k,n}$

and the following steps for sending data from node $Y_1\, Y_2 \ldots \alpha \ldots Y_{i-1}\, Z_i\, \delta_{k,n}$,

Step1: $Y_1\, Y_2 \ldots \alpha \ldots Y_{i-1}\, Z_i\, \delta_{k,n} \longrightarrow Z_i\, Y_2 \ldots \alpha \ldots Y_{i-1}\, Y_1\, \delta_{k,n}$
Step2: $Z_i\, Y_2 \ldots \alpha \ldots Y_{i-1}\, Y_1\, \delta_{k,n} \longrightarrow \alpha\, Y_2 \ldots Z_i \ldots Y_{i-1}\, Y_1\, \delta_{k,n}$
Step3: $\alpha\, Y_2 \ldots Z_i \ldots Y_{i-1}\, Y_1\, \delta_{k,n} \longrightarrow Y_1\, Y_2 \ldots Z_i \ldots Y_{i-1}\, \alpha\, \delta_{k,n}$

The nodes $X_1\, X_2\, \ldots\, Z_i\, \ldots\, X_{i-1}\, \alpha\, \delta_{k,n}$ and $Y_1\, Y_2\, \ldots\, Z_i\, \ldots\, Y_{i-1}\, \alpha\, \delta_{k,n}$ in $S_{k-1}(\alpha)$ are also consecutive neighboring nodes because exchanging α and Z_i symbols dose not affect the ordering of nodes.

***Rule* 3.** In transmission from nodes in $S_{k-1}(i)$ to nodes in $S_{k-1}(j)$, where i is the minimum symbol in the corresponding symbols set and j is the greatest one, k-2 exchange steps are required within the $S_{k-1}(j)$ to send data in the *corresponding* nodes to each other.

If the contents of the node $X_2 \ldots X_i\, X_1\, \delta_{k,n}$ is transmitted by function *send* to the $S_{k\text{-}1}(X_i)$, where $X_i > X_{i-1} > X_{i-2} \ldots > X_1$ in $S_{k-1}(X_1)$, the following steps are performed in the first phase of the *send* function:

$$\begin{array}{lll}
\text{Step1: } X_2\, X_3 \ldots X_i\, X_1\, \delta_{k,n} & \longrightarrow & X_1\, X_3 \ldots X_i\, X_2\, \delta_{k,n} \\
\text{Step2: } X_1\, X_3 \ldots X_i\, X_2\, \delta_{k,n} & \longrightarrow & X_i\, X_3 \ldots X_1\, X_2\, \delta_{k,n} \\
\text{Step3: } X_i\, X_3 \ldots X_1\, X_2\, \delta_{k,n} & \longrightarrow & X_2\, X_3 \ldots X_1\, X_i\, \delta_{k,n}
\end{array}$$

It's clear that $X_2\, X_3 \ldots X_i\, X_1\, \delta_{k,n}$ has the greatest index among other nodes in $S_{k\text{-}1}(X_1)$. Thus, in a correct transmission, the contents of this node should be transmitted to a node in $S_{k\text{-}1}(X_i)$ which has the greatest index (according to the processor ordering), but the use of the proposed algorithm doesn't accomplish this task in the first 3 steps. To do so, *K*-2 exchange steps are required in $S_{k\text{-}1}(X_i)$ as follows:

$$\left.\begin{array}{lll}
\text{Step1: } X_2\, X_3 \ldots X_1\, X_i\, \delta_{k,n} & \longrightarrow & X_3\, X_2 \ldots X_1\, X_i\, \delta_{k,n} \\
\text{Step2: } X_3\, X_2 \ldots X_1\, X_i\, \delta_{k,n} & \longrightarrow & X_4\, X_2\, X_3 \ldots X_1\, X_i\, \delta_{k,n} \\
\quad\vdots & & \quad\vdots \\
\text{Step } k\text{-2: } X_{i-1} X_2\, X_3 \ldots X_1\, X_i\, \delta_{k,n} & \longrightarrow & X_1 X_2\, X_3 \ldots X_{i-1}\, X_i\, \delta_{k,n}
\end{array}\right\} \begin{array}{l} k\text{-2 steps} \\ \text{are required} \end{array}$$

From Rules 1, 2 and 3, it can be concluded that each node in $S_{k\text{-}1}(i)$ is sent to its corresponding node in $S_{k\text{-}1}(j)$ by *send* function, if *i* and *j* are in the descending order except when *i* is the minimum and *j* is the greatest symbol in corresponding symbol set. The following transmission sequence shows a correct order of transmission.

$$S_{k-1}(X_i) \longrightarrow S_{k-1}(X_{i-1}) \longrightarrow \ldots\ldots \longrightarrow S_{k-1}(X_1) \longrightarrow S_{k-1}(X_i)$$

where $X_i > X_{i-1} > X_{i-2} \ldots > X_1$.

3 The Parallel Algorithm

The parallel algorithm consists of three main phases: *Initialization Phase, Cluster Assignment Phase,* and *Centers update Phase.* The number of patterns in this algorithm, *N*, the number of clusters, *K,* and the number of features, *M,* should satisfy conditions $N.M{=}S!$, $K.M{=}T!$, and $M{=}R!$. If the number of patterns, clusters or features are not in a factorial manner, one can add enough number of dummy entries so that the above conditions satisfy and the clustering results are not affected [5].

3.1 The Initialization Phase

During this phase, two index numbers are associated to each PE according to the mentioned processor ordering scheme. The first index shows the order of the PE in the host network S_s and the latter one is the order of the PE in the corresponding S_T. Then the patterns are associated with different S_R's in such a way that the *i*-th feature of a pattern resides in a PE whose index number satisfies the condition $index \overset{m}{\equiv} i$. Then the first S_T in S_S is considered as the master cluster window (the choice of the initial cluster centers in the master cluster window is arbitrary), and its contents are copied into all other S_T's, so that the current cluster center selection is reported to all the other cluster windows. The register R_1 of each node is used to store the distance of node to its cluster. The register R_3 represents the value of the node distance to the current cluster. The registers F_1 and C_1 are used to store the values of features and their cluster numbers.

3.2 The Cluster Assignment Phase

The aim of this phase is to compute the distance $d_2\ (i,k)$ of each pattern in each cluster window from the current selection of cluster centers, and to choose the minimum distance to all cluster centers. We then assign this cluster to the pattern according to the selected choice.

First, the distances between the features and the current centers available in each node of a S_R are computed in a parallel fashion among all S_R's in the network as $(\textit{Pattern feature - Center feature})^2$.

Then by the use of a function called *Group Accumulate* [1], the value of $d_2\ (i,k)$ which represents the distance between the i-th pattern and the current center is calculated and compared to the old value; the smaller one is selected as the cluster where this pattern belongs to.

In the second step, the values of the S_R's present in all S_{R+1}'s available in the S_S are rotated once via the *send* function in parallel, as previously described. These steps will be repeated R+1 times, until all the S_R's present in all S_{R+1}'s get each other's data.

The next step would be to rotate the data values of S_{R+1}'s in all S_{R+2}'s in parallel once, and repeat the first and second steps $R+2$ times. The addition of the levels of sub-graphs and their rotation of them continues until the S_{T+1} is reached; in other words, we reach one level higher than the cluster windows (S_T's).

By the end of the *Cluster Assignment Phase*, all the patterns have been assigned their cluster membership in the corresponding higher order node (according to the PE's ordering). These steps are described in [2].

3.3 Centers Update Phase

As mentioned before, all the cluster windows (S_T's) have been indexed such that every node contains a variable T which is equal to $\left\lfloor \frac{Index}{R!} \right\rfloor$, which shows that what cluster center data each S_R in a cluster window is responsible for. This value is a pre-computed constant for each S_R (which would contain a feature vector). This phase has two steps: *Broadcast Cluster Center step* and *Cluster Center Update step.*

3.3.1 Broadcast Cluster Center Step

In this phase, all the S_R's in S_S broadcast their cluster numbers which have been computed in the previous phase and stored in their highest indexed node.

As mentioned earlier, the center number of patterns is stored in a node with highest index within the S_R's. Let $X_1\ X_2\ \ldots\ X_L\ \delta_{R,T}$ be the node with highest index in corresponding S_R. The following optimal broadcasting algorithm [4] can broadcast the content of this node to the processors in $S_R\ (X_L)$.

3.3.2. Cluster Center Update Step

In this phase, all the S_R's in S_{R+1} exchange their values $R+1$ times. In each rotation step, the contents of nodes in S_R's , which include cluster number and the feature value is transmitted to its corresponding node in the consecutive S_R. In each node in S_R's, if the cluster number T is equal to the cluster number it receives (from the previous window S_R), the PE adds its feature value to the feature value it gets, otherwise it does nothing.

In the next step, the dimension of the last step is increased once, in fact S_{R+1}'s exchange data $R+2$ times inside the corresponding S_{R+2}. In each exchange operation, the above steps are repeated again until S_{T+1} is reached.

Through the last step, all S_T's in a S_{T+1} exchange their nodes' contents with their corresponding S_T's, T+1 times. Since corresponding S_R's in two different S_T's (S_R's with similar values of T) contain similar cluster center information, the nodes of each S_T just add their former contents to the newly received ones; there is no need for any comparison or similar operation.

The last step is repeated until we reach the S_S. There will be [S(S+1)/2-T(T+1)/2+K] additions and *send* operations. The pseudo code of this phase is also present in [2].

4 Conclusions

The star graph was proposed as an attractive alternative to the hypercube topology for interconnection between processors in parallel computers. It has been extensively studied in different aspects and many algorithms have been designed for it. In this paper, a squared error clustering algorithm for a star multi-computer was presented. This algorithm is fast and requires a little amount of memory per processing node. This algorithm completes in $O(K+ S^2 - T^2)$ steps for a clustering problem of N patterns, with M features per pattern, and K clusters, where $N.M = S!$, $K.M = T!$, and $M=R!$, on an $N.M$–node multiprocessor with star topology.

References

[1] H. Sarbazi-Azad, M. Ould-Khaoua, L.M. Mackenzie, and S.G. Akl “A parallel algorithm for Lagrange interpolation the star graph”, *Journal of Parallel and Distributed Computing* 62, 605-621 (2002).

[2] M. Fazeli, H. Sarbazi-Azad and R.Farivar “Parallel Clustering on the Star Graph”, *Technical Report, IPM school of Computer Science* (2005).

[3] S. Akl, K. Qiu, ”A novel routing scheme on the star and pancake networks and its applications”, *Parallel Computing* 19(1), 95-101 (1993).

[4] P. Berthone, A. Ferreira, and S. Perennes, “*Optimal Information Dissemination in Star and Pancake networks", IEEE TPDS* 7, 1292-1300 (1996).

[5] S. Ranka and S. Sahni, “Clustering on a hypercube multicomputer*”, IEEE TPDS* 2(2), 71-82 (1991).

[6] D.H. Ballard and C.M. Brown, *Computer vision*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[7] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.

[8] K.S. Fu, *Syntactic Methods in Pattern Recognition.* New York: Academic Press, 1974.

[9] K.Fukunaga, *Introduction to Statistical Pattern Recognition.* New York: Academic Press, 1972.

[10] A. Rosenfeld and A.C. Kak, *Digital Picture Processing,* New York: Academic, 1982.

[11] J.T. Tou and R.C. Gonzalez, *Pattern Recognition principles.* Reading MA: Addison-Wesley, 1974.

Hierarchical Parallel Simulated Annealing and Its Applications*

Shiming Xu[1], Wenguang Chen[1], Weimin Zheng[1],
Tao Wang[2], and Yimin Zhang[2]

[1] Dept. of Computer Science and Technology, Tsinghua Univ., Beijing, China
[2] Intel China Research Center, Beijing, China

Abstract. In this paper we propose a new parallelization scheme for Simulated Annealing — Hierarchical Parallel SA (HPSA). This new scheme features coarse-granularity in parallelization, directed at message-passing systems such as clusters. It combines heuristics such as adaptive clustering with SA to achieve more efficiency in local search. Through experiments with various optimization problems and comparison with some available schemes, we show that HPSA is a powerful general-purposed optimization method. It can also serve as a framework for meta-heuristics to gain broader application.

Keywords: Simulated Annealing, Parallelization, Metaheuristics, Hierarchical Clustering.

1 Introduction

Simulated Annealing(SA), firstly proposed in [7], is a randomized optimization algorithm widely applied to various combinatorial and continuous problems. Compared with other randomized algorithms, such as GA, Tabu Search, various evolutionary algorithms, it possesses a formal proof of convergence to global minima[6] under some restrictions on cooling scheduling and temperature parameters[10]. Despite this strictness, SA in practice retains the ability to avoid local minimum and to locate near-optimal solutions.

SA is computation-intensive algorithm and features sequential intrinsics; there has been much work on its parallelization [4,3,8,11,2]. With different parallel granularity, these parallel schemes are targeted at various kinds of parallel machines. Schemes of coarse granularity usually have to deal with scalability problems. We've designed a new parallel SA scheme in which processes are organized in a three-level hierarchy, addressing scalability problems effectively while achieving better coverage over the search space. Experiments show that it outperforms conventional parallel SA in either convergence speed or solution quality. This article is organized as follows: in Section 2 we will have a short overview of sequential and parallel SA and summarize related works. Detailed design and implementation of HPSA is presented afterwards in Section 3. In Section 4 we

* This project is supported by NSFC 60273007.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 293–300, 2005.

show HPSA outperforms available parallel SA in either speed or solution quality through various experiments. Finally We conclude that HPSA could serve as a general-purposed optimization scheme and point out our future work.

2 Sequential SA and Its Parallelization

2.1 Sequential SA

Simulated Annealing[7,1,10] is an optimization algorithm in analogy to the annealing process in metallurgy. For a formal description of SA, we give definition over these terms:

$$\begin{array}{rl} \mathcal{S}: & \text{Search Space;} \\ \mathcal{C}ost: & \mathcal{S} \rightarrow \mathbb{R}, \text{ Cost Function Defined over } \mathcal{S}; \\ \mathcal{N}: & \mathcal{S} \rightarrow 2^{\mathcal{S}}, \text{ Neighborhood Function;} \\ T: & \text{Temperature, } T \in \mathbb{R}^{+}. \end{array}$$

SA is used to locate a solution s_m in $\mathcal{S}$ that minimize function $\mathcal{C}ost$, given the neighborhood relation $\mathcal{N}$. Usually, $\mathcal{N}$ is symmetric over $\mathcal{S}$: $\forall s \in \mathcal{S}, t \in \mathcal{S}$, $t \in \mathcal{N}(s) \rightarrow s \in \mathcal{N}(t)$. The basic idea of SA is to find an initial point in $\mathcal{S}$ and an initial temperature T_0, then conduct a random local search process within $\mathcal{S}$ under the control of T. The process carries on until T approaches zero close enough. A basic flow chart of SA is shown in Fig.1.

```
PROCEDURE Sequential_SA
BEGIN
  $s \leftarrow$ Initial Solution in $\mathcal{S}$
  $T \leftarrow$ Initial Temperature $T_0$
  DO
    DO
      $s^* \leftarrow \mathcal{N}eighbor(s)$
      $\Delta\mathcal{C} \leftarrow \mathcal{C}ost(s^*)$ - $\mathcal{C}ost(s)$
      IF $\Delta\mathcal{C} < 0$ OR Accept($\Delta\mathcal{C}$,T) THEN
          $s \leftarrow s^*$
      END IF
    UNTIL Equilibrium
    $T \leftarrow$ Decrement($T$)
  UNTIL Frozen
END PROCEDURE
```

Fig. 1. Sequential SA

The outer loop of SA generally deals temperature. It starts from T_0 and terminates when T is low enough, which also terminates the algorithm. The inner loop(Metropolis Loop) which is conducted under a certain temperature, mainly deals with local search. A solution s^* in $\mathcal{N}(s)$, is generated and judged by $\mathcal{C}ost(s^*)$. If s^* is better, i.e., of lower cost, s is replaced by s^*. If it is worse, it is accepted statistically according to the Metropolis criteria[7].

2.2 Parallelization of SA

According to the classification of parallelization of Metaheuristics in [5], parallel schemes for SA fall into three categories:

- Fine granularity parallelization for inner loop
  - Functional parallelization on move evaluation
  - Data parallelization of multiple-move evaluation
- Parallelization based on search space partitioning
- Multiple concurrent runs exploring the solution space

Since Type 1 schemes [3,8,4,2,3] feature fine granularity, they fit SMP or SIMD machines. The high communication frequency between processes hampers the effectiveness of such schemes on loosely-coupled systems, such as clusters or even distributed systems. Type 2 schemes require an effective segmentation over the search space so that final output can be summarized directly basing on partial results from concurrent processes[5]. These schemes are problem-dependent, so there's much constraints applying them to general problems. In Type 3 schemes processes are organized in non-intersected subsets, which we call clusters, to conduct search process, while communication between processes follows some patterns. For further description of Type 3 schemes we define:

$\mathcal{P}$: $\{\, p_i \mid 1\leq$ i$\leq$ N$\}$, set of processes;
$\mathcal{C}$: $\{\, c_i \mid c_i \in 2^{\mathcal{P}},\ c_i \neq \emptyset,\ \bigcup_i c_i = \mathcal{P},\ c_i \bigcap c_j = \emptyset$ for $i \neq j,\ 1 \leq i,j \leq m\}$,
set of clusters formed from $\mathcal{P}$.

These parallel schemes posses coarse granularity. Each process p_i in $\mathcal{P}$ initiates with a randomly chosen solution in $\mathcal{S}$ and carries on with its own chain until SA terminates. During the search process, local information is dynamically interchanged among process clusters c_j (here we assume $p_i \in c_j$) after all the processes within c_j has undergone certain steps of tempering, so that processes within c_j gain a better knowledge of the search space. Usually a solution s' is chosen or created for all the processes within c_j to carry on instead of their original solutions s_i. Process clusters could dynamically adapt during the search process.

MMC-PSA in [9] is a representative of Type 3 schemes. In MMC-PSA $\mathcal{C} \equiv \{\mathcal{P}\}$, i.e., only one constant process set exists. The replacement strategy is to replace solutions of all the processes with the best one s_best. While this replacement scheme's intuitively beneficial, currently best solution s_best may well be a local minimum. If current solutions of all processes in $\mathcal{P}$ are to be overridden, there's a possibility that processes which may potentially achieve global minimum s_m are deviated and lose adjacency to s_m. Given an extra large search space with many local minima, it is more probable for MMC-PSA to get trapped into a local minimum with fair cost, which is not our objective. Also the communication pattern of MMC-PSA does not fit large-scale systems. Especially, in asynchronous MMC-PSA, maintaining a globally accessible best solution is extremely costly in a distributed environment. HPSA is designed with these problems in notion. It's similar to MMC-PSA in that it is also based on multiple chains. Through dividing computation power over potential areas in the search space and confining most communication within process clusters, HPSA solves scalability problems faced by MMC-PSA and other similar schemes.

3 Hierarchical Parallel SA

HPSA is targeted at message passing systems, typically cluster environments. Generally HPSA can be classified as a coarse-grained, i.e., Type 3 scheme. It is similar to MMC-PSA in that it is also based on Multiple Markov Chain. Main design considerations are listed below:

- Processes include $\mathcal{P}$, a set worker processes, and a *farmer* process;
- $\mathcal{P}$ is dynamically divided into clusters: c_i's;
- *Farmer* is responsible for dynamically organizing clusters, i.e. changing $\mathcal{C}$, to achieve optimal distribution of processes, keeping:
- Processes within the same cluster have adjacent solutions, hence keeping high reachability within each cluster and minimizing the possibility of killing potential ones;
- Communication is either intra-cluster or between cluster and *farmer*.

3.1 Main Structure

Farmer process is mainly responsible for setting up and maintaining clusters. When the algorithm begins, no cluster exists. Dissociated processes, which do not belong to any cluster report to *farmer* directly. When all processes have reported to *farmer*, clustering decision is made and processes are informed of the cluster they belong. Each process is uniquely associated with a cluster, which is confined with an MPI communicator. In each cluster a head process is created to report to *farmer* at intervals about information of local search. *Farmer* decides to reshuffle clusters when a certain number of clusters have reported to have undergone great changes from their original positions. On the decision of reshuffling, *farmer* responds cluster heads with a message flag which indicates dismissal, which is broadcasted within the cluster. Processes which have received messages with dismissal flag on will become dissociated and report to *farmer* afterwards, just like when the algorithm begins. After all the clusters have been noticed of dismissal, *farmer* enters the phase same to the time when the algorithm initiates. A cluster reports to *farmer* that it's quitting when all of its processes have reported to have ended the annealing processes. When all the clusters have reported quitting, *farmer* quits, terminating the algorithm.

Communications in HPSA fall into two categories: intra-cluster communication and communication between clusters and *farmer*. Intra-cluster communication is carried out at the interval of n tempering iterations. Within a cluster the communication is synchronous, i.e., a process synchronizes with others to find out their best solution. Afterwards, processes will continue local search from this best solution. Under synchronous communication, there's no need to keep record of the globally best solution; also we are free from the overhead of exclusively accessing it.

Inter-cluster communication is fully asynchronous. Non-associated processes report to *farmer* as soon as they've reached local equilibrium under current temperature, sending out their current solutions; afterwards they wait for the cluster assignment from *farmer*. Heads of clusters report to *farmer* when local synchronization times has reached a threshold. After sending out local information to *farmer*, head processes wait until *farmer* replies. *Farmer* would either reply indicating the cluster to either carry on annealing or dismiss. On receiving dismissal messages, head process would dismiss all its fellow processes within the cluster and they will all enter non-associated state.

All the processes in HPSA are organized into a multiple-level hierarchy. When clusters are formed, it contains three levels: the highest level contains *farmer* process only; the secondary level contains all cluster head processes; the lowest layer contains ordinary working processes. When clusters are disassociated, it is a two-level structure. Under either mode, communication is controlled within directly-adjacent nodes in the hierarchy.

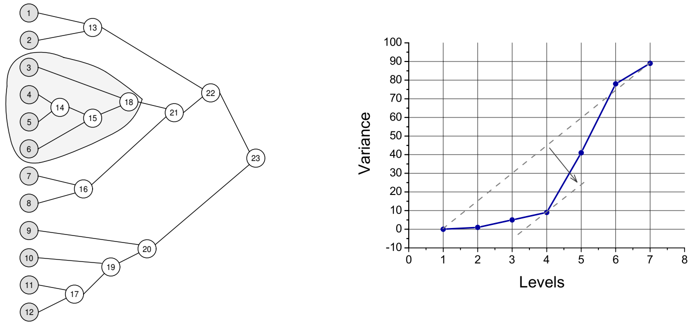
3.2 Clustering Decision in HPSA

On *farmer* we adopt Agglomerative Hierarchical Clustering to organize processes into clusters. So the process of building $\mathcal{C}$ can be divided into two steps:

- Building Hierarchical Clustering Tree
- Forming $\mathcal{C}$

Fig.2a is an example of dendrogram of hierarchical clustering. As is shown, configurations to be clustered are labelled from 1 to 12. A full tree is formed with internal nodes labelled from 13 to 23, according to their generation time during the clustering process. While conventionally in hierarchical clustering a stop criterion is used to terminate the clustering process, such as cluster count reaches a threshold, in HPSA we decide to build the whole cluster tree since available process count is usually small and not likely to exceed several hundred and the overhead of building the whole tree is trivial. Cluster identification is specific to problems. For problems such as Protein structure prediction in [12], a quantitive threshold may be provided basing on experiences. For other problems of implicit distance measurement, threshold may be provided heuristically, for example, $1/10\ of\ Radius$.

As for those problems for which no distance threshold is available, other heuristics can be applied to identify clusters. For example, clustering decisions can be made basing on variance changes between different clustering options. For example, clustering decisions can be made basing on the variation series from leaf node up to the root in the clustering tree. See Fig.2 for an example. The variance trajectory of $Node-4$ in Fig.2a is shown in Fig.2b. Usually the variation series is non-descending. So we can detect one-step changes in variation series and put the clustering barrier between the pair of nodes with greater slope. In the previous example, all the nodes under $Node-18$ will form a cluster.



a. Dendrogram Example

b. Trajectory of Node-4

Fig. 2. Hierarchical Clustering Example

4 Experiments

4.1 Implementation and Configurations

HPSA is implemented in MPI to support message-passing environments such as clusters. We have tested HPSA over various TSP problems.

For symmetric TSP problems, the definition of neighborhood structure and distance between solutions varies according to implementations. In HPSA we use conventional neighborhood definition for TSP[3]. With this local topology, it requires much computation to attain the distance between solutions. In HPSA we introduce a method to approximate distance between different TSP solutions: the ratio of uncovered cities by common sub-chains among all the cities of two solutions. For comparison we have implemented MMC-PSA [9,11] and MIR-PSA(Multiple Independent Run).

Experiments are carried out on an 8-node cluster, each node featuring 4-way SMP of Pentium-III Xeon 700MHz CPU and 1GB Ram. The software environment is Linux 2.4.20 and mpich-1.2.5. All nodes are connected by 100Mb/s switch. On the cluster totally 64 MPI processes are engaged in the parallel SA, including the *farmer* process.

4.2 Test Results

We have randomly picked several TSP benchmarks from TSP-LIB: *eil*101, *tsp*225, *ch*150, *kroA*100 and *kroC*100, with best solutions known. Two aspects of HPSA are evaluated: the First Hit Time(FHT) of a certain cost level and Quality of final result. Table.1 shows the test result of FHT, and cost-levels were selected as 102%,105% and 110%. Since the annealing processes are different only in the initial temperature, so the percentages of FHT in the whole annealing process is listed. The average FHT of 10 independent runs are retrieved from each test suit for HPSA, MMC-PSA and MIR-PSA. We have also tested effects of fixed scheduling on HPSA. Given a fixed initial temperature,

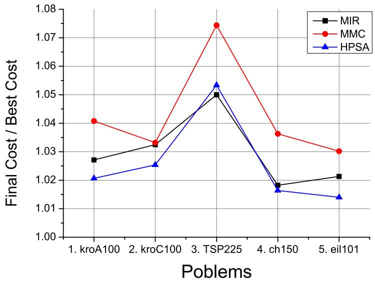


Fig. 3. Test Result II

especially one of a low value, the quenching process would take shorter time. The quality of final result generated by different parallel SA for given problems are listed in Fig.3.

Table 1. FHT Results

FHT	102%			105%			110%		
Problem	HPSA	MMC	MIR	HPSA	MMC	MIR	HPSA	MMC	MIR
ch150	83.0%	83.5%	87.0%	66.2%	64.2%	66.3%	44.0%	44.9%	34.6%
eil101	69.6%	73.2%	68.2%	57.5%	60.0%	51.9%	44.3%	48.3%	45.9%
kroA100	74.8%	77.2%	75.0%	61.2%	62.2%	64.8%	40.1%	37.0%	41%
kroC100	70.3%	70.1%	72.0%	57.7%	57.3%	61.4%	42.2%	33.1%	37.2%
tsp225	87.7%	93.3	91.0%	79.5%	59.5%	77.0%	50.0%	41.7%	47.8%

From Table.1 we can see that HPSA gains a margin over MMC-PSA and MIR-PSA if we use a lower cost level. But when cost level rises, there are more chances that any of the three may overtake the other two. Also given a fixed schedule, the quality of final result averaged by 10 runs, as is in Fig.3, shows that HPSA outperforms MMC-PSA and MIR-PSA. The fact that MIR-PSA outperforms MMC-PSA is congruent with the tuition that given a low starting temperature, MMC is more likely to kill potential processes.

The running time saved by HPSA is trivial according to our experiment results. Most of the time MMC-PSA and HPSA consume similar amount of time. Through localizing communication by assigning clusters of processes to adjacent processing units, HPSA may gain further timing-advantages over MMC-PSA.

5 Conclusion and Future Work

HPSA is a parallel SA scheme that is located between MMC-PSA and MIR-PSA. By dynamically clustering processes and manage them in a two-level hierarchy, it easily handles the scalability problem most conventional parallel SA schemes face. Through experiments we show that for TSP problems, HPSA gains advantages over MMC-PSA and MIR-PSA on the large. With further growth of distributed systems, HPSA is a more promising algorithm among parallel SA schemes.

For our future work, clustering criterion of HPSA is to be refined so that it can handle problems with speculative distance threshold is provided, which may not be accurate enough and has to be refined. Mixed clustering schemes would be more adaptive, combining both heuristics and experiential results for cluster identification. In future work we will apply HPSA to various contemporary applications, such as protein 3D structure prediction. Since HPSA can serve as an general-purposed optimization method, we will also put much emphasis on its interface design, so that we can cut down implementation efforts of applying it to other problems.

References

1. E.H.L. Aarts and J.H.M. Korst. *"Simulated Annealing and Boltzmann Machines"*. 1989.
2. A. Bevilacqua. "A Methodological Approach to Parallel Simulated Annealing on an SMP System". *J. of Parallel and Distributed Computing*, April 2002.
3. H. Chen, N.S. Flann, and D.W. Watson. "Parallel Genetic Simulated Annealing: A Massively Parallel SIMD Algorithm". *IEEE Trans. on Parallel and Distributed Systems*, 9(2), Febrary 1998.
4. R. Diekmann, R. Luling, and J. Simon. "Problem Independent Distributed Simulated Annealing and its Applications". *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, 1992.
5. Fred Glover and Gary A. Konchenberger. *"Handbook of metaheuristics"*. Boston, Kluwer Academic Press, 2003.
6. V. Granville, M. Krivunek, and J. P. Rasson. "Simulated Annealing : A Proof of Convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16:652–656, June 1994.
7. S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. "Optimization by Simulated Annealing". *Science*, May 1983.
8. G. Kliewer and S. Tschöke. "A General Software Library for Parallel Simulated Annealing". *IPDPS 2000*, pages 55–61, 2000.
9. S. Lee and K.G. Lee. "Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains". *IEEE Trans. on Parallel and Distributed Systems*, 7(10):993–1008, October 1996.
10. R. Otten and L. van Ginneken. *"The Annealing Algorithm"*. Kluwer Academic Publishers, March 1989.
11. D. Janaki Ram, T. H. Sreenivas, and K. Ganapathy Subramaniam. "Parallel Simulated Annealing Algorithms". *Journal of Parallel and Distributed Annealing*, 1996.
12. K.T. Simons, C. Kooperberg, E. Huang, and D. Baker. "Assembly of Protein Tertiary Structures from Fragmens with Similar Local Sequences using Simulated Annealing and Bayesian Scoring Functions". *J. of Molecular Biology*, 1997.

Multi-color Difference Schemes of Helmholtz Equation and Its Parallel Fast Solver over 3-D Dodecahedron Partitions*

Jiachang Sun

R & D Center for Parallel Software,
Institute of Software, Chinese Academy of Sciences, Beijing, China, 100080
sun@mail.rdcps.ac.cn

Abstract. In this paper, the problem of partitioning parallel dodecahedrons in 3D is examined. Two schemes are introduced and their convergence rate discussed. A parallel fast solver was implemented and tested experimentally, with the performance results presented.

1 Parallel Dodecahedron Partition in 3D

Give a three linear independent vectors: e_1, e_2, e_3, we set bi-orthogonal vectors n_1, n_2, n_3;

$$(e_j, n_k) = \delta_{j,k} \ (j, k = 1, 2, 3), \quad n_4 = n_1 - n_2, \quad n_5 = n_2 - n_3, \quad n_6 = n_3 - n_1.$$

There are six normals of six planes via the four directions e_1, e_2, e_3 and $e_4 = -e_1 - e_2 - e_3$. For any 3-D point P, we define $P = (t_1, t_2, t_3, t_4, t_5, t_6)$ by

$$t_1 = (P, n_1), \ t_2 = (P, n_2), \ t_3 = (P, n_3), \ t_4 = (P, n_4), \ t_5 = (P, n_5), \ t_6 = (P, n_6)$$

with $t_4 = t_1 - t_2, \quad t_5 = t_2 - t_3, \quad t_6 = t_3 - t_1$.

A basic parallel dodecahedron domain is defined as follows

$$\Omega = \{P | P = (t_1, t_2, t_3, t_4, t_5, t_6) | -1 \le t_\nu \le 1, (1 \le \nu \le 6), t_4 = t_1 - t_2,$$
$$t_5 = t_2 - t_3, \ t_6 = t_3 - t_1\} \tag{1}$$

The basic parallel dodecahedron domain has 14 vertices, 12 hyper-planes and 24 edges on the boundary.

It is worth to note that all of parallel dodecahedrons form a tiling of R^2 shown as Figure 1 and many crystal polyhedrons in material science can be composed from such kinds of dodecahedrons.

* Project supported by National Natural Science Foundation of China (No. 10431050) and the Major Basic Project of China ”High Performance Scientific Computing”.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 301–308, 2005.

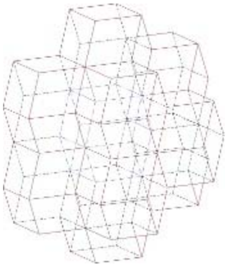


Fig. 1. A dodecahedron partition

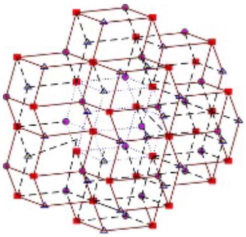


Fig. 2. Three colors ordering

2 Two Schemes for Laplacian Operator over Rhombic-Dodecahedron Partitions

In 3-D parallel dodecahedron domain case, the Laplacian operator can be can expressed using the following two operators

$$\Delta = (n_1, n_1)(e_1, \nabla)^2 + (n_2, n_2)(e_2, \nabla)^2 + (n_3, n_3)(e_3, \nabla)^2$$
$$+2(n_1, n_2)(e_1, \nabla)(e_2, \nabla) + 2(n_2, n_3)(e_2, \nabla)(e_3, \nabla) + 2(n_3, n_1)(e_3, \nabla)(e_1, \nabla),$$

and

$$\Delta = \sum_{j=1}^{4} A_j (e_j, \nabla)^2 + A_{12}(e_1 + e_2, \nabla)^2 + A_{23}(e_2 + e_3, \nabla)^2 + A_{31}(e_3 + e_1, \nabla)^2 \quad (2)$$

where

$$A_1 = (e_2 \times e_3, e_3 \times e_4) + (e_3 \times e_4, e_4 \times e_2) + (e_4 \times e_2, e_2 \times e_3),$$
$$A_2 = (e_3 \times e_4, e_4 \times e_1) + (e_4 \times e_1, e_1 \times e_3) + (e_1 \times e_3, e_3 \times e_4),$$
$$A_3 = (e_4 \times e_1, e_1 \times e_2) + (e_1 \times e_2, e_2 \times e_4) + (e_2 \times e_4, e_4 \times e_1),$$
$$A_4 = (e_1 \times e_2, e_2 \times e_3) + (e_2 \times e_3, e_3 \times e_1) + (e_3 \times e_1, e_1 \times e_2),$$
$$A_{12} = (e_1 \times e_3, e_2 \times e_4) + (e_2 \times e_3, e_1 \times e_4),$$
$$A_{23} = (e_2 \times e_1, e_3 \times e_4) + (e_3 \times e_1, e_2 \times e_4),$$
$$A_{31} = (e_3 \times e_2, e_1 \times e_4) + (e_1 \times e_2, e_3 \times e_4).$$

Based on the above two operator identities, we may derive some 13-point or 15-point difference schemes, respectively. The 15-point scheme corresponds piecewise linear finite element of 3-D Laplace equation within the rhombic dodecahedron partition. In this special discrete case,

$$(e_j, e_k) = (1 - \frac{4}{3}\delta_{j,k})h^2, \quad j,k = 1,2,3,4., \quad A_{12} = A_{23} = A_{31} = 0,$$

the above 15-point scheme degenerates to 9-point scheme. Let

$$c_1 = e_1 - e_4,\ c_2 = e_2 - e_4,\ c_3 = e_3 - e_4,\ c_4 = e_1 - e_2,\ c_5 = e_1 - e_3,\ c_6 = e_2 - e_3$$

it is natural to lead to a 13-point scheme with second order accuracy over the rhombic-dodecahedron partition case as follows

$$L_{13}u(P) = 12u(P) - \sum_{i=1}^{6}(u(P+c_i) + u(P-c_i)) = -\frac{16}{3}h^2(\Delta u(P) + O(h^2)) \quad (3)$$

Note that in the rhombic-dodecahedron partition case al points can be divided into three colors in the sense that for each fixed point there are no neighbor points of this point belong to the same color. We may denote the three color to be yellow, black and red. It is interesting that among the three color points the are divided into two groups. The first two colors belong to a group, on which the number of neighbor points equals to four and all neighbors are red color. And the number of neighbors for the red color points equals to eight. Based on the above three colors ordering we may derive the following so-called 5-5-9 scheme

$$L_5^Y u(P) = 4u(P) - \sum_{i=1}^{4} u(P+e_i) = -\frac{2}{3}h^2(\Delta u(P) + \frac{h}{9}ru(P) + O(h^2));$$

$$L_5^B u(P) = 4u(P) - \sum_{i=1}^{4} u(P-e_i) = -\frac{2}{3}h^2(\Delta u(P) - \frac{h}{9}ru(P) + O(h^2));$$

$$L_9^R u(P) = 8u(P) - \sum_{i=1}^{4}(u(P+e_i) + u(P-e_i)) = -\frac{4}{3}h^2(\Delta u(P) + O(h^2)) \quad (4)$$

where r is a third order differential operator

$$r = 2\frac{\partial^3}{\partial z^3} - 3\frac{\partial^3}{\partial y^2 z} - 3\sqrt{2}\frac{\partial^3}{\partial xy^2} - 3\frac{\partial^3}{\partial x^2 z} + \sqrt{2}\frac{\partial^3}{\partial x^3}$$

With matrix notation the above scheme can be written as

$$\begin{pmatrix} 4I & 0 & A_{YR} \\ 0 & 4I & A_{BR} \\ A_{YR}^T & A_{BR}^T & 8I \end{pmatrix} \begin{pmatrix} u_Y \\ u_B \\ u_R \end{pmatrix} = -\frac{2}{3}h^2 \left[\begin{pmatrix} \Delta u_Y \\ \Delta u_B \\ 2\Delta u_R \end{pmatrix} + \frac{h}{9} \begin{pmatrix} ru_Y \\ -ru_B \\ 0 \end{pmatrix} + O(h^2) \right] \quad (5)$$

where two matrices A_{YR} and A_{BR} are sparse with four non-zero terms (-1) at most in each row and column. Hence for Dirichlet boundary conditions the matrix A is non singular because of its weak diagonal dominant.

3 Convergence Rate

Lemma 1. *If*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

is invertible as well as A_{11} *and* A_{22}*, then*

$$A^{-1} = \begin{pmatrix} A_1^{-1} & -A_1^{-1}A_{12}A_{22}^{-1} \\ -A_2^{-1}A_{21}A_{11}^{-1} & A_2^{-1} \end{pmatrix}$$

or

$$A^{-1} = \begin{pmatrix} A_1^{-1} & -A_{11}^{-1}A_{12}A_2^{-1} \\ -A_{22}^{-1}A_{21}A_1^{-1} & A_2^{-1} \end{pmatrix}$$

where

$$A_1 = A_{11} - A_{12}A_{22}^{-1}A_{21}, \quad A_2 = A_{22} - A_{21}A_{11}^{-1}A_{12}.$$

Repeating to referee Lemma 1 leads to the following

Lemma 2. *If*

$$A = \begin{pmatrix} 4I & 0 & A_{YR} \\ 0 & 4I & A_{BR} \\ A_{RY} & A_{RB} & 8I \end{pmatrix}$$

is invertible then

$$A^{-1} = \begin{pmatrix} A_Y^{-1} & \frac{1}{8}A_Y^{-1}A_{YR}A_{RB}\tilde{A}_{22}^{-1} & -\frac{1}{4}A_{YR}A_R^{-1} \\ \frac{1}{8}A_B^{-1}A_{BR}A_{RY}\tilde{A}_{11}^{-1} & A_B^{-1} & -\frac{1}{4}A_{BR}A_R^{-1} \\ -\frac{1}{4}A_R^{-1}A_{RY} & -\frac{1}{4}A_R^{-1}A_{RB} & A_R^{-1} \end{pmatrix}$$

where

$$A_R = 8I - \frac{1}{4}(A_{RY}A_{YR} + A_{RB}A_{BR}), \quad A_Y = \tilde{A}_{11} - \tilde{A}_{12}\tilde{A}_{22}^{-1}\tilde{A}_{21},$$

$$A_B = \tilde{A}_{22} - \tilde{A}_{21}\tilde{A}_{11}^{-1}\tilde{A}_{12}$$

$$\tilde{A}_{11} = 4I - \frac{1}{8}A_{YR}A_{RY}, \quad \tilde{A}_{12} = -\frac{1}{8}A_{YR}A_{RB}, \tilde{A}_{21} = -\frac{1}{8}A_{BR}A_{RY},$$

$$\tilde{A}_{22} = 4I - \frac{1}{8}A_{BR}A_{RB}.$$

Based on the above Lemma, it is easy to verify the following convergence theorem.

Theorem 1. *Even though the above 5-5-9 scheme (4) only has first order local truncation error, it also has second order global convergence rate as well as 13-point scheme (3).*

4 Approximate Matrix Eigen-Decomposition Preconditioning

As is well known, an approximate sparse inverse may be a good preconditioning [1]. Now we propose that a reasonable approximate eigen-decomposition, based on fast algorithms, also can be taken as a preconditioner. In this case, it needn't require the preconditioner B to be sparse, but the working amount of Bu must less $O(N^2)$, e.g. $O(NLogN)$, where N is the matrix order.

Suppose A_o is an approximation of A. Let $E = I - A_o^{-1}A$, if $\rho(E) < 1$, then

$$A^{-1} = \{I - E\}^{-1}A_o^{-1} = \{I + \sum_{k=1}^{\infty} E^k\}A_o^{-1}$$

We may define various levels of preconditioners for the matrix A

$$B_0 = A_o^{-1}, \qquad B_k = \{I + \sum_{j=1}^{k} E^j\}A_o^{-1}, \quad (k = 1, \cdots).$$

If A is symmetry, then

$$\frac{(B_0Au, u)}{(u, u)} = 1 - \frac{(Eu, u)}{(u, u)}, \qquad \frac{(B_kAu, u)}{(u, u)} = 1 - \frac{(E^{k+1}u, u)}{(u, u)} \quad (k = 1, \cdots).$$

As an example, let $A = A_o + \varepsilon Q, \quad E = -\varepsilon Q$, then

$$\frac{(B_0Au, u)}{(u, u)} = 1 + \varepsilon\frac{(Qu, u)}{(u, u)}, \qquad \frac{(B_kAu, u)}{(u, u)} = 1 - (-\varepsilon)^{k+1}\frac{(Q^{k+1}u, u)}{(u, u)},$$

$$(k = 1, \cdots).$$

Thus, to be an efficient preconditioner, B must be an approximate inverse of A in some sense and Bu must be done easily. The second reason can explain why so many people interested in taking sparse matrices as approximate inverse preconditioners. However, in numerical PDE problems, the discrete Green function is completely dense, it is hard to get high efficient sparse approximate inverse preconditioners directly. In some cases we may find an approximate eigen-decomposition. The left question is can we find a fast algorithm for Bu in magnitude of $O(NLogN)$. A typical successful example is to solve Laplace equation in a cube domain by using the traditional FFT.

5 Parallel Fast Solver and Numerical Experiments

Based on the above analysis now we turn to find a preconditioner

$$B = W'DW, \quad W = (W_{jk}) \quad D = diag(d_k)$$

where the eigen-function matrix

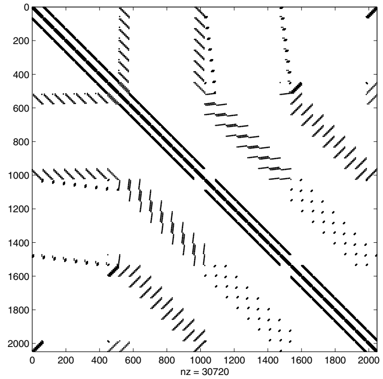


Fig. 3. 3D 15-point stiffness matrix

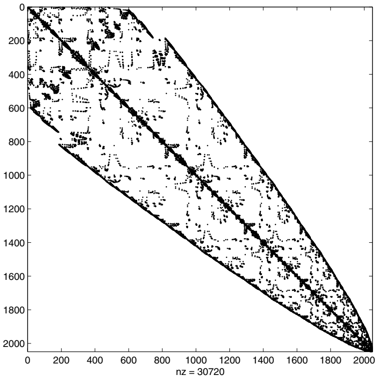


Fig. 4. Reordering towards the original

$$W_{jk} = e^{i\frac{\pi}{2}\mathbf{j}\cdot\mathbf{k}} = i^{((3j_1-j_2-j_3)k_1+(3j_2-j_3-j_1)k_2+(3j_3-j_1-j_2)k_3)h}$$

and the related approximate eigenvalue diagonal matrix

$$\frac{1}{d_k} = 8 - 2\cos(k_1+k_4-k_6)h\pi - 2\cos(k_2-k_4+k_5)h\pi - 2\cos(k_3-k_5+k_6)h\pi$$

$$-2\cos(k_1+k_2+k_3)h\pi + Ch^2 q_o.$$

where the constant C is depends on the volume of Ω_{3D} and schemes.

Figure 3 and Figure 4 represent the non zero structure of the discrete Helmholtz system over 3-D dodecahedron with periodic conditions, according to natural ordering and reordering towards the original point $(0,0,0)$, respectively.

Once one obtains an eigen-decomposition for a preconditioner $B = W^{-1}DW$, the left key problem is to find a fast multiplication of $z = Wr$.

Based on our extended Fast Fourier Transform algorithm over parallel dodecahedron partition, see [5] and [6], it is not hard to design a parallel fast solver for the above two schemes.

Test: Fast 3-D Helmholtz solver on a unit rhombic dodecahedron domain with six direction periodic boundary conditions.

