

LNCS 2834

Xingming Zhou
Stefan Jähnichen
Ming Xu
Jiannong Cao (Eds.)

Advanced Parallel Pro- Technology

**5th International Workshop
Xiamen, China, September
Proceedings**

Lecture Notes in Computer Science 2834
Edited by G. Goos, J. Hartmanis, and J. van Leeuwen

Springer

Berlin

Heidelberg

New York

Hong Kong

London

Milan

Paris

Tokyo

Xingming Zhou Stefan Jähnichen
Ming Xu Jiannong Cao (Eds.)

Advanced Parallel Processing Technologies

5th International Workshop, APPT 2003
Xiamen, China, September 17-19, 2003
Proceedings



Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Xingming Zhou
Ming Xu
National Laboratory for Parallel and Distributed Processing
Changsha, Hunan 410073, China
E-mail: xmzhou@nudt.edu, xuming64@cs.hn.cn

Stefan Jähnichen
Institute for Computer Architecture and Software Technology at Fraunhofer Society
(FhG FIRST)
Kekuléstr. 7, 12489 Berlin, Germany
E-mail: jaehn@first.fraunhofer.de

Jiannong Cao
Hong Kong Polytechnic University
Department of Computing
Hung Hom, Kowloon, Hong Kong
E-mail: csjcao@comp.polyu.edu.hk

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress.

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliografie;
detailed bibliographic data is available in the Internet at <<http://dnb.ddb.de>>.

CR Subject Classification (1998): D, B, C, F.1-3, G.1-2

ISSN 0302-9743

ISBN 3-540-20054-1 Springer-Verlag Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer-Verlag. Violations are liable for prosecution under the German Copyright Law.

Springer-Verlag Berlin Heidelberg New York
a member of BertelsmannSpringer Science+Business Media GmbH

<http://www.springer.de>

© Springer-Verlag Berlin Heidelberg 2003
Printed in Germany

Typesetting: Camera-ready by author, data conversion by PTP-Berlin GmbH
Printed on acid-free paper SPIN: 10954836 06/3142 5 4 3 2 1 0

Preface

This volume contains the papers presented at the 5th International Workshop on Advanced Parallel Processing Technologies, APPT 2003. This series of workshops is designed to strengthen the cooperation between the German and Chinese institutions active in the area of these technologies. It has continued to grow, providing an excellent forum for reporting advances in parallel processing technologies. The 5th workshop itself addressed the entire gamut of related topics, ranging from the architectural aspects of parallel computer hardware and system software to the applied technologies for novel applications.

For this workshop, we received over 191 full submissions from researchers all over the world. All the papers were peer-reviewed in depth and qualitatively graded on their relevance, originality, significance, presentation, and the overall appropriateness for their acceptance. Any concerns raised were discussed in the program committee. The organizing committee did an excellent job in selecting 78 papers (Among them, 21 were short ones) for presentation. In short, the papers included here represent the forefront of research from China, Germany, and the other countries.

The convocation of APPT 2003 was made possible by the collective efforts of many people and organizations. We would like to express our special thanks to the Architecture Professional Committee of China Computer Federation (APC-CCF), the National Natural Science Foundation of China (NSFC), and the Xiamen Software Park. Without the extensive support from many communities for both the technical program and the local arrangements, we would not have been able to hold the workshop in Xiamen. Our thanks also go to Springer-Verlag for its assistance in putting the proceedings together.

We would like to take this opportunity to thank all the authors, many of whom travelled great distances to participate in this workshop and make their valuable contributions. We would also like to express our gratitude to the program committee members and all the other reviewers for the time and work they put into the thorough reviewing of the large number of papers submitted. Last but not least, our thanks also go to the local organizing committee for the great job it did in making the local arrangements and organizing an attractive social program.

September 2003

Xingming Zhou

Organization

General Chair

Prof. Xingming Zhou

National Laboratory for Parallel and Distributed Processing, China
(Member of Chinese Academy of Sciences, China)

Co-chairs

Prof. Dr. S. Jähnichen

Institute for Computer Architecture and Software Technology at Fraunhofer Society, (FhG FIRST), Germany

Prof. Xicheng Lu

National Laboratory for Parallel and Distributed Processing, China
(Member of Chinese Academy of Engineering, China)

Program Committee Members

Prof. Binxing Fang

Harbin Institute of Technology (China)

Prof. Xiaoming Li

Peking University (China)

Prof. Xinda Lu

Shanghai Jiao Tong University (China)

Prof. Siwei Luo

Northern Jiao Tong University (China)

Prof. Weimin Zheng

Tsinghua University (China)

Prof. Chengde Han

Institute of Computing, CAS (China)

Prof. Xinsong Liu

Electronical Sciences University (China)

Prof. Ming Xu

National Lab for Parallel and Distributed Processing (China)

Prof. Y. K. Chan

City University of Hong Kong (Hong Kong)

Dr. Jiannong Cao

Hong Kong Polytechnic University

(Hong Kong)

Prof. Chengzheng Sun

Griffith University (Australia)

Dr. Wentong Cai

Nanyang Technological University

(Singapore)

Dr. Kang Zhang

University of Texas at Dallas (USA)

Dr. Cheng-Zhong Xu

Wayne State University (USA)

Prof. Yi Pan

Georgia State University (USA)

Prof. Dr. A. Bode

Technical University of Munich (Germany)

Prof. Dr. M. Dal Cin

University of Erlangen (Germany)

Prof. Dr. K.-E. Grosspietsch

GMD St. Augustin (Germany)

Prof. Dr. R. Hartenstein

University of Kaiserslautern (Germany)

Prof. Dr. J. Kaiser

University of Ulm (Germany)

VIII Organization

Prof. Dr. K. Kleine	Jena University of Applied Sciences (Germany)
Prof. Dr. E. Maehle	Medical University of Luebeck (Germany)
Prof. Dr. W. Schröder-Preikschat	Otto von Guericke University, Magdeburg (Germany)
Prof. Dr. D. Tavangarian	University of Rostock (Germany)
Prof. Dr. W. Zimmermann	Martin Luther University, Halle-Wittenberg (Germany)

Organization Committee Members

Prof. Zhigang Luo (Chair)	National Laboratory for Parallel and Distributed Processing, China
Dr. Xiaodong Wang	National Laboratory for Parallel and Distributed Processing, China
Mr. Bing Liu	Information Engineering College, Xiamen Software Park, China
Dr. Jens Gerlach	Institute for Computer Architecture and Software Technology at Fraunhofer Society, (FhG FIRST), Germany

Table of Contents

Part I Architecture

Using Split Queues to Improve the Performance of Parallel Switch	3
<i>Xiaofeng Hu, Zhigang Sun, Xicheng Lu, Jinshu Su</i>	
LEAP: A Data Driven Loop Engine on Array Processor.....	12
<i>Yong Dou, Xicheng Lu</i>	
A New Architecture of a Fast Floating-Point Multiplier	23
<i>Xu Zhou, Zhimin Tang</i>	
A Highly Efficient FC-SAN Based on Load Stream	31
<i>Jiwu Shu, Jun Yao, Changdong Fu, Weimin Zheng</i>	
A New High-Performance Distributed Shared I/O System	41
<i>Qiong Li, Guangming Liu, Yufeng Guo, Hengzhu Liu</i>	
IA64 Oriented OpenMP Compiler: Design and Implementation of Fortran Front End	50
<i>Hongshan Jiang, Suqin Zhang, Jinlan Tian</i>	
An Alternative Superscalar Architecture with Integer Execution Units Only	57
<i>Kenneth K.C. Lee, K.-E. Grosspietsch, Y.K. Chan</i>	
Paper withdrawn	66
A High Efficiency Distributed Mutual Exclusion Algorithm	75
<i>Dan Liu, Xinsong Liu, Zhijie Qiu, Gongjun Yan</i>	
The Security Architecture of the Java Operating System JX – A Security Architecture for Distributed Parallel Computing	85
<i>Christian Wawersich, Meik Felser, Michael Golm, Jürgen Kleinöder</i>	
Simultaneous Multithreading Trace Processors	96
<i>K.F. Wang, Zhenzhou Ji, Mingzeng Hu</i>	
A VLSI Architecture Design of 1-D DWT	104
<i>Zhiyang Cao, Zhenzhou Ji, Mingzeng Hu</i>	
Overcoming Static Register Pressure for Software Pipelining in the Itanium Architecture	109
<i>Haibo Lin, Wenlong Li, Zhizhong Tang</i>	

Separating Data Storage, Data Computation, and Resource Management One from Another in Operating Systems	114
<i>Fuyan Liu, Jinyuan You</i>	
A High Performance Design and Implementation of the Virtual Interface Architecture	125
<i>Yu Chen, Zhenqiang Jiao, Jun Xie, Zhihui Du, Peng Liu, Ziyu Zhu</i>	
A Portable Debugger for PVM / MPI Programs on IA64 Cluster	136
<i>Xi Qian, Jian Liu, Weimin Zheng</i>	
Optimization of Asynchronous Volume Replication Protocol	140
<i>Huanqing Dong, Zhanhuai Li</i>	
Predicate Analysis Based on Path Information	147
<i>Li Shen, Zhiying Wang, Jianzhuang Lu</i>	
kd-Clos: New No-Blocking Permutation Network	152
<i>Mei Ming, Dong Li, Bo Fu</i>	
LilyTask: A Task-Oriented Parallel Computation Model	157
<i>Tao Wang, Xiaoming Li</i>	
A Method of Data Assignment on Heterogeneous Disk System	162
<i>Jingli Zhou, Dong Xiang, Shengsheng Yu, Lin Zhong, Jian Gu</i>	
Apply Aggregate I/O to Improve Performance of Network Storage Based on IP	167
<i>Qiang Cao, Changsheng Xie</i>	
Orthogonal Design Method for Optimal Cache Configuration	172
<i>Hongsong Chen, Zhenzhou Ji, Mingzeng Hu</i>	
A Probabilistically Correct Election Protocol in Asynchronous Distributed Systems	177
<i>Sung-Hoon Park</i>	

Part II Software and Theory

A Formal Specification and Method for MAS as a Distributed System	189
<i>Yan Qi, Xin Wang, Wei Yan, Xinjun Mao, Zhichang Qi</i>	
Optimal Fixed Priority Assignment with Limited Priority Levels	194
<i>Xuelian Bin, Yuhai Yang, Shiyao Jin</i>	
A Proof Assistant for Mobile Processes	204
<i>Xutao Du, Zhoujun Li</i>	

Data Space Fusion Based Approach for Effective Alignment of Computation and Data	215
<i>Jun Xia, Xuejun Yang, Huadong Dai</i>	
Optimization Parameter Selection by Means of Limited Execution and Genetic Algorithms.....	226
<i>Yonggang Che, Zhenghua Wang, Xiaomei Li</i>	
Study on CORBA-Based Load Balance Algorithm	236
<i>Gongxuan Zhang, Jianfang Ge, Changan Jiang</i>	
Parallel Algorithm for Mining Maximal Frequent Patterns.....	241
<i>Hui Wang, Zhiting Xiao, Hongjun Zhang, Shengyi Jiang</i>	
Design of Cluster Safe File System	249
<i>Gang Zheng, Kai Ding, Zhenxing He</i>	
Graph Scaling: A Technique for Automating Program Construction and Deployment in ClusterGOP	254
<i>Fan Chan, Jiannong Cao, Yudong Sun</i>	
Pattern Classification with Parallel Processing of the Cellular Neural Networks-Based Dynamic Programming	265
<i>Hyongsuk Kim, Taewan Oh, Sangik Na, Changbae Yoon</i>	
An Effective Molecular Algorithm for Solving the Satisfiability Problem	274
<i>Wen Yu, Weimin Zheng</i>	
Scheduling Outages in Distributed Environments.....	281
<i>Anthony Butler, Hema Sharda, David Taniar</i>	
An Improved Parallel Algorithm for Certain Toeplitz Cyclic Tridiagonal Systems on Distributed-Memory Multicomputer	292
<i>Xuebo Zhang, Zhigang Luo, Xiaomei Li</i>	
Generic Programming for Scientific Computing in C++, Java TM , and C#	301
<i>Jens Gerlach, Joachim Kneis</i>	
A General Metric of Load Balancing in δ -Range	311
<i>Xuejun Yang, Huadong Dai, Yuhua Tang, Xiaodong Yi</i>	
Lattice Boltzmann Simulations of Fluid Flows	322
<i>Baochang Shi, Nangzhong He, Nengchao Wang, Zhaoli Guo, Weibin Guo</i>	