Figure 5 lists CPU time comparison of our approximate eigen decomposition preconditioning algorithm to the usual ILU, run with the famous software PETsc

Table 1. Iteration counts and CPU time (Sec.) comparison for a rhombic dodecahedron domain

N	CG	HFFT	ILU(0)	ILU(1)	ILU(2)	ILU(4)	ILU(8)
4	0.0014	0.0018	0.0027	0.0065	0.0197	0.0664	0.0792
8	0.0638	0.0127	0.0707	0.1066	0.2026	1.3643	15.176
16	1.1868	0.1831	1.0544	1.2791	1.9904	9.1244	130.154
32	20.107	1.8897	14.977	17.302	23.373	72.985	2454.5
64	328.01	18.133	230.40	231.81	284.90	/	/

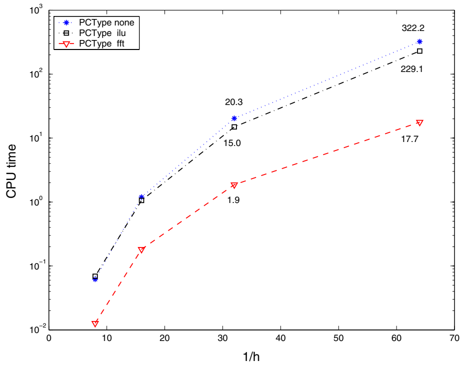


Fig. 5. Dodecahedron iteration: CPU comparison

Table 2. Iteration counts and CPU time (Sec.) comparison for non rhombic dodecahedron

	Iter.			CPU		
N	CG	ILU(0)	HFT	CG	ILU(k)	HFT
8	69	26	16	0.1017	0.0903	0.1140
16	138	50	19	1.9603	1.5171	1.9380
32	272	91	21	33.417	21.935	21.372
64	532	179	22	528.841	355.67	214.65

[2]. Table 1 shows, in this case, high level ILU(k) (k¿0) does not work, our fast algorithm is one magnitude faster than the traditional ILU for $h = \frac{1}{64}$.

Test 4: Fast 3-D Helmholtz solver on a non rhombic dodecahedron domain with six direction periodic boundary conditions. Table 2 shows in this case high level ILU(k) (k¿0) does not work, our fast solver still works with lower efficiency than rhombic dodecahedron domain case.

More parallel numerical experiments will be reported on the conference.

Acknowledgement. The figures and part computing are done by Yao Jifeng and Yang Chao.

References

1. L. Yu. Kolotilina, A. Yu. Yeremin, Factorized sparse approximate inverse preconditionings I: theory *SIAM Journal on Matrix Analysis and Applications* 14 (1993) 45 - 58.
2. PETc home page, *http://www-unix.mcs.anl.gov/petsd/petsc-2/, 2004*
3. Youself Saad, ILUs and Factorized Approximate Inverses are Strongly. - Bollhöfer, 2000.
4. Jiachang Sun, Multivariate Fourier series over a class of non tensor-product partition domains, *Journal of Computational Mathematics*, 21 (2003) 53–62.

5. Jiachang Sun, Jifeng Yao , Fast Generalized discrete Fourier transforms on hexagon domains, *Mathetitica Numerica Sinica*, 25(2004), N0.3, 351-366.
6. Jifeng Yao and Jiachang Sun, HFFT on parallel dodecahedron domains and its parallel implementation, *Journal on Numerical Methods and Computer Applications* 25(2004),303-314.
7. Jiachang Sun and Huiyuan Li, Generalized Fourier transform on an arbitrary tirangular domain, *Advances in Computational Mathematics* , 22(2005),223-248.
8. Jiachang Sun, Approximate eigen-decomposition preconditioners for solving numerical PDE problems, *Applied Mathematics and Computation (to appear)*

GridMD: Program Architecture for Distributed Molecular Simulation

Ilya Valuev

Institute for High Energy Densities of the Russian Academy of Science,
Izhorskaya 13/19, 127412 Moscow, Russia

Abstract. In the present work we describe architectural concepts of the distributed molecular simulation package GridMD. The main purpose of this work is to underline the construction patterns which may help to generalize the design of an application for extensive atomistic simulations. The issues such as design-time parallel execution implication, flexibility and extensions, portability to Grid environments and maximal adaptation of existing third-party codes and resources are addressed. The library is being currently developed, with gradually growing number of available components and tools. The basic GridMD engine is a free software and is distributed under the terms of wxWidgets library license [1].

1 Motivation and Strategy

The main subject of atomistic simulations is to study the microscopic behavior of a system of particles and to deduce physically important quantities from the microscopic model. Function of potential energy depending on particle coordinates may be taken from physical models of different kinds: classical, semi-empirical (with large number of parameters to be fitted for any specific system) or *ab-initio* (with much less number of fitting parameters and more generality). Having the potential defined, the next step is to extract system properties from it either by solving equations of motion (Molecular Dynamics) or by sampling the ensembles of phase space configurations selected by some criteria (Monte Carlo methods), or by searching the suitable configuration in phase space (transition state search, geometry optimization, ligand design, etc.). Sometimes the methods of exploring the system are combined in complicated numerical experiments. The most popular experiments in MD and MC have however relatively simple scenarios: take the system in some initial state and propagate it through the chain of other states by Newton equations solution or temperature-conditioned random process. Physically most important part of the model is the definition of the interaction potential. Looking at the simulation from the higher level as a tool to obtain physically significant results, researchers face the problem of process and data management which they must solve spending much effort on developing complex codes. Statistical averaging and variation of experiment parameters are always necessary to produce reliable data from numerical simulation [2]. Another fact is large simulation times for complex problems and the

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 309–314, 2005.

need to distribute computations. The alternative of taking existing simulation package and adjusting it for the problem may also require significant time effort to overcome the limits and inconsistencies in the third-party code.

The idea to create “another parallel Molecular Dynamics package” may seem not very promising taking in account large amount of fruitful work done in the field by Molecular Simulation community [3,4,5,6,7]. However, exactly this popularity of the subject and availability of different codes inspired us to begin developing Grid M(olecular) D(ynamics) library (GridMD)[8] to serve as flexible integration tool which may utilize as much of the existing models and methods as possible and integrate them into a single framework with distributed execution capabilities. The main strategy of GridMD is to have clear interfaces for all components, representing common simulation aspects, which may be especially important for simulation techniques [9] using combined potentials. In the current work we will cover mostly the upper layers of GridMD, responsible for distributed execution, leaving the description of MD-specific layers for further publications.

The GridMD code is designed as C++ class library providing construction tools for atomistic simulations. The library is intended mainly for usage by programmers who want to quickly build the simulation application. This purpose is somewhat analogous to the purpose of GUI library supplying its tools and patterns for easy development of applications with graphic interface. The library is as much platform-independent and flexible as possible. Any tools and components, which are platform-dependent or require additional third-party components (for example MPI libraries, Grid interfaces, MD packages etc.) may be used as extensions but are not strictly required. This simplifies selective usage in research and academic purposes and encourages friendly learning the library from simple aspects to more complicated. The intended target application is a statically compiled executable which is transferable to other hosts with the same OS. The components of the library have clear interplay patterns [10] and the third-party tools ranging from interaction potentials to Grid execution environments can be interfaced on different layers.

2 Numerical Experiments

The general structure of GridMD *Experiment* concept is shown on Fig. 1. The application framework logic assumes the definition of one experiment per application,

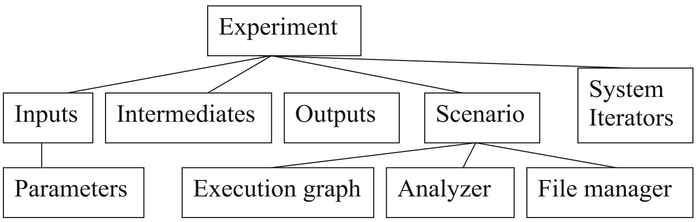


Fig. 1. Logical composition of *Experiment* component

this is however not strictly required. The *Experiment* consists of the following main components: *Inputs*, *Outputs* and *Intermediates* are data objects (files, numeric or symbolic parameters); *Scenario* contains the strategy of how the experiment is executed and this knowledge is encapsulated in *Execution Graph*; *System Iterators* are concrete linear process strategies driving the (atomic) system through the chain of states, the actual Molecular Simulation is performed by *mdIterators*.

Input Parameters. The numeric and symbolic input *Parameters* play special role, because they may be used as basis for creating experiment scenarios. The parameters may originate from different levels of the simulation (interaction potential parameters, propagator parameters, physical initial conditions, algorithmic parameters). GridMD provides a mechanism of registering the parameters via named *Variable* concept to make the *Experiment* aware of the available parameters and automatically create the corresponding entries in application configuration files if required. The typical parameter-based scenarios are parameter variation, averaging over a randomly selected sets in parameter space, parameter fitting. GridMD provides the developer with a set of predefined basic scenarios. *Scenario and Stages.* The *Scenario* concept standardizes the logic of application execution and allows to split the experiment into a set of separate processes (stages). The key point is that the stages may depend on each other through input and output but are not communicating during the execution. This process dependence is illustrated by execution graphs (Fig. 2). The lines show the process *Stages* with arrows indicating time direction of execution flow, the nodes represent process dependencies. Generally the stages coming out

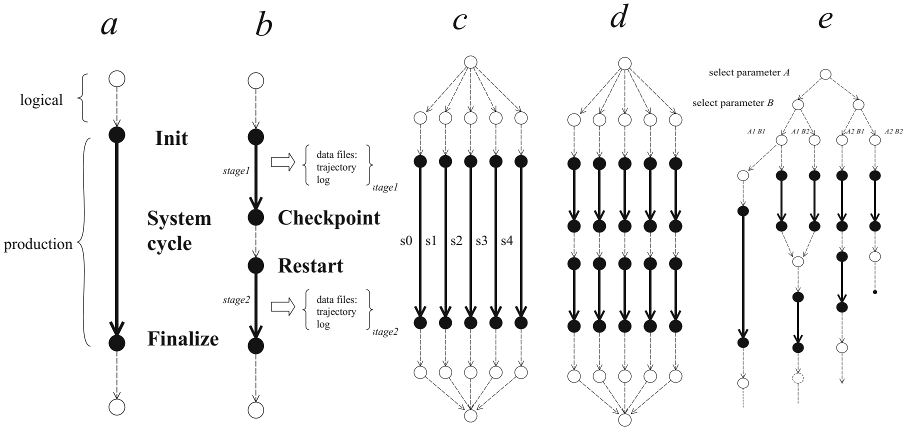


Fig. 2. Basic execution graph elements used in Molecular Simulations: *a* – simple linear single-process stage, *b* – chain of processes with intermediate data generation and checkpoints. The intermediate data is indexed by stage ids. Different experiment scenarios with their execution graphs: *d* – branching of single processes, the most frequently used scenario for parametric modeling, *e* – branching with checkpointing, important for lengthy simulations and distributed environments with time-limited queues, *f* – general tangled incompletely determined execution.

of a node depend on the stages coming in, meaning that the outgoing stages can not be started until the ingoing are *successfully* completed. Note that concrete *Scenario* implementation may have its own understanding of a *successfully completed node*, sometimes not requiring that all ingoing stages are finished (important for unreliable distributed environments). We distinguish the *logical* nodes and stages which may be introduced to *Execution graph* for structural purposes only and not reflecting encapsulated computational activity (white circles and thin lines) and *production* nodes and stages that represent computations (black circles and thick lines). The production stages, connecting mandatory *Init* and *Finalize* production nodes may be transferred to other hosts for remote execution.

Execution graphs can take different forms, generally the execution graph is not completely defined in advance by the application start. The *Scenario* component maintains the graph and can be divided into manager and worker parts. The worker part is used to start the required stage of the scenario. To simplify the development, GridMD uses single-application paradigm, so the worker and manager parts are contained in one executable. When distributed processing is required, the executable must be transferred to a remote system, switching to worker mode is usually based on the command line parameter passed to the executable. The *Scenario* component has four major tasks:

- execution graph construction before or in course of experiment execution, storage of up-to date graph state;
- determining the *pending* for execution production stages on the basis of the current graph state or terminating the execution if no stages are left;
- updating the graph state according to the results of the stage execution;
- invoking actual production stage execution for the pending stage by the request, identifying the stage.

The first three tasks are *manager* tasks and the last one is the *worker* task. The nodes and stages are indexed by symbolic identifiers, uniquely specifying their positions in the graph. The up to date graph itself is accessible for browsing by worker application components, which may wish to identify what part of the work is to be done based on the position of the stage in the graph.

Stage identifiers may contain enough information for starting a production stage, not requiring the implementation components to go into execution graph details. The rules for creating and parsing symbolic identifiers may depend on the *Scenario*. Although GridMD supports very general formulation of execution graphs, the main attention is paid to the most popular and clear forms (Fig. 2 *c, d*). For example, for the form from Fig. 2 *c*, used for parameter sweep, the symbolic stage identifier contains the branch number, making the definition of parameter set to start with in worker mode straightforward.

File Management. The node and stage identifiers may be used for the purpose of indexing files generated or required by worker components. This is done by scenario *File Manager.* By default it only composes the fully qualified local system file names from the identifiers and specified reference names, but also may be used for remote storage management. For example, the remote Grid data storage for intermediate data may be preferable in the cases when the phase space trajectory has to be recorded by production stages of Molecular Simulation experiment. The costs of data transfer to the local system may be high in case of limited local bandwidth. This transfer is usually also unnecessary because the trajectory data is used as intermediate for experiment result analysis, which may be as well performed remotely. Thus the intermediate data storage management can be delegated to the experiment *Scenario* and execution graph-based file management system. This system must then accomplish a task of locating the file by its name and graph id and is easily implemented, for example, on the basis of such services as Globus RLS [11].

Analyzer. Another component of the *Scenario* is *Analyzer.* It is included separately because frequently the *analysis phase* of Molecular Simulation is performed after the creation of intermediate trajectory data. It is normally less demanding computationally than the *production phase* but may be repeated several times with different parameters. The analysis scenario is connected with production scenario and uses by default the same execution graph. The *Analyzer* component is used to construct and tune the analysis phase of the experiment, which logically is just another version the the same experiment. GridMD provides a set of standard analysis tools for the trajectories generated by MD: property extraction as function of time, time correlations, particle correlations, distributions. These tools support ensemble averaging for execution graphs of the forms from Fig. 2 *c, d.* and may be accomplished by the same executable in analysis phase. The only difference between analysis and production stages of the scenario is the "MD propagator" of the system: the Newton equations solver is replaced by trajectory file reader, which loads the trajectory files generated by production stages. Some *Analyzer* components (such as temperature or energy log writers) may be as well used in production phase for monitoring purposes.

System Iterator. The *System Iterator* component is the core of any production stage, encapsulating the strategy of iterative cycle with possibility of checkpointing. This component may be requested to perform a number of pending iteration cycles and then record its complete state, supplying the transferrable files required for restarting. It must also inform the *Scenario* component about the termination when no more pending iterations are left. Optionally *System Iterator* may supply information about the total number of its cycles and/or the computational cost per cycle, expressed in normalized units. Note that this information is not always known, but is very useful in job splitting strategy. The chain scenarios, which can split the jobs in time can utilize this information to control the duration of each chain.

System iterator designed for Molecular Simulations (*mdSimulator*) is supplied with GridMD. It serves also as a manager for the physical part of the experiment, having atomic systems, interaction potentials, propagators and other model tools as components.

3 Integration to the Grid

As described in the previous sections, the design of GridMD implies easy Grid integration, because the application itself generates a sequence of jobs to be executed. The aims of distributed execution environment are then limited to the actual transfer and execution of jobs, and also accompanying tasks such as job monitoring and scheduling. Global job execution is managed in GridMD by *JobSpooler* component which receives the stage execution requests from *Experiment*, converts them to appropriate system or external commands submitting jobs and informs the *Experiment* of the stage execution status.

The execution environment tool to be used as primary testbed for Grid interface of GridMD is NIMROD/G [12,13]. This project utilizes concepts of computational experiments and parameter variation which are very close to that of GridMD. The parameter sweep scenarios, shown on the Fig. 2 *c* may be directly mapped to NIMROD experiments, and the scenarios from the Fig. 2 *d* can be converted to a sequence of NIMROD experiments. The web portal functionality of NIMROD provides monitoring and resource selection facilities which are not managed by GridMD but necessary for efficient operation.

Acknowledgements

This work was performed with support from the Russian Foundation for Basic Research (grants RFBR-03-07-90272-v, NWO-047.016.007/ RFBR-04-01-89006) and the Russian Academy of Science (RAS program N17).

References

1. `http://www.wxwidgets.org`
2. A. Yu. Kuksin, I. V. Morozov, G. E. Norman, V. V. Stegailov, and I. A. Valuev, *to appear in* Molecular Simulation (2005)
3. Charm++ website: `http://charm.cs.uiuc.edu/`
4. GAMESS grid portal: `https://gridport.npaci.edu/gamess/`
5. Takashi Amisaki, Shin-Ichi Fujiwara, *Proceedings of the* 2004 Symposium on Applications and the Internet-Workshops (2004), 614
6. J.Pytlinski, L.Skorwider, K.Benedyczak, M.Wronski, P.Bala, V.Huber in P.M.A.Sloot et al. (Eds.): ICCS 2003, LNCS 2658, 307–315, Springer-Verlag, 2003
7. Y. Li and M. Mascagni, *Grid-based Monte Carlo Application* Lecture Notes in Computer Science, 2536:13-25, GRID2002, Baltimore, 2002.
8. GridMD development website: `http://biolab1.mipt.ru/gridmd`
9. I. Valuev, Comput. Phys. Comm. **169** (2005) 60-63.
10. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns. Elements of Reusable Object-Oriented Software* Addison Wesley Longman, 1995
11. `http://www-unix.globus.org/toolkit/docs/4.0/data/rls/`
12. NIMROD website: `http://www.csse.monash.edu.au/~davida/nimrod/`
13. Sudholt, W., Baldridge, K., Abramson, D., Enticott, C. and Garic, S. New Generation Computing **22** (2004) 125-135.

Visuel: A Novel Performance Monitoring and Analysis Toolkit for Cluster and Grid Environments*

Kuan-Ching Li[1,**], Hsiang-Yao Cheng[1], Chao-Tung Yang[2], Ching-Hsien Hsu[3], Hsiao-Hsi Wang[1], Chia-Wen Hsu[1], Sheng-Shiang Hung[1], Chia-Fu Chang[1], Chun-Chieh Liu[1], and Yu-Hwa Pan[1]

[1] Parallel and Distributed Processing Center,
Department of Computer Science and Information Management,
Providence University, Taichung 43301, Taiwan
kuancli@pu.edu.tw

[2] High Performance Computing Laboratory,
Department of Computer Science and Information Engineering,
Tunghai University, Taichung 40704, Taiwan
ctyang@thu.edu.tw

[3] Department of Computer Science and Information Engineering,
Chung Hua University, Hsinchu 300, Taiwan
chh@chu.edu.tw

Abstract. The computing power provided by high performance low-cost PC-based Cluster and Grid platforms are attractive, and they are equal or superior to supercomputers and mainframes widely available. In this research paper, we present the design rationale and implementation of Visuel, a toolkit for performance measurement and analysis of MPI parallel programs and real time resources monitoring in cluster and grid computing environments. The proposed toolkit is web-based interface to show performance activities of all computing nodes involved in the execution of a MPI parallel program, such as CPU and memory usage levels of each computing node, and monitors all computing nodes of a computing platform by displaying real time performance data. In addition, this toolkit is able to display comparative performance data charts of multiple executions of MPI parallel application under investigation, which facilitates the "what-if" analysis. The usage of this toolkit shows that it outperforms in easing the process of investigation of parallel applications.

Keywords: Monitoring, MPI Parallel Program, Distributed Computing, Performance Visualization.

[*] This research is supported in part by National Science Council, Taiwan, under grants no. NSC93-2213-E-126-010 and NSC94-2213-E-126-005, and National Center for High Performance Computing, Taiwan, under "Taiwan Knowledge Innovation National Grid" Project.

[**] Corresponding author.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 315 – 325, 2005.

1 Introduction

In recent years, the cluster computing technology has become a cost-effective computing infrastructure, because it aggregates resources of computational power, communication and storage [6, 8]. It is also considered a very attractive platform for low-cost supercomputing.

Cluster of workstations are easy to build, cost effective and highly scalable. It consists of a number of personal computers or workstations that are interconnected through a high-speed network (Gigabit Ethernet, Myrinet or Infiniband) for information exchange and coordination among them. They run commodity operating systems, such as Linux. In addition, we can connect a number of cluster platforms to form a grid platform, which advantage is to obtain more computational power at low cost. Grid computing offers a model for solving massive computational problems using large numbers of computers arranged as clusters embedded in a distributed infrastructure. Grid computing has the design goal of solving large problems as any single supercomputer, whilst retaining the flexibility to work on multiple smaller problems. It involves sharing heterogeneous resources (based on different platforms, hardware/software, computer architecture, computer languages), located in different places belonging to different administrative domains over a network using open standards.

With advances in networking technology, interconnecting PCs and workstations is not a problem anymore. Despite of this fact, there is still much to do in the software domain. Parallel programs can behave in a number of unexpected ways, because of their complex structure, the number of computing nodes used to execute the application in a cluster or grid platform, the dataset used by the parallel code, the regularity of applications and algorithms in space and time, the heterogeneity of software and hardware platforms, among a number of other reasons. In addition, effective partitioning, allocation and scheduling of application programs on a network of workstations are crucial to achieve high performance. Thus, the performance is very sensitive to the strategy used to distribute data to the processors or clients [1].

There are several performance monitoring toolkits available, to visualize graphically the performance of an application's execution, e.g., VAMPIR [11] and DIMENAS [9]. One way to improve the performance of a parallel application is to analyze its performance metrics, e.g., CPU load, memory usage, I/O load, among others, and see what happened with the execution of that particular MPI parallel application at given conditions. In this paper, we designed and implemented Visuel toolkit for cluster and grid environments, providing in this way graphical performance data visualizations of MPI parallel applications executed in either cluster or grid platforms.

The remainder of this paper is organized as follows. In section 2 is discussed some related researches in performance and monitoring tools for distributed systems. Section 3 introduces the Visuel toolkit and its implementation for cluster and grid platforms. Later in section 4, a MPI parallel program is executed using Visuel toolkit, showing performance data and comparative performance data charts. Finally, in section 5, a brief conclusion and future researches are presented.

2 Background

Several performance-monitoring tools are available as recent researches in distributed platforms, in order to achieve higher performance and to visualize graphically the performance data of an application's execution, e.g., VAMPIR [11] and DIMEMAS [9].

A number of monitor tools that generate HTML pages containing performance graphical images and data are also available. MRTG (Multi Router Traffic Grapher) [4], based on Perl and C. It is a tool to monitor the traffic load on network links. It generates HTML pages containing PNG images, which provide a live visual representation of the traffic. It consists of Perl script that uses SNMP to read the traffic counters and a fast C program that logs the traffic data. RRD (Round Robin Database) [7] is a system that stores and displays time-series data (e.g., network bandwidth, machine-room temperature, average load). The RRD stores in a compact way that does not expand over time.

The advantage of MRTG over RRD is that it is easier to use, whilst RRD has more graphical display options than MRTG. However, the main disadvantage is that MRTG has fixed format data (it can only shown the data over time), and it depends fully on the use of SNMP to obtain the data, otherwise, it cannot work. The advantage of MDS is easy to get information and quick, but it does not have graphical display options.

Several well-known tools such as Ganglia Cluster Toolkit [2] and CACTI [5] are particular implementations of RRD tool developed by independent research teams around the world. As mentioned before, tools such as VAMPIR, Ganglia, DIMEMAS, and CACTI can only show the data over time of each one of the computer nodes in a cluster system, not possible to show in particular periods, such as the start and end of execution of an application in a grid system.

3 System Overview

The Visuel toolkit is designed and implemented based on RRD tool [7]. The main reason why we started to work on this toolkit is that we need a tool to visualize the performance data of MPI application during its execution, the moment it starts until the moment it finishes. In addition, we need a tool which we can perform "what-if" analysis, that is, to compare performance results of a parallel application during its development stage, and know whether it is cost-effective using a given number of computing nodes.

Visuel toolkit is scalable, i.e., it is able to measure long running MPI applications on as much computing nodes in either cluster or grid platforms as they are involved in the computations. It is able to generate from minutes to several hours MPI parallel programs' executions. This toolkit supports heterogeneous and homogeneous clusters of workstations; this tool can work on any platform where RRD tool is able to run and installed.

In next subsections, design and implementation of this monitoring and analysis performance toolkit are introduced and discussed.

3.1 Components of the System

The Visuel performance toolkit is composed of two components. The first one is Performance Visualization Manager (VM), which provides to the user graphical visualization of application execution's data, while the second component is the DP*Graph Code Visualization Manager (CVM), responsible to bring parallel timing graph representation using DP*Graph, as discussed in [13]. The Visuel toolkit scheme is shown in figure 1.

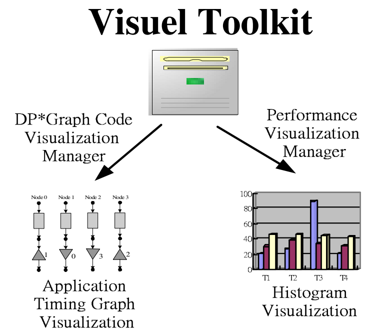

Visuel Toolkit
DP*Graph Code Visualization Manager
Performance Visualization Manager
Node 0
Node 1
Node 2
Node 3
1
0
3
2
Application Timing Graph Visualization
100
80
60
40
20
0
T1
T2
T3
T4
Histogram Visualization

Fig. 1. Visuel toolkit and its components

Essentially, the visualizations are processed in three steps, as shown in figure 2.

1. rrdtool create: set up a new Round Robin Database (RRD),
2. rrdtool update: store new data values into an RRD,
3. rrdtool graph: create a graph from data stored in one or several RRD.

Create
Update
Graph

Fig. 2. Visualization creation process in Visuel toolkit

The Performance Visualization Manager (VM)'s main objective is to monitor the amount of resources used to execute a MPI parallel program, from its start to end points, e.g., CPU load, memory usage, network bandwidth. These data are used for performance analysis and tuning as next step.

Different from other monitoring tools that provides performance data since the system is on, not being able to provide specific measurements in a specific time. Additionally, we cannot have in our chart past time data for analysis. The application developer can use performance data obtained from successive executions to perform code tuning, in order to observe the performance improvements in most recent tuned parallel program.

3.2 Execution Data Collection

The data collection and later visualization are performed according to following steps:

Step 1: for the purpose of record the performance data selected at this initial step, the RRD database is built every time on those computing nodes involved in our computation.

Step 2: before executing our MPI parallel program, the master node should execute MDF (Monitor Daemon Parent process), which goal is to fork MDC (Monitor Daemon Child process) to each of involved computing node. The MDC in each computing node involved is going to detect when master node starts with the distribution of tasks (segments of code of MPI parallel program), its job is at this moment recording the performance data, originated from the execution of MPI parallel program. Before the beginning of execution, each involved computing node is in "waiting" state, since the tasks did not reach to the computing nodes yet. See figure 3 for details.

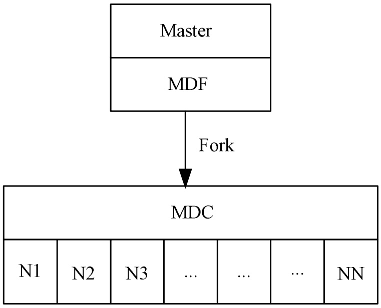


Fig. 3. Master node (MDF) parent process forked to a number of child processes (MDC), which is equal to the number of computing nodes involved in the computation

Step 3: As the MPI parallel program is started to run in each of involved computing nodes, MDC in each computing node is acknowledged. MDC will get defined system resource usage from each computing node, and through network file system protocol, these performance data are written back to RRD database in Master Node and log file.

At the moment of overlapping two data charts of the same MPI parallel program executed, since their execution time are different, they will appear side by side in different execution times. The log file is used to correct this problem, and it helps us in overlapping the two performance data charts in the same execution, beginning at the same start point. As MDF detects the end of execution of MPI parallel program, this process will broadcast a message to each involved computing node for the sake of stopping monitoring the computing nodes and the process of obtaining performance data can be stopped. See figure 4 for detailed explanations of this step.

Step 4: During the programmer starts the performance tuning process, by reviewing several executions of the same MPI parallel program, the programmer can choose some of several executions of this MPI parallel program to display a combined data chart of these selected executions. The scheme in figure 5 shows the details of this selection process.

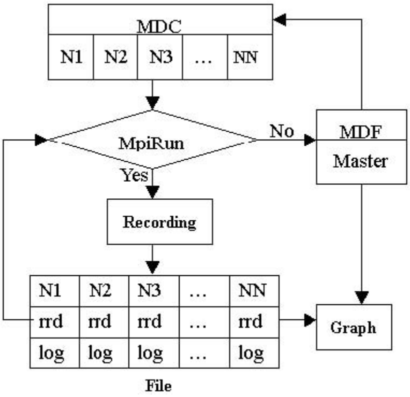


Fig. 4. File system in RRD database and log file

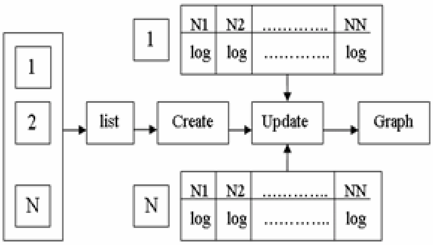


Fig. 5. Selection of specific executions of a MPI parallel program during its development

Step 5: The Visuel toolkit will clean up pending processes by checking each of computing nodes, since these only cause marginal errors in performance data. Otherwise, it is possible to cause programmers misinterpret obtained results.

3.3 MDS Data Acquisition and Visualization

MDS (Monitoring and Discovery System) is information services provider of Globus Toolkit [11, 12]. It is based on LDAP protocol, which assists in acquiring system information. It is divided to 2 parts. The former one is Information Providers (IPs), which is used for information collection. The latter is GRIS, which is used to search specific information we need. *Get Info* is a collection of scripts used to collect all computing nodes' GRIS. First, we must run a slapd daemon. Later, we utilize grid-info-search or globus-job-run to get every computing nodes' info and save them as a log file.

Several versions of MDS available does not support Globus Toolkit in MacOS, and consequently, if the computing node is Apple's RISC-based processor, we use globus-job-run to execute the customized script we developed for MacOS, while any other computing node that run MDS, we use globus-info-seach to collect those information we need.

Information system is divided into three parts:

Part 1: Information Providers (IPs) are used for information collection. User cannot get information by IPs directly. It must be used by GRIS,

Part 2: Grid Resource Information Service (GRIS) that is used to search specific information we need,

Part 3: Grid Index Information Service (GIIS) is used to find where nodes are and we can via this to use other nodes' GRIS. GIIS doesn't need run on every node, being just needed only on the master node.

We have two ways to connect GRIS. One is to connect directly. Use this way we have to connect every nodes by ourselves. The other is to connect by GIIS. If via GIIS, we only use one instruction and then will return all nodes' information we need, as show in figure 6.

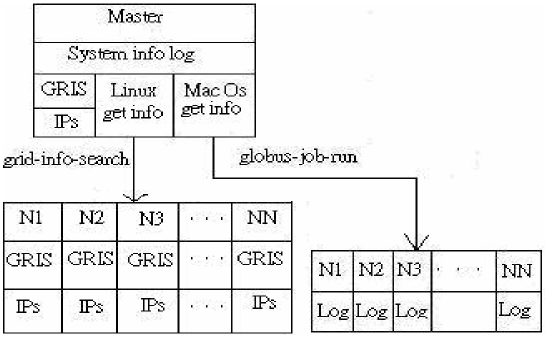
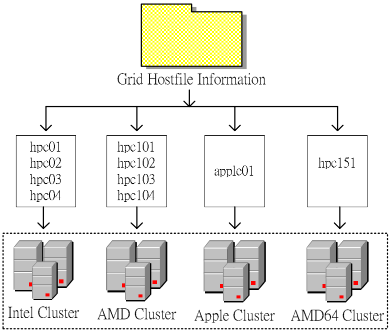


Fig. 6. MDS usage scheme to obtain information



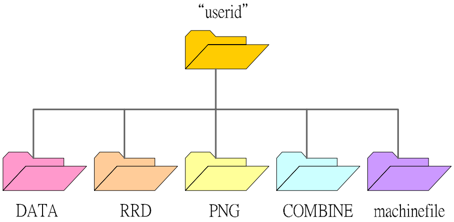


Fig. 7. Hostfile in the grid-hostfile

Fig. 8. List of folders inside main folder

Get Info is a collection of scripts used to collect all computing nodes' GRIS. Hostfiles are referenced by these scripts, and it is used to record the number of computing nodes we have. We use cluster platform name to be the hostfile name. All of these information are stored in a folder named grid-hostfile, as shown in figure 7.

The process to add/removing computing nodes in Visuel toolkit has three steps. In the first step, user must input cluster platform name and computing node name. As second step, some scripts are executed to check the computing node, which is authenticated and added. Finally, in the last phase, users just refresh the webpage, when it will be shown in the computing node webpage together with other computing nodes.

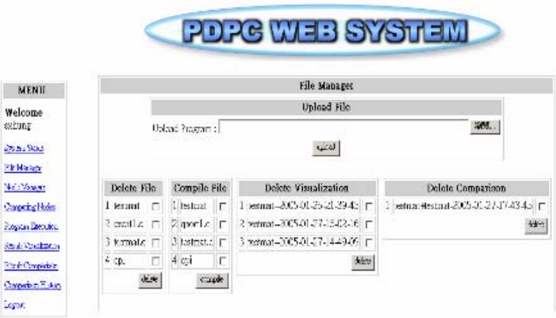
4 Using Visuel Toolkit

We have used Visuel toolkit to study several MPI parallel applications. In this research paper, we will show the execution and visualization of a MPI parallel matrix multiplication program.

The experimental environment is a grid platform built using a number of cluster platforms available in our laboratory. The first cluster platform, namely Intel Heterogeneous Cluster, is built using a number of Intel processors of different speeds (P2 300MHz to P4 2.8GHz) and amounts of memory (from 128MB to 768MB), interconnected via Fast Ethernet. The second cluster platform is a homogeneous platform, built up using 17 PCs with AMD Athlon 2400+ CPUs and 1GB memory in each node, interconnected via Gigabit Ethernet. The third homogeneous system, named Apple Cluster, is built up using 2 nodes with PowerPC(970) 1.6GHz CPU and 1.25GB memory. Finally, the fourth homogeneous cluster system is built using 4 computing nodes, where each of them contains 1 AMD Sempron 64-bit 2800+ CPU and 1GB memory, interconnected via Gigabit Ethernet.

As user logs in the management system, the user will see a list of programs, executables and visualizations of his research in the main screen. The user can remove or edit older or unused files, to compile recent created new parallel programs, choose to delete older and unused visualized files and choose to delete older and unused compared files to be deleted. See figure 9 for the visualization of the user's workplace.

Before proceeding with manual selection of computing nodes, the Visuel toolkit shows the listing of all computing nodes available, its system information, speed, numbers of CPUs, memory capacity and OS kernel installed.



Delete File		
1	testmat	☐
2	cmatl.c	☐
3	testmat.c	☐
4	cpi	☐

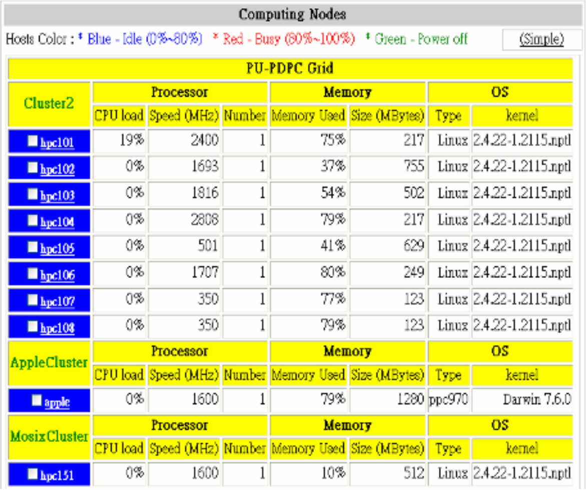
Compile File		
1	testmat	☐
2	qsort.c	☐
3	testmat.c	☐
4	cpi	☐

Delete Visualization		
1	testmat–2005-01-25-21-29-43	☐
2	testmat–2005-01-27-15-02-16	☐
3	testmat–2005-01-27-14-49-06	☐

Delete Comparison		
1	testmat:testmat-2005-01-27-17-43-43	☐

Fig. 9. User's workplace screen shot

Under little usage, the background of machine name is Blue, meaning that CPU usage is between 0%~80%. When under quite full utilization, the color is Red, while color Green is displayed that specific computing node is off. See figure 10 for computing node page of Visuel Toolkit for our local PDPC/PU Grid platform.



Computing Nodes

Hosts Color : * Blue - Idle (0%~80%) * Red - Busy (80%~100%) * Green - Power off (Simple)

PU-PDPC Grid							
Cluster2	Processor			Memory		OS	
	CPU load	Speed (MHz)	Number	Memory Used	Size (MBytes)	Type	kernel
■ hpc101	19%	2400	1	75%	217	Linux	2.4.22-1.2115.nptl
■ hpc102	0%	1693	1	37%	755	Linux	2.4.22-1.2115.nptl
■ hpc103	0%	1816	1	54%	502	Linux	2.4.22-1.2115.nptl
■ hpc104	0%	2808	1	79%	217	Linux	2.4.22-1.2115.nptl
■ hpc105	0%	501	1	41%	629	Linux	2.4.22-1.2115.nptl
■ hpc106	0%	1707	1	80%	249	Linux	2.4.22-1.2115.nptl
■ hpc107	0%	350	1	77%	123	Linux	2.4.22-1.2115.nptl
■ hpc108	0%	350	1	79%	123	Linux	2.4.22-1.2115.nptl
AppleCluster	Processor			Memory		OS	
	CPU load	Speed (MHz)	Number	Memory Used	Size (MBytes)	Type	kernel
■ apple	0%	1600	1	79%	1280	ppc970	Darwin 7.6.0
MosixCluster	Processor			Memory		OS	
	CPU load	Speed (MHz)	Number	Memory Used	Size (MBytes)	Type	kernel
■ hpc151	0%	1600	1	10%	512	Linux	2.4.22-1.2115.nptl

Fig. 10. PDPC/PU Grid platform computing node selection webpage

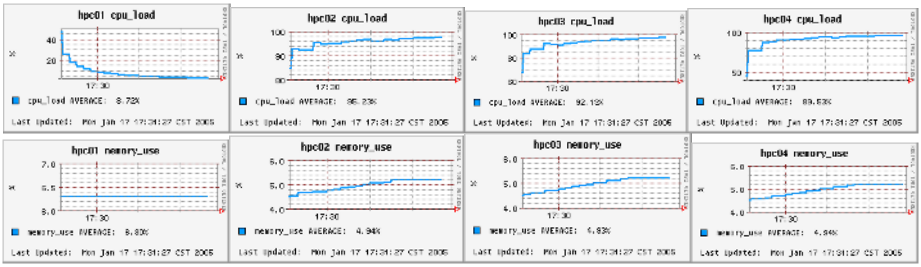


Fig. 11. Parallel program performance visualization

Once finished the execution of MPI parallel application, performance data of selected execution are available and it is able to be displayed anytime. As in figure 11, specific performance data of each computing node is displayed separately.

The developer can perform "what-if" analysis, that is, comparison of several runs of same parallel program, for example, minor changes in his program code, modifications in the loop levels, data distribution. The Visuel toolkit allows the developer to perform comparisons of different runs, by calling previous results. Note that performance data of each computing node are draw for each performance data category, e.g., CPU load, memory usage. Example of comparisons can be seen in figure 12.

The first experiment involves the development and execution of parallel versions of matrix multiplication program, using four computing nodes of our grid platform.

The execution of parallel application is shown in figure 11, while figure 12 shows the comparison of two different versions of the matrix multiplication program, where the “red line” executes “ijk” and the “black line” executed “ikj” shown in this chart.

Performance evaluations are quite easy using Visuel toolkit. By looking at each selected computing node, it is possible to see and compare performance “before” and “after” modifications in the developer’s parallel program.

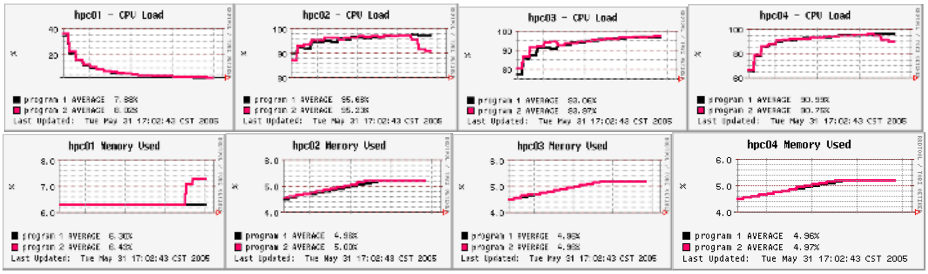


Fig. 12. Performance data comparison screenshot

5 Conclusions and Future Work

We show in this paper the viability of implementing a toolkit that brings to developers performance data visualizations originating from executions of his parallel applications. In addition, this toolkit is a place where the developer can perform “what-if” analysis on his parallel application, in order to tune the application either to achieve to higher performance or to fulfill the developer’s parameters.

As future work, several directions of this research are ongoing. The first activity is to include this performance visualization in our PDPC/PU (Parallel and Distributed Processing Center/Providence University) webportal. This webportal provides access of registered members to run their parallel applications on cluster and grid computing platforms in a secure way, as also easier will be the management to system administrators.

Once this visualization tool is included in the computing system, the developer is able to develop and modify his parallel program as much as he needed, and work on performance analysis of his parallel program with successive attempts under “what-if” analysis, efforts to try to obtain higher performance of his parallel program.

As part of our investigation, we will analyze possibilities that thread migration and thread level parallelism techniques are included in DP*Graph Code Visualization Manager, in order to assist with achieving higher performance in the developer’s parallel applications.

Another idea is to implement automatic computing nodes selection algorithm and integrate it in our cluster and grid computing platforms. Computing nodes in each site are chosen based on information provided in real-time basis, then the Visuel toolkit can also be used to perform “what-if” analysis, in the sense that which of selected computing nodes should execute what pieces of sequential code, since in a heterogeneous cluster and grid computing environments, the processors of computing nodes are different in speed. However, the automatically computing nodes selection problem is still an open challenge.

References

[1] L. Cheung and A.P. Reeves. High performance computing on a cluster of workstations, IEEE, 1992.

[2] Ganglia Cluster Toolkit. http://sourceforge.net

[3] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard, in http://www.mcs.anl.gov/mpi/mpicharticle/paper.html, Argonne National Laboratory, 1996.

[4] MRTG Webpage, in http://www.mrtg.org

[5] CACTI Webpage, in http://www.cacti.net/

[6] D.K. Panda and L.M. Ni, Special Issue on Workstation Clusters and Network-based Computing: Guest Editors' Introduction. Journal of Parallel and Distributed Computing, 40(1), January 1997.

[7] RRDtool Webpage, in http://www.rrdtool.org

[8] J. Sang, C.M. Kim, T.J. Kollar and Isaac Lopez, High-performance cluster computing over Gigabit/Fast Ethernet, Informatica, 23, pages 19-27, 1999.

[9] DIMEMAS Tool Webpage http://www.cepba.upc.es/dimemas/

[10] Pallas Products Webpage (VAMPIR tool) http://www.pallas.de/e/products/vampir

[11] Globus Toolkit Webpage, in http://www.globus.org/

[12] MDS Tool Webpage http://www-unix.globus.org/toolkit/mds/

[13] K.C. Li, H.C. Chang, C.T. Yang, L.M. Sato, C.Y. Yang, Y.Y. Wu, H.K. Liao, M.C. Hsieh, C.W. Tsai, and M.Y. Pel, "On Construction of a Visualization Toolkit for MPI Parallel Programs in Cluster Environments", in AINA'2005 The 19th IEEE International Conference on Advanced Information Networking and Applications, vol. II, Taipei, Taiwan, 2005.

Introduction to a New Tariff Mechanism for Charging for Computer Power in the Grid

Sena Seneviratne and David Levy

School of Electrical and Information Engineering,
The University of Sydney, Sydney, NSW, Australia
{auntvini, dlevy}@ee.usyd.edu.au

Abstract. In order to charge the computer power in the grid, we have made an effort to work towards a standard pricing unit. Currently there is a lot of work done towards the development of grid economy models and architectures. But when it comes to charging, the usual metric which has been popularly used is $ per CPU per Hour which seems to be too simple. Our effort is to make this metric more meaningful to both grid service provider and client. We argue that in a particular grid host the metric should reflect the true load consumed by the clients and the delays caused due to the other loads. Further it should eventually reflect the network, memory etc consumed by the client as well.

Previously we have studied about the prediction in the grid after introducing the division of the load average at the kernel level. This gave more meaning to the historical load collection as CPU historical load data had been collected separately for each login user. Interestingly, later on the division of load strategy has been helpful in the development of a meaningful tariff mechanism and would be demonstrated in this paper.

Eventually this fare mechanism would be used to predict the computational costs, which would certainly contribute to the scheduling in the grid.

1 Introduction

Varying CPU load has a significant effect on the running time of CPU-bound applications. Indeed, for certain types of applications the running time of a computer-bound task is linearly proportional to the average CPU load it encountered during the execution [5]. Important information is the composition of the users who are logged in a grid host at a particular time. The focus of this paper is to develop a tariff mechanism, which reflects the net load average and delays due to other loads etc. Our focus is on a grid of computers. Our contribution is to introduce a fairer approach for charging the client. This is better than the existing flat $/CPU/hour mechanism.

Working towards a tariff mechanism has been motivated by previous work done by David Abrahamson on Nimrod/G project [1, 2, 3]. They have proposed economic based models for managing resource allocation in Grid computing environments. The Nimrod-G has been one of the main landmarks in this regard. The economic approach provided a fair basis in successfully managing decentralization and heterogeneity that is present in human economies. The models can be based on bartering or prices. In

M. Hobbs, A. Goscinski, and Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 326 – 335, 2005.

the bartering-based model, all participants need to own resources and trade resources by exchanges. In the price-based model, the resources have a price, based on the demand, supply, value, and the wealth in the economic system.

But the dynamic nature of the grid encourages us to develop a standard yet a changing price unit. In a grid, which is implemented through Globus, a user will have to register himself with the provider and needs to be allocated a separate login account. Thus the user is only allowed to submit his HPC jobs to this particular account. Therefore if we can measure the load under that particular user-login, such collection of historical load profiles will influence the behavior of that of other user logins. This fact emphasizes the importance of the division of the load signal, which we have already conducted to improve the prediction solution in grid.

The division of load means after making some necessary changes to the kernel code, the load signal could be collected separately for each login user. Previously we have shown that using the division of the load signal we would predict the load signal better. The division of load signal is going to be helpful in the development of a better pricing/tariff mechanism

2 Problem Statement

Our aim is to introduce a dynamic pricing/tariff mechanism for the grid. Our analysis has been done at the source or service provider level.

We argue that the currently most popular charging method which is $ per CPU per hour is unfairly static. The grid is multi user system; this means multiple users should be able to submit jobs simultaneously, if they chose to do so. For example if there are 3 users who have been registered with a particular host/service provider, when the actual owner is not logged in, then all these 3 users are eligible to submit jobs. Under such complex situations it would be unfair to have a static charging mechanism.

3 Analysis of the Load Signal

3.1 The Load Signal and the Run Time of a Task

Dinda and O'Halloren has related the running time of a task [5], t_{nom} to the average load it experiences while it runs using the following continuous time model:

$$\frac{texec}{1+\frac{1}{texec}\int_{0}^{t_{exec}} z(t)dt} = tnom \tag{1}$$

Here $z(t)$ is the load signal, shifted such that $z(0)$ is the value of the signal at the current time, t_{now}. The t_{nom} is free load runtime and t_{exec} is under load normal runtime. It has been assumed that the majority of the workload runs at similar priority.

As the average background load, $la(t) = \frac{1}{t_{exec}} \int_{0}^{t_{exec}} z(t)dt$

Therefore, $texec = tnom * \{1 + la(t)\}$

$$\text{Thus,} \quad \frac{texec}{1 + la(t)} = tnom \tag{2}$$

4 Theory of Tariff Mechanism

4.1 Dynamism of Tariff Under Changing Load Profiles

A particular user runtime is get elongated as a result of all the other user submissions. If we can calculate the free load runtime from the elongated runtime and background load, then we would be able to charge in a fairer manner. This is because the charging should be based on the runtime of a particular job under free background load or no load.

4.1.1 Calculation of the Correction Factor for the Charge C $/CPU/Hours

We ague that a correction for the traditional grid or cluster computer charge $/CPU/hours is necessary as

There should be a consideration for the delays due to the background load.

The charges should also depend on the load average

The Calculation Process for a correction factor for delays due to background loads:

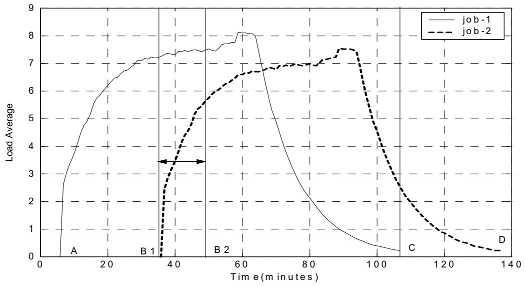


Fig. 1. Running under background loads

The Job-1 has been submitted to the grid node. After a time delay of AB1, Job-2 has been submitted to the same node. The load profile of job-1 has been plotted on the Fig. 1. Thereafter the load profile of job-2 has been added on with that of Job-1. Therefore what Figure 1 indicates is normal load diagrams. In Fig. 2 we have super-imposed (would be) free load curves of Job-1 and Job-2.

Let us say in Fig. 1 B1B2 is one sampling period in the kernel. In fact in Digital Unix this is 1s and in the most of other Unix and Linux this is 5s. We will say this distance is measured in sampling units.

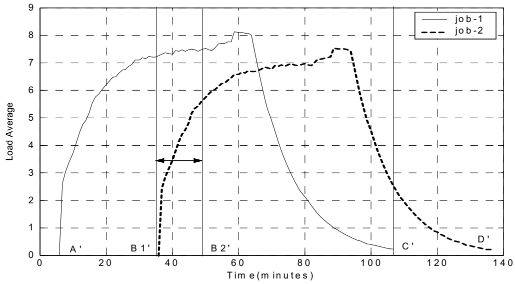


Fig. 2. Super imposed free load curves

Therefore

$B1B2 = sampling\ unit\ \ 1$

$B1'B2' = sampling\ \ unit\ \ (1 - \delta)$

We assume that sampling unit elongated by δ as a result of the background load. Further let us say the background load due to Job-1 at time t is $la1(t)$.

At time t, under the load $la1(t)$ 1 sample unit has been elongated to Δt_{exec_2}

Applying equation (2) for the task

$$\frac{\Delta t_{exec2\_normal}}{1 + la_1(t)} = \Delta t_{free\_load}$$

$$\frac{dy}{1 + la_1(y)} = dt \tag{3}$$

Therefore, $$\int_0^{t_{exe\_normal}} \frac{dy}{1 + la_1(y)} = \int_0^{t\_free} dt$$

Therefore, $$\frac{t_{exec2\_normal}}{1 + \dfrac{1}{t_{exec2\_normal}} \displaystyle\int_0^{t_{exec2\_normal}} z(t)dt} = t_{free\_load} \tag{4}$$

We can say from equation (3) that at time y for duration of dy

the new charges should be = $\dfrac{dy}{1 + la_1(y)} \times C$

Where C is the charge $ per CPU per seconds

We state that the client should be charged after reducing the delays due to background load. This means the client should be charged for the free load runtime. In general case, the total cost can be explained by the following equation.

The cost calculated based on the free load runtime= $t_{frre\_loadl} \times C$

Therefore $t_{frre\_loadl} \times C = (mF * C) * t_{exec2\_normal}$

From equation (4)

$$t_{frre\_loadl} = \frac{t_{exec2\_normal}}{1 + \frac{1}{t_{exec2\_normal}} \int_0^{t_{exec2\_normal}} z(t)dt}$$

$$\frac{t_{exec2\_normal}}{1 + \frac{1}{t_{exec2\_normal}} \int_0^{t_{exec2\_normal}} z(t)dt} \times C = (mF * C) * t_{exec2\_normal}$$

Therefore the resultant runtime multiplication factor,

$$\text{mF} = \frac{1}{1 + \frac{1}{t_{exec2\_normal}} \int_0^{t_{exec2\_normal}} z(t)dt} \qquad (5)$$

The existing Unix/Linux has the sampling rate of 5s and therefore the load average has been reported every 5s.

Let us assume for the period of 5s, error multiplication factor is stable.
Therefore for any 5s duration, new cost =
Cost calculated based on the free load runtime =

$$\int_0^5 \frac{dy}{1 + la_1(y)} \times C = 5 * mF_r \times C$$

Since there is no change of load average over 5s, $l_{a1(y)}$ is a constant.

$$l_{a1(y)} = \frac{1}{t_{exec}} \int_0^{t_{exec}} z(t)dt = \frac{1}{5} \int_0^5 z(t)dt$$

Therefore runtime multiplication factor $mF_r = \dfrac{1}{1 + \frac{1}{5} \int_0^5 z_r(t)dt}$ (6)

Therefore from the first 5s sampling to the Nth 5s sampling the runtime multiplication factors can be calculated as

$$mF_{r1}, mF_{r1}, mF_{r3}, \ldots\ldots. mF_{rn}$$

The Calculation Process for a correction factor for load average dependency:

The load signal or load average of a job at a particular time is an important factor for charging. In general load average 1 means 1 task is scheduled and load average 2 means 2 tasks are scheduled. As far as the supplier is concerned there should be an increase in charging when he increases the number of task on schedule.

Let us say our normal computer charge is C $/CPU/load average/second or C $/CPU/task/second. As far as a job is concerned if scheduled, both per load average and task depict the same meaning.

If we consider the total runtime then the resultant Load factor, Lf = average of load signal during the runtime.

Therefore $$Lf = l_a = \frac{1}{t_{exec}} \int_0^{t_{exec}} z(t)dt \qquad (7)$$

If we consider a certain 5s duration,

then average of load signal during the runtime $l_{a(5r)} = \frac{1}{5}\int_0^5 z(t)dt$

Load factor = $$Lf_r = \frac{1}{5}\int_0^5 z_{r1}(t)dt \qquad (8)$$

Therefore from the first 5s sampling to the Nth 5s sampling the Load factor can be calculated as

$Lf_1, Lf_2, Lf_3, \ldots\ldots. Lf_n$

Therefore at a certain 5s duration the correction factor = $Lf_r * mF_r$

At a certain 5s duration corrected computer charge = $C * (Lf_r * mF_r)$

The total correction factor = $Lf * mF$

Therefore corrected computer power charge= $C * (Lf * mF)$

5 Experimental Methodology and Results

Experiment 1:

In the first part of the experiment, the job 1 has been submitted to account U_1, and let the job run until it finishes. Then in the second part the job 1 has been submitted to U_1, and job 2 has been submitted to U_2 after time of T_{delay}. Thereafter in the 3rd part the job 1 has been submitted to U_1, and job 2 has been submitted to U_2 after time of T_{delay1} and job 3 has been submitted to U_3 after time of T_{delay2}. In all 3 occasions, the client job is job 1. The job 2 and job 3 are background jobs. Once the job 1 finishes, calculate the cost1 according to the runtime and C $/CPU/s. Thereafter using the multiplication correction factor in section 4 the new cost2 have been calculated. The costs have been compared. Please refer to figure 3 for details of actual runtimes and load averages.

Experiment 2:

Take 4 pcs in the middleware grid and submit jobs in the following manner

pc1- Job-A, background1: pc2- Job-A, background2: pc3- Job-A, background3: pc4- Job-A, background4: Then calculate the cost of the Job-A in each occasion.

6 Calculations

Experiment 1: we have used 1 pc. The C $/CPU/load average/second = 2 grid $

Experiment 1 part 1:

There is no background load, therefore from equation 6 the resultant mF=1

From equation 7 and Figure 3.1, resultant Lf = 0.77.

adjusted C = 2 * mF * Lf = 2 * 1 * 0.77 = 1.54

cost1= runtime*adjusted C = 850 * 1.54 = 1309 grid $

cost2 without correction = runtime * C=850 * 2 = 1700 grid $

The Figure 3.2 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.

Experiment 1 part 2:

From equation 6 and Figure 3.3, the resultant mF = 0.58.

From equation 7 and Figure 3.3, the resultant Lf = 0.83.

The adjusted C = 2 * mF * Lf = 2 * 0.58 * 0.83 = 0.9628

The cost1 = runtime * adjusted C = 1100 * 0.9628 = 1059.08 grid $

The cost2 without correction = runtime * C=1100*2 =2200 grid $

The Figure 3.4 shows the calculated profiles of mF and Lf using equations 6 and 8 every 5s.

Experiment 1 part 3:

From equation 6 and Figure 3.5 the resultant mF = 0.42.

From equation 7 and Figure 3.5 The resultant Lf = 0.86.

The adjusted C = 2 * mF * Lf = 2 * 0.42 * 0.86 = 0.7224

The cost1 = runtime * adjusted C = 1380 * 0.7224 = 996.91 grid $

The cost2 without correction = runtime * C=1380*2=2760 grid $

The Figure 3.6 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.

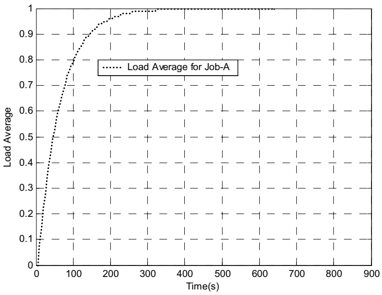


Fig. 3.1. of experiment 1.1

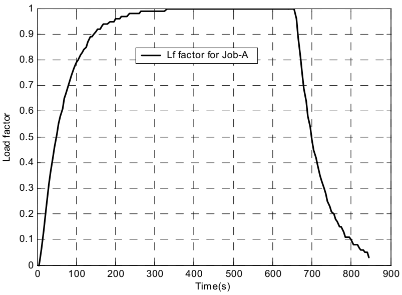


Fig. 3.2. of experiment 1.1

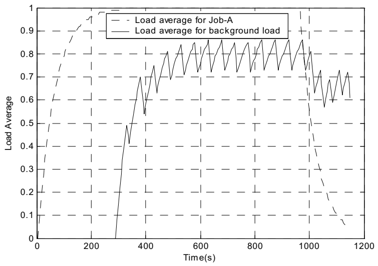


Fig. 3.3. of experiment 1.2

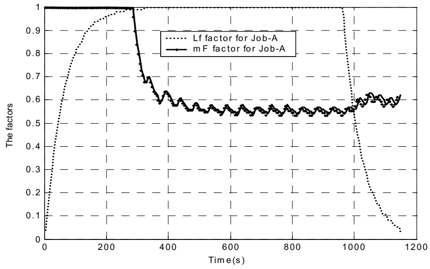


Fig. 3.4. of experiment 1.2

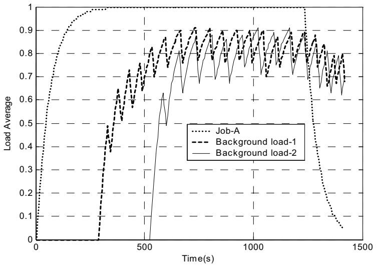


Fig. 3.5. of experiment 1.3

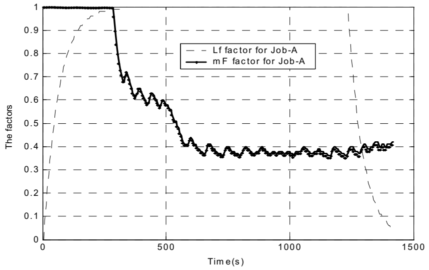


Fig. 3.6. of experiment 1.3

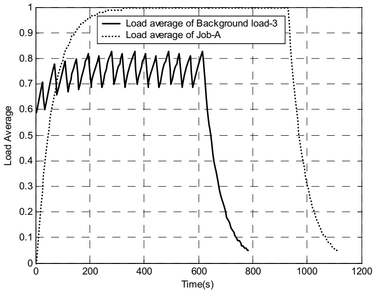


Fig. 4.5. of experiment 2.3

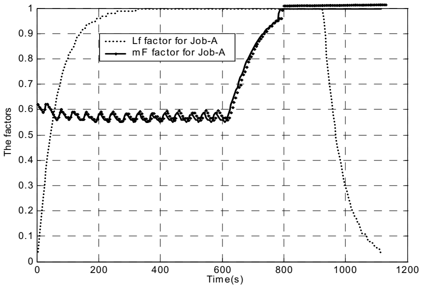


Fig. 4.6. of experiment 2.3

Experiment 2: we have used 4 pcs. The C1 \$/CPU/load average/second = 2 grid \$, The C2 \$/CPU/load average/second = 2.2 grid \$, The C3 \$/CPU/load average/second = 2.5 grid \$, The C4 \$/CPU/load average/second = 2.8 grid \$
Experiment 2.1:
The cost1 = runtime * adjusted C = 900 * 1.092 = 982.8 grid \$
The cost2 without correction = runtime * C= 900 * 2 = 1800 grid \$
The Figure 4.2 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.

Experiment 2.2:
The cost1 = runtime * adjusted C = 1000 * 1.2584 =1258.4 grid $
The cost2 without correction = runtime * C=1000 * 2.2 = 2200 grid $
The Figure 4.4 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.
Experiment 2.3:
The cost1 = runtime * adjusted C = 1200 * 1.2505 = 1500.6 grid $
The cost2 without correction = runtime * C=1200 * 2.5 = 3000 grid $
The Figure 4.6 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.
Experiment 2.4:
The cost1 = runtime * adjusted C = 1270 * 1.4336 = 1820.672 grid $
The cost2 without correction = runtime * C=1270 * 2.8= 3556 grid $
The Figure 4.8 shows the calculated profiles of mF and Lf using equations 5 and 8 every 5s.

7 Conclusion and Further Work

The multiplication factor mF and the load factor Lf have been derived considering the influence of background load on a particular job. In fact some background load always exists as some administrative jobs run as "root". In the experiment 1 we have demonstrated that with the change of background load mF is ever changing. The Lf too would change over time. In the experiment 2 we have submitted job-A to 4 pcs of the "Middleware grid" under different background load conditions.

In the experiment 1 we have first lounged job-A. The experiment has been repeated 3 times under different background loads. The total cost for job-A has been calculated and compared under experiment 1, section 6. In part 1 the runtime of Job-A is 850s sans any background load means mF=1. As the load changes the resultant Lf = 0.77. After considering the factors, the cost1 = 1309 grid $. But otherwise the cost2 becomes 1700 grid $. In part 2 with 1 background load resultant Lf = 0.83 and resultant mF = 0.58. The cost1 = 1059 grid $ and cost2 = 2200 grid $. In part 3 with 1 background load resultant Lf = 0.86 and resultant mF = 0.42. The cost1 = 996.9 grid $ and cost2 = 2760 grid $.

In part 1 as Job-A runs under free load mF=1. The difference between cost1 and cost2 is 391 grid $. This means one would have unfairly paid 391 grid $ extra, if the previous flat charging mechanism would have been used. But in part 2 with 1 background load this difference has increased to 1141 grid $. In part 3 with 2 background loads the difference has further increased to 1763 grid $. In part 2 and 3 as the background load increases the difference between cost1 and cost2 increases and such increments in difference further reinforce the necessity for a fairer charging mechanism.

In the experiment 2 we have run job-A in 4 different computers under different background load conditions. The charging of each computer differs. They charge 2, 2.2, 2.5, 2.8 grid $ respectively. Using the experimental results in section 5, the total running costs have been separately calculated and compared under experiment 2, section 6. In 2.1 with background load-1 resultant Lf = 0.78 and resultant mF = 0.70. The cost1 = 982.8 grid $ and cost2 = 1800 grid $. In 2.2 with background load-2 resultant Lf = 0.80 and resultant mF = 0.65. The cost1 = 1258.4 grid $ and cost2 = 2200 grid $. In 2.3 with background load-3 resultant Lf = 0.82 and resultant mF =

0.61. The cost1 = 1500.6 grid $ and cost2 = 3000 grid $. In 2.4 with background load-4 resultant Lf = 0.80 and resultant mF = 0.64. The cost1 = 1820.67 grid $ and cost2 = 3556 grid $. The least costs 982.8 grid $ is for pc1, yet had we used the flat charging mechanism it would have been 1800 grid $.

In a previous experiment [8] we have performed the prediction of the load profiles/ runtimes using free load profile of a particular job. The prediction results have been hence used for scheduling the jobs in the grids. In our future work we would be able to predict the set of costs for a particular job in the grid and that information too would be useful for better scheduling in the grid.

References

1. Buyya, R, Abramson, D. Giddy, J and Stockinger, H. "Economic Models for Resource Management and Scheduling in Grid Computing", Journal of Concurrency: Practice and Experience, Grid computing special issue 14/13-15, 2002, pp 1507 - 1542.
2. Abramson, D, Buuya, R. and Giddy, J. "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker", Future Generation Computer Systems. Volume 18, Issue 8, Oct-2002.
3. Abramson, D, Lewis, A. and Peachy, T., "Nimrod/O: A Tool for Automatic Design Optimization", The 4th International Conference on Algorithms & Architectures for Parallel Processing (ICA3PP 2000), Hong Kong, 11 - 13 December 2000.
4. C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications," presented at Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 11), Edinburgh, Scotland, 2002.
5. P. A. Dinda, "On line Prediction of Running time of Tasks", Journal of Cluster Computing, 5(2002).
6. Sena Seneviratne and David Levy 2004: Improving the Measurement of the Load Signal Through More Appropriate Sampling Rate, Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04), Las Vegas, Nevada, USA.
7. Neil J. Gunther, "Analyzing Computer Performance". Publisher: Springer-Verlag GmbH
8. Sena Seneviratne, David Levy "Enhanced Host Load Prediction by Division of User Load Signal for Grid Computing" submitted to Journal of Cluster Computing.

Host Load Prediction for Grid Computing Using Free Load Profiles

Sena Seneviratne and David Levy

School of Electrical and Information Engineering,
The University of Sydney, Sydney, NSW, Australia
{auntvini, dlevy}@ee.usyd.edu.au

Abstract. In Order to increase the overall performance, we have studied methods for improving load prediction, which would help improve load balancing in the Grid. Current software designed to handle distributed applications does focus on the problem of forecasting the computer's future load. The UNIX five-second-host load has been collected and used to predict the host load, but the solution of forecasting can be further improved if CPU historical load data had been collected separately for each login user. Another important aspect of historical data collection is that before submission to the grid, the user separates his HPC program into sizable parallel programs and test runs them supposedly on load free computers. This means the user can obtain the load profile of the parallel program on a load free computer together with other important information. Once the free load profile is known, load behaviour of a job under certain variable background load conditions can be predicted. Thus the forecast can be performed for each user before adding the weighted values towards the final solution of prediction. In this paper we have proved that load prediction using free load profiles provided better results. In fact once the user based load data are collected, the forecasting is somewhat like that of the Stock market.

1 Introduction

In a multi user host computer to which more than one user can submit their jobs, the applications are in active competition with unknown background workloads introduced by other users. Varying CPU load has a significant effect on the running time of CPU-bound applications. Indeed, for certain types of applications the running time of a computer-bound task is linearly proportional to the average CPU load it encountered during the execution [1, 5]. The focus of this paper is predicting the CPU load of shared computing resource. Our focus is on a grid of computers. Our contribution is to introduce a new approach for the collection of historical data, which is based on individual users in the system. That is to calculate and store the load averages against the individual users separately. Important information is the composition of the users who are logged in at a particular time. Then, we will introduce a new prediction methodology, which is based on free load profiles, which were obtained beforehand. Our work though focuses a grid of computer networks and has been based on the previous work done in the field of CPU load prediction by Dinda and O'Hallaron

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 336 – 344, 2005.

[1, 2, 3, 4, 5]. Further we have been quite influenced by the previous works of Byoung-Dai Lee and Jennifer M. Schopt [9]. At last but not least we have studied lessons from Rich Wolkski's work including Network Weather Service [3, 10].

2 Problem Statement

We argue that the individual CPU load can be predicted using free load profiles of that particular job and historical CPU load values collected for that particular individual user. The final CPU load prediction would be calculated from the weighted individual user load predictions. In this paper our aim is to prove that the free load profile technique is always give better predictions than previous such efforts in total load signal predicting [1] using only AR(16).

The important assumptions are that the set of application input parameters that can affect the application run time is known. We do not consider parallel applications with run times that are non deterministic or that depends on the distribution of the input data. Ex. Iterative Jacobi Computation [9].

3 Analysis of the Load Signal

3.1 Theory of the Load Signal

The existing Unix/Linux host load average or load signal has been calculated in the kernel after sampling the task-list run-queue of the CPU. In this case the processes that are runnable and therefore waiting in the queue, executing on CPU, or suspended (uninterruptible) waiting for some other external condition, have been counted for further calculation [7].

$$\text{Tasks (n) =TASK\_RUNNING (and Runnable) +} \atop \text{TASK\_UNINTERRUPTIBLE} \tag{A}$$

This can be written in more conventional mathematical notation as:

$$load(t) = load(t-1) * e^{-\sigma / r} + n(t) * (1 - e^{-\sigma / r}) . \tag{B}$$

If $load(t)$ is the current estimate of the load average (signal), $load(t-1)$ is the estimation of the load average from the previous sample, and $n(t)$ is the number of currently active Unix/Linux processes. The sampling period is σ and reporting period is r .

3.2 The Experimental Analysis of the Division of Load Signal

After making some necessary changes to the kernel code, the load signal can be collected separately for each login user. In a previous experiment [6, 8], we have proved that at any time t the collective divisible load, which is the resultant load signal, is equal to the total load signal as demonstrated by Figures 1 and 2.

i.e. Total $U = U1 + U2 + U3 + ... + Ur + ... + Un$

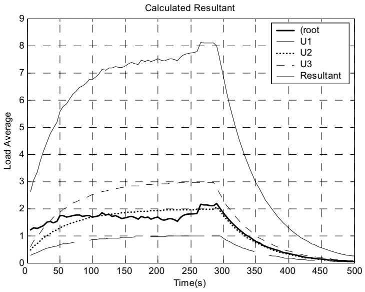


Fig. 1

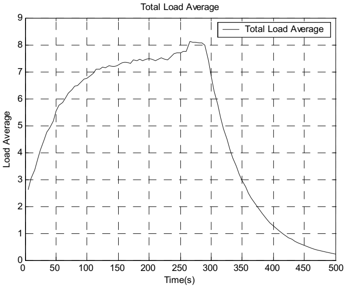


Fig. 2

4 Theory of Load Signal Prediction

4.1 The Load Signal and the Run Time of a Task

Dinda and O'Halloren has related the running time of a task [5], t_{nom} to the average load it experiences while it runs using the following continuous time model:

$$\frac{texec}{1+\frac{1}{texec}\int_{0}^{texec} z(t)dt} = tnom \tag{1}$$

Here $z(t)$ is the load signal, shifted such that $z(0)$ is the value of the signal at the current time, t_{now}. The t_{nom} is free load runtime and t_{exec} is normal runtime. It has been assumed that the majority of the workload runs at similar priority.

As the average background load, $la(t) = \frac{1}{t_{exec}} \int_{0}^{t_{exec}} z(t)dt$

Therefore, $texec = tnom * \{1 + la(t)\}$

$$\text{Thus,} \quad \frac{texec}{1+la(t)} = tnom \tag{2}$$

4.2 Prediction of Load Signal

We argue that the calculation of the predicted resultant CPU load is better than predicting the CPU load based on the historical total CPU load values which have been the traditional way of the prediction of the load average.

Suppose the users $U1$, $U2$, $U3$ and $U4$, ... Un are users at a particular time t Then at that time $t+1$ where times step is 5s.

$$\text{Final predicted resultant CPU load} = \sum_{r=1}^{n} (W_r) \text{ Predicted CPU load of } U_r + \text{CORRECTION} \qquad \text{(C)}$$

In the Separate Load Average case the historical values of $U_1, U_2, U_3, \ldots U_n,$ and root have been included in the prediction methodology. After predicting them separately, the resultant would be calculated as $0 \le W_r \le 1$. In general as at the time of prediction if $U_1, U_2, U_3, \ldots\ldots\ldots\ldots U_n,$ is 0 the historical data of such a component or components would not be taken into consideration for prediction assuming that they would not revive soon. This means any of the weighted values $W_{1,2,3,4 or 5 \ldots\ldots\ldots n}$ =0. If $U_1, U_2, U_3, \ldots\ldots\ldots\ldots U_n,$ NOT 0 then $W_{1,2,3,4 or 5 \ldots\ldots\ldots n}$ =1.

4.3 Prediction Methodology

4.3.1 Load Free Profile Matching

The load free profile of the application is the most important part of our prediction strategy. The following steps would show that how we can predict the future load, using load free profile of the application.

The Calculation Process:

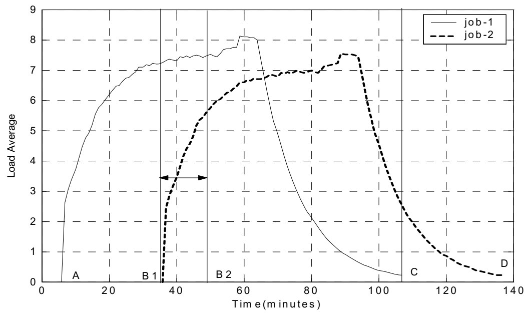


Fig. 3. Superimposed load curves

The Job-1 has been submitted to the grid node. After a time delay of AB1, Job-2 has been submitted to the same node. The load profile of job-1 when run on a load free machine has been plotted on the Figure 3. Thereafter the load free profile of job-2 has been superimposed with that of Job-1. Therefore what Figure 3 indicates is superimposed diagrams. Let us say in Figure 3, B1B2 is one sampling period in the kernel. In fact in Digital Unix this is 1s and in the most of other Unix and Linux this is 5s. We will say this distance is measured in sampling units.

Therefore

$$B1B2 = sampling\ unit\ \ 1$$

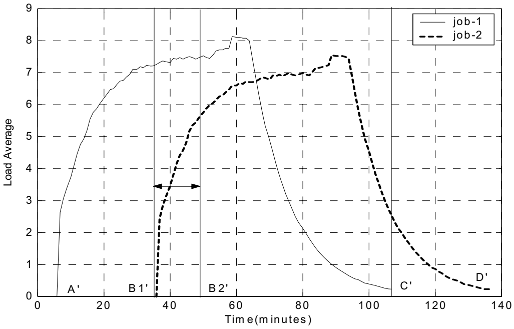


Fig. 4. Normal running under background load

In Fig. 4, we have assumed and plotted the might be plot of Job-1 and Job-2.

$B1'B2' = sampling\ unit\ (1+\delta)$

We assume that sampling unit elongated by δ as a result of the background load. Let us say the background load due to Job-1 at time t is $la1(t)$.

At time t , under the load $la1(t)$ 1 sample unit has been elongated to $\Delta texec\_2$ Applying equation (2) of section 4 for the task

$$\Delta texec\_2 = \Delta tsamp\_unit * \{1 + la1(t)\} \tag{3}$$

$dy = dt\{1 + la1(t)\}$

Our main assumption is that $la1(t)$ is same over 1 sampling unit and over $sampling\ unit\ (1+\delta)$

Therefore we would do the following integration.

$$\int_0^{texec} dy = \int_0^{tnorm} \{1 + la1(t)\}dt \tag{4}$$

Where t_{nom} is free load runtime and t_{exec} is normal runtime

$$texec = tnom + \int_0^{tnorm} la1(t)dt \tag{5}$$

From Figure 3 $Area\ of\ la1(t)\ from\ B1\ to\ C = \int_0^{tnorm} la1(t)dt$

Therefore $$t2\_exec = texec = tnom + tnom * la\_normal\_average\_load \tag{6}$$

That is under free load conditions.

$la\_normal\_average\_load = average\ of\ load\ average\ over\ time\ tnorm$

Under the influence of load of Job-1, B1'C' section has new $t2\_{exec} > t2\_{norm}$

Anyway still the maximum load average of Job-2 is the same. As a result of the additional load of Job-1 B1C has been elongated to become B1'C'.

The same can be said about the Job-1 as well.

The error generated due to our major assumption would be proven very small. Currently Unix/Linux uses sampling period of 5s. If we reduce it the error generated can be further reduced.

Usually, we expect up to 3 users would submit jobs to a node. In generally 3 users mean 3 jobs and 3 jobs means the maximum load average around 3. It is generally accepted and we too have empirically found that if the load average exceeds beyond 3, it would hinder performance.

5 Experimental Methodology and Results

Our infrastructure hardware consists of Pentium-3 hosts, which are part of the "Middleware Grid" of School of Electrical Engineering, University of Sydney. Currently "Middleware grid" consists of 27 Intel Pentium-3 Linux workstations. In each pc, it has been configured to measure the load signals separately, that is in accordance with the log in user.

Experiment 1: Firstly the job 1 has been submitted to account $U1,$ and job 2 has been submitted to $U2$ after time of $Tdelay$. The moments before the job 2 is submitted, the predicted run times of job 1 and job 2 has been calculated using the methodologies discussed in the section 4. Thereafter they have been compared against the actual runtimes of job 1 and job 2. By changing the value of $Tdelay$, the experiment has been repeated thrice.

Experiment 2: Firstly the job 1 has been submitted to $U1,$ and job 2 has been submitted to $U2$ after a time delay of $Tdelay1$. The moments before the job 2 is submitted, the predicted run times of job 1 and job 2 has been calculated using the methodologies discussed in the section 4. Thereafter the job 3 has been submitted to $U3$ after a time delay of $Tdelay2$. The moments before the job 3 is submitted, the predicted run times of job 1, job 2 and job 3 have been calculated using the methodologies discussed in the section 4. Thereafter they have been compared against the actual runtimes of job 1, job 2 and job 3. By changing the value of $Tdelay1$ and $Tdelay2$ the experiment has been repeated thrice.

Experiment 3: In this experiment, 10 computers of the middleware grid have been used. Firstly all 10 computers have been submitted with 10 different HPC jobs. Say the pcs are $p1,\ p2,\ p3, \ldots\ldots\ldots. p10$. The name of the grid user account is user5a. After $Tdelay1$ s 10 identical HPC jobs have been submitted the grid user account user5b. This time the jobs are to be done a particular task. They are Bio-informatics applications that analyses certain Gene expressions. We have borrowed them from *High Performance Computing Support Unit* University of New South Wales. The jobs are considered to be independent. Please refer to the table 2 for results.

Table 1. The Job submission and results of Experiment 3

Computer name	$p5$	$p6$	$p7$	$p8$	$p9$	$p10$
$user5a, Load\ \ free(J5-10)s$	2058	2175	2605	2685	2725	3035
$Delay \approx\ Tdelay$	298	308	315	930	1265	1300
$user5b, Load\ \ freeJ0$	885	885	885	885	885	885
$\Pr edicted\ \ runtime\ \ u'5a(s)$	2488	2688	3277	3357	3497	3671
$Measured\ \ runtime\ \ u'5a(s)$	2405	2518	3175	3220	3520	3750
$\Pr edicted\ \ runtime\ \ u'5b(s)$	1380	1442	1454	1458	1601	1651
$Measured\ \ runtime\ \ u'5b(s)$	1322	1400	1580	1445	1685	1700

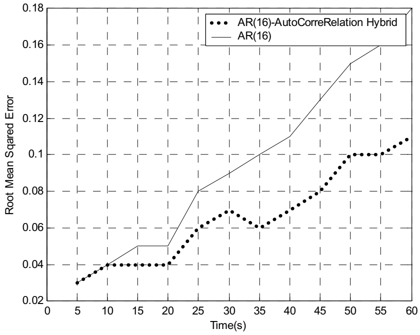


Fig. 5

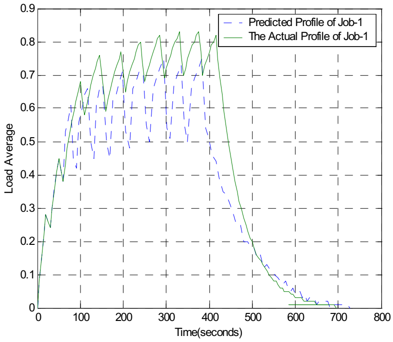


Fig. 6. Case 1.1a-Exp1

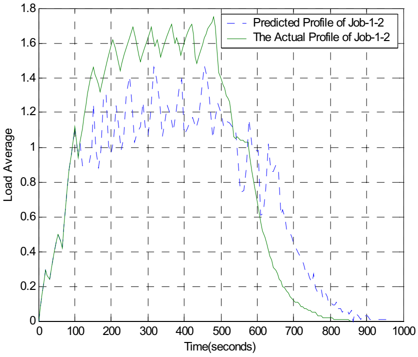


Fig. 6. Case 2.2a-Exp2

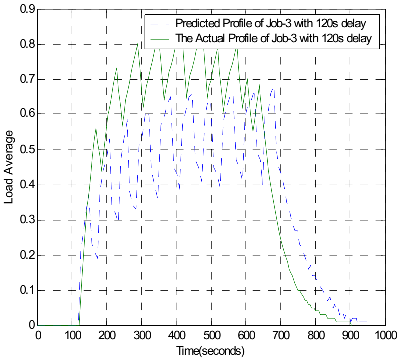


Fig. 6. Case 2.2b-Exp2

6 Conclusion and Further Work

The predictor we used is based on Free Load Profiles and the Division of Load Signal, which has been introduced to measure the load separately. Further as a result of the division of load signals our predictor will be able to weight the individual predicted results in accordance with a selected algorithm.

As Dinda has performed load prediction using AR(16) so that we have compared it with ours. In Figure 5 we have presented the plot of RMS error of AR(16) over 100 samples. In the same diagram we have plotted the case of the Hybrid method which is slightly better than AR(16). Our Hybrid method consists of curve matching technique and AR(16). When there is a repetition in the submission of the job it switches on to curve matching. In this comparison [8] we have predicted only 60s into the future, yet you find an ever-increasing RMS error. The problem with AR(16) is as the lead time increases margin of error too increases. We have shown that our new approach had helped to reduce the margin of error considerably. Apart from that our new approach can predict the total runtime of an application. In fact this is the main advantage over the existing AR(16). According to the results of current experiments we would predict up to 3600s (1 hour) with about 5% error.

In the experiment 1 we have first lounged job 1. After Tdealy1, we predicted the profile of job 2 before lounging it. Thereafter the actual load profile has been plotted using collected load signals. In Figure 6 Case1.1a shows the predicted and measured load curves of job1. In the complete diagrams Figure 6: Case1.1a, Case1.2a and Case1.3a depict the behavior of job1. The complete diagrams Figure 6: Case1.1b, Case1.2b and Case1.3b depict the behavior of job2. In all 3 cases the average prediction error in runtime is less than 5%.

In the experiment 2 we have first lounged job 1. After Tdealy1 we predicted the profile of job 2 before lounging job2. Thereafter after Tdealy2 predicted the profile of job 3 and then lounged. The actual load profile has been drawn using collected load signals. In Figure 6 Case2.2a shows the predicted and measured load curves of job1 and job 2. In Figure 6 Case2.2b shows the predicted and measured load curves of job 3. The complete diagrams of Figure 6: Case2.1a, Case2.2a and Case2.3a depict the combine behaviors of job 1 and job 2. The complete diagrams of Figure 6: Case2.1b, Case2.2b and Case2.3b depict the behavior of job 3. In all 3 cases the average prediction error in runtime is less than 10%.

In the experiment 3 we have tested our prediction methodology in the grid environment. We have used the 10 computers of the “Middleware grid”. It has been shown that most suitable 5 pcs were predicted as far as runtime is concerned. The table 1 shows the actual and predicted runtimes of the first set of jobs and the 2nd set of job. In this manner we will be able to predict the shortest runtime for the Bioinformatics applications that analyses certain Gene expressions.

The response time of the predictor itself is an important factor we have taken into consideration. Another factor is the execution time of the predictor. We have observed that although both these factors are somewhat higher than the AR (16), they are within tolerable range.

Improving the accuracy of the predictor is the major focus of this paper. We also must keep the response time and execution time of the predictor within an acceptable range. In fact they should be well less than the sampling period 5s. Future work will address the usage of prediction techniques for scheduling jobs in the “middleware grid”.

References

1. Dinda P.A. and O’ Hallaron, D.R. “Host Load Prediction Using Linear Models, Journal of Cluster Computing”, 3 (2000).
2. Dinda P.A. and D. R. O’ Hallaron:2000a “Realistic CPU Workloads Through Host Load Trace Playback”. In: Proc. Of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR2000), Vol 1915 of Lecture Notes in Computer Science, Rochester, New York, pp. 246-259.
3. R. Wolski, N. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Journal of Future Generation Computing Systems*, pp. 757-768, 1998.
4. C. Liu, L. Yang, I. Foster, and D. Angulo, "Design and Evaluation of a Resource Selection Framework for Grid Applications," presented at Proceedings of the 11th IEEE International Symposium on High-Performance Distributed Computing (HPDC 11), Edinburgh, Scotland, 2002.
5. P. A. Dinda, "On line Prediction of Running time of Tasks”, Journal of Cluster Computing, 5(2002).
6. Sena Seneviratne and David Levy 2004: Improving the Measurement of the Load Signal Through More Appropriate Sampling Rate, Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04), Las Vegas, Nevada, USA.
7. Neil J. Gunther, “Analyzing Computer Performance”. Publisher: Springer-Verlag GmbH
8. Sena Seneviratne, David Levy “Enhanced Host Load Prediction by Division of User Load Signal for Grid Computing” submitted to Journal of Cluster Computing.
9. Byoung-Dai Lee, Jennifer M. Schopf, “Run Time Prediction of Parallel Applications on Shared Enviorement” Proceedings of Cluster 2003, December 2003. Extended version is available at Argone National Laboratory Technical Report #ANL/MLS-P1088-0903, September 2003.
10. R. Wolski, "Dynamically Forecasting Network Performance Using the Network Weather Service," *Journal of Cluster Computing*, 1998.
11. Lingyun Yang, Jennifer M. Schopf, Ian Foster, “Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic environments”.

Active Link: Status Detection Mechanism for Distributed Service Based on Active Networks[*]

Zhan Tao[1], Zhou Xingshe[1], Liao Zhigang[2], and Chen Yan[2]

[1] Computer Science and Technology Department,
Northwest Polytechnical University, Xi'an 710072, China
[2] Institute of Computer Architecture & Network,
Xi'an Jiaotong University, Xi'an 710049, China

Abstract. For network service, it is obvious that "PUSH" method is more efficient than "PULL" method for bandwidth consuming. One problem for "PUSH" method is that the client is difficult to keep track with the status of server. Traditional polling method is bandwidth consuming and put much burden on server. Active link is an active network based service which builds a tree structure between clients and server. Different clients and service can share link information if they have the same middle nodes in the link path. This mechanism can reduce bandwidth consumption and burden of server.

1 Introduction

Internet is still expanding, as well as the type and number of services are increasing day by day. Many applications require clients to keep track with the server. So far, there are two popular way to achieve this: one is that client sends query timely to the server, this is called "PULL" method; the other way is that client wait silently for notification of changes from the server, this is called "PUSH" method. The difficulty for "PULL" method is to determine the interval of poll; if it is too long, it will lose freshness; if it is too short, it will bring heavy burden on network throughput. And both methods have a common shortage that one application cannot share information with other applications. The availability of service can be classified into two parts: the state of the service and the connectivity of the network. Active networking technology [1] provides activity for the network, it makes possibility to share the information between different applications. Active link is a service based on active network to achieve this goal. The mechanism has been successfully implemented on the ANTS 1.3.1[2].

2 Active Link

Active link provides an active method for status tracking between server and cache nodes. If a cache node intends to detect the state of server in traditional

[*] This paper is supported by the Nation Science Foundation of China (No.60173059).

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 345–350, 2005.

network, a typical way is to send a query message to detect whether the server is ok or not. The state can be divided into two parts: the state of network connection and the state of the server. When the server receives a query message, it will send a reply message. If the cache node receiving reply message, it means that the server is ok; if the cache node cannot receive a valid reply message within a given period, it will retry the above action for several times and if still no result back it can make a judge that the server is transiently out of service. If the cache node wants to trace the state of server it will send query message repeatedly to the server. This method has the following problems: (1)The major problem is bandwidth consumption and heavy burden on server. (2) If no reply message sending back, it's difficult for cache node to identify the fault is caused by the network connection or the server.

Active networking technology provides a way to solve these problems. In active link, the status tracking can be achieved by a dynamic service and it can be shared among different cache nodes, thus it can reduce network bandwidth and relieve the server from heavy burden of replying to all cache nodes.

2.1 Principles

Assume a cache node intends to keep track with server's state. Firstly, it sends request capsule to the server. As well as server receiving the request, it sends back a reply capsule; each node on the route from server to the cache node will activate the active link service, which will monitor the next and the previous node to see whether the neighbor is reachable. As shown in figure 1, this result in a dual link from the server to the cache node and a group of node pairs. In figure 1, it has generated three node pairs: (CacheNode, AN1), (AN1, AN2) and (AN2, Server). Each node in the pair will inspect the availability of the other. The availability of the server includes connectivity and service availability, and as to other nodes, it just includes connectivity. The normal state is that all nodes in the link connected together and the service is running. If the service on the server is down, the adjacent node, in this case AN2, will aware of the failure and notify the client along the link. AN2, AN1 and cache node will release the resource and may do their predefined procedure such as sending alert to web user. If the connectivity is broken, nodes in the same link pair cannot reach each other. Both of them will send a connection fail message along the two sublinks until reaching the cache node or server. All nodes that receive the message will release the resource. The client will fall in a temporary state and find another route to the server. In figure 1, assuming that pair (AN1, AN2) is broken, AN2 will release the resource and send a message to the server; the server then releases the resource too. AN1 does the similar action but sends a message to cache node; the node received the message then convert to a temporary state. It will try to find another route then; if it finds one, it resume to available state otherwise it will convert to connection fail state and it will send message to the server timely to detect the service.

Multiple cache nodes can share link in the same route end to end. As shown in figure 3, the route from C1 to S is [C1, R1, R5, S] and the route from C2

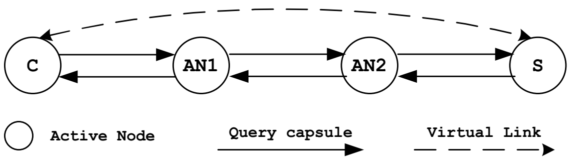


Fig. 1. Each node keeps track with adjacent nodes, and makes a virtual link between server and cache node

to S is [C2, R1, R5, S]; they can share pair (R1, R5) and (R5, S). R1 and R5, as they are active routers, can easily record share information and save network bandwidth especially in large scale networks.

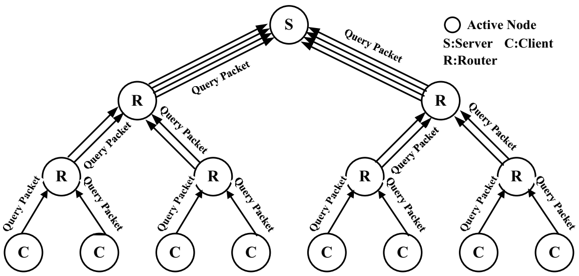


Fig. 2. Traditional query method will accumulate the flush; the server would burden heavily when cache nodes number increasing

Since every node in the route just keeps track with adjacent nodes, it is also easy to reuse them with different servers or different services running on the same server.

The one major advantage of active link is to share information among different links. The server's state is kept not only in cache node but also spread all over network; it becomes possible to share information among different cache nodes.

2.2 Status Tracking Information Sharing for One Server

The level of share is determined by (1) network topology and (2) deployment of the related server. Tree is a usual topology applied in Internet. For convenience, full binary tree is considered as network topology in following discussion. The server is the root node; all cache nodes make up the leaves and middle nodes (usually router) make the branch. Let T be a full binary tree. (From now on, the tree is always a full binary tree.) Let $m + 1$ be the levels of T and let n_j be the total node number in the level $j(0 \leq j \leq m)$. Then n_j should be: $n_j = 2^j$.