Part III Grid and Network

Design and Research of Strong-Mobile Agent Based Grid's Architecture.....	335
<i>Zhirou Zhang, Siwei Luo</i>	
A Cost-Based Online Scheduling Algorithm for Job Assignment on Computational Grids	343
<i>Chuliang Weng, Xinda Lu</i>	
Composition and Automation of Grid Services	352
<i>Zhihong Ren, Jiannong Cao, Alvin T.S. Chan, Jing Li</i>	
Algorithmic Skeletons for Metacomputing	363
<i>Martin Alt, Sergei Gorlatch</i>	
Grid Services Performance Tuning in OGSA	373
<i>Chuan He, Bin Du, Sanli Li</i>	
A Transaction Model for Grid Computing.....	382
<i>Feilong Tang, Minglu Li, Jian Cao</i>	
An Overview of Research on QoS Routing.....	387
<i>Yanxing Zheng, Wenhua Dou, Jing Tian, Mingyan Xiao</i>	
A Novel Model and Architecture on NMS – Dynamically Constructed Network Management	398
<i>Zhongzhi Luan, Depei Qian, Xingjun Zhang, Tao Liu, Heng Chen</i>	
QoS-Driven Multicast Tree Generation Using Genetic Algorithm	404
<i>Xingwei Wang, Hui Cheng, Jiannong Cao, Zhijun Wang, Min Huang</i>	
A Scalable Peer-to-Peer Network with Constant Degree	414
<i>Dongsheng Li, Xinxin Fang, Yijie Wang, Xicheng Lu, Kai Lu, Nong Xiao</i>	
Symmetric Distributed Server Architecture for Network Management System	425
<i>Xiaolin Lu</i>	
A Distributed Network Management Framework Based on NGI	430
<i>Jinxiang Zhang, Jilong Wang, Jianping Wu, Xing Li</i>	
Performance Evaluation of Scheme Integrating Mobile IP and NHRP over ATM Networks	435
<i>Tae-Young Byun, Min-Su Kim, Ki-Jun Han</i>	

Improving Availability of P2P Storage Systems	446
<i>Shuming Shi, Guangwen Yang, Jin Yu, Yongwei Wu, Dingxing Wang</i>	
Research and Implementation of Dynamic Web Services Composition	457
<i>Haiyan Sun, Xiaodong Wang, Bin Zhou, Peng Zou</i>	
Call Admission Control for Multimedia CDMA Networks under Imperfect Power Control	467
<i>Xiaoming Bo, Zujue Chen</i>	
Efficient Data Consistency Schemes in 2-Tier Cellular Networks	478
<i>Seungbum Lim, Jai-Hoon Kim, Young-Bae Ko</i>	
iHOPE: An Intelligent Handoff Protocol for Seamless Multimedia Service in Wireless Network	483
<i>Jang-Woon Baek, Dae-Wha Seo</i>	
Content and Cell Based Predictive Routing (CCPR) Protocol for Mobile Ad Hoc Networks	488
<i>Jörg Kaiser, Changling Liu</i>	
A Multiple Access Scheme Using Split and Merge Algorithm for Contention/Reservation-Based Wireless MAC Protocols	496
<i>Min-Su Kim, Tae-Young Byun, Ki-Jun Han</i>	

Part IV Applied Technologies

Reconfigurable Cipher Processing Framework and Implementation	509
<i>Jingfei Jiang, Xiaoqiang Ni, Minxuan Zhang</i>	
A Dynamic Reconfiguration Platform Based on Distributed Component Technology CCM	520
<i>Lei Dou, Quanyuan Wu, Yan Jia, Weihong Han</i>	
TMO-Based Object Group Framework for Supporting Distributed Object Management and Real-Time Services	525
<i>Chang-Sun Shin, Myoung-Suk Kang, Chang-Won Jeong, Su-Chong Joo</i>	
Extendable and Interchangeable Architecture Description of Distributed Systems Using UML and XML	536
<i>Changai Sun, Jiannong Cao, Maozhong Jin, Chao Liu, Michael R. Lyu</i>	
Dynamics in Hierarchical CSCW Systems	546
<i>Xiaodong Wang, Haiyan Sun, Ming Xu, Xingming Zhou</i>	
The Tenure Duty Method (TDM) in the Active Incident Recovery Research	557
<i>Zunguo Huang</i>	

A Combined Continuous-Time/Discrete-Event Computation Model for Heterogeneous Simulation Systems	565
<i>Andreas Franck, Volker Zerbe</i>	
Information Geometry on Modular and Hierarchical Neural Network	577
<i>Yunhui Liu, Siwei Luo, Aijun Li, Hua Huang, Jinwei Wen</i>	
A Novel Speed-Up Algorithm of Fractal Image Compression	582
<i>Yusong Tan, Xingming Zhou</i>	
Implementation and Evaluation of a Novel Parallel SAR Imaging Method on Clustering Systems.....	590
<i>Jizhong Han, Shichao Ma</i>	
Shape Registration Based on Modified Chain Codes	600
<i>Muhammad Bilal Ahmad, Jong-An Park, Min Hyuk Chang, Young-Suk Shim, Tae-Sun Choi</i>	
A Distributed Parallel Resampling Algorithm for Large Images	608
<i>Yanhua Jiang, Xuejun Yang, Huadong Dai, Huizhan Yi</i>	
Collaborative Supervision of Machine Vision Systems: Breaking a Sequential Bottleneck in the Supervised Learning Process	619
<i>Steve Drew, Sven Venema, Phil Sheridan, Chengzheng Sun</i>	
Parallel Storing and Querying XML Documents Using Relational DBMS	629
<i>Jie Qin, Shuqiang Yang, Wenhua Dou</i>	
A Variable Consistent Server Replication Model for Mobile Database	634
<i>Jinhui Xu, Ming Xu</i>	
Multi-scheduler Concurrency Control for Parallel Database Systems	643
<i>Sushant Goel, Hema Sharda, David Taniar</i>	
Optimal Broadcast Channel for Data Dissemination in Mobile Database Environment	655
<i>Agustinus Borgy Waluyo, Bala Srinivasan, David Taniar</i>	
Global System Image Architecture for Cluster Computing	665
<i>Hai Jin, Li Guo, Zongfen Han</i>	
Author Index	677

Part I

Architecture

Using Split Queues to Improve the Performance of Parallel Switch*

Xiaofeng Hu, Zhigang Sun, Xicheng Lu, and Jinshu Su

School of Computer,
National University of Defense Technology,
Changsha, Hunan,
P.R. China, 410073
`{xfhu, sunzhigang, xclu, jssu}@nudt.edu.cn`

Abstract. Parallel switch scales well with the growth of port density and line rate. PSIQC (Parallel Switch based on Input-Queued Crossbar) is a parallel switch that is scalable and simple to implement. But it needs large capacity high-speed memories to store cells, and the average cell latency is high under heavy load. This paper presents a revised version of PSIQC based on split queues that is initiated as SQ-PSIQC (Split Queued Parallel Switch based on Input-Queued Crossbar), and its scheduling algorithm SQ-RRDS (Split Queued Round Robin and Deterministic Sequence). SQ-PSIQC not only has all of the original characteristics, but also solves the two above-mentioned problems. In SQ-PSIQC the memory buffers are required to operate only at $1/m$ of the line rate, where m is the number of the middle switches. The simulation results show that SQ-PSIQC performs better than PSIQC in the average latency and throughput under any load, especially heavy load.