Let P denotes the amount of query packet that a cache node sent per second and let S denotes the size of a query packet. We assume the response packet has the same size with the query packet. In traditional way, the server will receive

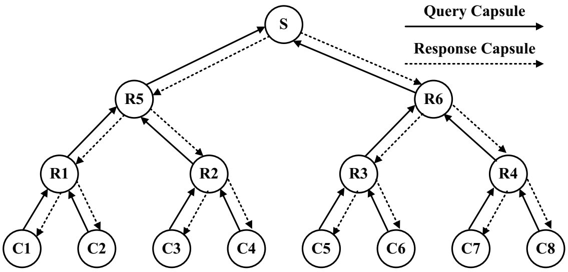


Fig. 3. By active link, the query and response are distributed to each active router; so the server has a fixed burden no matter the number of cache nodes

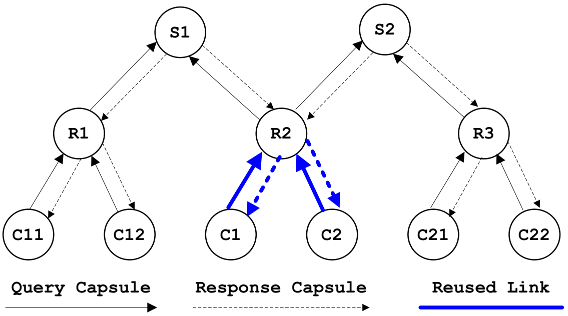


Fig. 4. Virtual link information can be shared for different service. The links colored blue are shared by two different servers.

query packets from all cache nodes and then send response packet to respective node. So the throughput in the server B_s is

$$B_s = 2PSn_m = 2^{m+1}PS \tag{1}$$

To compare, for active link, the throughput in the server $B_{s'}$ is

$$B_{s'} = 2PSn_1 = 4PS \tag{2}$$

It is obvious that Bs' is a constant whereas Bs will increase greatly if the levels of tree increase. The reason for this is that each middle node partake the total task. And all its children will share each middle node's information. Take the case of figure 3, the information of (R1, R5) can be shared by C1 and C2; and the information of (R5, S) can be shared by C1, C2, C3 and C4. The higher the middle node in the tree level, the more cache nodes can share the information from the node.

2.3 Information Share for Multi Server

Moreover, information can be shared among different servers. Figure 4 shows an example for sharing information between two services which deployed in different nodes. As figure 4 shows, C1 keeps track with S1 and S2 and these two links can share pair (C1, R2). The same thing is also occurred in node C2. It is obvious that the nearer the two servers are, the more the two link sets can share. To an extreme, the two servers are deployed in the same node and the clients can share all the connectivity information.

3 Implementation

We use ANTS (Active Network Transport System) [2][3] to implement a prototype of active link. There are two reasons to choose ANTS: one is that ANTS is one of the famous active networking runtime environments; the other is that it is implemented by JAVA, a promising language based on OOP (Object Oriented Programming).

In active link, each client tracks status of the server, and the routes of information tracking set up a virtual link.Figure 5 shows how a link establishing. It can be divided into three steps:

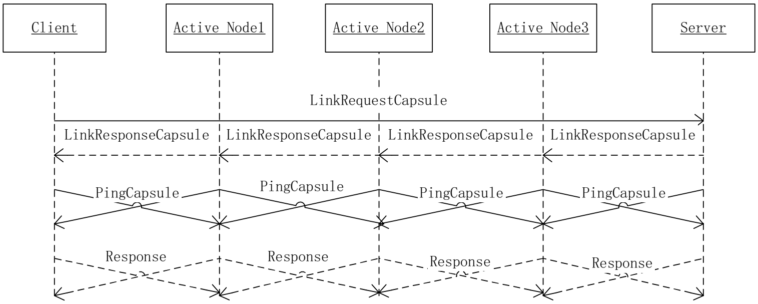


Fig. 5. The process of one link establishing

1. The cache node sends a RequestCapsule to server.
2. The server, if accepting the client request, sends a ResponseCapsule to the cache node.
3. Each node in the route from the cache to the server will invoke a 'ping' action by sending PingCapsule repeatedly to adjacent node(s) in the route. The node adjacent to the server will also send a PingServiceCapsule to detect the service's state.

After above steps the link will be established. The client should set the original link state to ok and do following works:

1. If all of ping actions, including the PingCapsule and the PingServiceCapsule, return true to each node, then the link is ok. otherwise,
2. If one or more ping action(s) fail(s), two neighboring nodes will receive no reply and they will send a DestroyCapsule to client and server reversely along the link respectively. Then each node in the link would release the resource allocated for the link.

3. When the client receives a DestroyCapsule, it sets its state to a temporary state. It will try to build another link to the server. If another link is available, it then changes its state to ok; otherwise it would change its state to failure.

4 Conclusion

Active link makes an effort on information share in active networks. Active link is an active network based service which builds a tree structure between clients and server. Different clients and service can share link information if they have the same middle nodes in the link path. This mechanism can reduce bandwidth consumption and burden of server.

References

1. David L Tennenhouse, Jonathan M Smith, W David Sincoskie et al.: A Survey of Active Network Research. IEEE Communication Magazine, Jan 1997, 35(1): 80-86.
2. David Wetheral.: Active Network Vision and Reality: Lesson from a capsule-based system. Proceedings of 17th ACM Symposium on Operating Systems Principles (SOSP '99), December 1999.
3. David Wetheral.: Service Introduction in Active Networks. PhD Thesis, Massachusetts Institute of Technology, April 1999.

Performance Monitoring for Distributed Service Oriented Grid Architecture

Liang Peng, Melvin Koh, Jie Song, and Simon See

Asia Pacific Science and Technology Center, Sun Microsystems Inc.,
Nanyang Center for Supercomputing and Visualization,
Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798
{pengliang, melvin, songjie, simon}@apstc.sun.com.sg

Abstract. Computational Grids provide an emerging highly distributed computing platform for scientific computing. Recently, service oriented architecture (SOA) is a trend of implementing software systems including Grid computing systems. SOA provides more flexibilities for Grid users at the service level. Since performance is still one of the major concerns in Grid environments, Grid service performance issues needs to be extensively investigated and studied. However, a lot of issues are still open to be explored and few work has been done on them.

In this paper, we propose a Grid service monitoring architecture for flexible monitoring on various Grid services. We implemented it for monitoring WSRF (Web Service Resource Framework) services in this paper. We show how the service oriented Grid monitor work with a simple example WSRF-compliant MathService. Moreover, the relationship of the monitor and Grid super scheduler is also analyzed. In this way, the scheduler may produce service performance oriented policies that ensure optimal quality of services for Grid applications.

Keywords: Grid Performance Monitoring, Grid Service Performance, Service Oriented Grid Architecture.

1 Introduction

In recently years, Grid computing is a fast developing technology as an approach to do high performance scientific and engineering computation. Grid performance evaluation and modeling, as an important supporting factor for Grid architecture, is still immature and not extensively explored. The performance information are usually not fully considered in Grid middleware design as well. The Grid middleware usually does not reveal or utilize performance information (e.g. performance model, evaluation results, etc) that can be used to improve the efficiency of Grid scheduling.

Scheduler oriented Grid performance evaluation is an approach that enables performance evaluation, monitoring, and modeling units closely cooperates with or even become a component of Grid scheduler so that the Grid scheduler is able to get more information about the performance of both computation resource and application and hence improve the scheduling policy for Grid jobs and ensure the quality of service.

Meanwhile, with the widespread emergence of Web services and service-oriented architecture [6] implementations in enterprise IT environments, service orientation is

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 351–356, 2005.

a trend for implementation of Grid architectures. In this environment, the Grid service performance problem can no longer be ignored.

In this paper, we propose a Grid service monitoring architecture for flexible monitoring on various Grid services. We implemented a prototype for monitoring WSRF (Web Service Resource Framework) services. We also show how the service oriented Grid monitor work with a simple example WSRF-compliant MathService. In addition, the relationship of the monitor and Grid super scheduler is also analyzed in order for the scheduler to produce service performance oriented policies that ensure optimal quality of services for Grid applications.

The remainder of this paper is organized as follows: section 2 introduces Grid service and performance in computational Grid environments and presents the Grid service performance monitoring architecture and its interaction with Grid super scheduler; The implementation details of our Grid service performance monitoring architecture and a simple example are introduced in section 3; Some related work are introduced in section 4 and finally we give a conclusion in section 5.

2 Service Oriented Grid Performance Architecture

2.1 Performance of Grid Services

The performance of Grid is not well defined, neither is the performance of Grid services. The obstacles are mainly the nature of Grid (heterogeneity, dynamism, wide distribution, etc) and these make traditional performance metrics not directly applicable to Grid environments.

In Grid environments, the following metrics can be considerred to monitor for Grid services:

- Availability: whether a Grid service is present and ready for immediate use.
- Throughput: how many requests can be serviced in a given time period.
- Latency (or response time): how much time elapses between the request and the response.
- Scalability: whether a Grid service is horizontally and vertically scalable.

2.2 Grid Service Performance Architecture

A Grid performance architecture consists of multiple components such as performance monitor, performance modeling, interfaces between performance components and Grid middleware (e.g. Grid super scheduler), etc. Figure 1 shows the monitoring components at the resource site. The Grid services located/running at the computing resource are monitored by local monitoring service called sensor. The sensor abstracts the lower level format of Grid services (e.g. WSRF services, OGSI/OGSA services, etc). When a Grid service monitoring request comes, the sensor invokes some local monitoring mechanisms to get the required information of the services. The information is then feed back to the requiring monitoring component. This is a "pull" mode of information retrieval. It can also be "push" mode, in which case the local sensor actively gets the monitoring data of Grid services and feed back to remote monitoring component even if

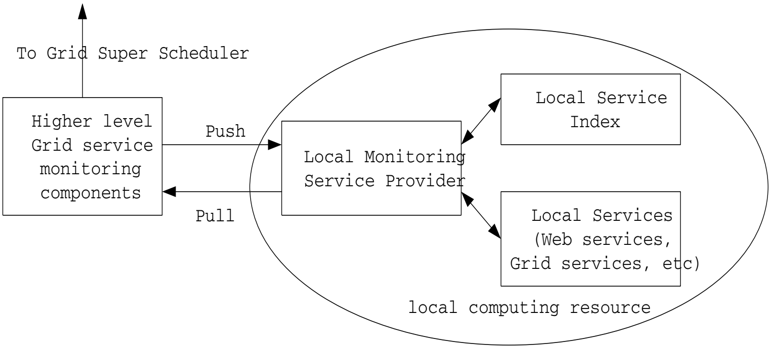

To Grid Super Scheduler
Higher level Grid service monitoring components
Push
Pull
Local Monitoring Service Provider
Local Service Index
Local Services (Web services, Grid services, etc)
local computing resource

Fig. 1. Service monitoring for Grid computing resources

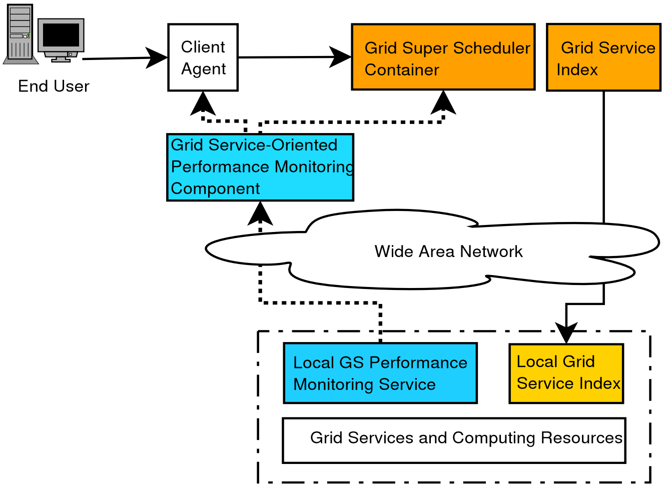

End User
Client Agent
Grid Super Scheduler Container
Grid Service Index
Grid Service-Oriented Performance Monitoring Component
Wide Area Network
Local GS Performance Monitoring Service
Local Grid Service Index
Grid Services and Computing Resources

Fig. 2. The Service Monitor and Grid Super Scheduler

there is no requirements. In practice, the implementation can be based on either mode, or the combination of both.

Grid service monitoring can be a relatively independent module. It can also a component that feeds back information to Grid super scheduler. Figure 2 shows the position of service monitor in Grid super scheduler architecture. The Grid user can monitor the performance of the Grid job, and the data collected by monitor can also be fed to performance modeling and performance tuning modules, which in turn affect the Grid performance evaluation. The Grid super scheduler takes the performance evaluation results into consideration, select the most appropriate resource, and distributes the job to the local scheduler of the selected resource.

In this scenario, the user submits the job to the computational Grid via his local agent. The local agent forwards the user's resource requirements to the super scheduler, which will interact with the virtual resource index to find the most suitable resource. Here the virtual resource index may contain the performance model (this can be achieved by, for example, analyzing historical benchmarking results) and corresponding parameters (e.g. hardware/software configuration) of the computing resources. The super scheduler may take the performance model into consideration and find the most appropriate resource for the user's job. The information of the selected resource will be sent back to the user's local agent, so that the local agent can submit the Grid job to the remote site with the selected resource. When the job is being executed, the job monitoring module is activated. The job status and some performance data are collected and sent back to the agent and the scheduler, so that the user is able to view the performance and status, while the super scheduler is able to update or adjust the existing performance model or policy accordingly. If possible, the super schedule might also do performance tuning and optimization by migrating the job to some other computing resources in case of the monitored performance is not as good as expected (this can happen especially for QoS guarantee). In this scenario, the performance issues are considered by the scheduler in almost every step of the job submission and execution. This ensures that the job scheduling policies are performance-aware and the overall Grid performance is optimal.

3 The Implementation and Example

3.1 WBEM and CIM

In implementation of the Grid monitoring service, we utilize the Web-Based Enterprise Management (WBEM [2]). WBEM is a set of Internet standards which gives the ability to interconnect between different management standards and environments. WBEM allows to manage both software (OS, applications) and hardware (computers, network devices) by creating a common player which unifies and simplifies management through WBEM compliant applications. There are a lot of WBEM implementations by different vendors including WBEMServices (Sun Microsystems), Pegasus (The Open Group), OpenWBEM (Caldera), SBLIM (IBM), WMI (Microsoft), etc.

The data specification model for WBEM is the Common Information Model [1], which is an objected oriented description of information. WBEM uses XML/CIM language for encoding CIM objects. The communication protocol for WBEM is HTTP.

The sensor we mentioned is a server called CIM Object Manager, which is the central element of the WBEM environment architecture.

3.2 Service Monitoring Based on WBEM/CIM

We implemented a service provider called "Grid service monitor" and plugged it into the WBEM/CIM architecture. As for the triggering mechanism, we use WS-Notification specified by WSRF. Figure 3 illustrates the Grid service monitoring architecture.

3.3 An Example for Performance Monitoring

As a testing example, a MathService is selected from GlobusToolkit4 user manual as a WSRF-compliant service to provide mathematical computing. Meanwhile, we use

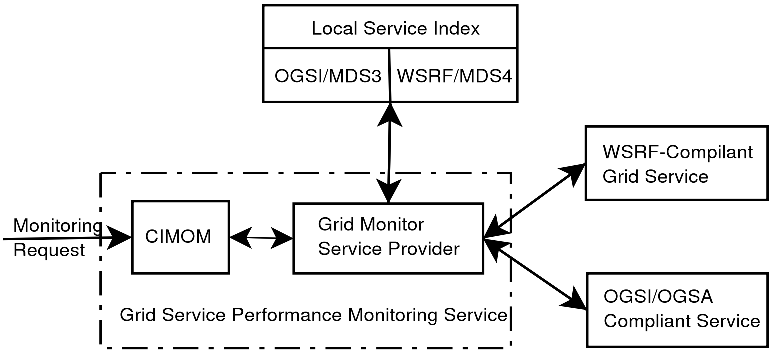
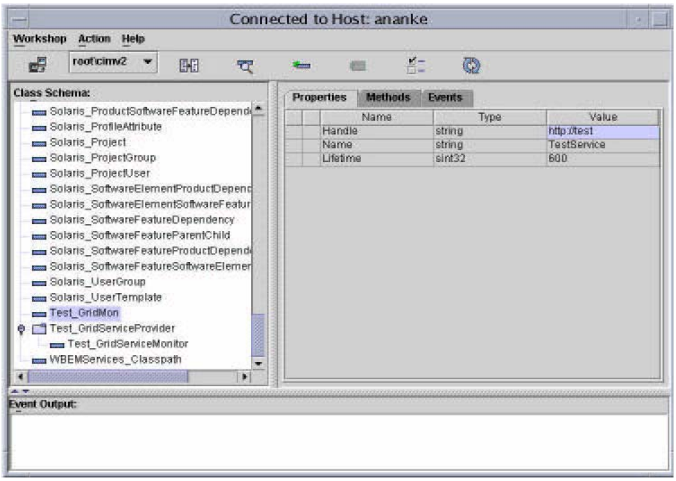

Local Service Index
OGSI/MDS3
WSRF/MDS4
WSRF-Compilant Grid Service
Monitoring Request
CIMOM
Grid Monitor Service Provider
Grid Service Performance Monitoring Service
OGSI/OGSA Compliant Service

Fig. 3. The Grid service monitoring implementation based on WBEM/CIM

our Grid service monitoring provider to monitor the status of the MathService. The MathService does nothing but some mathematical computations.

Figure 4 shows the monitored service information through CIM workshop service and



Class Schema:

```
Solaris_ProductSoftwareFeatureDepend
Solaris_ProfileAttribute
Solaris_Project
Solaris_ProjectGroup
Solaris_ProjectUser
Solaris_SoftwareElementProductDepenc
Solaris_SoftwareElementSoftwareFeatur
Solaris_SoftwareFeatureDependency
Solaris_SoftwareFeatureParentChild
Solaris_SoftwareFeatureProductDepend
Solaris_SoftwareFeatureSoftwareElemer
Solaris_UserGroup
Solaris_UserTemplate
Test_GridMon
Test_GridServiceProvider
    Test_GridServiceMonitor
WBEMServices_Classpath
```

Name	Type	Value
Handle	string	http://test
Name	string	TestService
Lifetime	sint32	600

Fig. 4. The service information monitored by CIM workshop service

4 Related Work

Very few work has been done on Grid service performance monitoring. GSMon [5], developed at Tsinghua university in China, is one of the few systems for monitoring Grid services. However, GSMon sticks to OGSI/OGSA services and does not provide an abstract layer for various Grid services.

There are some other research work focused on Grid performance monitoring and evaluation. Few of them are inherently embedded into a Grid super scheduler and most of them are stand-alone. Many of them are extended from local performance tools.

Netlogger [3] is a distributed application, host, and network logger. It can be used for performance and bottleneck analysis and correlating application performance with system information. But it is basically a central collector and does not scale well with the number of resources.

Network Weather service (NWS) [4] is a distributed system that periodically monitors and dynamically forecasts the performance of various networks. It scales well with the number of Grid resources, but it does not measure the Grid application performance.

Recently work include Grid job superscheduler architecture and performance in computational Grid environments by Shan et al. [7]. In their work they propose several different policies for superschedulers and use both real and synthetic workloads in simulation to evaluation the performance of the superschedulers. They also present several Grid performance metrics including response time and Grid efficiency.

5 Conclusion

In this paper present our work on Grid service performance monitoring. We provide some analysis of the performance of Grid services and then proposed an architecture design of our service oriented Grid performance monitoring and its prototype implementation utilizing WBEM/CIM and WS-Notification mechanism. The implementations of other components of the architecture are still in progress. Our future work may include how to adjust Grid super scheduling policies based on monitoring information fed back by Grid service performance monitors; defining the interfaces for interactions between monitors and schedulers, etc.

References

1. Common Information Model (CIM) Standards, Distributed Management Task Force. http://www.dmtf.org/standards/cim/.
2. Web Based Enterprise Management (WBEM) Services. http://wbemservices.sourceforge.net/.
3. D. G. et al. NetLogger: A Toolkit for Distributed System Performance Analysis. In *the Proceedings of the IEEE Mascots 2000 Conference*, 2000.
4. R. W. et al. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Systems*, 1998.
5. C. He, L. Gu, B. Du, Z. Huang, and S. Li. A WSLA-based monitoring system for Grid Service–GSMon. In *Proceedings of IEEE International Conference on Services Computing (SCC'04)*, pages 596–599, Shanghai, China, Sept. 2004.
6. M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing. *Communications of The ACM*, 46(10):25–28, Oct. 2003.
7. H. Shan, L. Oliker, and R. Biswas. Job Superscheduler Architecture and Performance in Computational Grid Environments. In *the Proceedings of ACM Super Computing 2003*, 2003.

Distributed Defense Against Distributed Denial-of-Service Attacks

Wei Shi, Yang Xiang, and Wanlei Zhou

School of Information Technology, Deakin University,
Melbourne Campus, Burwood 3125, Australia
{shiwei, yxi, wanlei}@deakin.edu.au

Abstract. Distributed defense is a promising way to neutralize the distributed Denial-of-Service attacks by detecting and responding the attacking sources widespread around the Internet. Components of the distributed defense system will cooperate with each other to combat the attacks. Compared with the centralized defense systems, distributed defense systems can discover the attacks more timely from both source end and victim end, fight the attacks with more resources and take advantage of more flexible strategies. This paper investigates 7 distributed defense systems which make use of various strategies to mitigate the DDoS attacks. Different architectures are designed in these 7 systems to provide distributed DDoS defense solutions. We evaluate these systems in terms of deployment, detection, response, security, robustness and implementation. For each criteria, we give a recommendation on which technologies are best suitable for a successful distributed defense system based on the analysis result. Finally we propose our idea on the design of an effective distributed defense system.

1 Introduction

Distributed denial-of-service attacks (DDoS) bring a tremendous threat to the Internet. The report form the Computer Emergency Response Team (CERT) says the security incidents each year have doubled since 1988 [1]. A large number of network servers, routers and hosts have been pulled down worldwide by the DDoS attack. Effective approaches to defeat DDoS attack are desperately demanded [2, 3].

Although some current solutions can detect DDoS attacks and drop the attacking packets in some circumstances, there is still no successful solution to DDoS attacks. Most of the current DDoS defense systems are centralized and isolated systems which are used to protect a single network. It is very difficult for the centralized defense systems to detect the attack before it was launched or at the beginning of the attacks. When the attacks are full-fledged, it becomes more difficult for defense systems to resist the flooding. And centralized defense systems themselves are more vulnerable to be attacked by the hackers. The centralized defense systems are mostly deployed on the victim network because of the economic reasons. Thus such defense systems are irresponsible systems which could only respond to the attacks, but not to stop the attacks.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 357 – 362, 2005.

Distributed defense systems overcome the shortcomings of centralized and isolated defense systems. Deployed on all around the Internet, distributed defense systems can detect the attacks before they are launched by inspecting the traffic on many edge networks in which the computers are compromised by hackers. The most important and attractive feature of the distributed defense system is that the components in the distributed defense system can cooperate with each other to fight against DDoS attacks.

This paper focuses on the distributed solutions to the anti-DDoS issue. To obtain an insight of the current methods used in the current distributed defense system, we evaluate 7 systems according to some criterion proposed by us. This evaluation is also beneficial to identify the weakness of the current distributed systems to motivate the development of better solutions. Based on the evaluation result, we propose a distributed defense system which will not only adopt the current mature and effective technologies, but employ some new methods to neutralize DDoS attacks in a more effective way.

2 Evaluations on the Strategies of Distributed Defense System

We sample 7 distributed defense systems. They are DefCOM (Defensive Cooperative Overlay Mesh) [4], IDIP (Intrusion Detection and Isolation Protocol) [5], ACC (Aggregate-based congestion control) [6], ASSYST (Active Security System) [7], Secure Overlay Services (SOS) [8], MANANET [9], COSSACK [10]. Different strategies are employed by these systems. We can not say one system is better than the other because different systems are applied onto different scenarios. However, we can compare the strategies used by these systems to get the insight of what strategies are more useful in the campaign with DDoS attacks. This section proposes some criterion to compare these strategies. Table 1 shows the summary of comparison among these 7 systems. We hope that the evaluation of these strategies can lead to the development of better strategies and even distributed defense systems.

Table 1. Comparison of 7 distributed defense system

Criterion	SOS	DefCOM	MANANet	COSSACK	IDIP	ACC	ASSYST
Deployment	Source /Victim	Throughout the network	Victim end as a group	Source /Victim	Distribute d groups	Throughout the network	Throughout the network
Security	IPSec	PKI	N/A	CA	IPSec	N/A	N/A
Detection	Filtering	Traffic tree discovery	PEIP	Spectral Analysis	Intrusion detection	Congestio n detection	Intrusion detection
Response	Rate-limiting	Rate-limiting	Rate-limiting	Dropping all packets	Dropping all packets	Rate-limiting	Dropping all packets
Robustness	Strong	Weak	Weak	Weak	Weak	Weak	Weak
Implementation	Difficult	Difficult	Difficult	Easy	Easy	Difficult	Difficult

2.1 Deployment

Since a distributed defense system has many nodes that can be homogenous or heterogeneous nodes, these nodes must be deployed at different locations in the network. The functionalities of defense nodes include detection of potential attack, alarm generating and multicasting, attack source finding, and attack traffic controlling. Different nodes can be deployed at the edge networks and core networks.

Some approaches such as DefCOM [4], ACC [6] and ASSYST [7] deploy their nodes throughout the network. This deployment requires that every participating node must be able to perform the detection and traffic controlling functions, communicate and coordinate well with each other. It could raise the unnecessary traffic burden at the intermediate nodes. Moreover, it could not be the best place to detect the attack at the intermediate nodes. We envision the best deployment is the mixture deployment at both source end and victim end. The reason for this deployment is that first, the victim end aggregates the most information for the detection and can achieve the most accurate detection true positive rate; second, by detecting preliminary attack signatures at source end allows the defense system to mitigate a DDoS attack at its initial phase; third, the source end traffic controlling can protect the network's availability to a max degree because not only the victim but also the rest of network can be free of network congestion.

2.2 Detection

We classify the detection functions into three categories, signature-based detection, traffic anomaly detection and traceback [11][12]. Signature-based detection is very accurate because it can find specific characteristics of attacks. It is applied in many current intrusion detection systems and some of the distributed defense system we mentioned in the related work [5, 14]. A main drawback of it is that it can only detect known attacks, but not the unknown/new attacks or some variants of previous attacks.

DDoS attacks bring network anomaly such as the sudden surge of network traffic volume, increase of the packets with random source IP addresses, and asymmetric amount of packets associated with some network protocol such as TCP SYN. Detection and filtering is a straightforward approach to defend such attack. The objective a successful distributed defense system should be the fast and sensitive detection by using a fine granularity detection method.

IP traceback is the ability to trace IP packets to their origins [11]. Among the traceback mechanisms, packet marking schemes are relatively easy to implement, and require a modest computation load and bandwidth [12]. Actually, packet marking traceback can be applied in two ways. One is the real-time traceback, which is to find the attacking sources during the attack and then punish the sources. Another application is DDoS detection and filtering. If the packets are marked, the information carried by the packets can be used to detect DDoS attacks [15].

2.3 Response

Rate-limiting is the most popular strategies used in the current distributed defense systems, such as in DefCOM, SOS, ACC and MANANet. Because no defense systems can detect the attacking packets with 100 percent accuracy, it is advisable to limit the rate of high-bandwidth flows rather than to drop all the suspicious packets. Rate-limiting also gives the defense system flexibility to adjust the limit to which the suspicious network traffic is suppressed. The disadvantage of the rate-limiting strategy is that it will allow a certain amount of attacking packets to pass through. This will bring problems when rate limiting is deployed on the network in which there are resource-demanding applications (e.g. video stream) and the bandwidth is not big enough. However, currently there seems to be no better solutions unless the detection accuracy can be improved to a satisfying extent.

2.4 Security

A distributed defense system must be able to protect the information to be exchanged from being intercepted by the hackers. Current security mechanisms such as IPSec, PKI, CA are sufficient to meet the requirement to obtain the above two goals. The examples of security implement can be found in [5, 8] (IPSec), [4] (PKI), and [10] (CA). Some research has been done to deal with the denial of service problems in the security protocols [13, 14]. Here we do not specifically consider how to defend the security architecture because we assume the motivation of the DDoS attacks is to prevent the legitimate users from accessing the desired resources, but not to crash the security architecture, which is more difficult to achieve.

2.5 Robustness

Here robustness means the degree to which the distributed defense system itself can resist the attacks. When the distributed defense system is deployed and is known to the hackers, they will launch attacks to the distributed defense system so that they are further attack the protect networks. Although the distributed defense system is less vulnerable to such attacks than the centralized defense system, it is still possible that the distributed defense system fails due to the attacks targeting it. Unfortunately this issue is less concerned in the design of the current distributed defense system.

2.6 Implementation

If a distributed system is in good design and has good experimental results, such system still can not be accepted by the security community if they are not easy to be implemented or even impossible to be implemented under current Internet infrastructure. Among the 7 distributed defense systems, DefCOM, SOS, ASSYST, ACC and MANANet need routers to support specific functions. We can see that a large portion of current distributed defense systems require the Internet infrastructure to be modified. That is one of the reasons why no successful solutions which can defeat DDoS attacks are available up to now.

3 Our Proposed Distributed Defense System

We propose a distributed defense system as shown in Figure 1.

Our system has the following characteristics:

(1) Deployed on source end networks and victim end networks. Nothing is required to be installed on intermediate networks (ISPs). Currently ISPs are not willing to deploy anti-DDoS systems on their networks because they can not obtain economic benefits from such systems. Not involving with ISPs, our system is more likely to be accepted by the security communities. And the components in our system will run on the servers connected to routers so we do not have the need to modify routers.

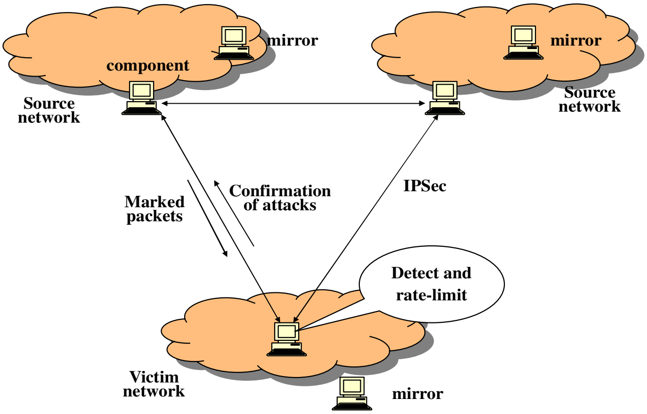


Fig. 1. Architecture of our distributed defense system

(2) Attack detection is done by analyzing the attack signatures and traffic anomaly. As for the analysis of the traffic anomaly, we will take advantage of neural network algorithms so as to potentially identify new attacks which can not be identified by analyzing the attacking signatures. And also we will use packet marking technology to improve the detection accuracy. The suspicious packets in the source network will be marked by the component monitoring that source network. When the marked packets reach the destination network, the component in the destination network will take this information from the marked packet into consideration when judging if there is a DDoS attack occurring and where the attacks originate from. If the marked packets are confirmed to be the attacking packet, attacking alarm will be sent to the component in the source network. By the combination of attacking signature detection, network traffic anomaly detection and packet marking, we anticipate that our system can achieve a high accuracy of detection of DDoS attacks.

(3) IPSec is used to encrypt the communications among different components in our distributed defense system

(4) Rate-limiting is taken advantage of to suppress the DDoS traffic

(5) Each component will have a corresponding mirror site to improve system's robustness.

4 Conclusions

In this paper, we investigate 7 distributed defense system against DDoS attacks and compare these systems according to the deployment, detection, response, security, robustness and implementation. Based on these discussions, we propose our distributed defense system which encompasses many superior features enabling our system a potentially better solution to the DDoS attacks. As an example of distributed defense system, MDAF shows strong capability to identify and filter out attack traffics and let most of legitimate traffics pass through.

References

[1] CERT/CC, “Security Statistics during 1988-2002”, Computer Emergency Response Team, Carnegie Mellon University, ttp://www.cert.org/stats/cert_atates.html, Pittsburgh, PA., Oct. 20, 2002.

[2] Cisco QoS and DDoS Engineering Issues for Adaptive Defense Network, MITRE. 7/25/2001,
http://www.mitre.org/support/papers/tech_papers_01/moore_cisco/index.shtml

[3] S. Gibson, “Distributed Reflection Denial-of-Service Attacks”, Gibson Research Corporation, http://grc.com/dos/drdos.htm, 2002.

[4] J. Mirkovic, M. Robinson, and P. Reiher, "Alliance Formation for DDoS Defense," New Security Paradigms Workshop 2003, pp.11-18, 2003.

[5] D. Schnackenberg, K. Djahandari, and D. Sterne, "Infrastructure for Intrusion Detection and Response," Proc. of the DARPA Information Survivability Conference and Exposition 2000, 2000.

[6] R. Mahajan, S. M. Bellovin, and S. Floyd, "Controlling High Bandwidth Aggregates in the Network," Computer Communications Review, Vol.32, No.3, pp.62-73, 2002.

[7] R. Canonico, D. Cotroneo, L. Peluso, S. P. Romano, and G. Ventre, "Programming Routers to Improve Network Security", Proc. of the OPENSIG 2001 Workshop Next Generation Network Programming, 2001.

[8] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure Overlay Services," Proc. of Network and Distributed System Security Symposium (NDSS 02), 2002.

[9] Cs3, Inc. MANAnet DDoS White Papers, http://www.cs3-inc.com/mananet.html

[10] C. Papadopoulos, R. Lindell, J. Mehringer, A. Hussain, and R. Govindan, "COSSACK: Coordinated Suppression of Simultaneous Attacks," DARPA Information Survivability Conference and Exposition III, pp.2-13, 2003.

[11] H. Aljifri, "IP Traceback: A New Denial-of-Service Deterrent?," IEEE Security & Privacy, Vol.1, No.3, pp.24-31, 2003.

[12] Y. Xiang, W. Zhou, and J. Rough, "Trace IP Packets by Flexible Deterministic Packet Marking (FDPM)", IEEE International Workshop on IP Operations & Management, 2004.

[13] P. Eronen, "Denial of Service in Public Key Protocols", Proc. of the Helsinki University of Technology Seminar on Network Security, 2000.

[14] J. Leiwo, T. Aura, and P. Nikander, "Towards Network Denial Of Service Resistant Protocols," 8th International Security Protocols Workshop, Cambridge, UK, pp.301-310, April 3-5, 2000.

[15] Y. Xiang, and W. Zhou, "Mark-aided Distributed Filtering by Using Neural Network for DDoS Defense", IEEE GLOBECOM 2005.

Security and Safety Assurance Architecture: Model and Implementation (Supporting Multiple Levels of Criticality)*

Li Zhongwen**

Information Science and Technology College, Xiamen University, Xiamen 361005, China
Zhongshan institute of UESTC, Zhongshan City, GuangDong Province, China
lizw@xmu.edu.cn

Abstract. A combined architecture is described to protect the system against malicious attacks as well as unplanned system failures. Discussions are laid on its characteristics, structure, safety assurance technologies. Safety kernel (shell) and integrity policy for criticality are used to ensure the system safety. Furthermore, to implement rules of integrity policy, the reflective technology based on metaobject is adopted and how to apply reflective technology to implement these rules is analyzed in details. Finally, an experiment illuminates the feasibility of the proposed architecture.

Keywords: Distributed Control System, Safety Kernel, Security and Safety Assurance Architecture; Integrity Policy for Criticality.

1 Introduction

The safety and security (in brief, we call them s&s in the rest of this paper) of network-based systems is considered as a crucial issue to guarantee the proper behavior of sophisticated distributed applications [1]. Systems that are now being built are frequently required to satisfy these properties simultaneously [2]. However, due in part to the evolutionary growth of the approaches to safety and security specification techniques, they have largely been developed in isolation. There are many nodes in distributed control system, of which the s&s assurance is divided into two parts: one is within domain[3]; the other is between domains. The safety requirement of these systems at least includes two aspects. One is to protect critical devices to avoid wrong damage resulting from software errors. The other is to ensure that a non-critical task (for example, the passenger information system) not be able to corrupt an extremely critical task (such as the automatic pilot)[4]. There are many solutions to these new

* This work is supported partly by This work is supported partly by Fujian young science & technology innovation foundation (2003J020), NCETXMU 2004 program, and Xiamen University research foundation(0630-E23011).

** Corresponding author.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 363 – 369, 2005.

safety problems. A compromise among the high cost solution and the non-effectiveness solution, is using safety kernel (shell)[5] and integrity policy. They are not needed to validate all the tasks with the same effort, but only those which accomplish critical tasks and those which provide data to critical devices. Integrity policy for criticality is different from Biba policy, the more critical a component is, the more it needs to be trusted so the higher must be its level of integrity.

This complexity constitutes the main problem in distributed control systems: how to deal with safety/security effects in an integrated way, as a basis for developing homogeneous protection solutions. In this paper, we further focus on the issues of mediation by enforcing a multilevel integrity policy and safety kernel (shell) technology, and base on our previous work [6] to present a assurance architecture PSAD which combine safety and security effects together.

2 The Structure of PSAD

2.1 The Exterior View of PSAD

PSAD has three levels, see figure 1:

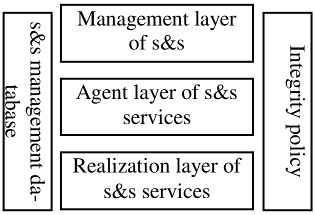


Fig. 1. Exterior view of PSAD

1. The management level: it is made up of modules to complete the installation, debugging, monitor, maintenance and rebuilding of PSAD.
2. The agent layer of security and safety assurance services: it consists of all the security and safety agents, and they cooperate with each other to ensure the security and safety of the system.
3. The realization level of security and safety assurance services.

Integrity policy for criticality defines rules on data flow that ensure that no low integrity data can corrupt high integrity objects. The s&s management database stores all related information concerning the security and safety of the system.

2.2 S& S Service

Figure 2 describes s&s service agents and their relationships in PASD.

- Appliance agent: it saves instructions of all types of appliances supported in domain and standard security and safety information; it also records security and safety requirements of appliances. In addition, it checks the rationality of

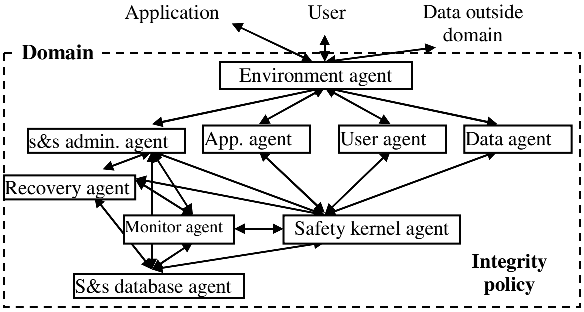


Fig. 2. Relationships of s&s service agents

security and safety requirements of appliances according to standard security and safety information and makes proper response.

- Safety kernel agent: It provides safety kernel service for protected devices.
- Monitor agent: it receives data collected by s&s management agent and records the data in security and safety management database through its agent.
- Environment agent: it carries through initial security check to users, appliance data and data from other domains.
- Outside domain data agent: it is used to deal with heterogeneous problems.

In PSAD, we quote five kinds of standard communication security service prescribed by International Organization for Standardization ISO for OSI environment. S&s safety services within domain:

- Safety kernel service: Safety kernel service is classified into four groups. They are used respectively in maintaining equipment control policies, policies for the state of Internet applications, diagnostic policies for equipment's mistakes and policies for responding to mistakes.
- Integrity authorization service: it refuses an access that can provoke a contamination of a higher or non-comparable integrity level.
- Database service: It quotes the database security services in open system environment. That is, apart from access control of OSI, data secrecy and integrality service, it also includes two kinds of services: to keep safety consistency of data stream and to prevent deducing data.
- Security and safety check service: it detects events relating to security and safety.

2.3 The Realization of Safety Service

Safety policies in safety kernel are described by FSM (finite state machine). As we analyze CRTOS[8] and RT-Linux, safety kernel mechanism can provide in many ways[6]. In our design, safety kernel is put in RTOS and the safety kernel mechanism is still provided by operating system. When no safety service is needed, the system will not provide such service.

We define an object is an entity that provides one or more services that can be requested by a client. It is composed of both internal data items (known as attributes) and methods (which provide services). We use three kinds of objects and flow control policies of these objects in paper [4]: SLOs (single level obejects), MLOs (multi-level objects) and VOs (validation objects). Here we study how to use metaobjects[7] to realize authorization based on integrity policy for criticality (see figure 3). In figure 3 the flow control policies are in the integrity management aware (IMA) which is inserted in operating system kernel. Assume component 1 with criticality level of 1 wants to invoke a read/write method of component 2 with level 3.

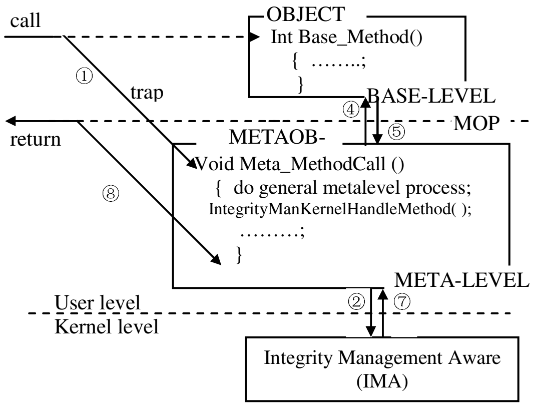


Fig. 3. Authorization realization based on integrity policy

① MetaObject intercepts information of this call req..
② MetaObject asks for IMA to authorized this call req..
③ IMA returns authorization information to MetaObject.
④ Meta_HandleMethod checks the validity of authorization information, and reasonable call request will be supported, namely, transferring this call request to BaseObject. Unreasonable call request will be refused.
⑤ BaseObject returns results to MetaObject.

⑥ and ⑦complete the authorization of results. If the results is reasonable, MetaObject transfers them to component 1 by process ⑧, else deals with errors and informs component 1.

3 Implementation on TCS[6]

Unlike paper [6], here three PCs and DC are used respectively to simulate TCS (see figure 4). IMA(Integrity management aware) of domain 3 is inserted on PC3 to test the feasibility of the proposed approach and the performance of separate IMA in the domain with more than one nodes. Hardware configurations are: ① PC1: Intel Pentium Ⅱ, PCI bus, 32M RAM; ② PC2: Intel Pentium Ⅲ, PCI bus, 128M RAM;③

PC3: Intel Pentium 120, PCI bus, 32M RAM;④ DC: i386 EX (40 MHz), CRTOS 2.0[8], 3712K. TCS' security and safety requirements includes: ① Without receiving PC1's instructions, PC2 or DC should make sure that traffic lights in the same cross don't be set to green simultaneously. ② To ensure data integrality in TCS.

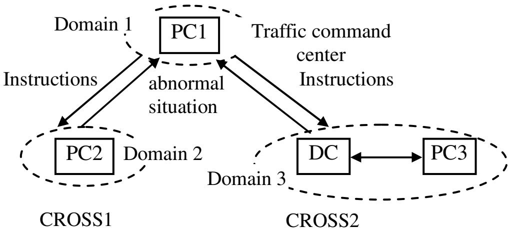


Fig. 4. Topology of TCS

Domain 1 is the mother domain of domain 2 and 3. Domain 2 and 3 is equal, their security & safety system have same structure. The structure of domains' s&s system is like that in [6]. Here we introduce the realization of integrity policy of domain 3. In domain 2 or domain 3, environment agent, safety kernel agent, management database agent are realized respectively by task SC, SK and Proc. SK provides safety kernel service. WD imitates SK's monitor, once it finds that SK behaves abnormally, it will stop the SK immediately and start another safety kernel SK', which is a back-up of SK. The COMM task issues orders to change the color of traffic lights which conflict with each other in their directions according to the instructions received by SC. Software Trafficlight simulates traffic light and changes its color in accordance with the orders from COMM. If the traffic light does not change the color for 5 times in succession according to orders from COMM, SC will send information M1, M2 or M3 to domain 1 and ask for its help. Compared with domain 2, domain 3 has an additional test, namely integrity policy test. The IMA used of integrity authorization is implemented as a kernel modular of RTLinux on PC3. Every object which appears at runtime on the host computer (object name, type, level, method types, and accessible objects for validation objects) is as following:

① SLO Trafficlight{Criticality=1; Sensor(r), Executive(w), COMM(w), SK/SK'(w)};

② SLO Proce{Criticality=2; Securitycheck(object), SC(w)};

③ SLO WD{Criticality=3, SK(r), SK(w), SK'(r), SK'(w)};

④ MLO IMA{Criticality=3; Authorization (Object)};

⑤ MLO SC{Criticality=3; Proce(r), COMM (w), SK(r), dataoutside(r), dataoutside(w)};

⑥ MLO COMM{Criticality=3; SC(r), SK/SK'(w), SK(r), Trafficlight(r)};

⑦ MLO SK'{Criticality=2; Safetycheck(Object), Trafficlight(r), Trafficlight(w)};

⑧ MLO SK {Criticality=2; Safetycheck(Object), Trafficlight(r), Trafficlight(w)};

⑨ VO FT-Sensor{Criticality=2; VSensor(r)};

If any non-referenced object appears, it is supposed to be at the lowest integrity level, with all methods in read-write mode. Sensor data are given a low integrity level since they are unreliable. It is common to use several redundant sensors to be able to tolerate the failure of some of them. Of course, a more trustable information than any single sensor will be produce by this way. So the criticality of VO FT-Sensor is set to 2.