1 Introduction

Today, most of the high performance routers use the centralized switch fabrics, such as the crossbar [1][2][3] or the shared memory [4][5]. Although widely deployed, the centralized switches are of poor scalability. The need for switches with more ports and faster line rates makes them impractical to implement [6][7][8][9][10]. To improve scalability, the parallel switching technology is proposed [7][8][9][11].

The parallel switch comprises of multiple input/output controllers and multiple identical switch modules operating at lower speed than external line. These switch modules operate independently and in parallel, so the parallel switch is scalable and easy to implement. There are two problems to be solved in designing parallel switch. One is to distribute the load equally, which is crucial to the performance of parallel switch. If the loads are balanced in the switch modules, the capacity of the parallel switch can be increased according to our demand. Another problem is to prevent packet reordering since mis-sequencing can cause TCP retransmits and deteriorate the performance of the TCP connections [12].

* Supported by: (1) the National Natural Science Foundation of China (No. 90104001); (2) the National High Technology Research and Development Plan of China (No. 2001AA112120).

PPS [7][8] (Parallel Packet Switch) and ADSA [9] (A Distributed Switch Architecture) are typical parallel switches. PPS's middle switch modules are output-queued switch modules, and the queue management policies are complicated in output ports. So, it is difficult to implement. As to ADSA, it uses input-queued switch modules. The modules are customized which communicate with external ports in the request-acknowledge way. The external ports need high-speed memories. Like PPS, the queue management policies in output ports are complicated. In addition, both PPS and ADSA need a timestamp or sequence in the cell head to restore the original cell order, which will waste bandwidth.

To solve the above-mentioned problems, we presented PSIQC (Parallel Switch based on Input-Queued Crossbar) in our previous work, which uses the general-purpose input-queued crossbar switch modules. PSIQC works in a distributed way. When a packet arrives, it is divided into fixed-length cells. Then, the cells are buffered at different queues in input port according to their destination ports. The input port selects a queue to serve in round robin order, and disperses its head-of-line cell into a crossbar switch module in a deterministic sequence. The crossbars forward the cells from the input ports to the output ports. After receiving the cells, the output port reads the cells in the same deterministic sequence and reassembles them as a packet. PSIQC has the following advantages: (1) It has good scalability and is easy to implement. (2) It can guarantee the packet order. (3) There is no communication cost. (4) On our simulations, it can guarantee load balancing in the switch modules, and the throughput and the average cell latency are close to those of the centralized crossbar switch.

However, despite of its advantages, the VOQs in input ports need high capacity memories operating at external line rate, which will make the implementation difficult. Additionally, the average cell latency is a little higher than that of the centralized crossbar under heavy load. To solve these problems, we have proposed SQ-PSIQC (Split Queued PSIQC) based on the queue splitting idea, a revised version of PSIQC, while keeping the characteristics of PSIQC.

In the rest of this paper, we describe the SQ-PSIQC switch model in section 2, introduce the input/output port scheduling algorithm SQ-RRDS in section 3, show the simulation results and compares the performance of SQ-PSIQC with PSIQC and centralized crossbar in section 4, and finally summarize our findings.

2 SQ-PSIQC Switch Model

SQ-PSIQC switch, shown in figure 1, is composed of N inputs/output controllers and m input-queued crossbar switch modules. The input port controller has a cell enqueuer (CQ) and a load dispatcher (LD). The output port controller has a cell integrator (CI). Supposing the external line rate is R , the operating rate of each switch module is $r=R/m$, and the time spent to transmit or receive a cell at rate R is a time slot. Denote the beginning time of the r th time slot as t_r . Define as a flow (i, j) the cells that are received from input port i and transmitted to output port j .

A distinct architectural difference between SQ-PSIQC and PSIQC is that there are $(m \cdot N)$ VOQs in each input controller in SQ-PSIQC instead of N VOQs in PSIQC. Each VOQ in PSIQC is divided into m VOQs corresponding to different switch modules respectively. Every N VOQs buffering cells destined to different output ports through the same switch module belong to a group. The VOQs in a group compete for access to the switch module with each other. We use $VOQ_{(i,j,l)}$ to denote the VOQs, where i represents the input port, j the output port, and l the middle switch module.

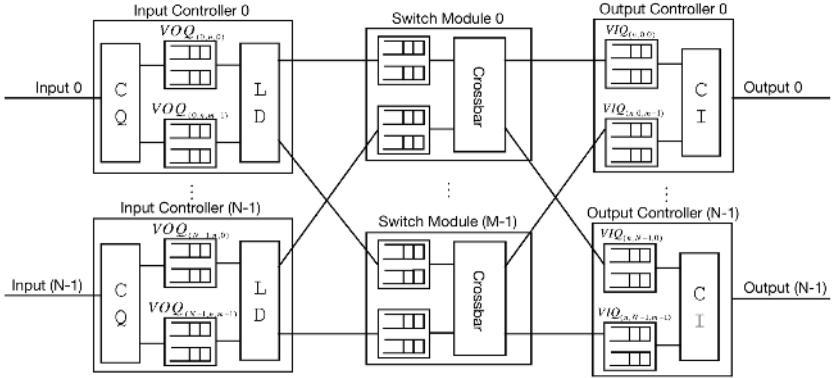


Fig. 1. SQ-PSIQC Switch Model

Each output controller manages $(m \cdot N)$ VIQs as PSIQC. These queues are used to buffer cells passing through different middle switches. We use $VIQ_{(i,j,l)}$ to denote VIQ.

CQ is responsible for deciding which switch modules will process the received cells and dispatching them into VOQs. LD schedules which cells will enter into switch modules and transmits them. Cells are then forwarded to destination ports by the switch modules and enter into VIQs. Finally CI transmits the cells to external line in order.

CQ, LD, switch modules, and CI compose the data path in SQ-PSIQC. They work in a distributed way and make their scheduling decisions independently. The SQ-RRDS algorithm determines their work process. CQ disperses received cells to the VOQs in a deterministic sequence. Both LD and CI work in a cycle of m time slots. During each cycle LD transmits up to m cells to the switch modules and decides which cells will be transmitted in the next cycle based on the VOQs' status. CI simultaneously transmits up to m cells to external line and decides which cells will be transmitted in the next cycle. It is the switch modules that forward the cells. The scheduling algorithm will influence the performance of SQ-PSIQC, so it should be effective and simple to implement. We will not be concerned at these algorithms in this paper. Since iSLIP [13] and ISP [14] are two typical input-queued crossbar scheduling algorithms that have been used in high performance commercial routers, we suppose that all the switch modules use one of them.

3 SQ-RRDS Algorithm

SQ-RRDS is a distributed algorithm including cell enqueueing algorithm, load dispatching algorithm and cell integrating algorithm. Each input and output controller runs SQ-RRDS based on its status.

3.1 Cell Enqueueing Algorithm

Definition 1 (Target Switch Module Pointer): *Indicates the switch module that flow (i, j) should select, denoted as $NM_{(i,j)}$.*

For any input controller $i (\forall i \in \{0, 1, \dots, N-1\})$, the cell enqueueing algorithm is described as following:

```

1 Initialization:    $NM_{(i,j)} = 0$  ,    $j = 0, 1, \dots, N-1$ 
2 Scheduling:
    output=Destination Port of the input cell
    mid =  $NM_{(i,output)}$ 
    Put the cell into  $VOQ_{(i,output,mid)}$ 
     $NM_{(i,output)} = (NM_{(i,output)} + 1) \bmod m$ 

```

Fig. 2. Cell Enqueueing Algorithm

3.2 Load Dispatching Algorithm

Definition 2 (Switch Module-Input Port Round Robin Pointer): *Indicates the VOQ having the highest priority within a VOQ group in an input port. Denote the round robin pointer corresponding the i th input port and the l th switch module as $RR_{(i,l)}$.*

For any input controller $i (\forall i \in \{0, 1, \dots, N-1\})$, the load dispatching algorithm is described as following:

```

1 Initialization:    $RR_{(i,l)} = 0$  ,    $l = 0, 1, \dots, m-1$ 
2 Scheduling:
    FOR (mid=0; mid<m; mid++)
        output =  $RR_{(i,mid)}$ 
        FOR (loop=0; loop<N; loop++)
            IF ( $VOQ_{(i,output,mid)}$  is not empty)
                Select the HoL cell in  $VOQ_{(i,output,mid)}$ 
                 $RR_{(i,mid)} = (RR_{(i,mid)} + 1) \bmod N$ 
                Continue to process the next group
            ELSE output = (output + 1) mod N

```

Fig. 3. Load Dispatching Algorithm

3.3 Cell Integrating Algorithm

Definition 3 (Full Block/Non-Full Block): For $l(\forall l \in \{0, 1, \dots, m-1\})$, if all the queues ($VIQ_{(i,j,l)}$) belonging to $flow(i, j)$ are not empty, the head-of-line cells of these queues compose a full block. Otherwise the head-of-line cells compose a non-full block.

Definition 4 (VIQ Pointer): Indicates the next VIQ to send cell for a flow in a deterministic sequence. We use $PVIQ_{(i,j)}$ to denote the VIQ pointer of $flow(i, j)$.

Definition 5 (Full Block Pointer): Indicates the next full block to be selected by CI . The full block pointer of CI_j is denoted as PFB_j .

Definition 6 (Non-Full Block Pointer): Indicates the next non-full block to be selected by CI . The non-full block pointer of CI_j is denoted as $PNFB_j$.

For any output controller $j(\forall j \in \{0, 1, \dots, N-1\})$, the cell integrating algorithm is described as following:

```

1 Initialization:
     $PVIQ_{(i,j)} = 0$ ,  $i = 0, 1, \dots, N-1$ 
     $PFB_j = 0$ 
     $PNFB_j = 0$ 
2 Scheduling:
    IF ( $CI_j$  has a full block)
        Select the full block with highest priority
        from  $PFB_j$  in the round robin way.
        (Suppose the input port of the full block is  $i$ )
         $PFB_j = (i+1) \bmod N$ 
         $l = PVIQ_{(i,j)}$ 
        FOR (loop=0; loop<N; loop++)
            Select the HoL cell in  $VIQ_{(i,j,l)}$ 
             $l = (l+1) \bmod m$ 
    ELSE
        Select the non-full block pointed by  $PNFB_j$ .
        (Suppose the input port of the block is  $i$ )
         $PNFB_j = (i+1) \bmod N$ 
         $l = PVIQ_{(i,j)}$ 
        WHILE ( $VIQ_{(i,j,l)}$  is not empty)
            Select the HoL cell in  $VIQ_{(i,j,l)}$ 
             $l = (l+1) \bmod m$ 
     $PVIQ_{(i,j)} = l$ 

```

Fig. 4. Cell Integrating Algorithm

The cell enqueueing and load dispatching algorithms in SQ-RRDS guarantee that the time interval between consecutive accesses to any VOQ is at least m time slots. So instead of $(R+r)$ in PSIQC, the memory bandwidth in SQ-PSIQC is reduced to $2R/m$ (one read operation and one write operation) that is inversely proportional to m and can be reduced further by increasing m . The memories in output port also operate at rate $2R/m$ as in PSIQC, because they are accessed at most once in every m time slots. From the above analysis we know that the bandwidth requirement for memories in input ports is greatly reduced by splitting the VOQs, and the first problem in PSIQC is solved. We will show in the next section that the performance, especially in the average cell latency, is remarkably improved in SQ-PSIQC. The only drawback of SQ-PSIQC is that it needs $(m \cdot N)$ VOQs in each input port while PSIQC just N VOQs.

4 Performance Evaluation

To validate the effectiveness of SQ-RRDS, we use an $N \times N$ centralized crossbar switch as the reference switch. The reference switch runs iSLIP algorithm, and its port rate is R . We also compare the performance of SQ-PSIQC with PSIQC.

Definition 7 (Algorithm Performance 3-Tuple $\langle E, P, D \rangle$): E is the load balancing factor, P is the ratio of throughput of parallel switch (SQ-PSIQC or PSIQC) to that of the centralized crossbar switch, and D is the average cell latency.

$$\begin{aligned} E(t_r, t_s) &= \frac{\max_{i=0, \dots, m-1} (k_i)}{\min_{i=0, \dots, m-1} (k_i)} \\ P(t_r, t_s) &= \frac{K_P}{K_C} \\ D(t_r, t_s) &= \sum_{j=1}^{K_P} d_j / K_P \end{aligned}$$

Here, k_i represents the number of the cells forwarded by the i th switch module in time interval (t_r, t_s) , K_P the total number of the cells forwarded by the parallel switch, K_C the number of the cells forwarded by the centralized switch, and d_j the j th cell c_j 's latency.