To prove the validity of the experiment (without IMA), we set three arrays for COMM: Data1, Data2 and Data3. Data1 consists of correct orders changing traffic lights' color. Data2 adds incorrect orders on the basis of Data1 in order to check the security assurance function of domain 2 if without interference from domain1. Data3 adds more than 5 incorrect orders on the basis of Data2 to check the correctness and security of function of domain 2 or domain 3 if domain1 has interfered them. Results of the experiment prove that the security & safety system are valid. The experiment is repeated 10 times for the same tasks and the results are averaged over these runs. After using security and safety system, the efficiency of domain 2 is 98% of the initial one. We use same data in domain 3 and repeat test for 10 times. The average invocations with IMA is 632 μs, however the average invocation without IMA is 95 μ s. The invocation overheads is consisted of : ①Communication delay induced by the intercept or trap of reflection technology; ②Time overhead dues to integrity authorization in IMA; ③Transfer overhead between user state and kernel state.

4 Conclusion

In this paper a architecture was described to protect the system against malicious attacks as well as unplanned system failures. In our simulation, the safety kernel and integrity policy enforcement mechanisms are inserted in a micro-kernel so that they cannot be bypassed and to minimize time overheads during object invocation. According to test data, the bulk of the integrity authorization overhead is due to the metaobject trapping mechanisms and the communications times they induce. We need to improve the performance of refection technology. Now we are working on our simulation system, there is more and more work left.

References

[1] Grosspietsch K E, Silayeva T A. A combined safety/security approach for co-operative distributed systems, Proceedings of the 18th international parallel and distributed processing symposium (IPDPS'04), 2004

[2] Eames D P, Moffett J. The integration of safety and security requirements, SAFECOMP'99, 1999, 468~480

[3] Qin Z G, Lin J D. Design and implementation of global security system in open system environment, Journal of applied sciences [in chinese], 1999, 17(3):27~32

[4] Totl E, Blanquare J B, Deswarte Y, *et al.* Supporting multiple levels of criticality, IEEE Symposium on fault tolerant computing systems, 1998: 70~79

[5] Sahraoui A E, Anderson E, Katwijk V, et al. Formal specification of safety shell in real-time control practice. Proceedings of the WRTP'S 2000, 25th IFAC workshop on real-time programming. Oxford: Elsevier. 2000: 117~123

[6] Li Z W, Qiu Z P. A new type of security and safety architecture for distributed system: Models and Implementation, Proceedings of the Third International Conference on Information Security (infosecu'04), 2004: 107-114

[7] Fabre J C, Perennou T. A metaobject architecture for fault-tolerant distributed systems: the FRIENDS approach, IEEE Trans. On Computers, 1998, 47(1):78~95

[8] Li Z W, Xiong G Z. Research and realization of safety kernel mechanism, Computer Science [in chinese], 2001, 28(4): 87~90

Modeling and Analysis of Worm and Killer-Worm Propagation Using the Divide-and-Conquer Strategy

Dan Wu, Dongyang Long, Changji Wang, and Zhanpeng Guan

Department of Computer Science, Zhongshan University,
Guangzhou 510275, Guangdong, PRC
wuliudan@163.com,{issldy, isswchj}@zsu.edu.cn

Abstract. A new approach to fight against Internet worms through the use the worm-killing worm has been presented. This paper attempts to model the interaction between the two worms using the divide-and-conquer strategy. We extends the idea of the killer-worm and divide it into three basic types. 1) Patching type: It only installs the patches on the susceptible machines; 2) Predator type: It only kills the worm (it may also patch the infected machines); 3) Composition type: It does both the jobs. The state transition diagram of the two worms and a mathematical model for every type are given. The results by dynamic simulation with the help of MATLAB are obtained.

1 Introduction

Since the Morris [1] worm arose in 1988, active worms have become a persistent security threat on the Internet. Active worms propagate by infecting computer systems and by using infected computers to spread the worms in an automated fashion. In 2001, the Code Red [2] [3] [4] and Nimda [5] [6] worms infected hundreds of thousands of systems, causing millions of dollars loss to our society. The SQL Slammer [7] [8] appeared on January 25th, 2003, and grew into its full-fledged state in just 10 minutes with its super fast scan rate. Only seven months later, on August 11th, the famous Blaster worm [9] broke out and spread out quickly on the Internet. Worm.Netsky, Worm.Lovgate, and Worm.Sasser etc. are all notorious and awesome worms in 2004 [10] [11].

Malicious mobile codes, known as computer viruses or worms, have become a significant social problem recently. To protect against malicious worms, traditional human-intervened response is no longer adequate to preempt the epidemic. Worm incidents all above have served as existent serious proofs. A new defense mechanism to counter the attack in time has been proposed for some while [12] [19] [22]. It is known as the good-will mobile code [12] or anti-worm [19]: worm-killing worm. We call it killer-worm in this paper. Its main characteristic aspect is that it spawns exactly as worms do. In fact, it is a worm, except that it is a good-will one that cures the infected and preventively patches vulnerable machines.

The main idea of the killer-worm is that through and only through worm-like spawning, it can par with fast worms in speed.ïComparing with the commercial antivirus applications nowadays, the killer-worm approach has one important advantage:

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 370 – 375, 2005.

It is not a C/S application model with bottleneck while updating the latest files. However, there are many controversial aspects too. How does the killer-worm gain entry? How to set up trust association among normally mutually distrustful administrative network? What additional security problems will arise? Despite these concerns, there were several reports [13] [14] [15] saying that some real killer-worm mobile codes, dubbed Code Green, CRclean, and Nachi, were actually released to the Internet to fight the Code Red and Blaster. No doubt the new era of killer-worm needs much more going on study. In this paper we put aside the ethics and other non-technical ramifications and focus only on the interaction between the worm and killer-worm. First, we try to recognize the killer-worm by the divide-and-conquer strategy, which helps us to know the effectiveness and efficiency of the killer-worm technology under different circumstances. And then we give mathematical models on the worm and killer-worm accordingly. We believe the result can help to design the killer-worm more properly.

The rest of this short paper is organized as follows. Session II discusses related work and the system assumption. Three types of killer-worms are analyzed in Session III with the divide-and-conquer strategy. Session IV presents and simulates different models with the help of MATLAB(the details are omitted). Discussion and conclusion will be in Session V and Session VI, respectively.

2 Related Work

The Internet is a huge dynamic system. The easy access and wide usage makes it a primary target for malicious activities. An Internet worm model provides insight into worm behavior. It helps to identify the weakness in the worm spreading chain and provides accurate prediction for the purpose of damage assessment for a new worm threat. Internet worms are similar to biological viruses in their self-replicating and propagation behaviors. Thus the mathematical techniques developed for the study of biological infectious diseases can be adapted to the study of computer Internet worm propagation. Our work bases on the simple epidemic model (SEM) in some extent. Zou etc. derive the simple epidemic model by using infinitesimal analysis in detail [16]. Other work on worm propagation includes the KM model [17], the two-factor model [18] [19] [22] etc. These all help us to model the two worms propagation.

The notation of worm-killing worm has been in the folklore for some time. In [12], the Lotka-volterra equations are employed to model the prey-predator dynamics between the worm and the killer-worm. The shortcoming of the prey-predator model is that it didn’t concern the environmental capacity, i.e., the number of susceptible hosts. Yang [19] proposed a SIAR model on the interaction between the two worms. However, they didn't distinguish the different behaviors of killer-worm which in reality could be varied according to given requirements. This paper attempts to model the interaction between the two worms using the divide-and-conquer strategy, which means we extend the killer-worm into several types in detail because of actual variable circumstances. We will discuss it later in session III and session IV.

3 The Divide-and-Conquer Strategy in Analysis of Killer-Worm

Before modeling on the propagation of the two worms, one important thing should be clear: what a killer-worm will exactly do. We know its behavior is designed by "good-will" so that we stop for some while and think about the "good-will" carefully. Here we use the “Divide-and-conquer” strategy. It means that we study different cases and try to define different types of killer-worm accordingly. The basic idea is that the killer-worm can be assigned different jobs. For example, it may immunize the susceptible node or kill the worm or do the both. With such considerations, we extend the idea of killer-worm and decide to divide it into three basic types:

- Patching type: It only installs the patches on the susceptible machines.
- Predator type: It only kills the worm (it may also patch the infected machines).
- Composition type: It does both the jobs and is the relatively strongest killer-worm.

We admit that there may be other more complicated types. Our proposal may be perfected in later work. In Session IV, we will analyze the application timing, present state transition diagram of the two worms and give a mathematical model for every type. We also show the results by dynamic simulation with the help of MATLAB. We shall omit the details of those results because of limitation of the paper length.

4 Modeling on the Two Worms Propagation

4.1 Case 1: Patching Type

Most of Internet worms always exploit the existing vulnerability in order to gain access to the system. An initial method to contain the spread of worm is to reduce the number of the victims as many as possible. And we believe that to be immunized ahead is much better than to be recovered from infection. A very common way to protect the susceptible machines is to patch them. In addition, we know that worm can spread in a very fast way (in tens of seconds!) [20]. As a result, the time taken for patching is very crucial. Nowadays current approaches to patch distribution are primarily centralized, namely the C/S application model. Hence both the server/push approach, in which servers broadcast patches to client machines, and the client/pull approach, in which clients download the patch from a server, suffer from bottlenecks when updating the latest files due to centralization.

Here we introduce a killer-worm that patches the system to solve the problem. It acts like a worm infects the system in order to catch up with the worm speed. Furthermore, patching type has the potential of containing the spread of a worm, by continually patching machines until a stable infection rate is reached. The reason is that both of the two worms are competing for the same target.

4.2 Case 2: Predator Type

It is natural to consider another type of killer-worm. We also call it predator [12]. Its job is to find out the worms and then kill them. Comparing with the patching type, this type only pays close attention to the infectious machines. They hunt for different targets. From mathematical model, we know that the predator is a fast way to clear the worm. This would also serve as a useful stopgap measure for containment of worms until a patch becomes available. The patch could then be distributed by traditional means or by another killer-worm of patching type. In reality, the predator can be either passive or active. Passive type waits for the worm locally while active type will go out and search the worm. And according to different strategy, the predator type may patch the victims or not. When the predator packaged with a patch, its data will increase and it costs more time to multiply them. However, in some case people may not be able to get proper patch in time to immunize the system. Hence we divide the predator type into two sub-types (a) Sub-type I: state transition diagram and (b) Sub-type I: two worms' model.

4.3 Case 3: Composition Type

Based on the patching type and the predator type, it is right time to discuss a more complicated case. We obtain a relatively stronger killer-worm by a compositive approach. Exactly the third type composites the former two types' functionalities. We call it the composition type. This type seems to be a subclass that inherits from the former two types as its super classes. Maybe the most perfect method is that we can control the killer-worm dynamically. This is left for our future work.

5 Discussion and Future Work

The main weakness of the results presented in this paper is that they are all based on simple system assumption and simulation. Real systems often display behaviors that are more complex and variable. In reality, all the system parameters, such as infection rate and death rate, should be evaluated based on detecting and monitoring system for more proper input data helps to display more veracious results. In our system, we also ignore other defense factors (such as human countermeasures, network capability etc.). Also, we haven't mentioned the time delay to release the killer-worm. It is reasonable and necessary to introduce it for we cannot unleash the killer-worm the moment the worm epidemic breaks out. Here we take the composition type (see 4.3) as an example. As killer-worm is introduced some delay time, say T, after the outbreak.

Other issues confronted by a person wishing to release a killer-worm onto a network is much more than what we have discussed in this paper. One is how the killer-worm gains entry into the target system. If the killer-worm exploits the same vulnerability as the worm, as was originally assumed and not all users on the network have given consent, then the release of the killer-worm is a criminal act. So there may eventually be some sort of legal authority that authorizes the release of killer-worms. This is very easy and suitable to Intranet that has centralized administrative power. Furthermore, we can develop a secure infrastructure that can support the entry of authorized killer-worm and aid in controlling the propagation of the killer-worm. Such infrastructure sounds alluring and abstracts us to make follow-up study in later work.

6 Conclusion

The results presented in this paper demonstrate that killer-worms have the potential to quickly clean-up networks infected by self-propagating malicious code and also immunize networks from future attacks. Killer-worms have a potential for becoming a practical emergency patch distribution mechanism, when many machines need to be quickly patched in the face new a worm. Simulation techniques could be used to tune the killer-worm's behavior prior to release so that killer-worms are quickly eliminated while the only minimum amount of necessary bandwidth is consumed. Killer-worms can potentially provide timely control on the spread of self-propagating worms, thereby reducing the monetary losses due to their unchecked spread.

Acknowledgments. This work was partially sponsored by the National Natural Science Foundation of China (Project No. 60273062) and the Guangdong Provincial Natural Science Foundation (Project No. 04205407).

References

1. E.H. Spafford, The internet worm incident. In ESEC'89 2nd European Software Engineering Conference, Coventry, United Kingdom, 1989.
2. eEye Digital Security, ANALYSIS: .ida "Code Red" Worm. http://www.eeye.com/html/Research/Advisories/AL20010717.html
3. eEye Digital Security, ANALYSIS: CodeRed II Worm. http://www.eeye.com/html/Research/Advisories/AL20010804.html
4. R. Russell, A. Machie, Code Red II Worm. Tech. Rep, Incident Analysis, Secrity Focus, Aug. 2001
5. A. Machie, J. Roculan, R. Russell, M. V. Velzen, Nimda Worm Analysis, Tech. Rep, Incident Analysis, Security Focus, Sept. 2001
6. CERT/CC, CERT® Advisory CA-2001-26, Nimda Worm. http://www.cert.org/advisories/CA-2001-26.html
7. CERT/CC, CERT® Advisory CA-2003-04 MS-SQL Server Worm, http://www.cert.org/advisories/CA-2003-04.html
8. D. Moore et al., The spread of the Sapphire/Slammer worm, a NANOG presentation, http://www.nanog.org/mtg-0302/ppt/worm.pdf.
9. EEye Digital Security. Blaster worm analysis. 2003. http://www.eeye.com/html/Research/Advisories/AL20030811.html
10. CCERT, CCERT advisory on W32.Sasser, http://www.ccert.edu.cn/notice/show.php?handle=102 (in Chinese)
11. db.Kingsoft.com., Worms report, 2004. http://db.kingsoft.com/c/2004/12/29/164830.shtml(in Chinese)
12. H. Toyoizumi, A. Kara, Predators: good will mobile codes combat against computer viruses, New Security Paradigms Workshop 2002, Sept. 23-26, Virginia Beach, USA
13. Herbert HexXer, Code Green, http://www.securityfocus.com/archive/82/211428
14. CCERT, CCERT advisory on W32, Nachi.Worm, http://www.ccert.edu.cn/announce/show.php?handle=93 (in Chinese)
15. Douglas Knowles, Frederic Perriot, Peter Szor, Symantec security response: W32/Nachi.A, http://www.f-prot.com/virusinfo/descriptions/nachi_A.html

16. Zou CC, Gong W, Towsley D, On the performance of Internet worm scanning strategies, Technical Report, TR-03-CSE-07, Electrical and Computer Engineering Department, University of Massachusetts, 2003
17. Frauenthal JC, Mathematical Modeling in Epidemiology, New York: Springer-Verlag, 1980
18. Zou CC, Gong W, Towsley D, Code Red worm propagation modeling and analysis, In: Proc. of the 9th ACM Symp. on Computer and Communication Security. Washington, 2002, 138~147
19. Yang Feng, Duan Haixin, Li Xing, Modeling and analysis on the interaction between the Internet worm and anti-worm, SCIENCE IN CHINA Ser. E Information Sciences, 2004, 34(8): 841~856 (in Chinese)
20. S. Staniford, V. Paxson, and N. Weaver, How to Own the Internet in Your Spare Time, In 11th Usenix Security Symposium, San Francisco, August, 2002
21. Zou CC, Gao L, Gong W, Towsley D, Monitoring and early warning for Internet worms, Technical Report, TR-CSE-03-01, Electrical and Computer Engineering Department, University of Massachusetts, 2003.
22. Wen WP, Qin SH, Jiang JC, Wang YJ, Research and Development of Internet Worms, J. of Software, V.15(2004)8, 1208-1219. (in Chinese)

An Efficient Reliable Architecture for Application Layer Anycast Service

Shui Yu and Wanlei Zhou

School of Information Techonolgy,
Deakin University, Burwood, Victoria 3125, Australia
{syu, wanlei}@deakin.edu.au

Abstract. Anycast is a new service in IPv6, and there are some open issues about the anycast service. In this paper, we focus on efficient and reliable aspects of application layer anycast. We apply the requirement based probing routing algorithm to replace the previous period based probing routingalgorithm for anycast resolvers. We employ the twin server model among the anycast servers, therefore, try to present a reliable service in the Internet environment. Our theoretical analysis shows that the proposed architecture works well, and it offers a more efficient routing performance and fault tolerance capability.

1 Introduction

With the dramatic development of computer network technologies, a lot of new application requirements appear, and researchers are trying to develop new protocols, models to meet the ever increasing and changing requirements. Partridge, Mendez, and Milliken [9] originally proposed the idea of anycast in the network layer. They defined IP anycast as a service to deliver an anycast datagram to one of the members of an anycast group. The idea of anycast met the requirements of mirrored or replicated servers in the Internet, therefore, a number of researches were quickly conducted in the area.

At the middle of 1990s, some researchers found the limitations of network-layer anycast, for example, inflexibility and limited supported by current routers, hence, they presented the idea of application-layer anycast [1], [2], [7], focusing the research on anycast in the application layer. Our previous research [12] proposed a requirement based application layer anycast routing algorithm. Compared with the periodical based anycast routing algorithms [2], [7] the proposed algorithm possesses some advantages, therefore, we will apply an improved requirement base algorithm in this paper for the proposed architecture.

Fault-tolerant distributed systems are designed to provide transparent, reliable and continuous service despite the failure of some of its components. Anycast servers are mirrored and distributed servers in the Internet environment. As we know, the Internet is dynamic and unstable with possible server crashes and link failures, therefore, an anycast service needs reliable and continuous service guarantee for anycast users. In this paper, we extend the twin server model [14] from the local area network to the Internet environment for anycast services, and apply the queuing theory to analyze the changes when an anycast server failure occurs. Paper [13] has explored this issue, and an enforced version will be deployed in this paper.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 376 – 385, 2005.

The remainder of the paper is organized as follows. Section 2 introduces related work and background. In section 3, we present the efficient and reliable architecture for anycast service, and related algorithms, the periodical probing routing algorithm and the twin server model algorithm. We compare the performance and capability of the proposed algorithms in section 4. Finally, section 5 summaries the paper and discusses the future work.

2 Related Work and Background

A number of anycast routing algorithms [2],[8],[11],[12] have been proposed. Paper [8] took use of round trip time on an anycast router for server selection decision for network-layer anycast. Paper [2] proposed a network status and server load mixed application-layer anycast algorithm, but the data of anycast resolver is updated periodically based on periodical probing on network performance and server load. Paper [12] created a requirement-based probing algorithm for application-layer anycast routing.

The critical problem of application-layer anycast is how to map an anycast query into one IPv4 address. Paper [2] presented 4 metrics about how anycast performs: 1) server response time, 2) server-to-user throughput, 3) server load, and 4) processor load. The paper identified four possible approaches to maintain replicated server performance information in the anycast servers' database: remote server performance probing; server pushing; probing for locally-maintained server performance; and user experience.

The topic of fault tolerance for distributed systems has been explored for many years. [5] introduced the concept of unreliable failure detectors and studied how they can be used to solve the consensus problem in asynchronous systems with crash failures. [6] studied the quality of service (QoS) of failure detectors. The paper focused on two issues in terms of QoS: a) how fast the failure detector detects actual failures, and b) how well it avoids false detections. The paper first proposed a set of QoS metrics to specify failure detectors for systems with probabilistic behaviours, such as the detection time for how fast a detector detects crashes, and the query accuracy for how well a detector avoids mistakes. The paper then presented a new failure detector algorithm and analyzed its QoS in terms of the proposed metrics.

[14] researched the fault-tolerant problem in the scenario of distributed operating system, and tried to provide continuous services in the case of a server or even a host failure, without or with little impact on the whole distributed system. For each service, two servers (twin servers) are maintained to provide the fault-tolerant service. If one server dies, its twin will continue its job. The background of the research is that the servers are located very "near", such as in a local area network and all the twined servers are symmetric computers.

Network traffic properties have been intensely studied for a quite long time. Examples of analysis of typical traffic behaviours can be found in [3],[10]. Traffic variables on an uncongested Internet wire exhibit a pervasive nonstationarity. As the rate of new TCP connections increases, arrival processes (packet and connection) tend locally toward Poisson, and time series variables (packet sizes, transferred file sizes, and connection round-trip times) tend locally toward independent [4]. Here the Poisson arrivals are given by

$$P\{X=k\}=\frac{\lambda^k}{k!}e^{-\lambda}, k=0,1,2,... \tag{1}$$

The statistical properties of the Internet congestion reveal long-tailed (lognormal) distribution of latencies [16]. Here the possibility of latency time T_L are given by

$$P(T_L)=\frac{1}{T_L\sigma\sqrt{2\pi}}\exp\left(-\frac{(\ln T_L)^2}{2\sigma^2}\right) \tag{2}$$

Where σ represents the workload of the network. Latencies are measured by performing a series of experiments in which the round-trip times of ping packets are averaged over many sent messages between two given nodes.

3 The Efficient and Reliable Architecture for Application-Layer Anycast

In this section, we combine our previous work on anycast routing algorithm [12] and fault tolerance research [13] to propose an efficient and reliable architecture for application layer anycast, shown as Figure 1. In the architecture, there is an application program, anycast resolver, running all the time for anycast routing services. We suppose that there are *N* servers in the anycast group, S_1, S_2, ..., S_n, which are distributed in the Internet, and there is one anycast resolver severing for the anycast group.

As we found in paper [2] that the foundation of anycast resolver algorithms is the remote server performance probing based on periodical testing, we name it as periodical probing routing algorithm. Paper [2] mixed the different methods together in practical applications. There are several disadvantages for the periodical probing algorithm: accuracy problem; networkload problem; completeness problem; and resolver server load problem. In this paper, we employ an algorithm, requirement based probing routing algorithm, which can overcome most of the disadvantages of the periodical probing routing algorithm. The main idea of requirement based probing routing algorithm is described as below.

- A client submits an anycast request to the anycast resolver for anycast routing service (step 1 in Fig. 1),
- The resolver will broadcast *N* probing packets, such as ping, to each member in the anycast group, respectively (step 2 in Fig. 1). In this case, the probed servers will respond for the probing requirements, respectively. If a server's workload is heavy or performance is bad, then the responding will last longer than a server whose workload is light or performance is good. Therefore, the probing packets can not only probe the servers' performance at that moment, but also the network workload at the same period. Based on the analysis, we define that the first responsive server as the "best" server in the anycast group, because the responsive time represents the network performance and server performance.
- The anycast resolver delivers the IPv4 address of the "best" server to the client (step 3 in Fig. 1).
- The client then tries to find the server using the traditional IPv4 procedures (step 4 in Fig. 1).

The advantages of requirement based probing routing algorithm include higher accuracy, better system performance, and less workload for both network and resolvers than the periodical probing routing algorithm. It is also practical and easy to implement. In section 4, we will present the performance comparison of the two categories of application-layer anycast routing algorithms.

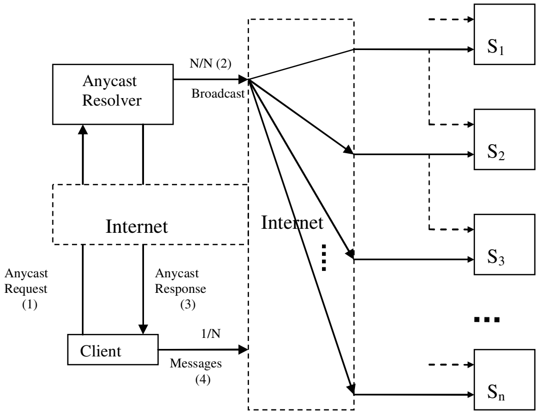


Fig. 1. An Efficient and Reliable Architecture for Application Layer Anycast

In order to provide transparent, high performance and reliable services, we organise the distributed mirrored anycast servers in pairs, and the anycast resolver takes the responsibility of deciding the pairs. For each server in an anycast group, say S_P, we try to find a backup server, S_T, from the other anycast servers in the same anycast group of S_P. Once S_P fails, then S_T will continue the uncompleted services of S_P. We name server S_P as primary server, and server S_T as twin server. For example, in Figure 1, *{S₁, S₂}* is a pair, and S_1 it the primary server, and S_2 is the twin server. More details of this fault-tolerance model are shown as Figure 2.

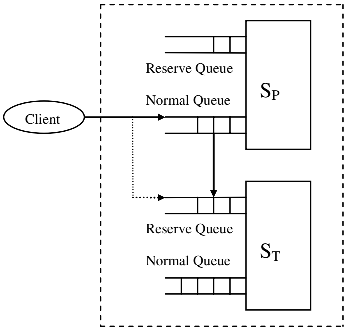


Fig. 2. The Reliable Model for Anycast Servers

```
The Algorithm at the Primary Server ( Sender )

$T_s$ = the mean service time for requests of primary server;
P = the pointer for the normal queue;
 t = 0; // t is the service time for a request
while ( True )
        if ( $t < 2T_s$) then
            MessageSend (p)
             t = 0;
        endif
        if ( $t >= 2T_s$ ) then
            MessageSend (" I am alive");
             t = 0;
        endif
end while.

The Algorithm at the Twin Server ( Receiver)

$T_s$ = the mean service time for requests of primary server;
P = the pointer for the reserve queue;
$T_d$ = the network delay between the two servers;
t = 0; // t is the interval between two coming messages
while (True)
      if ( $t < 2T_s + T_d$) and (MessageReceive () <> null) then
         p = MessageReceive (); // update the pointer
         t = 0;
       end if
      if ( $2T_s + T_d <= t < 2*(2T_s + T_d)$) and (MessageReceiver() = " I am alive" ) then
         t = 0;
      endif
       if ( $t > = 2* (2T_s + T_d)$) and ( MessageReceive() = null ) then
      // suspect that the primary server is dead
           QueueAppend (Normal Queue + Reserve Queue)
end while
```

List 1. The Algorithms for Reliable Anycast Servers

For each server, there are two queues, normal queue and reserve queue, for the incoming requests. If there comes an anycast request for server S_P, then the request will be stored in the normal queue of server S_P, at the same time, S_p sends a copy of the request its twin, S_T, and the copy will be stored in the reserve queue of S_P. each server takes the requests from its normal queue and executes the requests respectively. There is a pointer from the normal queue of the primary server to the reserve queue of the twin server to indicate the progress of the execution of the requests in the primary server. Once a request is executed successfully by the primary server, the copy of the request will be deleted from the reserve queue of the twin server by moving the pointer. Once the twin server finds that the primary server is down, it will push the request(s) in the reserve queue into its normal queue, therefore, the uncompleted requests of the primary server will be executed by the twin server. The procedure of the requests transfer is quick and transparent to the users.

The key issue in the reliable model is server failure detecting, therefore we propose a server failure detecting algorithm, shown as List 1. The purposes of the sever failure detecting algorithm are to synchronize the normal queue of the primary server and its counterpart reserve queue of the twin server, and to detect the crash failure of the primary server as well.

The main idea is that the primary server sends messages to the twin server, and the twin server decides to trust the primary server (*the primary server is alive*) or suspect the primary server (*the primary server is dead*). In order to avoid a request is executed twice by the primary server and the twin server respectively, once a request is completed successfully, the primary server will send that information to the twin server immediately. Therefore, the frequency of that kind of message transmission is high. If the twin server gets one of the messages, then it is true that the primary server is alive. It is possible that there is no that kind of message transmission for a long time. For this reason, we set two timers in the primary server and the twin server respectively to calculate the time consuming. Once a request is processed in the primary server, the server will send a message about the pointer to the twin server, and the later will adjust its pointer of the reserve queue. If a request's service time is longer than a given time (*2Ts in this paper, Ts is the mean service time of the primary server*), then the primary server will send a message (*I am alive*) to the twin server to hint that the primary is alive. On the other hand, if the twin server does not receive a message from the primary server for a long time (*2*(2Ts+T_d) in this paper, T_d is the average network delay*) then it will suspect the primary server and it will take the primary server's uncompleted duties.

4 Performance and Capability Analysis for the Architecture

In this section, we will compare the performance of the requirement based routing algorithm with the periodical probing routing algorithm based on queuing theory and the pervious researches. We also analyze the capability of the twin server model in the Internet environment.

4.1 Performance Comparison for Anycast Routing Algorithms

We compare the two categories of application-layer anycast routing algorithms based on research of statistics characteristics of the Internet traffic and the queuing theory. There are some assumptions for the calculations:

1) Customer arrival rate and the service rate are Poisson distributions.
2) The time unit for both algorithms is 1.
3) During the time unit of 1, there are N customers for both algorithms.
4) There is one server in the system acting as the resolver, and the service velocity, μ, can be obtained from formula (2).

$$\mu = \frac{1}{E(x)} = e^{-\frac{1}{2}\sigma^2} \tag{3}$$

There are two important parameters to measure the performance of a system. One is the average time used in the system for a customer, denoted as T_q. Another one is the average waiting time for all customers, denoted as T_w. For both algorithms, we will calculate these two parameters respectively. We use p as the subscript for the periodical probing algorithm, and r as the subscript for the requirement based probing algorithm.

We have obtained the result in [12] as follows,

$$T_{qp} = pT_{qp1} + (1-p)T_{qp2} = \frac{(1-p)e^{\frac{1}{2}\sigma^2}}{1-\frac{N}{1-p}e^{\frac{1}{2}\sigma^2}} \tag{4}$$

$$T_{wp} = pT_{wp1} + (1-p)T_{wp2} = \frac{Ne^{\sigma^2}}{1-\frac{N}{1-p}e^{\frac{1}{2}\sigma^2}} \tag{5}$$

$$T_{qr} = \frac{\frac{1}{\mu}}{1-e} = \frac{e^{\frac{1}{2}\sigma^2}}{1-Ne^{\frac{1}{2}\sigma^2}} \tag{6}$$

$$T_{wr} = \rho T_{qr} = \frac{\rho}{\mu(1-\rho)} = \frac{Ne^{\sigma^2}}{1-Ne^{\frac{1}{2}\sigma^2}} \tag{7}$$

Based on this, we can derive the following two conclusions:

Conclusion 1. $T_{qr} < T_{qp}$, $t \in (0, p_e)$, where $P_e = \frac{1-2Ne^{\frac{1}{2}\sigma^2}}{1-Ne^{\frac{1}{2}\sigma^2}}$.

Based on formula (4) and (6), we obtain the curves shown as Figure 3. If P locates in (0, p_e) then $T_{qr} < T_{qp}$, and if p locates in ($p_{e,\ 1}$] then $T_{qr} > T_{qp}$. That means when the net-workload becomes heavy ($\sigma\uparrow$), or there are more customers ($N\uparrow$), or both of these events happen, then P_e becomes smaller. That is when the above situation(s) happen, in a system's view, T_{qp} is less than T_{qr}, but in practice, we hope that P_e is close to time point 1, that means we hope the resolver's database update period is only a small part of the whole time unit, because during [P_e, 1], resolver will focus on database updating, therefore the performance of the service is poor. Based on the analysis, generally speaking, in most of the time unit, (0, P_e), the performance of the requirement-based probing algorithm is better than that of periodical probing algorithm; only in a very small part of the time unit, (P_e, 1), the former performance will be worse than the later.

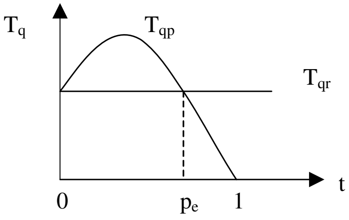


Fig. 3. The compare of T_{qp} and T_{qr}

Conclusion 2: $T_{wr} \le T_{wp}$

This shows that T_{wr} is always less than or equal to T_{wp}, namely the average waiting time of the requirement-based probing algorithm is always less than or equal to that of the periodical probing algorithm.

From the two conclusions, we can obtain that: in general, the requirement based probing routing algorithm is better than the periodical probing routing algorithm.

4.2 Performance Analysis for the Twin Server Model

In order to analyse the capability and performance for the twin server model, we model the system as Figure 4, where S_1 is the primary server, and S_2 is the twin server. Q_1 and Q_2 are the normal queues for S_1 and S_2 respectively. When S_1 crashes, the uncompleted jobs in Q_1 will be pushed into Q_2 to be executed. The specifications of the related parameters are described below.

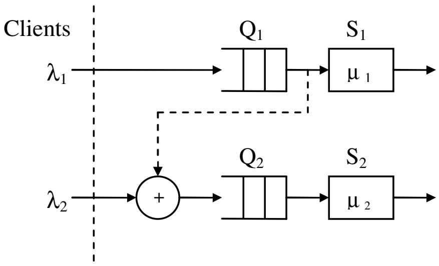


Fig. 4. The Reliable Model for Anycast Service

λ_i, $i = 1, 2$. arrival rate of Poisson arrival.
μ_i, $i = 1, 2$. mean service rate for each arrival.
T_i, $i = 1, 2$. mean service time for each arrival.
T_{qi}, $i = 1,2$. mean time a request spends in the system.

Here, $\mu_i \cdot T_i = 1$. The service time T_i includes two parts: the average network delay of traffic and the average computing delay of the server.

Based on the mathematics model and query theory, we have derived the following assertions in paper [13]

Assertion 1. If the workloads of n computers $(n \geq 2)$ are balanced, then in a given period [*0, T*] (*T* is sufficiently big), the sums of the related service time T_s of each computer are equivalent.

Assertion 2. If the workload of *n* computers $(n \geq 2)$ are balanced, then in a given period [*0, T*] (*T* is sufficiently big), the ratios of arrival rate to the service rate for each computer are the same.

Assertion 3. If the workload of *n* computers $(n \geq 2)$ are balanced, then in a given period [*0, T*] (*T* is sufficiently big), the relationship between $T_{q,}$ *mean time a request spends in the system*, and the arrival rate λ is reciprocal.

Based on the previous assertions, we obtain a very important conclusion:

Assertion 4. In our proposed fault tolerant anycast server model of Figure 4, if $\mu_2 >> \mu_1$, when the primary server crashes, in the following crash processing period, T_q for the requests of S_2 is decreased, but very close to that before the crash; T_q for the unfinished request(s) of S_1 is dramatically increased in the viewpoint of clients.

The conclusion of assertion 4 is applied by the anycast resolver in the proposed architecture to choose the server pair. There are more algorithms for the twin server model, such as, the twin server failure broadcasting algorithm. The interested readers please refer to paper [13].

5 Remarks and Future Work

In this paper, we proposed an efficient and reliable architecture for application layer anycast service. We applied the requirement based probing routing algorithm, instead of the periodical probing routing algorithm, and generally speaking, the employed algorithm is better in several aspects, such as, accuracy, network workload, and so on.

The anycast servers are distributed and mirrored in the unstable Internet environment, therefore, a fault-tolerant mechanism is highly expected for the anycast systems. We extended the twin server model to propose a reliable and efficient anycast service.

We have proved that the proposed architecture works well by modeling and mathematic analysis, and further, a prototype and the experiments in the Internet are the jobs for the future work.

References

1. S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application-layer Anycasting," Technology report, College of Computing, Georgia Institute of Technology, 1996
2. S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei, "Application-layer Anycasting," IEEE INFOCOM'97, April, 1997
3. R. Caceres, "Measurements of wide-area Internet Traffic," Tech. Report. UCB/CSD 89/550, Computer science Department, University of California, Berkeley, 1989.

4. Jin Cao,William S.Cleveland, Dong Lin, and Don X. Sun, “On the Nonstationarity of Internet Traffic,” Proc. ACM Sigmetrics `01, 102-112, 2001.
5. T. D. Chandra and S. Toueg, “Unreliable Failure Detectors for Reliable Distributed Systems.” Journal of the ACM, 43(2): 225-267, March 1996.
6. Wei Chen, Sam Toueg, and Marcos K. Aguilera, “On the Quality of Service of Failure Detectors,” Proceeding of International Conference on Dependable Systems and Network, New York, USA, June 25-28, 2000.
7. Z. Fei, S. Bhattacharjee, E. W. Zegura, and M. H. Ammar, “A Novel Server Selection Technique for Improving the Response Time of a Replicated Service,” IEEE INFOCOM’98.
8. Hirokazu Miura and Miki Yamamoto, “Server Selection Policy in Active Anycast,” IEICE Trans. Commun., Vol. E84.B, No. 10 October 2001.
9. C. Partridge, T. Mendez, and W. Milliken, “Host Anycast Service,” RFC 1546, Nov. 1993.
10. Vern Paxson, “End-to-End Internet Packet Dynamics,” IEEE/ACM Transactions on Networking, Vol.7, No.3, pp. 277-292, June 1999.
11. Dong Xuan, Weijia Jia, Wei Zhao, and Hongwen Zhu, “A Routing Protocol for Anycasting Massages,” IEEE Transactions on Parallel and Distributed Systems, Vol. 11, No. 6, June 2000.
12. Shui Yu , Wanlei Zhou, Fuchun Huang, and Mingjun Lan , "An Efficient Algorithm for Application-Layer Anycasting ", *The Fourth International Conference on Distributed Communications on the Web, Sydney, Australia, April 3-5, 2002*
13. Shui Yu , Wanlei Zhou, and Weijia Jia , "Fault-Tolerant Servers for Anycast Communication". *PDPTA'03, June 23-26, 2003, Las Vegas, USA*
14. Wanlei Zhou, Andrzej Goscinski, “Fault-Tolerant Servers for RHODOS System,” Journal of Systems and Software, Elsevier Science Publishing Co., Inc., New York, USA, Vol. 37, No. 3, pp 201-214, June, 1997

A Distributed Approach to Estimate Link-Level Loss Rates

Weiping Zhu

ADFA, The University of New South Wales, Australia

Abstract. Network tomography aims to obtain link-level characteristics, such as loss rate and average delay of a link, by end-to-end measurement. A number of methods have been proposed to estimate the loss rate of a link by end-to-end measurement, all of them, in principle, are based on parametric statistics to identify unknown parameters. In addition, they all used the traditional centralized data processing techniques to complete the estimation, which is time-consuming and unscaleable. In this paper, we put forward a distributed method to tackle the scalability problem. The proposed method, instead of estimating link-level characteristics directly, estimate path level characteristics first that can be executed in parallel and can be achieved by a distributed system. The path level characteristics obtained can be used to identify link-level ones later. The proposed method has been proved to be an maximum likelihood estimate.

1 Introduction

Link-level network characteristics, such as packet loss rate and average delay of a link, are important to network design and performance evaluation. However, due to technical and commercial reasons, those characteristics cannot be obtained directly from a network. To answer this challenge, research community starts to investigate other alternatives to obtain this information [1], [2], [3], [4], [5]. The most interesting alternative is called network tomography that aims to obtain network characteristics by sending probing packets from a source or a number of sources to a number of receivers, via the networks that we are interested in its characteristics. By observing the arrivals and their correlations at the selected receivers attached to the networks, it can find some network characteristics.

Since all observations are carried out at the designated receivers attached to endpoints, we only have an incomplete view of network reactions to the probes. We then rely on statistical inference, such as maximum likelihood estimate (MLE), to find out the characteristics of those links, including those cannot be directly observed. MINC is the pioneer to use the multicast-based approach to obtain the loss rates of a tree-like network [5], [6], [7]. It depends on a set of high-order polynomial that show the correlation of loss rates in a tree structure. By solving those polynomials with numeric methods, the loss rates of links can be obtained. Harfoush *et al.* proposed to use a unicast-based approach to discover link-level characteristics [8]. Their simulation confirms the usefulness

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 386–395, 2005.

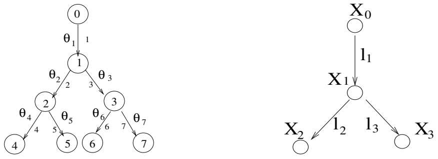
of their method. Similarly, Coates and Nowak also used the packet-pair technique to estimate link-level characteristics. They used EM algorithm to estimate the correlation between packet pairs, and then loss characteristics on links [9]. Zhu proposed to use Bayesian networks to estimate loss rates [10].

After examining probe propagation in a multicast tree, we have a insight about the loss rates between paths and links. The insight, called hierarchical dependency, forms the basis of a new inference method that divides the optimization process into a number of independent subtasks. Instead of estimating link loss rates directly, we can either estimate the loss rate of a path connecting the source to an internal node or estimate the loss rates of those subtrees that connect a node to the receivers attached to the subtrees first. It is then based on the the path loss rates identified to find the link-level loss rates. The insight further leads to two theorems that unveil the relationship between the loss rate of a link and the loss rates of the two paths that connect the source to the two ends of the link. Applying one of the two theorems on the obtained path loss rates, we can obtain the link-level loss rates. The proposed method greatly reduces the time spent on inference and turns the centralized processing model into a distributed one that not only reduces the time used in inference, but also reduce the amount of data exchanged between nodes. In fact, the complexity of a subtask is independent to the size of a network and equal to the complexity of the inference of a two-level tree that makes the method scalable. More importantly, the method is a MLE.

The rest of the paper is organized as follows. In Section 2, we present the fundamental of statistical inference and apply it to discover loss rates by end-to-end measurement. In Section 3, we present the hierarchical dependent insight in details and derive the two theorems. In addition, a distributed scheme used to identify link-level characteristics is presented. Section 4 covers the details of traffics used in our simulations and compares the results obtained from observation with the actual data collected from the simulator. Section 5 is devoted to concluding remark.

2 Statistical Inference

The multicast tree used to send probes to receivers can be abstracted by a three-element tuple (V, E, Θ) as shown in Figure 1(a). The first two elements



(a) Multicast Tree.

(b) Virtual Tree.

represent the nodes and links that have the same definitions as that in graph theory, i.e., $V = \{v_0, v_1, ...v_n\}$ is a set of nodes, which correspond to routers and switches in a network, $E = \{e_1, ..., e_n\}$ is a set of links that connect the elements of V to form a network. While, $\varTheta = \{\theta_1, ..., \theta_n\}$ is a n-element vector, one for a link to describe the network characteristics that we are interested in, and in loss tomography the characteristics is the loss rate of a link. As a regular tree, we assign a unique number to each link, starting from 1 to n, we also assign a unique number to each node, starting from 0 to n. The two sets of numbers map each other as follows: link 1 connects node 1's parent (node 0) to node 1, link 2 connects node 2's parent to node 2, and so on.

To find the loss rates of links, in particular for those that cannot be observed, a loss model should be assumed, which has some unknown parameters. The model describes the behavior of losses occurred on a link. Network tomography in this circumstance aims to determine those parameters from samples collected by receivers. If those parameters are identifiable, when we have enough samples, we should be able to determine them correctly. A number of methods, e.g. neural net, Monte-Carlo, Gaussian approximation, EM, etc., have been developed to identify the unknown parameters, and all of them adopt an iterative approximating approach to do it. Mathematically, this process can be expressed as:

$$\sup_{\varTheta} L(\varTheta) = \sup_{\varTheta} \sum_{y \in \varOmega_R} n(y) \log Pr(y; \varTheta) \tag{1}$$

where R denotes the set of receivers attached to leaf nodes, and $\varOmega_R$ denotes the possible observations of R. All the methods listed above aim to find a set of parameters embedded in (1)that can maximize (1). That is equivalent to search for a point in a n-dimensional space that can maximize (1), which can take considerable amount of time and take the risk to get a local maximum. Apart from those, to send all samples from receivers to a centralized server not only takes time but also needs bandwidth.

3 Decomposition

What we consider here is whether the inference process can be decomposed into a number of tasks that can be executed independently by different nodes. This attempt depends on whether the parameters embedded in (1) can be merged and decomposed. A merged parameter here means grouping some related parameters together, called meta-parameter, that can be identified first; and later the original parameters can be recovered from meta-parameters. After studying the uniqueness of multicast trees, we discover two key insights from probe propagated in a tree structure, namely *sibling independency* and *hierarchical dependency.* They provide the foundation to merge and decompose the parameter space.

3.1 Virtual Link

For each internal nodes, we can construct a two level tree, which has an input virtual link connecting the source to the node that carries probes from the source to the node, and a number of output virtual links, each for a subtree rooted at the node. Each internal node has a set of receivers connected to it. For instance, Figure 3.1 shows how the multicast tree shown in Figure 1(a) is decomposed into three two-level trees, where node 45 (67) represents the combined observation of node 4 (6) and 5 (7), the link between 0 and 2 (3) is the path between 0 and 2 (3) in Figure 1(a), the link between 1 and 45 (67) is a subtree in Figure 1(a).

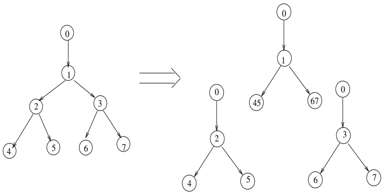


Fig. 1. Decomposed Figure 1

As stated, each link has a unique number, and every subtree can be named by the number of its root link and denoted by $T(i), i \in V$. Accordingly we use $V(i)$ to denote the set of nodes in $T(i)$. Let R denote the receivers attached to the multicast tree, and then $R(i) = R \cap V(i)$ be the set of receivers attached to the multicast subtree rooted at node i. In addition, let $path(i)$ denote the links that connect the source (node 0) to node i.

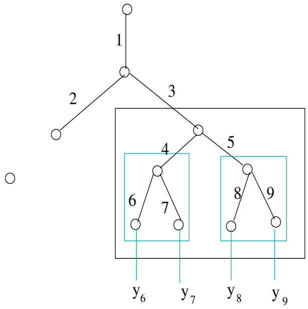


Fig. 2. A Multicast Tree and Subtrees

Every node, apart from the source, has a unique path connecting the source to itself. Apart from leaf nodes and the source, the rest of V that have a parent and children are grouped into a set called *internal set* and denoted by Int, $Int = V \setminus (R \cup 0)$. Let $|Int|$ represent the number of internal nodes. Node $i, (i \in Int)$, observes only those probes that are passed by $path(i)$. Therefore, from the viewpoint of node i, $path(i)$ can be viewed as the single input link that forwards probes from the source to the node. In addition to the input link, node i, since it is not a leaf node, has a number of children, denoted by $clink_i$, where $|clink_i|$ denotes the number of children. Each child of node i is a subtree rooted at node i, and can be viewed as an output link of node i. Let $clink_i(j)$ denote the output link that is rooted at node i, and via link j, i.e. subtree j is a child of node i. Based on the above views, each internal node can be virtually abstracted as a two level tree, called *node view* in the rest of the paper. A node view consists of only one input link that connects the source to the node, and a number of output links connecting the node to its receivers.