We compare the performance of SQ-PSIQC with PSIQC through simulation. In our simulation, the load is a uniform independent and identically distributed (i.i.d.) on-off arrival process modulated by a two-state Markov chain. The system configuration is that N equals 8, m equals 4, both the centralized switch and the switch modules use iSLIP algorithm, and the simulation runtime is 1000000 time slots.

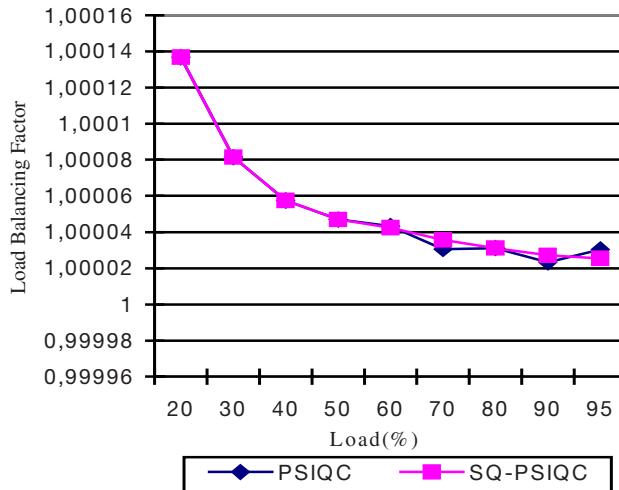
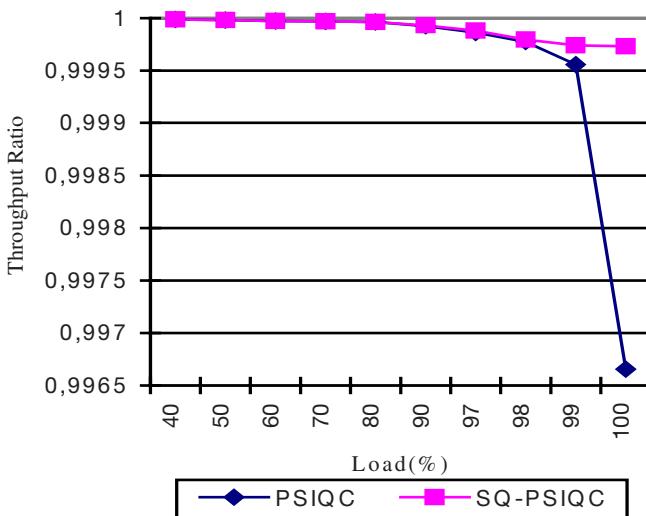
**Fig. 5.** Load Balancing Factor**Fig. 6.** Relative Throughput

Figure 5 shows that the load balancing factors of SQ-PSIQC and PSIQC are between (1.00002, 1.00014), which indicates the loads in switch modules are almost balanced. Figure 6 shows that the throughputs of centralized crossbar switch and SQ-PSIQC or PSIQC are very similar. Under heavy load, the throughput of SQ-PSIQC is larger than PSIQC. For example, under 100% load, the throughput ratio of SQ-PSIQC is 99.97%, while PSIQC's 99.66%.

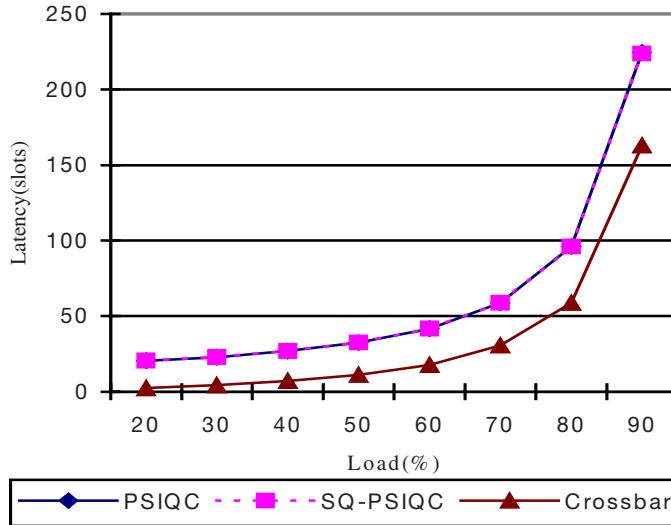


Fig. 7. Average Cell Latency under Low Load

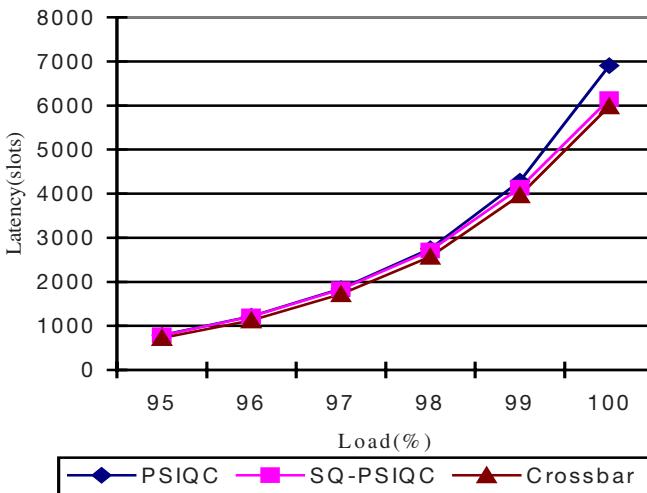


Fig. 8. Average Cell Latency under Heavy Load

Figure 7 and figure 8 show the average cell latencies under different load in SQ-PSIQC, PSIQC and centralized switch. The results are: (1) under low load, the average cell latency in SQ-PSIQC is a little lower than in PSIQC, and they are similar to that in centralized switch; (2) as load increases, the average latency in PSIQC grows up much more steeply than in SQ-PSIQC and centralized switch. Especially, when the load is 100%, the average latency in PSIQC is much larger than in centralized switch, while the average latencies in SQ-PSIQC and centralized switch are almost the same. From these two figures we know that the second problem is solved by SQ-PSIQC.