To estimate the loss rates of the input link and output links of a node, we need to combine the observations of a subtree that corresponds to an output link of the node. Let $\Psi(k)$ denote the view of subtree k for a probe, which equals to:

$$\Psi(k) = \begin{cases} 1, \exists i, \ i \in R(k), \ y_i = 1 \\ 0, \forall i, \ i \in R(k), \ y_i = 0 \end{cases} \tag{2}$$

where y_i corresponds to the observation of receiver i. Based on $\Psi(\cdot)$, we are able to estimate the loss rates of the virtual links by running EM or other algorithms. Let cd_k represent the effect of a trial on node k. After n trials, $R(k)$, the receivers attached to node k, provides its observations, $\mathbf{CD} = \{cd_k^1, cd_k^2, ..., cd_k^n\}$. Based on the data provided, the loss rates of the input and output links of the node can be estimated. More importantly, these estimation can be executed in parallel and in distributed manner.

In a distributed system, each receiver is responsible for an internal node since $|R| > |Int|$. The task assigned to a receiver should consider the locality of other receivers that form a group for an internal node. Due to the recursive nature of the tree structure, the task can be assigned easily. Then, each receiver according to the tasks assigned and its groups passes its observations to those nodes that assigned the tasks. Whenever a node receives all required observations, it can start its estimation independently.

3.2 Hierarchical Dependency

Applying EM or other methods on a node view, say i, we can obtain the loss rates for the input link and output links of the node. To obtain the link-level loss rates that we are interested in, we need to have a method that can compute the link-level loss rates from the related path loss rates. Two theorems that reveal the relationship between these two loss rates are discovered; in which let $f(i)$ denote the parent node of node i, and let p_i denote the input loss rate for node i, and then, $p_{f(i)}$ denotes the input loss rate of node $f(i)$. Further, let l_i denote the loss rate of link i that connects node $f(i)$ to node i. l_i can be obtained by the following theorem:

Theorem 1. *If we have the input loss rates of node i and node f(i), p_i and $p_{f(i)}$, respectively, the loss rate of link i is equal to:*

$$l_i = \frac{p_i - p_{f(i)}}{1 - p_{f(i)}} \tag{3}$$

Proof. Let e_i and e_{pi} denote loss events occurred on link i and $path(i)$, respectively. Based on probability addition rule, we have

$$\begin{aligned} P(e_{pi}) &= P(e_i \cup e_{f(i)}) \\ &= P(e_i) + P(e_{f(i)}) - P(e_i \cap e_{f(i)}) \\ &= P(e_i) + P(e_{f(i)}) - P(e_i)P(e_{f(i)}) \end{aligned} \tag{4}$$

since e_i is independent to $e_{f(i)}$, $P(e_i \cap e_{f(i)}) = P(e_i)P(e_{f(i)})$. Then, we have

$$P(e_i) = \frac{P(e_{pi}) - P(e_{f(i)})}{1 - P(e_{f(i)})}$$

Using p_i, $p_{f(i)}$, l_i to substitute $P(e_{pi}), P(e_{f(i)})$, and $P(e_i)$, respectively, we have (3).

Similarly, we have Theorem 2 to obtain the loss rates of a link from its related output links. Let $cp_i(k)$ denote the output loss rate of subtree k which is an output link of node i. Then, $cp_{f(i)}(i)$ is the loss rate of output link i, where link i connects node $f(i)$ to node i. Then, we have

Theorem 2. *If we have the loss rates for $cp_i(k)$ and $cp_{f(i)}(i)$, the loss rate of link i is equal to:*

$$l_i = \frac{cp_{f(i)}(i) - \prod_{k \in clink_i} cp_i(k)}{1 - \prod_{k \in clink_i} cp_i(k)} \tag{5}$$

Proof. Since $cp_{f(i)}(i) = l_i + (1 - l_i) \prod_{k \in clink_i} cp_i(k)$, we obtain (5) after reorganizing the equation.

In fact, these two theorems are dependent, from one can obtain the other. Thus, one should be regarded as a corollary of the other.

3.3 Distributed Algorithm

On the basis of the node view, the inference process can be decomposed into a number of tasks that can be executed in a distributed system, the number is equal to the number of internal nodes. For instance, Figure 3 a) shows a multicast tree used to send probes to the receivers attached to the leaf nodes. There are 6 internal nodes, i.e. 1, 2, 3, 6, 7, and 8, each has its own view when probes are sent from the source to the receivers. The view of node 6 can be represented by a two level tree a shown in Figure 3 b), where $\Psi(8)$ and $\Psi(9)$ represent the node based observations by $R(8) = \{12, 13\}$ and $R(9) = \{9\}$, respectively.

Based on the two Theorems, we put forward a distributed algorithm that adopts a divide-and-conquer approach to find link level loss rates, which has three phrases as follows:

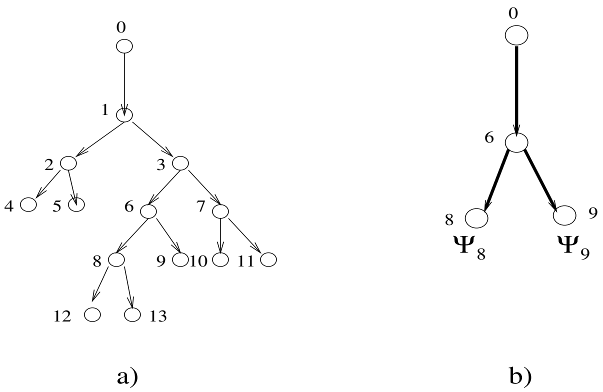


Fig. 3. A Multicast Tree and An Internal View

```
Procedure main {
        group  observation(T, D);
        path-rate estimation();
        link-rate calculation();
   }
```

1. Procedure *group_observation(T, D)* has two inputs parameters T and D, the former is the multicast tree used to propagate probes, the latter is the samples collected by receivers. The procedure groups observations for each internal node according to the tree structure and determine a receiver to do the estimation.
   This procedure can be carried out in a distributed manner from bottom up. For a group of receivers that share the same parent, they selects a receiver to do the estimation of the 2 level virtual tree created for the parent node. Those unselected receivers move one level up to join other receivers to compete the duty to estimate the path loss rates for their grand-parent. This process is continued until it reaches the source.
2. Procedure *path-rate_estimation()* assigns each node with its observations to a processor to infer path loss rates. The iterative approximating procedure used in [9] can be used here to estimate the loss rates of various pathes in the tree.
3. Except for node 1, the results obtained by other internal nodes are the loss rates of the corresponding paths. Therefore, after completing step 3, we can either use Theorem 1 or Theorem 2 to derive the loss rates for all internal links. The procedure called *link-rate_calculation()* is used to carry out this operation.

4 Simulation Result

To demonstrate the correctness of the formulas derived in the last section, we conducted a series of tests on a simulation environment built on *ns2* [11] that

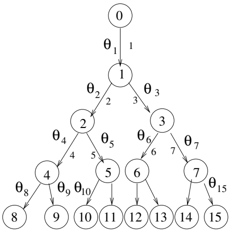


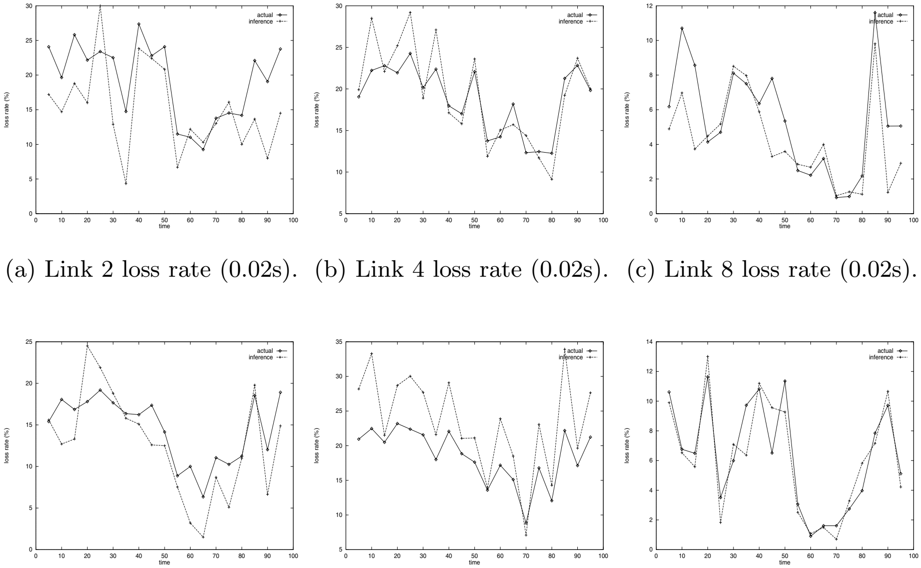
Fig. 4. Simulation Structure

has 16 nodes connected by 15 links as shown in Figure 4. Link 1, 2 and 3 have 3Mbps of bandwidth, while the others have 1.5Mbps. Apart from link 1 that has 2ms as its propagation delay, all others have 10ms of propagation delay. Each node has a FIFO queue attached to temporarily store packets. The queue length of a leaf node is 10, while a non-leaf node has a queue with a limit of 20 packets. The droptail policy is employed by all nodes to handle congestion, i.e. when a queue is full, newly arrived packets were dropped. A combined TCP and UDP traffics are added at different nodes as background traffic, where the left hand subtree is heavily loaded, but the right one is lightly loaded. Probe packets were periodically multicasted from node 0, the source, to the receivers attached to the leaf nodes.

Two sets of experiments were carried out on the simulation environment, apart from the frequency used to send probes, one has its interval as 0.01s, the other has its interval as 0.02s. Apart from the intervals, these two set experiments are identical. This arrangement aims to study the impact of sampling frequency on inference accuracy. In addition, we divide time into slots, each has 5 seconds. Based on the observations obtained in a slot, we use the formulas previously derived to calculate the loss rate for each link. The inferred loss rate is compared against the actual loss rate obtained from the simulator to evaluate the accuracy of the proposed method.

Since the right subtree is lightly loaded, no loss has been observed in all time slots. Then, zero loss rates have been assigned to all links of the right subtree. While, the situation in the left tree is very different. Figures 5(a), 5(b) and 5(c) shows the inferred loss rates against the actual loss rates at link 2, link 4 and link 8, respectively, when the probing interval is set to 0.02 second. These three figures clearly show the inferred results correctly reflect the actual loss trend of the background traffic.

On the same traffic setting, we carried out the second round simulation by doubling the number of probes sent to the receivers. The inference results against actual losses are plotted in Figures 5(d), 5(e) and 5(f). Each of these figures is a bit different from its counterpart in the previous round, including the actual curves. The difference is created by more probes sent into the network. Although those extra probes only slightly increase the traffic on those links, the increase of traffic triggers protocol actions at different points and leads to the change of loss rates and phases. For instance, such an increase may cause TCP streams to reduce their



(a) Link 2 loss rate (0.02s). (b) Link 4 loss rate (0.02s). (c) Link 8 loss rate (0.02s).

(d) Link 2 loss rate (0.01s). (e) Link 4 loss rate (0.01s). (f) Link 8 loss rate (0.01s).

window sizes earlier, and due to the synchronization effect for all TCP streams because the droptail policy was used for buffer management, the estimated loss on link 4 in the second is marginally higher than the actual one. Despite of this, the curves plotted in the two set of figures have very strong similarity.

5 Conclusion

In this paper, we present a scheme that allow us to divide the inference task of network tomography into a number independent subtasks that can be executed in parallel. The advantage of this scheme relies on decomposition to achieve its scalability, in which receivers that are geographically closely located work together to find out the characteristics within the network connecting them.

In this paper, we present a distributed algorithm to speed up the inference of the link-level loss rate by end-to-end measurement. The algorithm is built on a divide-and-conquer approach with the excellent parallel nature that makes it scalable. Our simulations show this approach is workable and accurate. Currently, we are investing methods to control the number of probes used to create informative observations.

References

1. Felix: Independent monitoring for network survivality.
Technical report, ftp://ftp.bellcore.com/pub/mwg/felix/index.html.
2. Ipma: Internet performance measurement and analysis.
Technical report, http://www.merit.edu/ipma.

3. J. Mahdavi, V. Paxson, A. Adams, and M. Mathis. Creating a scalable architecture for internet measurement. In *INET'98*.
4. Surveyor. Technical report, http://io.advanced.org/surveyor.
5. R. Cáceres, N.G. Duffield, J. Horowitz, and D. Towsley. Multicast-based inference of network-internal loss characteristics. *IEEE Trans. on Information Theory*, 45, 1999.
6. R. Cáceres, N.G. Duffield, S.B. Moon, and D. Towsley. Inference of Internal Loss Rates in the MBone . In *IEEE/ISOC Global Internet'99*, 1999.
7. R. Cáceres, N.G. Duffield, S.B. Moon, and D. Towsley. Inferring link-level performance from end-to-end multicast measurements. Technical report, University of Massachusetts, 1999.
8. K. Harfoush, A. Bestavros, and J. Byers. Robust identification of shared losses using end-to-end unicast probes. In *Technical Report BUCS-2000-013*, Boston University, 2000.
9. M. Coates and R. Nowak. Unicast network tomography using EM algorthms. Technical Report TR-0004, Rice University, September 2000.
10. W. Zhu. Using Bayesian Networks on Network Tomography. *Computer Communications, Elsevier Science, B.V.*, 26(2), 2003.
11. The network simulator 2. Technical report, www.isi.edu/nsnam/ns2.

Evaluation of Interconnection Network Performance Under Heavy Non-uniform Loads*

C. Izu[1], J. Miguel-Alonso[2], and J.A. Gregorio[3]

[1] Department of Computer Science, The University of Adelaide, SA 5005 Australia
cruz@cs.adelaide.edu.au
[2] Dep. of Computer Architecture and Technology,
The University of the Basque Country, 20080 San Sebastian, Spain
miguel@si.ehu.es
[3] Computer Architecture Research Group,
Universidad de Cantabria, 39005 Santander, Spain
monaster@unican.es

Abstract. Many simulation-based performance studies of interconnection networks are carried out using synthetic workloads under the assumption of independent traffic sources. We show that this assumption, although useful for some traffic patterns, can lead to deceptive performance results for loads beyond saturation. Network throughput varies so much amongst the network nodes that average throughput does not reflect anymore network performance as a whole. We propose the utilization of burst synchronized traffic sources that better reflect the coupled behavior of parallel applications at high loads. A performance study of a restrictive injection mechanism is used to illustrate the different results obtained using independent and non-independent traffic sources.

1 Introduction

Methods to evaluate the performance of an interconnection network range from the construction and measurement of its hardware prototype, to the utilization of overly simplified simulations. During the first stages of a new interconnection project, a fast simulation environment is critical, because it allows researchers to test and tune their design. Once a good tradeoff between expected performance and cost has been attained, the design can be rounded off using more detailed simulators. The evaluation of expensive prototypes goes just before the manufacture (and, again, evaluation) of the final product. In all these stages, evaluation has to be done using some kind of workload that resembles, with the higher possible fidelity, the actual workload that will be processed by the final network.

For practical reasons, most studies are carried out using synthetic workloads, running a simulator for a large number of cycles (simulated time) to get performance re-

* This work has been done with the support of the Ministerio de Educación y Ciencia, Spain, under grants TIN2004-07440-C02-01 and TIN2004-07440-C02-02, and also by the Diputación Foral de Gipuzkoa under grant OF-846/2004.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 396 – 405, 2005.

sults with the network in steady state. Although this may not be realistic, we consider the obtained results as indicators of the level of performance the network could provide under real conditions. For some SPLASH applications such as Radix or LU, it has been shown to be a reasonable approach [11].

A synthetic workload is defined by three parameters: the injection process, the spatial traffic pattern and the message size [4]. This can be done in a per-node basis, although very often all nodes share the same behavior. The spatial pattern determines the distribution of destinations for each source node. The injection process determines the temporal distribution (in other words, when a packet is generated). The size distribution determines the message length.

Traffic patterns include permutations such as bit-reversal or matrix transpose, uniform (also called random) and hot-spot. Each of them represents a worst-case scenario: uniform has no locality, permutations make an uneven use of resources, and hot-spot models nodes that receive a higher proportion of the traffic.

In general, we cannot assume that applications running on a parallel computer use fixed-size messages. However, networks often impose a maximum packet size and messages have to be segmented to fit in several of those packets. For this reason, most studies are done with fixed-size messages of 8-32 phits [2, 4, 11]. In some cases, message length follows a bimodal distribution which reflects network workload for a cc-NUMA system [10]. In this study, we will limit our discussion and experiments to fixed-size packets, although conclusions are valid for other length distributions.

Regarding the injection process, nodes are "programmed" to inject packets using some probability distribution (*independently* of the others). Injection times usually follow a Poisson or Bernoulli distribution, which are smooth over large time intervals. These widely used workloads treat each node as an independent traffic source (ITS).

The purpose of this paper is to show that performance results obtained with ITS for non-uniform traffic patterns under heavy loads process can be misleading because they do not reflect the way actual parallel applications make use of the communication subsystem: their processing nodes may advance tightly or loosely coupled, with all the possibilities in between *but they are never totally uncoupled*. To better illustrate this issue, we describe an experimental setup designed to evaluate the impact on network performance of a restrictive injection mechanism, and we compare the results obtained using ITS with those obtained using burst-synchronized traffic (BTS).

The rest of the paper is organized as follows. Section 2 defines all the relevant parameters of our experimental setup. Section 3 presents, discuss and compare the disparate results obtained using independent and non-independent traffic sources. Section 4 summarizes the findings of this work.

2 Evaluation Environment

In this section we define the experimental setup used to illustrate the impact of the choice of synthetic workload (focusing on the injection process) on the simulation results. First, we present the interconnection network as modeled for this study. Then we describe the context in which the injection process is analyzed, and the rest of simulation parameters.

2.1 The Simulated Network

For this work we use FSIN (Functional Simulator for Interconnection Networks), an in-house simulator, developed to simulate k-ary n-cube networks based on virtual cut-through (VCT) router architectures.

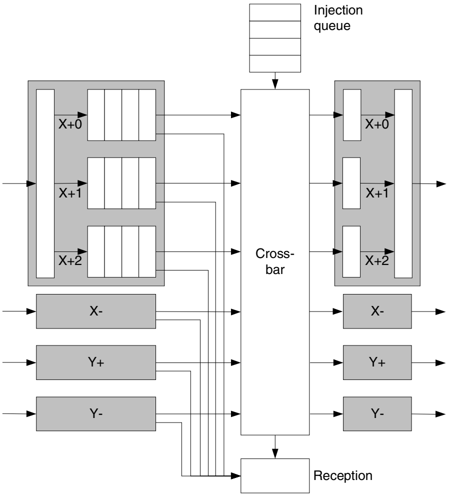


Fig. 1. Architecture of the adaptive VCT router used in the experiments

Fig. 1 shows the architecture of an adaptive virtual cut-through (VCT) router. It uses three VCs per physical channel, to map a deadlock-free oblivious (dimension-order routing) sub-network and a minimal adaptive sub-network. Each VC has a buffer with capacity for 8 packets (128 phits). One of the VCs is used for the escape sub-network, relying on Bubble Flow Control (BFC) [11] to avoid deadlock in each dimension. The adaptive sub-network uses the other two virtual channels. Any blocked packet in the adaptive sub-network can resort to an escape path to break a potential deadlock cycle [6]. Such combination provides low-cost, deadlock-free adaptive routing.

In order to reduce the number of figures and better focus our discussions, in this paper we show results for a 32x32 tori. However, conclusions are valid for other network configurations.

2.2 The Evaluation Context

The choice of synthetic workload has a definite influence on any kind of performance experiment we may carry out. In order to be more specific, and to show this influence in a particular (but relevant) context, we describe an experimental setup that was used to study the advantages of implementing restrictive injection techniques to prevent network congestion

Congestion control mechanisms limit injection when the network reaches a given level of congestion, which can be estimated locally or globally. In this paper, we apply a local method called *in-transit-priority restriction* (IPR): for a given fraction P of cycles, priority is given to in-transit traffic; in those cycles, injection of a new packet is only allowed if it does not compete with packets already in the network. P may vary from 0 (no restriction) to 1 (absolute priority to in-transit traffic), although in this paper we will consider only the two extreme cases. This method is used in IBM's BG/L [1] and in the Alpha 21364 network [8]. A more detailed discussion of congestion control mechanisms can be found in [7].

When studying congestion, which appears at high loads, the main figure of merit is the maximum sustained throughput for loads beyond saturation. However, unexpected results lead us to examine throughput figures in more detail and identify a significant level of throughput unfairness, which rends average values to be meaningless. That finding lead us to redefine the temporal distribution of packets for the synthetic workloads used in the experiments, as reported in the next section.

2.3 Network Workload

We have considered fixed-size packets of 16 phits. The traffic patterns used in the experiments are:

- **UN**: uniform traffic. Each node selects destinations randomly in a packet-by-packet basis.
- **TR**: transpose permutation. In a 2-D network, the node with coordinates (x, y) communicates with node (y, x).
- **SH**: perfect-shuffle permutation. The node with binary coordinates (a_{k-1}, a_{k-2}, ..., a_1, a_0) communicates with node (a_{k-2}, a_{k-3}, ..., a_0, a_{k-1})—i.e., rotate left 1 bit.

We use two types of injection processes:

- **Normal**: independent traffic sources, each one following a Bernoulli distribution with a parameter that depends on the applied load. This load is varied from 0 to 1 phit/cycle/node. The simulator runs for a warm-up period of 100,000 cycles, plus a measurement period of 100,000 cycles.
- **Burst-synchronized**: non-independent sources, to reflect the synchronized nature of parallel applications. The injection method is similar to that described in [2]. The same workload (*b* packets) is assigned to each source of traffic. A burst starts with an empty network. Nodes inject their *b* packets as fast as the network accepts them. The burst ends when all packets of all the traffic-generating nodes have been consumed. In the experiments, the simulator runs for 5 bursts of 1K packets.

3 Performance for Independent and Burst-Synchronized Traffic Sources

Most interconnection network simulators model the processing nodes as ITS which are continuously generating packets. Network performance is reported using two figures: latency (time from packet generation until its delivery) and throughput, which is measured as the number of packets delivered in a given time interval divided by the interval length and the network size. In other words, this is the average load accepted by the network (i.e., the network throughput), which is expected to be even amongst the network nodes.

In this section we will show such expectation is incorrect for non-uniform loads once the network has reached saturation, and we will question the validness of average throughput as the figure of merit under heavy loads. The evaluation of the impact that a restrictive injection mechanism (IPR) has on the performance of an adaptive VCT torus network is provided only to illustrate this issue. We could have selected different router architecture, topology or congestion-control mechanism. It would not matter because conclusions would be the same: throughput under non-uniform patterns for loads beyond saturation varies widely amongst the network nodes.

3.1 Network Performance Under Independent Traffic Sources

Fig. 2 represents network performance under three different traffic patterns (UN, TR and SH), with and without IPR, using a typical plot of average throughput versus applied load.

For the UN pattern, results show that utilization of a restrictive injection mechanism eliminates the throughput loss for loads beyond congestion. However, we cannot extend this conclusion to the permutations. In fact, results indicate that restrictive injection is counterproductive for TR and SH traffic under heavy load. This result was unexpected as non-uniform loads suffer more from congestion than UN, so we expect restrictive injection should be more effective, not less.

Another indicator of network performance is channel utilization: the higher the channel utilization, the better, because more resources are being productive. Let us focus on TR traffic without/with IPR. Fig. 2 indicates that, in saturation, throughput is higher without IPR. However, simulation results also indicate that channel utilization is higher with IPR. Which figure of merit is correct? How can channel utilization increase while delivering fewer packets? Does IPR increase performance, or not?

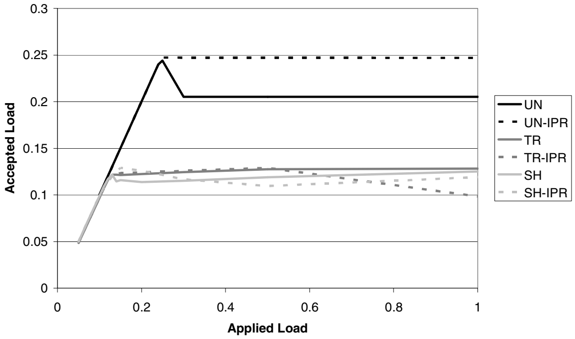


Fig. 2. Applied load vs. throughput (phits/cycle/node) for UN, TR and SH patterns, without/with IPR

3.2 Discussion of Performance Figures Under Independent Traffic Sources

In [4], Dally & Towles suggested that performance of a network for a given traffic pattern in which the node injection rate is not the same for all nodes should be reported as the lowest injection rate that matches the desired workload.

Following this approach, in Table 1 we report maximum, minimum and average injection rates for the six configurations under study. Notice the vast differences between these values for the TR and SH permutations.

Table 1. Maximum, minimum and average network throughput (phits/cycle/node), for applied loads beyond saturation, for UN, TR and SH patterns, without/with IPR

	UN		*TR*		*SH*	
	IPR off	*IPR on*	*IPR off*	*IPR on*	*IPR off*	*IPR on*
Max.	0,219	0,267	0,559	0,716	0,973	0,974
Min.	0,194	0,217	0,013	0,000	0,002	0,000
Avg.	0,205	0,243	0,132	0,098	0,125	0,119

Such large variations of throughput under TR and SH permutation patterns were also observed in other popular IN simulators such as Flexsim [12] and the Chaos simulator [3] for a range of network designs. Dally & Towles [4] state that average and minimum rate differ in some routers due to their unfair design, citing the chaos router with prioritizes traffic in its internal queue over incoming or new packets as an example of that unfairness.

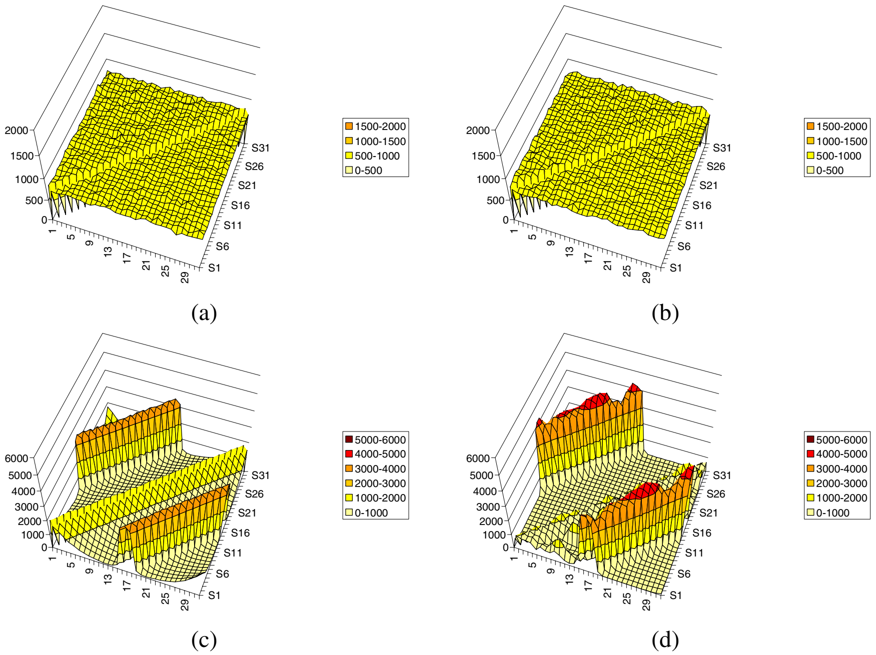
We should note that the time a packet awaits in an injection buffer before entering the network depends not only on the arbitration method, but also on the local router state. Under UN traffic, the network load is evenly distributed, so that all nodes have a similar view of network status and are able to inject packets at a similar rate. However, under non-uniform loads the degree of utilization of resources (buffers, output channels) may vary widely from one router to another. Therefore, at high loads, nodes connected to busy routers[1] have lower chances to inject their load than nodes in less used areas—a difference that causes wide variations in the number of packets injected by each node. In other words, the differences shown in Table 1 are not caused by an unfair routing or arbitration method, but by the fact that network resources are used unevenly by the applied workload, which is the case for all non-uniform loads.

Let us focus again on the TR permutation. In a 32x32 network, and assuming that all nodes inject at the same rate, the average distance packets traverse is 16.516. In fact, this is what the simulator reported when network load was below its saturation point. For those loads the map of packets injected per node is flat (except the nodes in the diagonal, which do not generate traffic for themselves), as shown in Fig. 3a.

The scenario changes drastically when saturation is reached. The simulator reflects this in a change in average distance (17.12) and in a very different map of injected packets (Fig. 3c). Note we have change nothing but the applied load. The problem is that some nodes can inject packets in their routers at very high rates, while others can hardly access the network because their routers devote most resources to passing-by packets. Fig. 3c shows that "lucky" nodes (those that have more opportunities to inject packets) are located close to the diagonal and in a pair of bands parallel to it. It gives the impression that the network is *unfair* for TR traffic.

We are interested to know why adding the IPR congestion control mechanism appears not to be beneficial in this scenario. Network response does not change below

[1] "Busy" routers are those that are traversed by numerous in-transit packets.



(a)

(b)

(c)

(d)

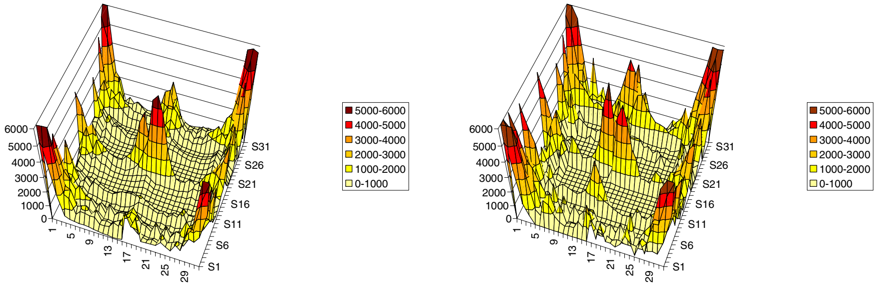
Fig. 3. Maps of injected packets for TR traffic. Each surface point (x, y) represents the number of packets a node with coordinates (x, y) injected in 100.000 cycles. (a) Below saturation, no IPR. (b) below saturation, IPR. (c) beyond saturation, no IPR. (d) beyond saturation, IPR.

saturation (Fig. 3b) but for loads beyond saturation network unfairness is worst as shown in Fig. 3d: the “lucky” area close to the diagonal shrinks, and the two parallel bands are narrower and taller than without IPR[2]. As nodes in these bands are injecting packets addressed to distant destinations, the average distance rises up to 23.47. In other words, IPR magnifies the fairness problem.

For other non-uniform workloads results are similar. As an example, Fig. 4 shows the maps of injected packets for the SH pattern for loads beyond saturation, without and with IPR.

In conclusion, simulations report again that the implementation of a congestion-control mechanism is counterproductive for all traffic patterns under study—except for UN. Although IPR increases channel utilization, the number of packets delivered per cycle diminishes. This unexpected result is explained by the fact that network unfairness favours packets that travel longer paths.

[2] A digression of interest: we have a collection of nodes capable of injecting more than 4000 packets in 100.000 cycles, while some others are unable to inject a single one (they suffer starvation).. Any router that imposes restrictions to the injection of new packets may suffer from starvation. Although the adaptive router (without IPR) is starvation-free, it exhibits a high degree of unfairness under non-uniform traffic.



(a)

(b)

Fig. 4. Maps of injected packets for SH traffic beyond saturation. (a) IPR off. (b) IPR on.

3.3 Performance of the Network Under Burst-Synchronized Traffic

The above conclusion could be considered correct as numerous previous works using this simulation methodology and workload. But luckily in this case we have several indicators (channel utilization, average distance and unfairness) that something is wrong. And what we think is wrong is the synthetic workload used.

Application processes are somehow *coupled*, because they work to perform a given task in a cooperative way. Most (if not all) applications use synchronization barriers, perform collective operations or use other mechanisms as described in [5] that make all the processes advance at a similar rate. It is true that worst-case performance for data exchanges is important (as shown in [9]) because it may halt progress of computation nodes, which are not able to perform additional operations, or communicate any further, until the data exchange has been completed. However, we cannot conceive a realistic scenario in which, *in the same parallel application*, a process is sending packets to its selected destination *ad infinitum* while other nodes do the same at a *much* smaller rate.

We consider burst-synchronized injection as described in section 2.3 to be a better alternative to model the communication structure of a parallel application at heavy loads. We have made a complete performance analysis similar to that reflected in Fig. 2, but using burst-synchronized traffic (BTS). Fig. 5 shows the time to complete 5 bursts of 1K packets for the six scenarios under consideration. For comparison purposes, Table 2 shows their throughput computed as the total workload delivered divided by the completion time. For this workload, maps of injected packets are meaningless (all nodes inject exactly the same number of packets), and the reported average distance traversed by packets is always the expected one[3]. Under burst-synchronized workload, the use of restricting injection policies is positive for the three traffic patterns: the time to deliver the 5 bursts of packets is lower with IPR than without it. As we expected, IPR is more effective for TR, a pattern that suffers badly from network congestion.

[3] In this context starvation is not an issue: if the network somehow favors some nodes, they will send their workload faster than others, but will eventually stop, allowing the rest of the nodes to progress faster, until all of them have sent their packets.

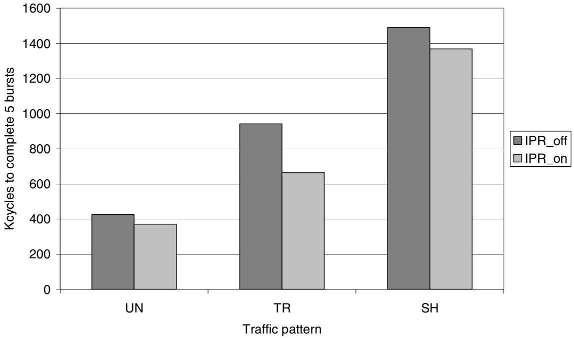


Fig. 5. Time to deliver 5 bursts of 1K packets for UN, TR and SH patterns, without/with IPR

Table 2. Network throughput (phits/cycle/node) averaged for 5 bursts under UN, TR and SH patterns, without/with IPR

UN		*TR*		*SH*	
IPR off	*IPR on*	*IPR off*	*IPR on*	*IPR off*	*IPR on*
0,192	0,220	0,087	0,123	0,055	0,060

The performance reported under BTS contradicts the results obtained under ITS. Which one is *correct*? As both are based on synthetic workloads, both are just approximations to the reality. But the behavior of real parallel applications at heavy loads is clearly closer to the burst-synchronized source model than to the independent source model. In fact, tests carried out with real applications show that this congestion control mechanism does improve throughput under heavy loads.

4 Conclusions and Future Work

Performance of interconnection networks is evaluated using a widely accepted set of synthetic workloads which model uniform, hot spot and traffic permutation patterns. Each node generates packets independently following a Poisson or Bernoulli distribution.

Evaluation of a congestion control mechanism using these workloads lead us to identify the vast differences in network throughput observed by each processing node at heavy non-uniform loads. This network unfairness is not caused by the mechanism itself but by the uneven nature of the workload. Consequently, we question the validity of average peak throughput as the figure of merit under non-uniform heavy loads and independent traffic sources (ITS). In fact, changing the injection model to burst synchronized sources (BTS), a workload closer to the pattern generated by real parallel applications, leads to different conclusions about the goodness of that congestion control mechanism under non-uniform loads.

In short, the ITS model fails to reflect the communication behavior of loosely coupled parallel applications. This leads to incorrect conclusions when evaluating *any* router mechanism at loads beyond saturation. BTS is used instead to model the synchronized behavior exhibited by coupled parallel applications at high loads.

We are conscious that further characterization of application workloads is needed to guide the development of synthetic workloads that reflect the communication structure (various levels of message coupling) that exist in most parallel applications.

References

[1] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, P. Heidelberger, S. Singh, B. Steinmacher-Burrow, T. Takken, P. Vranas. “Design and Analysis of the BlueGene/L Torus Interconnection Network” IBM Research Report RC23025 (W0312-022) December 3, 2003.

[2] T. J. Callahan, S.C. Goldstein. “NIFDY: A Low Overhead, High Throughput Network Interface”. Proc. of the 22nd Annual International Symposium on Computer Architecture, ISCA '95, Santa Margherita Ligure, Italy. pp. 230-241, June, 1995

[3] The Chaotic Routing Project at the U. of Washington. Chaos Router Simulator. Available at http://www.cs.washington.edu/research/projects/lis/chaos/www/chaos.html

[4] W.J. Dally & B. Towles. Principles and Practices on Interconnection Networks. Morgan Kaufmann, 2004.

[5] P. Dinda, B. Garcia, K. Leung, The Measured Network Traffic of Compiler-Parallelized Programs, Proceedings of the 30th International Conference on Parallel Processing (ICPP 2001) pp 175-184..

[6] J. Duato. “A Necessary and Sufficient Condition for Deadlock-Free Routing in Cut-Through and Store-and-Forward Networks”. IEEE Trans. on Parallel and Distributed Systems, vol. 7, no. 8, pp. 841-854, 1996.

[7] C. Izu, J. Miguel-Alonso, J.A. Gregorio. "Packet Injection Mechanisms and their Impact on Network Throughput". Technical report EHU-KAT-IK-01-05. Department of Computer Architecture and Technology, The University of the Basque Country. Available at http://www.sc.ehu.es/acwmialj/papers/ehu_kat_ik_01_05.pdf.

[8] S. Mukherjee, P. Bannon, S. Lang, A. Spink and David Webb, “The Alpha 21364 Network Architecture”, IEEE Micro v. 21, n. 1 pp 26-35, 2002.

[9] F. Petrini, D. Kerbyson and S. Pakin. “The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q”. In IEEE/ACM SC2003, Phoenix, AZ, November 2003.

[10] V. Puente, J.A. Gregorio, R. Beivide and C. Izu, “On the Design of a High-Performance Adaptive Router for CC-NUMA Multiprocessors”, IEEE Trans. on Parallel and Distributed Systems, Vol. 14, NO. 5, May 2003.

[11] V. Puente, C. Izu, R. Beivide, J.A. Gregorio, F. Vallejo and J.M. Prellezo (2001). “The Adaptative Bubble Router”. Journal of Parallel and Distributed Computing. Vol 61 - n. 9.

[12] SMART group at the U. of Southern California. FlexSim 1.2. Available at http://ceng.usc.edu/smart/FlexSim/flexsim.html

Analytical Models of Probability Distributions for MPI Point-to-Point Communication Times on Distributed Memory Parallel Computers

D.A. Grove and P.D. Coddington

School of Computer Science, University of Adelaide,
Adelaide, SA 5005, Australia
paulc@cs.adelaide.edu.au

Abstract. Measurement and modelling of distributions of data communication times is commonly done for telecommunication networks, but this has not previously been done for message passing communications on parallel computers. We have used the MPIBench program to measure distributions of point-to-point MPI communication times for two different parallel computers, with a low-end Ethernet network and a high-end Quadrics network respectively. Here we present and discuss the results of efforts to fit the measured distributions with standard probability distribution functions such as exponential, lognormal, Erlang, gamma, Pearson 5 and Weibull distributions.

1 Introduction

There has been a lot of research on measuring and modelling network traffic in telephone networks, wide-area networks such as the Internet, and local area (e.g. Ethernet) networks. However, there has been comparatively little work on measuring and modelling message passing communications for parallel computer networks. In particular, we are not aware of any previous work on the measurement and modelling of the distribution of point-to-point communication times for parallel computers due to the effects of contention in the communication network. Modelling of message passing times on a parallel computer is typically focussed on average communication times, and standard programs for benchmarking the communications performance of message passing routines (such as Mpptest, MPBench and SKaMPI) provide only average communication times for point-to-point communications between two processors, which does not give any indication of the effects of contention in the network.

We developed a new MPI communications benchmark called MPIBench [1,2], which provides highly accurate and detailed benchmarks of the performance of MPI message-passing routines. MPIBench can generate histograms that show the distribution of completion times for individual MPI communication routines, not just average times. It also takes into account contention effects by running point-to-point communications on N processors, with processor p communicating with processor (p+N/2) mod N. This provides greater insight into the performance of

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 406–415, 2005.

MPI routines and parallel programs that use MPI, since in some situations the variance and the tail of the distribution are just as important as the average.

The histograms generated by MPIBench are a measure of the probability distribution function (PDF) for MPI communication times. These PDFs can be used to provide more accurate modelling and estimation of the performance of parallel programs. Standard techniques for performance modelling of parallel programs use average communication times, thereby ignoring the variation due to contention. We have developed a system called the Performance Evaluating Virtual Parallel Machine (PEVPM) [3] that samples from PDFs generated by MPIBench to more accurately model message passing communications times, and we have found that this provides more accurate estimates of parallel program performance than using average communication times.

In this paper, we provide the first examination of the quantitative nature of the observed PDFs for point-to-point (send/receive) communications on parallel computers, in order to determine if and how they can be characterised by analytical models. Good analytical models would be very useful, for modelling the performance of parallel programs using tools such as PEVPM, and also to provide a deeper general understanding of the performance characteristics of message passing communication on parallel computers.

2 Analytical Models

Some example PDFs of message passing communication times that have been measured using MPIBench are presented in Figures 1-4. Figures 1 and 2 show results from Perseus, a commodity Linux PC cluster with a switched Fast Ethernet network. Figures 3 and 4 show results from the APAC SC, an AlphaServer SC with a Quadrics QsNet communications network. Note that the measured distributions are quite noisy. They usually become smoother with increasing number of measurements, although this takes correspondingly longer to run the benchmarks. Many more examples of probability distributions of communication performance for a variety of parallel computers, data sizes and numbers of processors can be found in Grove's thesis [2].

Qualitatively, the distributions have a hard lower bound, usually a normal-shaped middle and taper out with an unbounded tail. The lower bound is determined by the minimum message latency that is possible under perfect conditions. The shape of the middle-part of the curve is determined by contention effects. In reality, the right-hand tail does not actually extend to infinity because of the discrete nature of the distribution and protocol timeouts, however in the analytical models, the probabilities associated with the tail become astronomically small very quickly.

A number of common distribution functions exhibit these broad properties. These include exponential, Erlang, gamma, Pearson 5, lognormal and Weibull distributions [4]. Unlike the normal distribution, these distributions are asymmetric in general and cannot be distinguished by their mean and variance alone. In addition to mean and variance, which are also known as the first and second

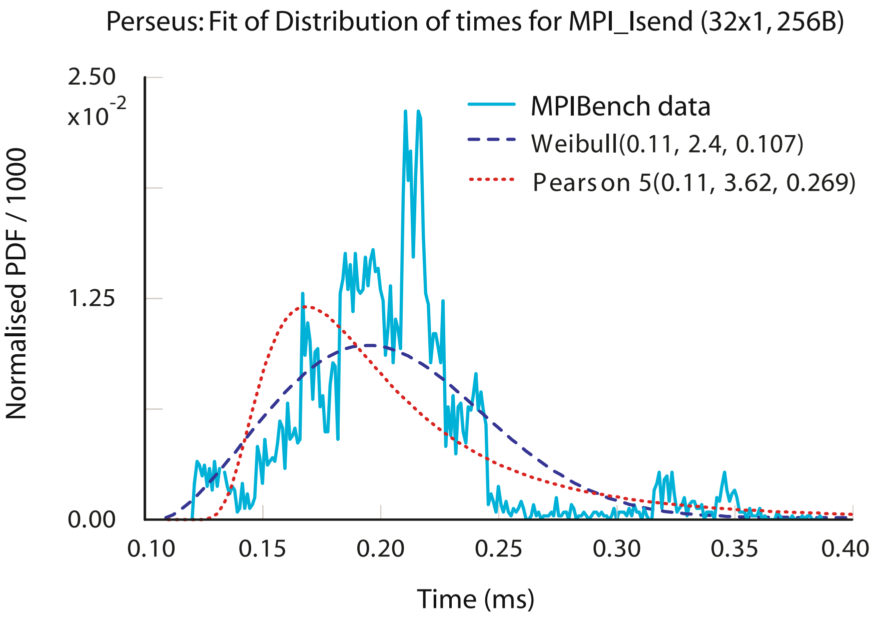


Fig. 1. Pearson 5- and Weibull-fitted performance profiles for 512 byte `MPI_Isend` messages with 32x1 processes on Perseus

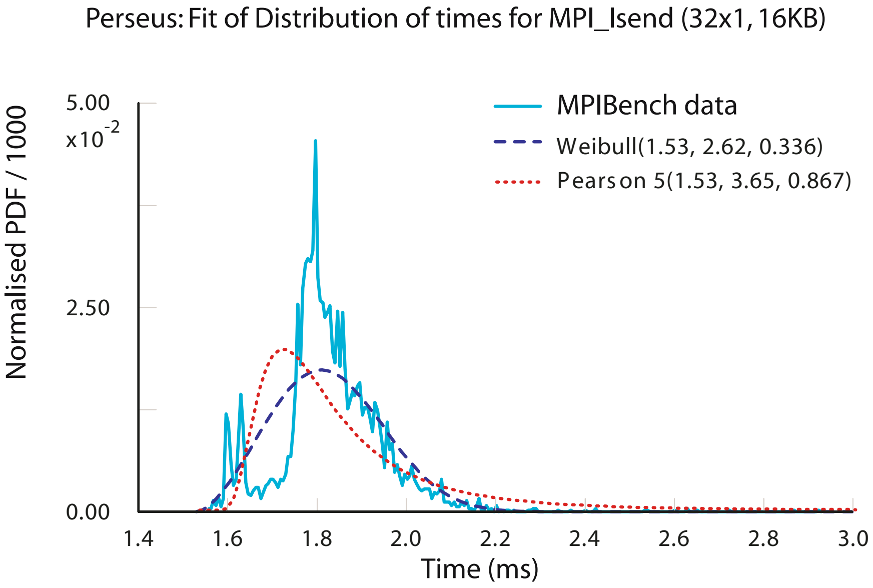


Fig. 2. Pearson 5- and Weibull-fitted performance profiles for 16 Kbyte `MPI_Isend` messages with 32x1 processes on Perseus

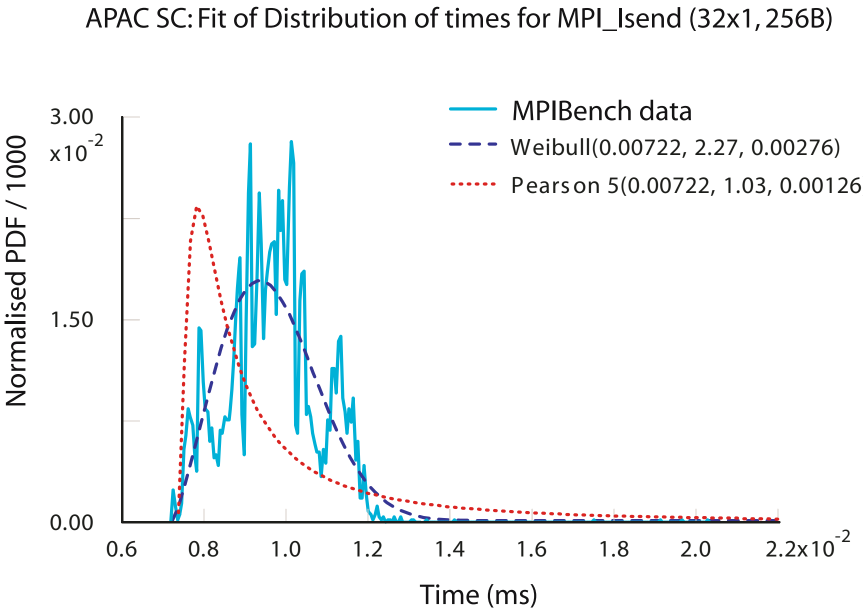


Fig. 3. Pearson 5- and Weibull-fitted performance profiles for 512 byte MPI_Isend messages with 32x1 processes on APAC SC

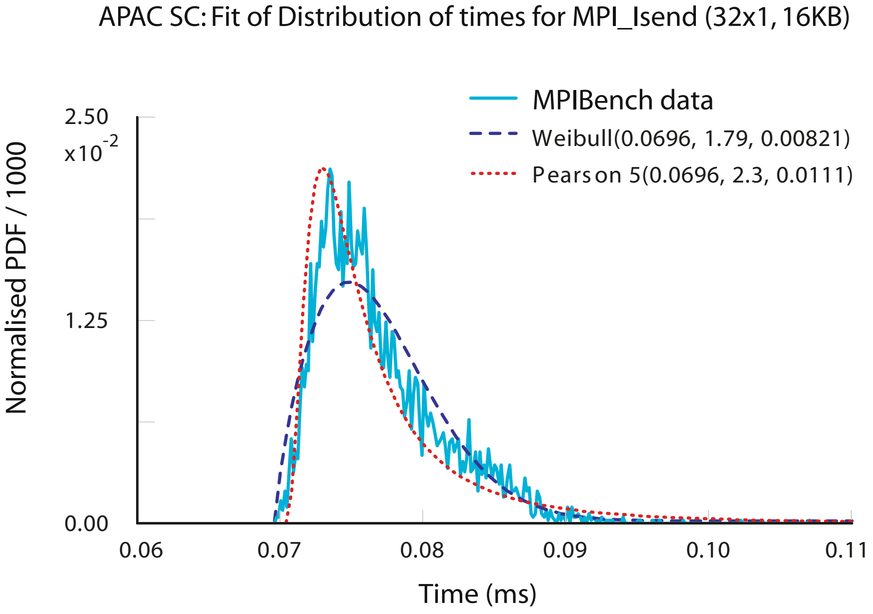


Fig. 4. Pearson 5- and Weibull-fitted performance profiles for 16 Kbyte MPI_Isend messages with 32x1 processes on APAC SC

moments of a distribution, these distributions must be differentiated by their third and fourth order moments, known as skewness and kurtosis respectively. The skewness statistic describes the degree of symmetry of a distribution. A positively skewed (right-skewed) distribution rises rapidly, reaches its maximum and falls slowly with a pronounced right-tail. A negatively skewed (left-skewed) distribution rises slowly reaches through a pronounced left-tail, reaches its maximum and falls rapidly. The kurtosis statistic describes the peakedness/flatness of a distribution near its mode, relative to the normal distribution.

The distribution functions listed above are defined by at most three parameters, usually known as the scale parameter, the shape parameter and the location parameter. The scale parameter defines where the bulk of the distribution lies, or how stretched out the distribution is. In the case of the normal distribution, the scale parameter is the standard deviation. Unsurprisingly, the shape parameter defines the shape of a distribution. Some distributions, for example the normal distribution, do not have a shape parameter because they have a predefined shape that does not change. Finally, the location parameter shifts the origin of a distribution either left or right. Without a location parameter (or with a location parameter of zero) all of the distributions listed above have a domain of $(0,\infty]$ so the location parameter can be used to model the lower bound on message latency. Determining which scale and shape parameters should be used to model the PDF of communication performance is less clear.

Rather than blindly trying to fit observed data to known analytical distributions, it is more useful to first examine how the assumptions of those analytical expressions mesh with the underlying traffic patterns and contention that are fundamental to message passing programming on distributed memory parallel computers. Historically, the most frequently used model for the time instants at which events are observed has been the Poisson process. In particular, this model is used in the telecommunications industry to model the interarrival and service times of telephone calls. From these roots, it has been commonly applied to modelling data transmission in computer networks. A Poisson process is characterised by a sequence of randomly spaced events, where the arrival time of the next event is independent of, but probabilistically like, the time of the previous event. The Poisson distribution gives the probability that a given number of events will occur within a certain time interval. In relation to a communication network, when a large number of packet arrival events occur in a short period of time (due to the inherent randomness of interarrival times) communication buffers will become very full. Hence the time that a packet can spend waiting for transmission can be large. With this in mind, a Poisson process provides a model of network contention that can be used to determine the interarrival time and service time (i.e. end-to-end latency) of message-passing operations. Interarrival times and service times of a Poisson process are both exponentially distributed.

The Poisson process provides an attractive modelling formalism because it has a number of properties that greatly simplify its evaluation. However it fails as a realistic model for network traffic, and in particular message-passing traffic for parallel programs, because the assumption of independent communications events is not true. Data communication is often very bursty and is self-similar in nature [5,6,7]. Message-passing programs, due to their frequent synchronisation (either explicit or implicit), are even more so. This means that contention between seemingly unrelated processes is not truly independent. For this reason, researchers have suggested that Poisson processes are inappropriate for modelling data communication. A number of recent studies, mostly focussed on wide-area networks, have found that service times for data traffic are much better modelled by heavy-tailed distributions such as Erlang, lognormal or Weibull distributions [8,9].

The Erlang distribution [4] was initially developed to model the workings of telephone exchanges. It was specifically designed to model the situation where the likelihood of immediate process completion increases with the amount of processing that has already been done. In particular it describes the waiting time until the m^{th} event of a process that occurs randomly over time. This makes the Erlang distribution particularly good at modelling transmission times in the face of contention. Erlang distributions are defined by their location parameter x, positive integer shape factor m and scale parameter β. The case of $m = 1$ reduces an Erlang distribution to an exponential.

The Erlang distribution is actually a special case of the gamma distribution, which is identical, except that the shape factor m may take on non-integer values. Also related to the gamma distribution is the Pearson 5 distribution, which is sometimes called the inverse gamma distribution, since there is a reciprocal relationship between a Pearson 5 random variable and a gamma random variable [4]. The Pearson 5 distribution is particularly useful for modelling time delays where some minimum delay value is almost assured and the maximum time is unbounded and variably long [10]. This makes it an attractive candidate for modelling message-passing time.

The lognormal distribution [4] results from the product of many independent random variables, where overall distribution values are based on the cumulative effect of many small perturbations in those variables. This theoretical underpinning also fits well with the idea of contention, where mutually excluded access to shared resources can increase the chance of further contention, thus causing increasingly lengthy delays. The lognormal distribution looks like a normal curve that has been right-skewed, and has a finite lower bound.

Both the gamma family (including Erlang and Pearson 5) and lognormal processes provide (different) potential theoretical models for the effects of random contention on message-passing service times. However, a lack of strict randomness in the underlying process being modelled (in this case contention) could lead to negatively skewed data, which cannot be fit by either gamma family or lognormal models. The Weibull distribution is a very versatile, general-purpose distribution that can be used in these cases [4]. Depending on the values of the parameters, the Weibull distribution can be used to model a variety of behaviours. For example, setting the scale parameter $\beta = 1$, the Weibull distribution reduces to an exponential distribution; $\beta < 1$ produces a exponential-like curve, except that it begins higher and diminishes faster. Using $1 < \beta < 3.6$ results in a distribution that looks much like a gamma or lognormal, i.e. monotonically rising until the mode, and then monotonically decreasing with a pronounced right-tail. For $\beta = 3.6$ the coefficient of skewness approaches zero, and the curve approximates a normal distribution, but with a finite lower bound. Uniquely, for $\beta > 3.6$ the distribution is negatively skewed, i.e. most data is found in the right-hand side of the distribution, despite a left-bounded tail.

All of these distributions are special cases of a three-parameter distribution called the generalised gamma function [4], however this function is not often used in modelling due to its complexity.

3 Comparison of Different Distribution Functions

The data from a broad selection of distributions generated by MPIBench were input into a statistics program called Stat::Fit [11] and analysed to determine which of the analytical distribution(s) listed above could best describe them. Stat::Fit is very simple to use, in particular via its Auto::Fit function which automatically fits data to different distributions, provides an absolute measure of each distribution's acceptability and ranks the results. Stat::Fit uses Maximum Likelihood Estimation (MLE) [12] for parameter estimation, which determines the parameter values that maximise the probability of obtaining the sample data. MLE is considered the most accurate parameter estimation method for 100 or more samples. Stat::Fit uses χ^2, Kolmogorov-Smirnov and Anderson-Darling tests to provide goodness of fit measures. Notably, the Kolmogorov-Smirnov test provides the best metric over a wide range of distributions and the Anderson-Darling test provides the best metric for heavy-tailed distributions [13]. Importantly, it is known that all of these tests can become too sensitive for a large number (say more than 1000) data points and thus occasionally reject proposed distributions that in reality provide useful fits [11].

For each investigated distribution, 500 data points were randomly selected from the measured results from MPIBench and used for fitting. Auto::Fit MLE and goodness of fit analyses were performed. Tables 1 and 2 show goodness of fit results for Perseus and the APAC SC for different distribution functions for three different message sizes and different numbers of nodes (with 1 process running

Table 1. Results of automated fits to MPIBench communications times from Perseus. Lower values indicate better fits. F indicates that the automated fit function failed to make an acceptable fit.

data size	128 bytes				4K bytes				16K bytes			
nodes	2	16	32	average	2	16	32	average	2	16	32	average
erlang	0.293	0.160	0.101	0.180	0.78	F	F	F	2.59	F	141	F
exponential	0.293	0.210	0.183	0.229	68.2	F	F	F	62.2	F	126	F
gamma	0.262	0.16	0.101	0.174	0.78	F	F	F	2.45	F	66.6	F
lognormal	0.309	0.176	0.0879	0.191	0.84	54.0	69.8	41.5	2.59	30.1	8.75	13.8
pearson5	0.368	0.282	0.173	0.274	1.73	56.5	108	55.4	2.88	32.5	6.87	14.1
weibull	0.238	0.141	0.118	0.166	0.78	36.2	27.0	21.3	2.09	22.9	38.1	21.0

Table 2. Results of automated fits to MPIBench communications times from the APAC SC. Lower values indicate better fits. F indicates that the automated fit function failed to make an acceptable fit.

data size	128 bytes				4K bytes				16K bytes			
nodes	2	16	32	average	2	16	32	average	2	16	32	average
erlang	F	F	F	F	62.4	F	0.98	F	21.4	1.09	1.51	14.0
exponential	F	F	F	F	67.0	66.2	77.1	70.1	42.2	19.0	24.5	28.6
gamma	F	F	F	F	50.7	2.37	0.96	18.0	22.0	19.0	1.52	14.2
lognormal	0.334	0.131	0.146	0.204	54.4	2.01	0.93	19.1	24.7	5.94	1.9	10.8
pearson5	0.336	0.171	0.238	0.248	63.0	38.8	1.48	34.4	25.4	3.50	1.95	10.3
weibull	0.331	0.0961	0.095	0.174	55.6	1.66	1.01	19.4	21.2	17.7	1.95	13.6

per node). The Kolmogorov-Smirnov test was used for the smallest message size, and the Anderson-Darling test for the other message sizes (note that the two tests have very different fit metrics). An F indicates that the automated fit function failed to make an acceptable fit. Examples of some fits that were obtained for small and large message sizes for 32 processes are plotted in Figures 1-4.

Significantly, the results support a common interpretation for the behaviour of point-to-point message-passing performance on both of the machines that were examined. The performance distributions observed for the smallest message size are essentially normal-shaped, although they necessarily have a bounded lower limit. The best fit in each of these cases was provided by a Weibull distribution with a shape parameter near 3.6 (i.e. close to normal), although lognormal and Pearson 5 distributions also provide a reasonable approximation. Importantly, however, the standard deviation for each of these distributions is comparatively small, about half of the minimum message latency for a zero byte message on the same system. These normal-shaped distributions are consistent with random rather than contention delays, for example caused during context switching, polling for message arrivals or physical transmission.

For larger messages where contention is more prevalent, the Pearson 5 and lognnormal distributions provided the best fits. In comparison, Weibull (or other) distributions could not be used because they were too heavy-tailed and lacked the peakedness to fit the observed data well.

Most of the cases where some functions failed to give acceptable fits are distributions with slightly negative skew. Weibull fits these quite well, and lognormal and Pearson 5 are able to give acceptable fits.

4 Conclusions and Further Work

MPIBench allows, for the first time, the measurement of probability distributions of message passing communication times on parallel computers. This provides useful insight into the variability of communication times due to contention effects, and also allows for more accurate modelling of the performance of parallel programs than is achievable by just using averages of communication times.

It is interesting to investigate what kinds of analytical functions best describe the measured distributions. This is commonly done for telecommunications networks, but to our knowledge has never been done for message passing communications on parallel computers.

For small message sizes where contention effects should be negligible, communication times follow a Weibull distribution that is close to normal but with a bounded minimum time, indicating random rather than contention delays. The lognormal and Pearson 5 distributions also provide reasonable fits where there is low contention.

Increasing the message size or the number of processes increases the level of contention and creates a more heavily skewed distribution. The Pearson 5 or lognormal distributions best fit these results, although from preliminary studies of additional data, Pearson 5 seems to give better results when the contention level is high and the distributions are broader. For both the low-end (Fast Ethernet) and high-end (QsNet) communication networks that we have examined, it appears that the performance variation in message-passing time under a normal contention level can be explained based on the roots of the Pearson 5 distribution; i.e. variation occurs as the result of a transmission process that has a high chance of succeeding in minimum time, yet has a small chance of being continually delayed, in this case due to contention in the communications network.

The lognormal and Pearson 5 distributions provide the most accurate fits over the broadest spectrum of conditions. Even in the few cases where a Weibull distribution would provide a better fit, Pearson 5 and lognormal are an acceptable alternative to use for modelling communications time for applications such as PEVPM that provide performance prediction of parallel programs.

In future work, we will undertake a more detailed statistical analysis of a wider variety of data, with different message sizes, numbers of processes, numbers of processes per node for clusters consisting of multi-processor nodes, and different networks and parallel architectures, including shared memory machines.

We are working on changing MPIBench so that it can produce distributions for each process, rather than the current approach of combining the results from all processes. We have seen situations where distributions appear to differ between processes, so we expect to get better fits to results from individual processes, as well as more detailed insight into message passing performance.

We also plan to extend MPIBench so that it can automatically fit measured data to appropriate distributions (e.g. Pearson 5 and/or lognormal and/or Weibull). This output would be particularly useful as input to systems such as PEVPM that could use these distributions to provide more accurate predictions of the performance of parallel programs.

It would be very useful to be able to develop a model that could estimate the parameters of a Pearson 5 or lognormal distribution as a function of known quantities such as the message size, the number of processes, and the latency and bandwidth of the network. This would enable us to use PEVPM to generate accurate estimates of the performance of parallel programs on machine configurations for which we cannot run MPIBench to generate measured distributions of communication times. For example, we could see how a parallel program would scale to very large numbers of processors, or how reducing the latency or increasing the bandwidth of the communications network would affect the performance of the program. This is the main goal of our future work on this project.

Acknowledgements

This research was supported in part by the Advanced Computational Systems and Research Data Networks Cooperative Research Centres, and a Merit Allocation Scheme award of time on the APAC National Facility.

References

1. Grove, D.A.: Precise MPI performance measurement using MPIBench. In: Proc. of HPC Asia. (2001)
2. Grove, D.A.: Performance Modelling of Message-Passing Parallel Programs. PhD thesis, University of Adelaide (2003)
3. Grove, D., Coddington, P.: Modeling message-passing programs with a Performance Evaluating Virtual Parallel Machine. Performance Evaluation **60** (2005) 165–187
4. Johnson, N., Kotz, S., Balakrishnan, N.: Continuous Univariate Distributions. Wiley, New York (1995)
5. Leland, W., Taqqu, M., Willinger, W., Wilson, D.: On the self-similar nature of Ethernet traffic (extended version). IEEE/ACM Transactions on Networking **2** (1994) 1–15
6. Park, K. and Willinger, W. (eds): Self-Similar Network Traffic and Performance Evaluation. Wiley, New York (2000)
7. Willinger, W., Taqqu, M., Sherman, R., Wilson, D.: Self-similarity through high-variability: Statistical analysis of Ethernet LAN traffic at the source level. IEEE/ ACM Transactions on Networking **5** (1997) 71–85
8. Feldman, A., Whitt, W.: Fitting mixtures of exponentials to long-tail distributions to analyze network performance models. Performance Evaluation **31** (1998) 963–976
9. Zwart, A.P.: Queueing Systems with Heavy Tails. PhD thesis, Eindhoven University of Technology (2001)
10. Law, A.M., Kelton, W.D.: Simulation Modeling & Analysis. McGraw-Hill, New York (1991)
11. Greer Mountain Software: (Stat::Fit software, version 1.1) Available from `http://www.geerms.com/`.
12. Dodson, B.: Weibull Analysis with Software. ASQ Quality Press, Milwaukee (1995)
13. Anderson, T., Darling, D.: Asymptotic theory of certain goodness-of-fit criteria based on stochastic processes. Annals of Mathematical Statistics **23** (1954) 193–212

Communication Data Multiplexing in Distributed Simulation*

Jong Sik Lee

School of Computer Science and Engineering,Inha University,
#253, YongHyun-Dong, Nam-Ku,
Incheon 402-751, South Korea
jslee@inha.ac.kr

Abstract. Complex and large-scale distributed systems are characterized by numerous interactive data communication among distributed components over network. This paper proposes a communication data multiplexing approach as an efficient message traffic reduction scheme. This approach encodes joint outputs of sender components into a single message and applies to a distributed system involving components moving and interacting in multi-dimensional space. For performance evaluation, this paper applies uses a projectile/missile case study with realistic multi-dimensional dynamics. This paper investigates variation of system accuracy and network bandwidth requirement, while a ratio of active components and a time granule are varied. Analytical and empirical data clearly indicate the advantages of multiplexing in saving communication resources in a large-scale distributed simulation. In addition, this paper discusses effectiveness and limitation of the multiplexing approach while considering the tradeoff between system accuracy and performance.

1 Introduction

Complex and large-scale distributed systems are characterized by numerous interactive data exchanges among entities distributed between computers networked together. A method to support the reduction of the interactive messages among entities is called a "message traffic reduction approach." [1] It is the goal of a message traffic reduction scheme that a large-scale distributed system executes within reasonable communication and computation resources. Current message traffic reduction approaches are Quantization [2], Predictive quantization [3], Multiplexing predictive quantization [4], and Interest-based quantization [5]. This paper proposes a dynamic multiplexing approach as an efficient message traffic reduction approach applicable to simulations involving large numbers of moving and interacting entities in multi-dimensional space. Using the multiplexing approach, this paper introduces an effective mean of transmitting messages that consist of samples from a multi-dimensional space. In order to study the strengths and limitations of multiplexing, we investigate the dependence of simulation accuracy and performance on the activity of sending components. An analysis suggests optimal ratios of active to non-active components and time granule sizes which were confirmed in a realistic experimental distributed simulation of missile-to-missile interaction. This

* This work was supported by INHA UNIVERSITY Research Grant.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 416 – 421, 2005.

paper is organized as follows. Section 2 presents a dynamic multiplexing approach and suggests why it is potentially a more efficient mean of message traffic reduction while discussing its advantages and limitations. Section 3 discusses our experimentation with a projectile/missile application and evaluates the performance of the dynamic multiplexing approach. Section 4 presents our conclusions.

2 Interest-Based Quantization: Multiplexing and Predictive

The multiplexing predictive interest-based quantization is an extension created by adding a multiplexing approach to the predictive interest-based quantization. In distributed system with a large number of entities, there will be many entities assigned to each federate. Sender and receiver federates encapsulate a large number of similar component models. In this case, the multiplexing approach is very effective. This approach requires two components: sender multiplexer and receiver de-multiplexer. The sender multiplexer gathers the messages outputted from the sender agents within a time granule into a large message, which is sent to the receiver de-multiplexer in the other receiver federate over network. The receiver de-multiplexer separates the large multiplexed message to the small-unmultiplexed messages and distributes the small messages to the proper receiver agents. As the number of sender and receiver pairs increases, through this multiplexing approach, tremendous communication bits are saved.

2.1 Dynamic Multiplexing

The multiplexed message size is constant in previous multiplexing which is called fixed multiplexing. This paper proposes the dynamic multiplexing in which the message size varies with the number of active senders. Active sender indicates a sender which has an output in a certain time granule. This section introduces the dynamic approach in multiplexing, compares it with the fixed multiplexing and analyzes the performance of dynamic approach with network bandwidth requirement. Fig. 1 illustrates the operation of the dynamic multiplexing using predictive quantization. The dynamic sender multiplexer only collects the encoded bits from active senders. At a given event time, the number of active senders varies and the number of transmitted data bits is not fixed. Different from fixed multiplexing, additional bits (SL) are needed to represent active senders. The number of data bits for an active sender is calculated by adding the additional bits (log2 Npair < SL) and the encoded bits (SQ). Usually, a is less than 1 since all senders are not active senders at any given event time.

The network loading for any global state transition of a sender federate using dynamic multiplexing is:

Network bandwidth requirement using dynamic multiplexing

$$= S_{OH} + a\, N_{pair} \times \ (S_Q + S_L) \text{ (bits)} \tag{1}$$

The multiplexer collects the encoded bits and the active bits from each encoder. The encoded bits are the bits required to represent the message dimension alternatives. Let S_Q be the number of the encoded bits. Then by:

$$S_Q = \lceil D \times \log_2 3 \rceil = \lceil 1.7 \times D \rceil \quad (D = 1,2,3,\cdots (\mathit{Number\ of\ Dimensions})) \tag{2}$$

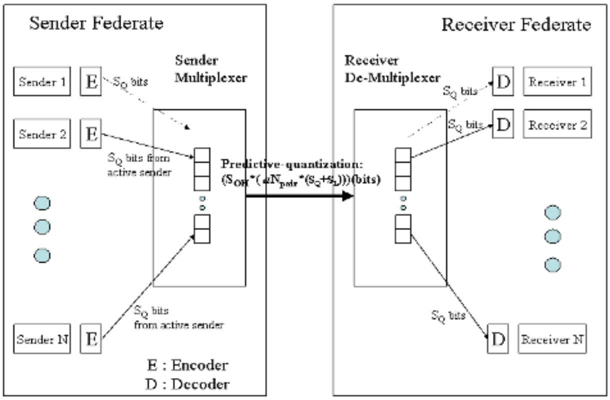


Fig. 1. Dynamic multiplexing with the predictive quantization (S_{OH} : the number of overhead bits for a packet; S_Q: the quantized and encoded data bit size; S_L: the encoded data bit size for sender ID; N_{pair} : the number of pair components; a: the ratio of active components)

For example, if a message has three-dimensional values in the predictive quantization, five bits ($\log_2 3^3 < 5 = S_Q$) are required to represent the message dimension alternatives. The active bit indicates whether a sender is active or inactive. An active sender is one that has a boundary crossing at a given event time and generates an output event. A receiver de-multiplexer checks the active bit of each sender and sends the encoded bits of active senders to the respective decoders. In fixed multiplexing, for any global state transitions of a sender federate at any given event time, the network loading is fixed and calculated by:

$$\text{Network bandwidth requirement for fixed multiplexing} \\ = S_{OH} + N_{pair} \times (S_Q + 1) \quad \text{(bits)} \tag{3}$$

However, the bits assigned for inactive senders can be wasted in fixed multiplexing. The fixed receiver de-multiplexer knows which sender sends certain encoded bits since the bit stream order in the multiplexed bits with fixed size follows a fixed ordering of the senders. Therefore, the additional bits representing which sender sends are not needed.

2.2 Performance Analysis

In order to analyze the performance of the dynamic multiplexing with predictive interest-based quantization, we investigate three performance factors: a) Network bandwidth requirement (e.g. number of bits required) for *N* component pairs; b) Reduction ratio to the number of bits needed for the non-multiplexing, non-predictive quantization with *N* component pairs; c) Reduction ratio to the number of bits needed for the non-multiplexing, non-predictive quantization with actual data bit sizes and 1000 component pairs. The reduction ratio specified by:

$$\text{Reduction Ratio} = \frac{\text{A}}{\text{B}} \tag{4}$$

A: # bits needed for the non-multiplexing and non-predictive quantization
B: # bits needed for the dynamic multiplexing with predictive interest-based quantization

The analysis is given in Table 1 where we consider all six combinations of quantization (non-predictive and predictive) and multiplexing (fixed and dynamic).

Table 1. Analysis with six combinations of quantization (non-predictive and predictive) and multiplexing (fixed and dynamic), (S_{OH}: the number of overhead bits for a packet (160 bits); S_D: the non-quantized data bit size (64*3 bits for double precision real numbers for three dimensions); S_Q: the quantized and encoded data bit size (5 bits for three dimensions ($\log_2 3^3 < 5 = S_Q$)); S_L: the encoded data bit size for sender ID (10 bits for 1000 N_{pair}($\log_2 1000 < 10 = S_Q$)); N_{pair} : the number of pair components (1000), a: the ratio of active components), Ratio with value (N_{pair} =1000, S_{OH}=160 bits, S_D=64*3bits, S_Q= 5 bits, S_L= 10 bits).

Scheme	# bits required for $N_{pair}(a<1)$	Ratio to Non-predictive quantization for large $N_{pair}(a<1)$	Ratio with value
Comb #1	$aN_{pair}\ (S_{OH} + S_D)$	1	1
Comb #2	$aN_{pair}\ (S_{OH} + S_Q)$	$(S_{OH} + S_D) / (S_{OH} + S_Q)$	2.172
Comb #3	$(S_{OH}+N_{pair}(S_D+1))$	$a(S_{OH} + S_D) /(S_D + 1)$	1.851 a
Comb #4	$(S_{OH}+N_{pair}(S_Q+1))$	$a(S_{OH}+S_D) /(S_Q+1)$	58.823 a
Comb #5	$(S_{OH}+aN_{pair}(S_D+S_L))$	$(S_{OH} + S_D) / (S_D + S_L)$	1.754
Comb #6	$(S_{OH}+aN_{pair}(S_Q+S_L))$	$(S_{OH} + S_D) / (S_Q + S_L)$	29.412

The combinations in Table 1 are defined as following: Comb #1: Non-predictive quantization; Comb #2: Predictive quantization and Non-multiplexing; Comb #3: Fixed multiplexing and Non-predictive quantization; Comb #4: Fixed multiplexing and Predictive quantization; Comb #5: Dynamic multiplexing and Non-predictive quantization; Comb #6: Dynamic multiplexing and Predictive quantization. The predictive quantization without multiplexing performs 2.172 times reduction in network load relative to non-predictive quantization. In the multiplexing non-predictive quantization scheme, the reduction ratio (approx. 1.851 a) is performed by combining the actual double value outputs into one message. Greater advantage is obtained from the multiplexing predictive quantization, which combines the encoded data bit size (5 bits for three dimensional data of message and 10 bits for sender ID) per component into one message. When the fixed multiplexing predictive quantization scheme is used, in order to make the reduction ratio higher above 29.412 times, at least 50 (%) active components are required. For the dynamic multiplexing predictive quantization, the reduction ratio is 29.412.

3 Experimentation and Performance Evaluation

The projectile/missile application [6] with the geocentric-equatorial coordinate system is used to evaluate the performance of the dynamic multiplexing approach. The projectile is a ballistic flight and accounts for gravitational effects, drag, and motion of rotation of the earth relative to it. A missile is assigned a projectile, and it follows its projectile until it hits its projectile. To evaluate the accuracy of the dynamic multiplexing approach, we use the previously developed basic system which is considered the standard system in which no error occurs. The system with the dynamic multiplexing approach includes two federates: projectile and missile. Each federate is assigned to each different computer and the experimental computers are connected in LAN environment. The dynamic multiplexing system works on the DEVS/GDDM Environment [7].

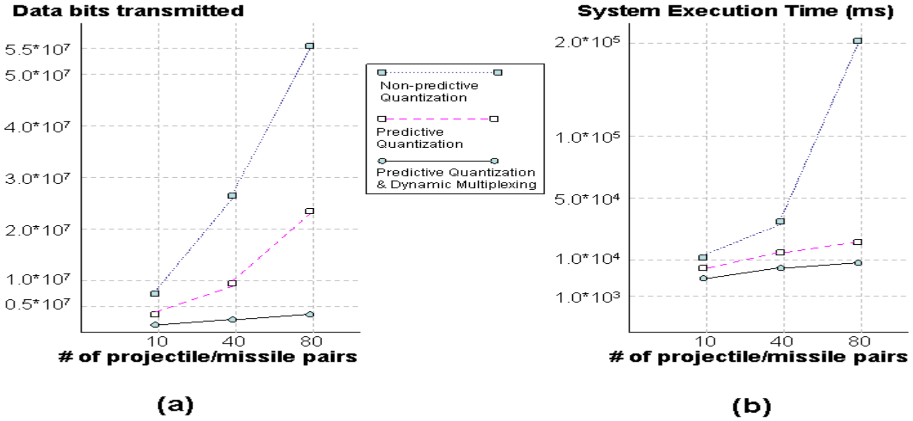


Fig. 2. (a) Transmitted data bits, (b) System execution time

In order to evaluate actual system execution performance of the dynamic multiplexing interest-based quantization scheme, we compared passed data bits and system execution time of a non-predictive quantization system, a predictive quantization system and a dynamic multiplexing predictive quantization system. Fig. 2 (a) shows the transmitted data bits of those three systems while the numbers of component pairs are varied. As the number of component pairs increases, the transmitted data bits of the non-predictive and predictive quantization systems increase significantly, and the dynamic multiplexing predictive quantization system tremendously reduces the transmitted data bits. In two non-multiplexing systems, the predictive quantization system shows the more reduction of transmitted data bits than that of the non-predictive quantization system. Fig. 2 (b) illustrates the variation of system execution time of those three systems in varying number of component pairs. In the dynamic multiplexing system, the system execution time increases slowly and proportionally to the transmitted data bits, as the number of component pairs increases.

4 Conclusion

This paper proposed a dynamic multiplexing approach as an efficient message traffic reduction scheme. Especially, the approach is applied to the predictive interest-based quantization and is more applicable to a system involving distributed components moving and interacting in multi-dimensional space. To evaluate the performance, we realized the dynamic multiplexing approach with the predictive interest-based quantization. We compared the dynamic approach to the fixed approach, analyzed the advantages and limitations, and experimented. Those approaches were applied to the projectile/missile application with realistic multi-dimensional dynamics and evaluated with the network bandwidth requirement. The analytical and experimental results showed that the dynamic multiplexing was very effective in saving the inter-federate data transmission and actual system execution time in distributed system.

References

1. Bassiouni, M.A., et al.: Performance and Reliability Analysis of Relevance Filtering for Scalable Distributed Interactive Simulation., ACM Trans. on Model. and Comp. Sim. (TOMACS) (1997) 293-331
2. Zeigler, B.P., J.S. Lee: Theory of Quantized Systems: Formal Basis for DEVS/HLA Distributed Simulation Environment. Enabling Technology for Simulation Science (II), SPIE AeoroSense 98. Orlando, FL (1998)
3. Bernard. P. Zeigler, H. J. Cho, J. S. Lee, Y. K. Cho, Hessam Sarjoughian, et al.: Predictive Contract Methodology and Federation Performance. in SIW. Orlando, FL (1999)
4. Bernard P. Zeigler, Hyup J. Cho, Jeong G. Kim, Hessam Sarjoughian, Jong S. Lee: Quantization-based filtering in distributed discrete event simulation. Journal of Parallel and Distributed Computing, 62 (2002) 1629-1647
5. Jong S. Lee and Bernard. P. Zeigler: Space-based Communication Data Management in Scalable Distributed simulation. Journal of Parallel and Distributed Computing, 62 (2002) 336-365
6. Erwin Kreyszig.: Advanced Engineering Mathematics, 7th Edition, John Wiley& Sons Inc, New York (1993)
7. Jong Sik Lee, Bernard. P. Zeigler: Design and Development of Data distribution Management Environment. Journal of Society Computer Simulation, (77)1-2. Simulation (2002) 39-52

Novel Adaptive Subcarrier Power and Bit Allocation Using Wavelet Packet Parallel Architecture*

Ren Ren[1,2] and Shihua Zhu[1]

[1] Sch. of Electronic & Information Eng., Xian Jiaotong University,
Xi'an, 710049,P.R. China
[2] Dept. of Phy., Xian Jiaotong University, Xi'an, 710049,P.R. China
renren@mail.xjtu.edu.cn

Abstract. To realize the requirement of next mobile communication, the adaptive allocation schema of power and bit is presented by using discrete wavelet packet transform (DWPT). The subcarrier modulation and rate allocation method is used for OFDM-DS/CDMA system. According to the downlink channel feedback bit error rate (BER) acquired from uplink channel, an optimal wavelet packet multicarrier modulation allocation is established. For the given BER and QoS, the transmitting power is minimum. The system realizes the high frequency spectrum efficiency. The result shows that the adaptive wavelet packet algorithm not only has the fast convergence rate, but also achieves minimum complexity. The allocation proposed has better performance compared with traditional OFDM system. It is valuable that the system can adaptively adjust the power and bit rate to achieve minimum total transmission power in high rate and efficiency.

1 Introduction

Along with the development of the next generation mobile communications, multimedia and broadband, the wireless communication system requires advanced rate transmission, fast processing ability so as to improve communication reliability and efficiency. Because of frequency selection Rayleigh fading produced by multipath [1], slow fading of obstruct, and space fading. The wireless channel have time-variable properties [2]-[4]. Now, there are different kinds of multi-carrier orthogonal transform to be proposed to get over this defect, such as DCT multi-carrier modulation based on discrete cosine transform, DFT discrete Fourier transform, and DWT discrete wavelet transform multi-carrier. The paper proposed a kind of discrete orthogonal wavelet multi-carrier modulation. Because of orthogonal parallel multiplexing communication mode, orthogonal wavelet will not lead to the noise increasing as well as obtain parallel processing efficient. The wavelet packet transform could supply a group wavelet packet multi-carrier schema. This idea can divide the channel into series orthogonal subband by using super resolution wavelet packet. Each subband is flat fading. Adaptive DWPT is able to adjust the transmission schema to receiver SNR and channel properties. Different user adopts their own bit

* Supported by National Natural Science Foundation of China 60372055, and Xian Jiaotong university Foundation xjj2004013.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 422 – 428, 2005.

rates, modulation grade and communication qualities. The subcarrier bandwidth is less than channel correlative bandwidth. Finally, the multipath effect is reduced for each subcarrier. Each subband has flat faded and ICI is down.

The wavelet packet modulation (WPM) stresses the fact that wavelet packet bit and power allocation has two advantages over others. The frequency selection MIMO fade channel becomes group of flat fading sub-channel by WPM. The signal spectrum enables overlapping, and higher spectrum efficiency is obtained. Moreover, we can adaptively adjust the bit and power allocation in each subcarrier according to SNR and subchannel state. WPM has super resolution property in time-frequency domain. The processes of modulation, filter and demodulation are the signal decomposition and reconstruction process of wavelet packet. The subchannel with better SNR employs higher modulation grade transferring more bits/symbol. The subchannel with high interference uses lower modulation grade transferring little bits/symbol. The paper adopts indefinite searching way to realize less calculation amount and achieve better spectrum efficient and robust based on optimum algorithm [5]-[10]. IWPT and WPT implementations are used as specific modulation and de-modulation.

2 Signal Model

2.1 Subcarriers Modulation Model Based on Orthogonal Wavelet Packet

Wavelet packet modulation process has ability to process time variant signal. The window function size can be adjusted. By IWPT, input bit can be transformed into time frequency domain symbol as well as it can be reconstructed in time-domain by WPT. we can obtain clear resolution wherever the signals are high or low time frequency domain. The transmitter data can ideally be decomposed by wavelet packet.

Multicarrier modulation system and subband bit and power allocation are shown in detail as Fig.1. In the transmitter, the signal of user is through serial/parallel transform and modulated by QAM. Data is decomposed into different subchannel by wavelet packet. Then the signal is sampled, filtered and reconstructed to run p/s transform. In addition, the signal time sequence is spread by frequency synthesizer using wavelet PN pseudorandom code. Finally, it is transmitted by D/A shaped smooth filter.

The subcarrier bit allocation system chart is shown as Fig.1. Suppose that system has k users, each user data rate is R_k bit/symbol. In the transmitter, the data streams are assigned to different subcarrier by each user. If the subcarrier bandwidth is smaller than the correlation bandwidth of channel, the channel is mutually independent. We can use the subcarrier bit algorithm to determinate transmitter modulation system. According to the allocated bit in each subcarrier, the modulation constellation size is decided. The power of each channel is determined by communication SNR and modulated mode. The M dimensions receiver vector $R = HU + n$, $n = [n_1, n_2, ..., n_M]^T$, the n-th user U_n is distributed to different subcarrier with bit scheme. **H** is the channel response and **n** is noise. It is assumed that $P_n(k)$ is power in m-th transmitter antenna $\mathrm{P = U^T(k)U(k) = \sum_{n=1}^{N} P_n(k)}$. The transmitter signal $f(t) = U \in L^2(R)$ can be decomposed by the wavelet modulation degree.

The $\Phi(x)$ and $\psi(x)$ are wavelet scale and wavelet function respectively. Signal space $L^2(R)$ is orthogonally divided into subspaces W_j, $L^2(R) = \underset{j\in Z}{\oplus} W_j = ...\oplus W_{-2} \oplus W_{-1} \oplus W_0 \oplus W_1 \oplus W_2 \oplus ...\cdot$

For the signal $f(t) = U \in L^2(R)$, the $f(t)$ can be decomposed in wavelet packet base in unique way. Considered that base $\psi_0(t) = \phi(t)$, $\psi_1(t) = \psi(t)$.

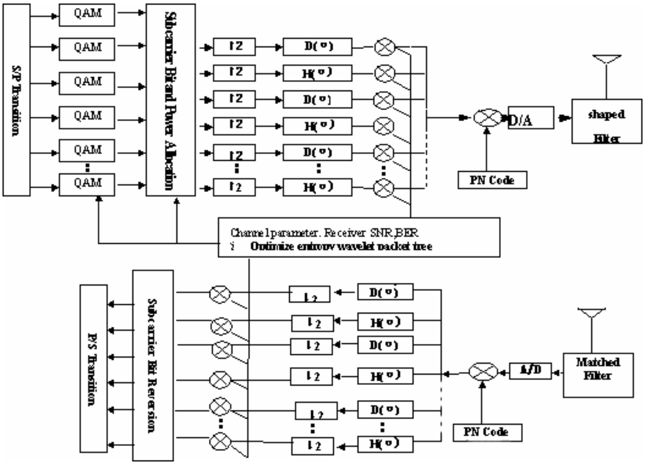


Fig. 1. The adaptive wavelet packet modulation system of bit and power allocation

The orthogonal wavelet packet is defined

$$\psi_{2l}(t) = \sum_k p_k \psi_l(2t-k) \qquad \psi_{2l+1}(t) = \sum_k q_k \psi_l(2t-k) \tag{1}$$

where p_k, q_k is decomposed sequence. $f_{l+1}(t) = f_l(t) + g_l(t)$ is expressed as

$$f_j(t) = \sum_k c_k^j \phi(2^j t - k) \qquad g_j(t) = \sum_k d_k^j \psi(2^j t - k) \quad \text{j=0,1,2,...} \tag{2}$$

Orthogonal scale function is $\phi(t)$ and $\psi(t)$ is wavelet function. The data streams $f(t) = U$ can be expressed in the base of $\psi_{2l}(t)$ and $\psi_{2l+1}(t)$, where $\{\psi_n\}$ is the wavelet function. For $n \in Z$, $clos_{L^2(R)}$ is signal space.

$$S_j^n = clos_{L^2(R)}(2^{j/2}\psi_n(2^j t - k) : k \in Z) \qquad j \in Z, n \in Z \tag{3}$$

The decomposition and modulation algorithm is shown as Fig.2.

$$f_{k+1}(t) = \sum_k [p_{l-2k} d_k^{j,2n} + q_{l-2k} d_k^{j,2n+1}]\psi_n(2^j t - l) \tag{4}$$

The reconstruction and de modulation algorithm is

$$\sum_k a_{k-2l} d_k^{j+1,n} = d_l^{j,2n} = f_k^{2n}(t) \text{ ,} \sum_k b_{k-2l} d_k^{j+1,n} = d_l^{j,2n+1} = f_k^{2n+1}(t) \tag{5}$$

where $d_l^{j,2n}$ and $d_l^{j,2n+1}$ are the decomposition coefficient.

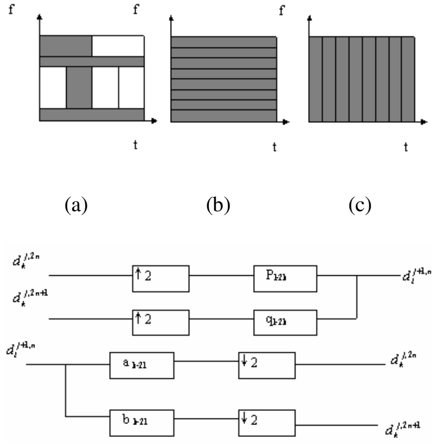
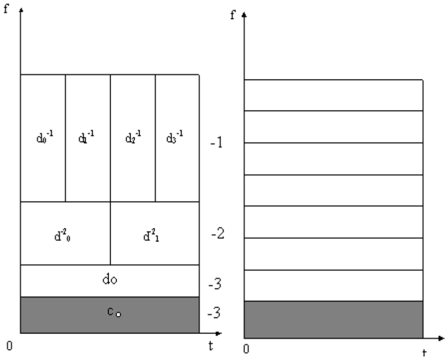


Fig. 2. The modulated and demodulated processing of wavelet packet



(a) mode 1(WPM) (b) mode 2(OFDM)

Fig. 3. WPM and OFDM packet structure

The sub-carrier bit and power allocation algorithm object is what the total power of the all sub-carriers of every user are minimum in definite QoS and data rate. The input data stream is modulated by QAM and decomposed by orthogonal wavelet packet. The adaptive WPM allocation has better time frequency domain performance compared with traditional OFDM system shown as Fig. 2 (a)(b)(c) and Fig. 3(a)(b).

2.2 Adaptive Algorithm of Wavelet Packet with Bit and Power Allocation

Multi bit rates and adaptive power allocation are realized by Lagrangian optimal algorithm. The constraint condition is total power and transmitter bits/symbol rate. The wavelet packet decomposition and reconstruction are changed with channel QoS and receiver SNR.

Each subcarrier $\{2^{\frac{j-k}{2}} \psi_{2^k+m}(2^{j-k}t-l) : l \in Z\}$ has their own allocation of power and bit rate. With the increase of the subcarrier number, the bandwidth is smaller than channel correlation bandwidth. The Rayleigh fading is reduced and can be thought to be a flat fading channel with smaller ICI.

We adopts water-filling algorithm and adaptive power and bit rate allocation. it can achieve minimum transmission power and the maximum channel capacity of Shannon theory. Ordinary, the suboptimum non-precision searching algorithm is accepted.

Considering that per symbol emit total bit B, the whole bit is allocated for each subcarrier by wavelet packet modulated. Subcarrier bit allocation is optimized to make transmitter power to be minimum. The object function is total bit of all subcarriers $p = \min_{c_{k,m}} \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{p_k(b_{k,n})}{h_{k,n}^2}$, $k \in \{1,...,k\}$, $c_{k,n}$ is bits/symbol, $b_{k,n}$ is gain.

The adaptive algorithm flow is shown as following.

$$y_i(t) = w_i^T(t-1)r_1(t)$$

$$\hat{r}_i(t) = w_i(t-1)y_i(t)$$

$$e_i(t) = r_i(t) - w_i^T(t-1)y_i(t)$$

$$r_{i+1}(t) = r_1(t) - \mu_i e_i(t) y_i(t)$$

i=i+1

Loop

where w_i is weigh coefficient, μ_i is step. r (t) is receiver's data, power or bit rate.