5 Conclusion

We present the SQ-PSIQC parallel switch, a revised version of PSIQC, to reduce the bandwidth requirement for the memories in the external ports and improve the average cell latency, which are two problems to be solved in PSIQC. Each VOQ in PSIQC is divided into m VOQs, and the cells are distributed into these queues in the round robin manner. In this way, the memories need operate only at $1/m$ of external line rate, inversely proportional to the number of switch modules. The LD and CI operate in cycles of m time slots, which make them simple to implement. By separating the cells to groups in SQ-PSIQC, the LD can schedule the cells in each group to the switch module independently and in parallel, thereby further reducing the implementation complexity. So, SQ-PSIQC has excellent scalability. The simulation results show that SQ-PSIQC can reduce the average cell latency and improve throughput, especially under heavy load. At the same time, SQ-PSIQC can prevent packet reordering and balance the load on switch modules.

References

1. N. McKeown. A fast switched backplane for a gigabit switched router. *Business Communications Review*, December 1997.
2. N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, M. Horowitz. Tiny Tera: a packet switch core. *IEEE Micro*, 1997, vol.17, No.1: 26–33
3. C. Partridge, P. Carvey, E. Burgess, et al. A 50Gb/s IP router. *IEEE/ACM Transactions on Networking*, 1998, vol.6 No.3: 237–248
4. M. Arpacı, J. Copeland. Buffer Management for Shared-Memory ATM Switches. *IEEE Communications Surveys*, 1Q 2000.
5. C. Semeria. Internet backbone routers and evolving Internet design. White paper, <http://www.juniper.com>
6. Sundar Iyer, Rui Zhang, Nick McKeown. Routers with a Single Stage of Buffering. In: Proc. of ACM SIGCOMM'02, Pittsburgh. September 2002.
7. S. Iyer, A. Awadallah, N. McKeown. Analysis of a packet switch with memories running slower than the line rate. In: Proc. of IEEE Infocom'00, Tel-Aviv, Israel. March 2000.
8. S. Iyer and N. McKeown. Making Parallel Packet Switches Practical. In: Proc. of IEEE Infocom'01, Alaska. April 2001.
9. W. Wang, L. Dong, and W. Wolf. A Distributed Switch Architecture with Dynamic Load-balancing and Parallel Input-Queued Crossbars for Terabit Switch Fabrics. In: Proc. of IEEE Infocom'02, New York. June 2002.
10. I. Keslassy and N. McKeown. Maintaining Packet Order in Two-Stage Switches. In: Proc. of IEEE Infocom'02, New York. June 2002.
11. F. Chiussi, D. Khotimsky, S. Krishnan. Generalized Inverse Multiplexing of Switched ATM Connections. In: Proc. of IEEE Globecom'98, Sydney, Australia. November 1998.
12. J. Bennett, C. Partridge, and N. Shectman. Packet Reordering is not Pathological Network Behavior. *IEEE/ACM Transactions on Networking*, 1999, vol.7, No.6: 789–798
13. N. McKeown. iSLIP: a scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 1999, vol.7, No.2: 188–201
14. Sun Zhigang, Su Jinshu, Lu Xicheng. ISP: a high performance crossbar arbitrating algorithm. *Journal of Computer Research and Development (In Chinese)*, 2000, vol.23, No.10: 1078–1082

LEAP: A Data Driven Loop Engine on Array Processor*

Yong Dou and Xicheng Lu

National Laboratory for Parallel and Distributed Processing of China
Changsha, Hunan, P. R. China, 410073
yongdou@163.net

Abstract. This paper presents a novel architecture for array processor, called LEAP, which is a set of simple processing elements. The targeted programs are innermost loops. By using the technique called if-conversion, the control dependence can be converted to data dependence to prediction variables. Then an innermost loop can be represented by a data dependence graph, where the vertex supports the expression statements of high level programming languages. By mapping the data dependence graph to fixed PEs, each PE steps the loop iteration automatically and independently at the runtime. The execution forms multiple pipelining chains. One example program FIR and corresponding transformation are given to show the execution process of the LEAP array. The performance analysis shows the effectiveness of the LEAP architecture, compared with CISC and RISC architectures.

1 Introduction

Integrating multiple computing elements on a single chip is a promising approach to effectively utilizing the hardware resources of the huge chip. By partitioned into independent physical regions, each of its small processors takes up a relatively small area on a large processor chip. Minimizing the length of its wires and simplifying the design of critical paths, the architecture of chip multiprocessor is a high performance and economical solution to the problem of designing microprocessors with upwards of a billion transistors. [1],[2]

This paper describes a novel architecture model for single chip array processors, called LEAP (Loop Engine for Array Processors), targeted at applications with inherent data-parallelism, high regularity requirements.

The LEAP array is a set of simple processing elements, connected by high speed networks, each PE contains an ALU for integer or float point computation, instruction registers, a small local cache, data buffers and loop control logic. The main idea of LEAP is mapping the data-flow graph of the innermost loop body to array processors. Each node of the data-flow graph, which represents a basic expression statement in high level programming language, is assigned to a PE, and the assignment is fixed during the whole execution of the loop. At the run-time, each PE steps the loop iteration automatically and independently. The data are fetched from memory, then

* This work was supported in part by NSFC (National Natural Science Foundation of China) under contract 69933030.

streamed directly from the source PEs to the destinations according to the edges in the data-flow graph, and finally written back to memory. The execution forms multiple pipeline chains. Data-driven processing units are designed to select data and compute from the data buffers whenever the data are ready. A dedicated communication network is used to send data directly between the producer and consumer along the routing depicted by the data dependency edges. This eliminates the centralized register file to store temporary results.

The LEAP array executes instructions according to FIMD mode (Fixed Instruction Multiple Data). The instruction on each PE is fixed during the whole execution process. The unique feature of FIMD model brings many advantages over other architecture models. Obviously, the first is eliminating memory accesses for loading instructions at runtime. The second is preserving the memory accesses in some regular orders. This can be easily used to design distributed share memory hierarchies without cache coherence problems. The third is that routing conflicts in the network can be avoided, because the routing information can be determined at the compiling time according to the fixed edges of the data-flow graph. Finally, parallelisms are exploited from sequential programs without the burden to parallelize programs by hand or by some complex compilers. Three levels of parallelism are exploited in the LEAP array: pipeline parallelism between loop iterations, parallelism between expression statements and parallelism between operations in an instruction.

The rest of the paper is organized as follows. Section 2 presents the program model for the LEAP architecture. In section 3, we describe the LEAP architecture, including array architecture and PE architecture. In Section 4, we will give an example program to illustrate the execution process of LEAP, and evaluate the effectiveness of LEAP architecture. Section 5 concludes the paper.