Supposing object function is $\text{Min}\sum_{m=1}^{M} P_m(b_m)$, and convergence condition $\sum_{m-1}^{M} b_m = B$, $b_m \in Z$, m=1,2,…,M. P_m is the lowest transmitter power in each subcarrier, where b_m is m-th subcarrier bits/symbol. The bit and power in sub-carrier is adaptively decided by downlink feedback SNR and BER of receiver. Let $\hat{b}_m = \log_2(1 + \nu(SNR)_m)$, where ν is regulative coefficient, $(SNR)_m$ is m-th subchannel SNR. The allocation bit and power can be derived by ν , b_m bits/symbol is modulated by MQAM and MPSK.

In addition, the cost function J is solved by optimum non-precision algorithm.

$$J(p_m) = \min_{SNR,BER}\{\sum_{m=1}^{M}\frac{p_m}{h_m^2} + \sum_n \lambda_i(\sum_{m=1}^{M} b_{n,m} - r)\} \tag{6}$$

The iterative computing $b_{k+1,n} = b_{k,n} - \mu\frac{\partial L}{\partial b_{k,n}}$ and $\lambda_{k+1} = \lambda_k + \mu(\sum_{n=1}^{N}(b_{k,n} - r_k))$ (1) Initialization user bit allocation $c_{k,n}$=0, $\lambda_k = 0$;(2) Calculation $b_{k,n}$, choice modulation grade with $b_{k,n}$. (3) Calculation λ_k and summation $b_{k,n}$, if it doesn't reach the user set rate, recirculation.

3 The Simulation and Result

In mobile downlink environment, the simulation parameters of WPM is 4QAM, 16QAM, 64QAM constellation, subband number 16, and Rayleigh channel adopting binary Haar wavelet. It is assumed that the wavelet packet sizes are 32 samples per packet, and system has randomly interference on the base-band system. Under four interferences source, two paths, same noise in each packet signal, the OFDM channel fading is become flat in each subband, whereas, the WPM subband are not affected in WPM packet. Using 16QAM, it is shown as in Fig.4 that the BER of adaptive WPM packet is close to 16QAM curve. The adaptive WPM BER is 4.5 dB is better than WPM. The WPM BER reduces further than OFDM. The performance of simulation illuminates adaptive WPM have advantage of others such as WPM, and OFDM.

Based on definite QoS and BER, adaptive WPM with bit and power algorithm has merits over equal bit method, optimum OFDM-TDMA and OFDM-CDMA. The result of bit error order is described as follows. Under same BER, WPM adaptive bit and power allocation is 3 dB better than OFDM-CDMA. The bit SNR WPM-CDMA is 10 dB better than WPM-TDMA, and 25 dB than equal bit OFDM-TDMA. Above all, the WPM-CDMA method significantly out performs the traditional OFDM and equal power allocation.

Table 1. Simulation specifications

Frequency	5GHz
Wavelet species	Haar and Daubechies wavelet
Number of subcarriers	16
Constellation	4QAM,16QAM, 64QAM
Channel bandwidth	20MHz
Sample per packet	32
Mobile speed	100Km/hour
Channel distribution	Rayleigh
Interferences source	4
Path number	**2**

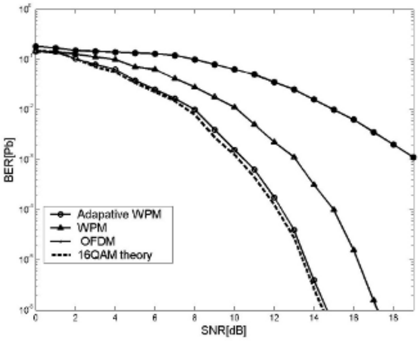


Fig. 4. BER performance in interference environment based on adaptive WPM, WPM, and OFDM

4 Conclusion

The paper proposed an adaptive WPM bit and power allocation model for downlink channel wireless mobile system to solve the ICI and ISI. The parallel algorithm has inferior iterative times, minor calculation and better global convergence. The results have shown that the schema obtains major SNR gain per bit. Furthermore, the adaptive wavelet packet achieves bandwidth utilizing effectively in high and low frequency. According to the channel SNR, BER and QoS, the bit rate, power allocation and wavelet packet coefficient weigh are adaptive to be adjusted to channel quality. The simulation indicates that WPM parallel algorithm is validated and has fast speed compared with OFDM. The WPM schema is feasible plan in communication system.

References

[1] W.C. Takes Jr.: *Microwave mobile communications*, Wiley. New York .(1974)

[2] S. Hara:Overview of multicarrier CDMA, *IEEE Commun. Mag.*, DEC, (1997).126-144

[3] M. Helard: Multicarrier CDMA techniques for future broadband wireless networks, *Ann. Telecommun.*. Vol.56, (2001)260-274

[4] S. Hara:Design and performance of multicarrier CDMA system in frequency-selective Rayleigh fading channels, *IEEE Trans. Veh. Technol.*, Vol.48.Sept .(1999)1584-1595

[5] E. Kerherve: OFDM bandwidth estimation using Morlet's wavelet decomposition, EUROCOMM 2000.Information Sys. for Enhanced Public Safety and Security. *IEEE/AFCEA*. May.(2000) 62-66

[6] Y. zhang: Performance of Wavelet Packet Based Multicarrier Modulation DS-CDMA System In Frequency Selective Rayleigh Fading Channel, *Inter. Rept. OHIO uni.*. Oct.. (1998) 82-89.

[7] R.M.Nasir: Adaptive wavelet packet basis selection for zerotree image coding, *IEEE.Trans.on image proceeding*. Vol 12(12). Dec. (2003) 1462-1472

[8] S.Hara : Overview of Muiticarrier CDMA, *IEEE.Comm. Magazine*. Dec, (1997) 126-133

[9] H.Nikolaj: Wavelets and time-frequency analysis, *Proc.of IEEE*. Vol.84(4), (1996) 523-540

[10] S.D.Sandberg: Overlapped Discrete Multitone Modulation for High Speed Copper Wire Communications, *IEEE J.Select. Areas Comm.*. Vol.13. Dec., (1995) 1571-1585

A Low–Level Communication Library for Java HPC

Sang Boem Lim[1], Bryan Carpenter[2],
Geoffrey Fox[3], and Han-Ku Lee[4,⋆]

[1] Korea Institute of Science and Technology Information,
(KISTI),Daejeon, Korea
slim@kisti.re.kr

[2] OMII, University of Southampton, Southampton SO17 1BJ, UK
dbc@ecs.soton.ac.uk

[3] Pervasive Technology Labs at Indiana University,
Bloomington, IN 47404-3730
gcf@indiana.edu

[4] School of Internet and Multimedia Engineering,
Konkuk University, Seoul, Korea
hlee@konkuk.ac.kr

Abstract. Designing a simple but powerful low-level communication library for Java HPC environments is an important task. We introduce new low-level communication library for Java HPC, called *mpjdev*. The mpjdev API is designed with the goal that it can be implemented *portably* on network platforms and *efficiently* on parallel hardware. Unlike MPI which is intended for the application developer, mpjdev is meant for library developers. Application level communication may be implemented on top of mpjdev. The mpjdev API itself might be implemented on top of Java sockets in a portable network implementation, or-on HPC platforms-through a JNI (Java Native Interface) to a subset of MPI.

1 Introduction

HPJava [1] is an environment for scientific and parallel programming using Java. It is based on an extended version of the Java language. HPJava incorporates all of the Java language as a subset. This means any ordinary Java class can be invoked from an HPJava program without recompilation. Moreover, a translated and compiled HPJava program is a standard Java class file that can be executed by a distributed collection of Java Virtual Machines.

Locally held elements of *multiarrays* and *distributed arrays* can be accessed using some special syntax provided by HPJava. HPJava does not provide any special syntax for accessing non-local elements. Non-local elements can only be accessed by making explicit library calls. This policy in the HPJava language, attempts to leverage successful library-based approaches to SPMD parallel computing. This idea is in very much in the spirit of MPI, with its explicit point-to-point and collective communications. HPJava raises the level of abstraction a

[⋆] Correspondence author.

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 429–434, 2005.

notch, and adds excellent support for development of libraries that manipulate distributed arrays. But it still exposes a multi-threaded, non-shared-memory, execution model to programmer. Advantages of this approach include flexibility for the programmer, and ease of compilation, because the compiler does not have to analyze and optimize communication patterns.

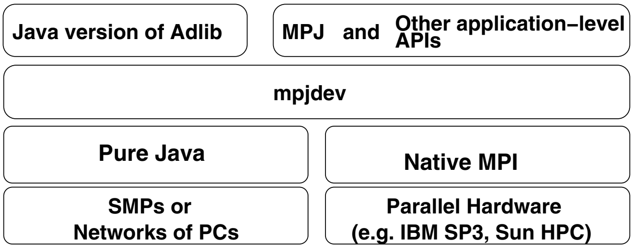

Java version of Adlib
MPJ and Other application–level APIs
mpjdev
Pure Java
Native MPI
SMPs or Networks of PCs
Parallel Hardware (e.g. IBM SP3, Sun HPC)

Fig. 1. An HPJava communication stack

The mpjdev [2] [3] API is designed with the goal that it can be implemented *portably* on network platforms and *efficiently* on parallel hardware. Unlike MPI which is intended for the application developer, mpjdev is meant for library developers. Application level communication libraries like the Java version of Adlib (or MPJ [1]) may be implemented on top of mpjdev. The mpjdev API itself might be implemented on top of Java sockets in a portable network implementation, or-on HPC platforms-through JNI (Java Native Interface) to a subset of MPI. The positioning of the mpjdev API is illustrated in Figure 1. Currently not all the communication stack in this figure is implemented. The Java version of Adlib, the pure Java implementation on SMPs, and native the MPI implementation are developed and included in the current HPJava or mpiJava releases. The rest of the stack may be filled in the future.

2 Communications API

In MPI there is a rich set of communication modes. Point-to-point communication and collective communication are two main communication modes of MPI. Point-to-point communication support blocking and non-blocking communication modes. Blocking communication mode includes one blocking mode receive, **MPI_RECV**, and four different send communication modes. Blocking send communication modes include standard mode, **MPI_SEND**, synchronous mode, **MPI_SSEND**, ready mode, **MPI_RSEND**, and buffered mode, **MPI_ BSEND**. Non-blocking communication mode also uses one receives, **MPI_IRECV** and the same four modes as blocking send: standard, **MPI_ISEND**, synchronous, **MPI_ISSEND**, ready, **MPI_IRSEND**, and buffered, **MPI_IBSEND**. Collective communication also includes various communication modes. It has characteristic collective modes like broad-

```
public class Comm {

  public void size() { ... }
  public void id() { ... }
  public void dup() { ... }
  public void create(int [] ids) { ... }
  public void free() { ... }

  public void send(Buffer buf, int dest, int tag) { ... }
  public Status recv(Buffer buf, int src, int tag) { ... }
  public Request isend(Buffer buf, int dest, int tag) { ... }
  public Request irecv(Buffer buf, int dest, int tag) { ... }

  public static String [] init(String[] args) { ... }
  public static void finish() { ... }

  . . .
}
```

Fig. 2. The public interface of mpjdev `Comm` class

cast, **MPI_BCAST**, gather, **MPI_GATHER**, and scatter, **MPI_SCATER**. Global reduction operations are also included in collective communication.

The mpjdev API is much simpler. It only includes point-to-point communications. Currently the only messaging modes for mpjdev are standard blocking mode (like **MPI_SEND**, **MPI_RECV**) and standard non-blocking mode (like **MPI_ISEND**, **MPI_IRECV**), together with a couple of ”wait” primitives.

The communicator class, **Comm**, is very similar to the one in MPI but it has a reduced number of functionalities. It has communication methods like **send()**, **recv()**, **isend()**, and **irecv()**, and defines constants **ANY_SOURCE**, and **ANY_TAG** as static variables. Figure 2 shows the public interface of **Comm** class.

We can get the number of processes that are spanned by this communicator by calling **size()** (similar to **MPI_COMM_SIZE**). Current id of process relative to this communicator is returned by **id()** (similar to **MPI_COMM_RANK**).

The two methods **send()** and **recv()** are blocking communication modes. These two methods block until the communication finishes. The method **send()** sends a message containing the contents of **buf** to the destination described by **dest** and message tag value **tag**.

The method **recv()** receives a message from matching source described by **src** with matching tag value **tag** and copies contents of message to the receive buffer, **buf**. The receiver may use wildcard value **ANY_SOURCE** for **src** and **ANY_TAG** for **tag** instead specifying **src** and **tag** values. These indicate that a receiver accepts any source and/or tag of send. The **Comm** class also has the initial communicator, **WORLD**, like **MPI_COMM_WORLD** in MPI and other utility methods. The capacity of receive buffer must be large enough to

```
public class Request {
  public Status iwait() { ... }

  public Status iwaitany(Request [] reqs) { ... }
  . . .
}
```

Fig. 3. The public interface of `Request` class

accept these contents. It initializes the **source** and **tag** fields of the returned **Status** class which describes a completed communication.

The functionalities of **send()** and **recv()** methods are same as standard mode point–to–point communication of MPI (**MPI_SEND** and **MPI_RECV**). A **recv()** will be blocked until the send if posted. A **send()** will be blocked until the message have been safely stored away. Internal buffering is not guaranteed in **send()**, and the message may be copied directly into the matching receive buffer. If no **recv()** is posted, **send()** is allowed to block indefinitely, depending on the availability of internal buffering in the implementation. The programmer must allow for this–this is a low-level API for experts.

The other two communication methods **isend()** and **irecv()** are non-blocking versions of **send()** and **recv()**. These are equivalent to **MPI_ISEND** and **MPI_IRECV** in MPI. Unlike blocking send, a non-blocking send returns immediately after its call and does not wait for completion. To complete the communication a separate send complete call (like **iwait()** and **iwaitany()** methods in the **Request** class) is needed. A non-blocking receive also work similarly. The **wait()** operations block exactly as for the blocking versions of **send()** and **recv()** (e.g. the **wait()** operation for an **isend()** is allowed to block indefinitely if no matching receive is posted). The method **dup()** creates a new communicator the spanning the same set of processes, but with a distinct communication context. We can also create a new communicator spanning a selected set of processes selected using the **create()** method. The ids of array **ids** contain a list of ids relative to this communicator. Processes that are outside of the group will get a null result. The new communicator has a distinct communication context. By calling the **free()** method, we can destroy this communicator (like **MPI_COMM_FREE in MPI**). This method is called usually when this communicator is no longer in use. It frees any resources that used by this communicator.

We should call static **init()** method once before calling any other methods in communicator. This static method initializes mpjdev and makes it ready to use. The static method **finish()** (which is equivalent of **MPI_FINALIZE**) is the last method should be called in mpjdev.

The other important class is Request (Figure 3). This class is used for non-blocking communications to ensure completion of non-blocking send and receive. We wait for a single non-blocking communication to complete by calling **iwait()** method. This method returns when the operation identified by the current class is complete. The other method **iwaitany()** waits for one non-blocking communication from a set of requests **reqs** to complete. This method returns when one of the operations associated with the active requests in the array **reqs** has completed. After completion of **iwait()** or **iwaitany()** call, the source and tag fields of the returned status object are initialized. One more field, index, is initialized for **iwaitway()** method. This field indicates the index of the selected request in the **reqs** array.

3 Message Format

This section describes the message format used by mpjdev. The specification here doesn't define how a message vector which contained in the **Buffer** object is stored internally-for example it may be as a Java **byte** [] array or it may be as a C **char** [] array, accessed through native methods. But this section does define the organization of data in the buffer. It is the responsibility of the user to ensure that sufficient space is available in the buffer to hold the desired message. Trying to write too much data to a buffer causes an exception to be thrown. Likewise, trying to receive a message into a buffer that is too small will cause an exception to be thrown. These features are (arguably) in the spirit of MPI.

A message is divided into two main parts. The *primary payload* is used to store message elements of primitive type. *The secondary payload* is intended to hold the data from object elements in the message (although other uses for the secondary payload are conceivable). The size of the primary payload is limited by the fixed capacity of the buffer, as discussed above. The size of the secondary payload, if it is non-empty, is likely to be determined "dynamically"-for example as objects are written to the buffer.

The message starts with a short *primary header*, defining an *encoding scheme* used in headers and primary payload, and the total number of data bytes in the primary payload. Only one byte is allocated in the message to describe the encoding scheme: currently the only encoding schemes supported or envisaged are *big-endian* and *little-endian.* This is to allow for native implementations of the buffer operations, which (unlike standard Java read/write operations) may use either byte order. A message is divided into zero or more *sections.* Each section contains a fixed number of elements of homogeneous type. The elements in a section will all have identical *primitive* Java type, or they will all have **Object** type (in the latter case the exact classes of the objects need *not* be homogeneous within the section).

Each section has a short header in the primary payload, specifying the type of the elements, and the number of elements in the section. For sections with primitive type, the header is followed by the actual data. For sections with object type, the header is the only representation of the section appearing in the primary payload–the actual data will go in the secondary payload. After the primary payload there is a *secondary header.* The secondary header defines the number of bytes of data in the secondary payload. The secondary header is followed in the logical message by the secondary payload. The mpjdev specification says nothing about the *layout* of the secondary payload. In practice this layout will be determined by the Java Object Serialization specification.

4 Discussion

We have explored enabling parallel, high-performance computation–in particular development of scientific software in the network-aware programming language, Java. Traditionally, this kind of computing was done in Fortran. Arguably, Fortran is becoming a marginalized language, with limited economic incentive for vendors to produce modern development environments, optimizing compilers for new hardware, or other kinds of associated software expected by today's programmers. Java looks like a promising alternative for the future.

Java introduces implementation issues for message-passing APIs that do not occur in conventional programming languages. One important issue is how to transfer data between the Java program and the network while reducing overheads of the Java Native Interface. As contribution toward new low-level APIs, we developed a low-level Java API for HPC message passing, called mpjdev. The mpjdev API is a device level communication library. This library is developed with HPJava in mind, but it is a standalone library and could be used by other systems. We discussed message buffer and communication APIs of mpjdev and also format of a message. To evaluate current communication libraries, we did various performance tests. We developed small kernel level applications and a full application for performance test. We got reasonable performance on simple applications without any serious optimization. We also evaluated a communication performance of the high- and low-level libraries for future optimization.

References

1. HPJava project home page. www.hpjava.org.
2. Sang Boem Lim. *Platforms for HPJava: Runtime Support for Scalable Programming in Java.* PhD thesis, Florida State University, June 2003.
3. Sang Boem Lim, Bryan Carpenter, Geoffrey Fox, and Han-Ku Lee. A device level communication library for the hpjava programming language. In *the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, November 2003.

Object-Oriented Design and Implementations of 3G-324M Protocol Stack*

Weijia Jia Haohuan Fu and Ji Shen

Department of Computer Science, City University of Hong Kong
83 Tat Chee Avenue, Kowloon, Hong Kong, SAR China
itjia@cityu.edu.hk

Abstract. This paper describes an object-oriented design and efficient implementation of 3G-324M protocol stack for real-time multimedia transmission. In particular, we discuss the implementations of 324M class hierarchical structure that includes classes H.245 (control) and H.223 (multiplexing) protocols. Our implementation is efficient and has been tested in a realistic 3G infrastructures in Hong Kong as well as in some China industries for optimizations of processing and transmission of real-time video, audio and data.

1 Introduction

The mobile communication market has grown with an explosive rate recent years. The number of mobile subscribers worldwide increased from 300 million in 1997 to 800 million in 2001 [1]. With wider bandwidth of third-generation (3G) and increasing number of multimedia service categories, it can be seen that the number of subscribers is and will be having a high-speed increase, especially, when 3G is launched in China in the near future.

3G wireless multimedia communications are particularly referred to as International Mobile Telecommunications 2000 (IMT-2000) that has been deployed and developed substantially. ITU-T H.324 [2] is an umbrella protocol defined by International Telecommunications Union (ITU) to enable multimedia communication over low-bit rate terminals (in the following, we will drop "ITU-T" from the prefix of standard names for simplicity). H.324 and several mobile specific annexes are usually referred to as H.324M (M stands for *mobility*). The 3rd Generation Partnership Project (3GPP) is a body that comprises wireless infrastructure, handset and service providers throughout the world. 3GPP has adopted the H.324M with some modifications in codecs and error handling requirements to create the 3G-324M standard for circuit-switched 3G networks. In order to support the enhanced and delay sensitive video services among heterogeneous terminals, 3G video phones or terminals are required to support 3G-324M protocol stack. 3G-324M currently operates with WCDMA air interface, but can also operate on other 3G technologies because the 3G-324M call

* This effort is partially sponsored by City University of Hong Kong strategic grants 7001587, 7001709 and 7001777 and the National Basic Research Program (973) MOST of China under Grant No. 2003CB317003..

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 435 – 441, 2005.

setup is able to reuse the underlined air interface protocol that the hand-held device uses. We have developed and implemented an efficient mobile multimedia transmission protocol stack based on 3G-324M standards using C++. This paper discusses some efficient techniques and experiences of implementations for 3G-324M protocol stack, especially about object-oriented approaches. Our implementations have been tested in a realistic heterogeneous 3G communication environment in Hong Kong infrastructure and some China industries for transmission of real-time video, audio and data and its performance is satisfactory.

The paper is structured into 5 sections. Sec. 2 introduces the 3G-324M protocol stack. Object-oriented design, implementations and optimizations of control protocol H.245 and data transmission multiplexing protocol H.223 in 3G-324M are discussed in Sec. 3. Sec. 4 gives the performance analysis of the implementations. We summarize our major results and experience and point out the future directions in Sec. 5.

2 H.324 and 3G-324M

The whole protocol stack of 3G-324M is shown in Fig. 1. ITU-T H.324 is a standard made by ITU-T for low bit rate multimedia communications, while H.245 and H.223 are two main parts under H.324 and have given specific descriptions about the procedures of message transformation and data transmission multiplexing. However, H.324 is originally defined for multimedia communication operated in Public Switched Telephony Networks (PSTN), some of the specifications of this standard are not quite appropriate for the mobile terminals with low processing capability and power-constraints.

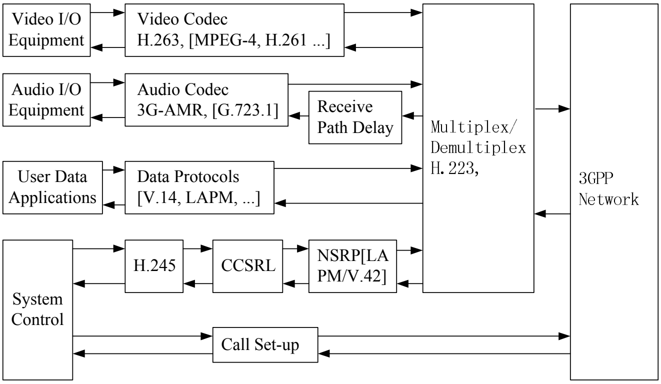


Fig. 1. 3G-324M Protocol Stack

H.324 and its annex C are referred to as H.324M for mobile terminals. Thus H.324M is also an "umbrella standard" in respect with other standards which specify mandatory and optional video and audio codecs, the messages to be used for call set-up, control and tear-down (H.245 [3]) and the way that audio, video, control and other data are multiplexed and demultiplexed (H.223 [4]). H.324M terminals offering audio communication will support G.731.1 audio codec [10]. Video communication offered in H.324M terminals will support H.263 [8] and H.261 [9] video codecs. H.324M terminals offering multimedia data conferencing should also support T.120 protocol [7]. In addition, other video and audio codecs and data protocols can optionally be used via negotiation through exchange of the H.245 control messages. Note that the differences between 3G-324M and H.324M mainly lie in codecs (voice by AMR-Adaptive Multi Rate Speech Codec, and video by H.263 or MPEG-4) and error handling requirements (H.223 Annex A and Annex B as mandatory) [14]. Therefore, 3G-324M inherits H.324M basically but must use AMR for speech codec. The AMR speech coder consists of the multi-rate speech coder, a source controlled rate scheme including a voice activity detector and a comfort noise generation system, and an error concealment mechanisms to combat the effects of transmission errors and lost packets [13].

3 OO Designs and Implementations of 3G-324M Protocol Stack

3.1 Object Class Structure of 3G-324 Protocol

In the 3G-324M protocol stack, the most upper-level interface is the H.324 class. The system structure is illustrated in the following skeleton codes. The functions illustrated here will be detailed in the subsequent sections.

```
    class H324
    { private:
            H245* h245;   //pointer to H245 entity
            H223* h223;   //pointer to H223 entity
    public:
            void Start();  //This function starts the execution of H324.
            BOOL StartH245();  //This function starts the execution of H245.
            BOOL HandleVideo(BYTE* data, int size); //Interface for video and audio
handling.
            BOOL HandleAudio(BYTE* data, int size);
            H324();            //construction and destruction functions
            H324(CDialog* dlg);
            virtual ~H324();
    }
```

Class H.324 holds a pointer to a H245 object and also a pointer to a H223 object. Therefore, H223 and H245 objects can be regarded as the composition classes within class H324. Similar to the design of class H324, classes H223 and H245 should provide the interface of method invocations for other classes as the H245 object has to send the H.245 control messages through H223 object whereas H223 object must also forward the receive the control messages in responding to the H245 object. Thus H223 object maintains a pointer to the H245 objects in the instance variable *m_pH245* and class H245 also maintains an interface to H223 using a pointer. For efficiency of implementation, the interface inherits class H245_SE rather than H245 as an instance (data) member within H245.

3.2 Hierarchy Class H.245

H.245 standard has been defined to be independent of the underlying transport mechanism, but is intended to be used with a reliable transport layer, which provides guaranteed delivery of correct data. H.245 specifies syntax and semantics of terminal control messages as well as the procedures for in-band negotiation at the start or during communication. The messages cover receiving and transmitting capabilities as well as mode preference, logical channel signaling and control. The message syntax is defined using ASN.1 formatted data [12] and is transformed into bit-stream based on an ASN.1 encoding standard of Packed Encoding Rules (PER) [11].

H.245 defines a general message type *MultimediaSystemControlMessage* (MSCM). Four major types of special messages are defined in MSCM as *request*, *response*, *command* and *indication*. A *request* message results in a specific action and requires an immediate response. A *response* message responds to a request message. A *command* message requires an action but no explicit response. An *indication* message contains information that does not require action or response. Messages with various types are transformed into MSCM for uniform processing.

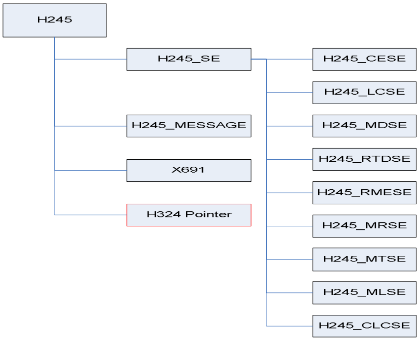


Fig. 3. Hierarchical class structure of H.245

In H.245, object *Signaling Entity* (SE) is referred to as a procedure that is responsible for special functions. It is designed as a state machine and changes its current state upon reaction to an event occurrence. With the SE objects, H.245 class hierarchy is illustrated in Fig. 3 in which all H245 procedures are packaged in H245 class. It has a member variable of H245_MESSAGE which provides message definitions and operations. A member variable of X691 is used for PER encoding/decoding and 9 member variables derived from H245_SE, each of which stands for one signaling entity object in H.245. There is also a pointer referring to upper layer protocol H.324. The class hierarchy can be further illustrated below:

H245_MESSAGE Class defines all the H245 messages and corresponding encoding/decoding functions.

H245_SE and H245_*SE: H245_SE is an abstract class and specifies the basic features and functions of a signaling entity object. H245_*SE implements H245_SE, realizing the procedures of individual signaling entities. Therefore, all subclasses H245_*SE inherit H245_SE class. In summary, SE object maps to the following SE objects:

H245_MSDSE	MasterSlaveDetermination
H245_CESE	CapabilityExchange
H245_LCSE	LogicalChannel

In addition, Class H245_SE includes some functions used in each signaling entity and inherited by the SE classes as shown below:

```
class H245_SE  public H245
{ …
public:
void h245_receive_primitive();  // check primitive event
BOOL h245_send_message(void *msg,int len); // send message to the peer
H245_SE();                  // constructor
virtual ~H245_SE();  // destructor
};
```

In each individual SE class, there are some structures, defining SE primitives and status used in this SE. For example, in a CESE, the structure below stands for TRANSFER.indications and records the primitive parameters as defined in H245 standard.

```
typedef struct H245_PRIMITIVE_DATA_TRANSFER_INDICATION

{
        ObjectID                    protocolIdentifier;
        MultiplexCapability         multiplexCapability;
        _setof13                    *capabilityTable;
        _setof14                    *capabilityDescriptors;
} H245_PRIMITIVE_DATA_TRANSFER_INDICATION;
```

With the hierarchical design of the message processing, the implementation of the protocol stack is very clear and efficient.

3.3 Class H.223

H.223 class provides low delay and overhead by using segmentation, reassembly and combination of information from different logical channels into a single packet. It performs the multiplexing of multimedia data into bit-streams before transmission to an air-interface. H.223 consists of Multiplex (MUX) Layer and Adaptation Layer (AL). AL is actually an interface for upper-layer applications and deals with different sources separately. The MUX layer performs the actual multiplexing. In this layer, data traffic from different sources can be multiplexed into one packet according to some rules which are exchanged by two terminals during the initialization of communication. H223 class is designed to hold the following functions: (1) Provide sending and receiving interface for video transmission, such as send/receive closing flags to accomplish the level-setup procedure of H324 class and control messages for H245 class etc. (2) An interface to the multimedia devices or functions for handling the captured video/audio data, performing multiplexing, send them to the peer terminal, and demultiplexing the video/audio data from the received data stream and forward them to the upper-layer multimedia devices. The following figure shows the general structure of class H223.

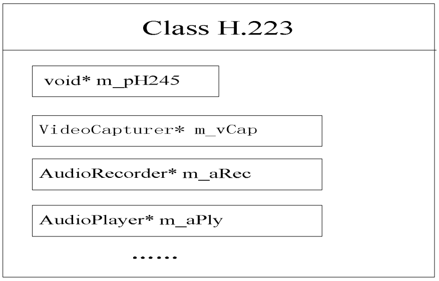


Fig. 5. Class H223 and major instance variables

The pointer *m_pH245* points to H245 object with the interface to notify H245 object of incoming H.245 control messages. The pointer *m_vCap*, ponting to the VideoCapturer class, indicates the interface to manage the operations (such as initialization, start/stop of capturing, etc.) of the video capturing device. Pointers *m_aRec* and *m_aPly* separately point to classes AudioRecorder and AudioPlayer to provide the interface for managing the audio devices.

4 Conclusions

In our implementation, object oriented approach is used for efficient control and modularization of overall protocol stack. Therefore, the complex structure of the implementation is reduced. We have discussed the point-to-point multimedia transmission implementation for the 3G terminals. With the key technologies used mentioned above, our implementation has gained a satisfactory performance. Currently we are implementing the multipoint multimedia transmission using in the applications of video conferencing or group meeting. The implementations of video-conferencing in 3G terminals can be a major challenge as multipoint communications require more resources, facing the 64kbps for a single streaming in current 3G setting in Hong Kong.

References

[1] ITU-R Rec. PDNR WP8F, “Vision, Framework and Overall Objectives of the Future Development of IMT-2000 and Systems beyond IMT-2000”, 2002.

[2] ITU-T Rec. H.324, “Terminal for low bit rate multimedia communication”, March 2002.

[3] ITU-T Rec. H.245, “Control protocol for multimedia communication”, July 2003.

[4] ITU-T Rec. H.223, “Multiplexing protocol for low bit rate mobile multimedia communication”, July 2001.

[5] B. Han, H. Fu, J. Shen, P. O. Au and W. Jia, “Design and Implementation of 3G-324M - An Event-Driven Approach”, IEEE VTC’04 Fall.

[6] H. Fu, B. Han, P. Au and W. Jia, “Efficient Data Transmission Multiplexing in 3G Mobile Systems”, Globe Mobile Congress 2004.

[7] ITU-T Rec. T.120, “Data protocols for multimedia data conferencing”, 1996.

[8] ITU-T Rec. H.263, “Video coding for low bit-rate communication”, 1998.

[9] ITU-T Rec. H.261, “Video codec for audiovisual services at p×64kbit/s”, 1993.

[10] ITU-T Rec. G.723.1, "Speech coders: Dual rate speech coder for multimedia communications transmitting at 5.3 and 6.3 kbit/s", 1996.

[11] ITU-T Rec. X.691, "Information technology–ASN.1 encoding rules–Specification of Packed Encoding Rules (PER)", 1997.

[12] ITU-T Rec. X.680, "Information Technology – Abstract Syntax Notation One (ASN.1) – Specification of basic notation", 1994.

[13] 3GPP TS 26.071 V4.0.0, "AMR Speech Codec; General Description", 2001.

[14] 3GPP TS 26.111 V5.1.0, "Codec for circuit switched multimedia telephony service: Modifications to H.324", June, 2003.

[15] Ly Q., Huang B., Wang F., "The mechanism of ASN.1 encoding & decoding implementation in network protocols", *Proceeding of International Conference on Information Technology: Coding and Computing*, pp 622-626, Apr. 2003.

Efficient Techniques and Hardware Analysis for Mesh-Connected Processors

Wu Jigang[1], Thambipillai Srikanthan[1], and Schröder Heiko[2]

[1] Centre for High Performance Embedded Systems,
Nanyang Technological University, Singapore, 639798
[2] School of Computer Science and Information Technology,
RMIT, Melbourne, Australia

Abstract. This paper proposes efficient techniques to reconfigure a multi-processor array, which embedded in a 6-port switch lattice in the form of a rectangular grid. It has been shown that the proposed architecture with 6-port switches eliminate gate delays and notably increase the harvest when compared with one using 4-port switches. A new rerouting algorithm combines the latest techniques to maximize harvest without increase in reconfiguration time. Experimental results show that the new reconfiguration algorithm consistently outperforms the most efficient algorithm proposed in literature.

Keywords: Mesh, parallel processing, reconfiguration, fault-tolerance, algorithm.

1 Introduction

The mesh-connected processor array has a regular and modular structure and allows fast parallel implementation of most signal and image processing algorithms. In this paper, the original array after manufacturing is called a host array which may contain faulty elements. A degradable sub-array of the host array, which contains no faulty element, is called a target array or logical array. The rows (columns) in the host array and target array are called physical rows (columns) and logical rows (columns), respectively. we consider the following reconfiguration problem:

Given an $m \times n$ mesh-connected host array H, integers r and c, find a $m' \times n'$ fault-free subarray T under the row and column rerouting scheme[4] such that $m' \geq r$ and $n' \geq c$.

In this paper, we focus on the design and analysis of efficient heuristic algorithms for the problem since it is NP-complete[2]. Many related research results have been presented. Recently, [2] studied the problems under different routing constraints. [3] and [4] proposed the greedy algorithms and [5] improved the running time of the reconfiguration algorithms in [4], without loss of performance. The algorithms in [2-5] are based on the array connected by 4-port switches, and each processor has two internal bypass links. In order to increase the harvest and to minimize gate delays, we combine one 4-port switch and one bypass link to form

M. Hobbs, A. Goscinski, and W. Zhou. (Eds.): ICA3PP 2005, LNCS 3719, pp. 442–446, 2005.

one 6-port switch. Unlike the old architecture, the new one supports rerouting two neighboring elements lying in same physical row into same logical column to obtain higher harvest. At the same time, there are no extra gate delay in bypass function due to the introduction of a bypass link within the 6-port switch. In reconfiguration algorithm, we present a new rerouting algorithm based on the architecture. The time complexity of the new algorithm is controlled in the same order as that of the algorithm in [4], and the performance improves significantly.

2 Architecture and Hardware Overhead

In old architecture (Fig.1, left), there is an extra two-gate delay when row (column) bypass needed, and two neighboring elements lying in same physical row cannot be rerouted into same logical column. If the element is faulty, it is highly probable that the internal bypass links are also faulty.

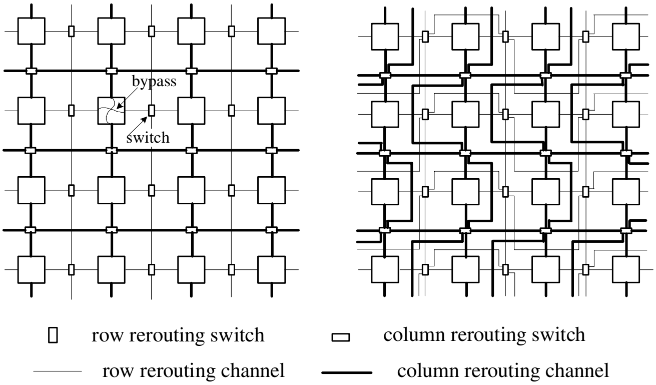


Fig. 1. 4 × 4 arrays linked by 4-port switches and by 6-port switches, respectively

We propose a new switch model to replace the 4-port switch model. The new model combines one 4-port switch and one bypass link to form one 6-port switch, which consists of pass gates to establish all possible connection pair among the six input rails. In other words, any pair combination of the six ports can be used to establish a pair wise connection. The only restriction is that no port can be connected to more than 1 port. Internal bypass links through elements are not involved as an element can be bypassed through tracks that run externally. This ensures that gate delays are avoided when a faulty element is bypassed. Unlike old architectures, the proposed architecture (Fig.1 right) is capable of allocating two neighboring fault-free elements on the same physical row into same logical column. The new architecture overcomes the drawbacks of previous approaches, be it at the expense of a small increase in hardware.

In order to fully address the likely increase in the chip area, we deduce a reasonable estimate on the likely penalty when the 4-port switches are replaced with 6-port switches. Our analysis is based on the area complexity of about 1700 gates for a 4-port switch with an 8-bit bus [6]. In [6], the entire hardware cost (in gates) of the $m \times m$ mesh array can be given as

$$\begin{aligned} G(m,p) &= 1700m(m-1) + (2160+p)m^2 + 3480m \\ &= (3860+p)m^2 + 1780m. \end{aligned}$$

The ratio of the switching circuits to the mesh array is formulated by

$$O_{switch}(m,p) = G(m,0)/G(m,p),$$

where p is the number of the gates for one processor. For an $m \times m$ mesh array connected by 4-port switches,

$$O_{switch}(m,p) = (3860m + 1780)/((3860+p)m + 1780).$$

Assuming that the area of a 6-port switch is about 50% more than that for the 4-port switch, the ratio for the case of 6-port switch is given as $1.5(3860m + 1780)/((3860+p)m+1780)$. Now, assuming the gate count for a processor (*i.e.*, p) to be 50000 gates, the chip area increases by a mere 3.59% for a 256×256 array with 6-port switches. Hence, following the traditional approach cited in the literature [1-5], it is reasonable to ignore the additional area incurred by switches, especially, for the case of that one processor is several orders of magnitude larger than the switch.

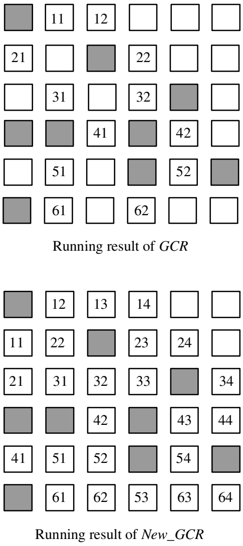
3 Algorithms

The most efficient algorithm under the constraints of *row and column rerouting* is the algorithm in [4], denoted as *RCRT* in this paper. The greedy algorithm is based on the old architecture. The crucial procedure, called *GCR*, is used for finding a target array that contains a set of selected logical rows. We improve *GCR* in this section.

Suppose v is not available as it is faulty. Its *upper (lower) neighbor* is defined as the element $er(i,j)$, where $i = row(v) - 1$ $(row(v)+1)$, and $j = col(v)$. Let $Adj(u) = \{v : v \in R_{i+1}\ \ and\ \ |col(u) - col(v)| \leq 1\}$, where the elements in $Adj(u)$ are ordered in increasing column numbers for each $u \in R_i$. Our algorithm, denoted as *New_GCR*, attempts to connect the element u to the leftmost element v of $Adj(u)$ that has not been previously examined. In *GCR*[4], if this step fails in doing so, a logical column containing the current element u cannot be formed and backtracking occurs. But in *New_GCR*, the upper (lower) neighbor of v will be examined to compensate v whenever possible. The upper neighbor of v is examined first. If the upper neighbor of v is not available, then the lower neighbor is examined. *New_GCR* backtracks to the previous element p, connected to u, only if the local compensation fails. It then attempts to connect p to the leftmost element of $Adj(p) - \{u\}$ that has not been previously examined.

Fig.2 outlines *New_GCR* and shows an running example for comparison between *GCR* and *New_GCR*. Following the analysis for *GCR*, it can be deduced that the running time of *New_GCR* is linear.

```
Input: the host arrary $H$ and the logical rows $R_0, R_1, ..., R_{k-1}$
Output: the target array with $n$ columns.
Procedure New_GCR($H, R_0, R_1, ..., R_{k-1}, n$ )
begin
    Unmark each $u$ in $R_0, R_1, ..., R_{k-1}$; $pred(u)$:=$nil$; $n$:=0;
    while there are unmarked elements in $R_0$ do
    begin
        $cur_0$ := the leftmost unmarked element in $R_0$ ,
        find the initial $cur$ in $R_{0;}$ mark $cur$;
        repeat
            if there are unmarked elements in $Adj(cur)$
            then begin
                $v$:= leftmost unmarked element in $Adj(cur)$;
                compensate $v$ with its neighbor;
                if ( $v$ is fault-free ) and (
                        $row(cur)<row(v)$ or (
                            $row(cur)=row(v)$ and
                            $col(pred(cur)) \neq col(v)$ ) )
                then begin $pred(v)$:=$cur$; $cur$:=$v$; mark $v$ end
                else Restore the changed elements
                     into their original state;
                end /* of if */
            else if $cur \notin R_1$ then $cur$:=$pred(cur)$; /* backtrack to p */
        until ($cur \in R_{k-1}$ ) or ($cur \in R_0$);
        if ($cur \in R_{k-1}$ ) then $n$:=$n$+1; /* new logical column obtained */
    end; /* of while*/
end.
```



Running result of *GCR*

Running result of *New_GCR*

Fig. 2. The formal description of *New_GCR* and running example, in which *GCR* produces a 6 × 2 target array while *New_GCR* can produces one of 6 × 4

We can obtain *New_RCRT* by replacing *GCR* in *RCRT*[4] with *New_GCR*. The time complexity of *New_RCRT* is the same as that of *RCRT* as the time complexity of *New_GCR* is the same as that of *GCR*. The further improvement for *New_RCRT* is to employ the techniques in [5]. Assume that *New_GCR* terminates at the logical row R_β in the previous iteration. Unlike *RCRT*, *New_RCRT* can directly uses R_β as the row to be excluded in the current iteration. In fact, the row R_β should be excluded with higher priority than R_γ in the current iteration. If $\gamma \neq \beta$, it is the row R_β, not R_γ, that stops *RCRT* from constructing a larger size submesh. Moreover, the implementation of this method of selecting R_β can be embedded into *New_GCR*.

4 Experimental Results and Conclusions

We implemented *New_RCRT* and *RCRT* in C on a personal computer—Intel Pentium-III 500 MHz. The average performance comparisons of both algorithms are shown in Fig.3, where *harvest* and *degradation*[4] for a range of faults are highlighted. It can be see that the average harvest of *New_RCRT* is greater than 95% and the average degradation is less than 14% for each type of random instances. In conclusion,the proposed architecture overcomes the drawback of previously reported ones to provide better harvest through improved connectivity. Rerouting via internal bypass links is avoided to eliminate gate delays. New techniques to enhance the performance of the new algorithm have been pro-

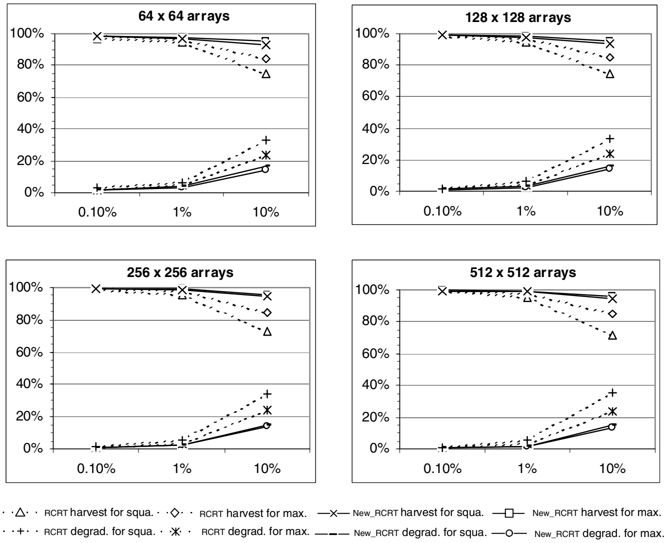


Fig. 3. Average *harvest* and *degradation* comparisons, 20 random instances with fault size 0.1%, 1% and 10% (shown in x axis) for different sized arrays

posed to benefit from the new architecture. Experimental results show that the proposed approach is superior to other alternatives reported in the literature.

References

1. T. Horita and I. Takanami, "Fault-tolerant processor arrays based on the 1.5-track switches with flexible spare distributions", *IEEE Trans. on Computers*, vol. 49, no, 6, pp. 542-552, June 2000.
2. S. Y. Kuo and I. Y. Chen, "Efficient reconfiguration algorithms for degradable VLSI/WSI arrays," *IEEE Trans. Computer-Aided Design*, vol. 11, no, 10, pp. 1289-1300, Oct. 1992.
3. C. P. Low and H. W. Leong, "On the reconfiguration of degradable VLSI/WSI arrays," *IEEE Trans. Computer-Aided Design of integrated circuits and systems*, vol. 16, no. 10, pp. 1213-1221, Oct. 1997.
4. C. P. Low, "An efficient reconfiguration algorithm for degradable VLSI/WSI arrays," *IEEE Trans. on Computers*, vol. 49, no. 6, pp.553-559, June 2000.
5. Wu Jigang and T. Srikanthan, "An Improved Reconfiguration Algorithm for Degradable VLSI arrays," *Journal of Systems Architecture*, vol.49, pp. 23-31, 2003.
6. M. Fukushi, S. Horiguchi, "Self-Reconfigurable Mesh Array System on FPGA", in Proc. of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2000, pp.240-248.

Author Index