2 Program Model and Program Transformation

2.1 Program Model

A program can contain multiple levels of loops. For simplicity in this paper, we only deal with the innermost *for* loops. A normalized innermost *for* loop consists of a loop body and an index variable that is explicitly defined by initial value, limit value and step value. The statements in the loop body are classified into three kinds, conditional branch statements, (*if*, *if else*, *switch case*), stream control statements (*goto*, *continue*, *break*), and expression statements ($X = Y + Z - W * \dots$). We assume the innermost loop does not contain loop statements, procedure calls and feedback jump statements.

In general, an expression statement with more than two operands can be represented by a chain of multiple basic expression statements, such as $X = Y \text{ op } Z$. The basic expression statement is executed in three phases, firstly, loading two operands Y and Z , then computing, and finally storing the results to the location X . During the execution, the values of the two operands may be from memory, temporary registers or directly from results of other statements.

The execution of a loop can be represented by multiple instances of the loop body, which are controlled by index variable stepping from initial value to the limit value. A loop slice, LS [i], is defined as an instance of the loop body with the value of index variable i . An instance of expression statement E with value i is denoted as a

statement slice $SS(E)[i]$. A loop slice is the set of statement slices with the same index value.

2.2 Transformation from Control Flow Graph to Data Dependence Graph [3]

Using a technique called if-conversion [3], any control dependence can be converted into a data dependence, which then allows an easier way to represent the dependence and reason about them. To support the control dependence conversion, a predication execution mechanism must be implemented. First, branch statements, like *if else*, are replaced by *test* statements. The *test* statement produces a logic result, true or false to a predication variable. Then, all statements in the *if else* blocks are constrained by the predication variable which value enables or disables the execution of statements. At runtime, the result of *test* statement has three possible values, *True*, *False* or *None*, *None* means the *test* statement itself is disabled. But for consistency with other statements, it still produces a result. The statement *test* is equivalent to the assignment to prediction variables, and the constrained conditions to the execution are uses of these variables. Prediction variables are defined and used as temporary variables, which only incur data dependence within a loop slice.

After the if-conversion, each statement in the CFG (Control Flow Graph) consists of five operation fields. The first field is CEC (Conditional Execution Code) field that denotes the conditions enabling the execution of statement. It is represented by one prediction variable. LD field is used for loading operands. It contains three parts, value, base address and offset. If a variable is a scalar, its offset is 0. Otherwise the offset can be the value of an address expression or the value of other variables, which means indirect address access. Each basic expression has two LD fields, called LD1 and LD2. The ST field is used for storing results. The address is similar to LD field. The COMPUTE field is used to represent the calculation in the statement.

2.3 Transformation to Data Dependence Graph [4]

Base on the above CFG enhanced with prediction variables, the data dependence graph (DDG) can be constructed. $DDG = \langle V, E \rangle$, V is still the set of vertices, where vertices represent the basic expression statements. While E is the set of edges, where edges represent the data dependence between statements. The source and destination of an edge are marked by a 3 fields tuple. The first field represents the name of statements, the middle field represents the name of operation field in the statement, and the last field represents the position within the operation field, value or offset, as shown in below:

$$\langle \text{Node X, Field Y, Position 0} \rangle \xrightarrow{\delta^* d} \langle \text{Node A, Field B, Position 1} \rangle$$

The $*$ represents four types of data dependence, flow dependence, anti dependence, output dependence and input dependence. The d represents the dependence distance.

Edge $(X \delta^* d Y)$ means a flow edge from position X to Y , semantic of the flow edge is that assuming the value of index I , the result created in the position X at the loop iteration I will be used by statement Y at the iteration $I+d$.

Edge $(X \delta^a d Y)$ means an anti edge from position X to Y , semantic of the anti edge is that assuming the value of index I , the operand value used in the position X at the

loop iteration I will be redefined by statement Y at the iteration I+d. In sequential execution, the semantic can be easily preserved, but for parallel execution of multiple iterations, a synchronization will be used to guarantee the above semantic.

Edge ($X \delta^d Y$) means an output edge from position X to Y, semantic of the output edge is that assuming the value of index I, the operand value defined in the position X at the loop iteration I will be redefined by statement Y at the iteration I+d.

Edge ($X \delta^d Y$) means an input edge from position X to Y, semantic of the input edge is that assuming the value of index I, the operand value used in the position X at the loop iteration I will be reused by statement Y at the iteration I+d. The input dependence can be used to link multiple load operations to a chain to reduce the number of memory accesses.

According to the dependence distance d , variables in the loop can be divided into two classes, **regular variable** and **irregular variable**. For regular variables, the value of d can be determined at compiler time and is kept constant during the execution of the loop. For irregular variables, the dependence distance is either related with another variable or dynamic at runtime. For example, indirect address $X[Y[I]]$ or $X[I*I]$. The semantics of sequentially accessing irregular variables must be guaranteed by strict synchronization operations between accesses of irregular variables.

The key idea of LEAP is mapping the DDG to the array. Vertices are realized by long instructions of PEs. Edges are implemented by messages to transfer values or synchronization signals. Multiple loop slices can be issued in pipeline mode. Statement slices with the regular variables can be executed concurrently, even for the loop with irregular variables, the part of regular variables can be executed in parallel, and only accesses to irregular variables are restricted to sequential mode. The exchanges of data and synchronization are determined completely by the data dependence relationship in the DDG.

3 LEAP Architecture

3.1 LEAP Array Architecture

As shown in Figure 1, the LEAP array processor is designed as a coprocessor attached with a host processor. The host processor can be a microprocessor sharing main memory with the array processor through multiple memory channels. The main functions of the host processor are to execute operating system calls and programs that are not suitable for LEAP array, to span innermost loops to array processors, and to process the loop exit from the array.

The LEAP array processors are a set of processing elements, called PE. They are connected by memory network (MemNet) and data dependence network (DDNet), as depicted in Figure 2. The MemNet provides channels for PEs to access main memory by connecting the local data cache of PE with external storage channels in mesh topology. DDNet connecting the input/output channel of each PE is used to exchange the data dependence information between PEs, including data value, synchronization signals and values of prediction variables. The topology of DDNet is flexible. It can be mesh or multiple dimensions network. But a configurable network will be more suitable than a fixed network, since the data dependence graph of the innermost loop

has fixed structure during the whole execution. A configurable network will provide point to point interconnection between PEs for high network bandwidth. In figure 2, we only show a fixed mesh topology of DDNet for the reason of clarity.

A loop control unit, LCU is interfaced between the host and DDNet. The main functions of LCU are to record the loop commitment and downloading instructions to PEs. When the index value reaches the limit of loop boundary and all store operations of the final loop slice have been completed, LCU informs host that the loop execution is over.

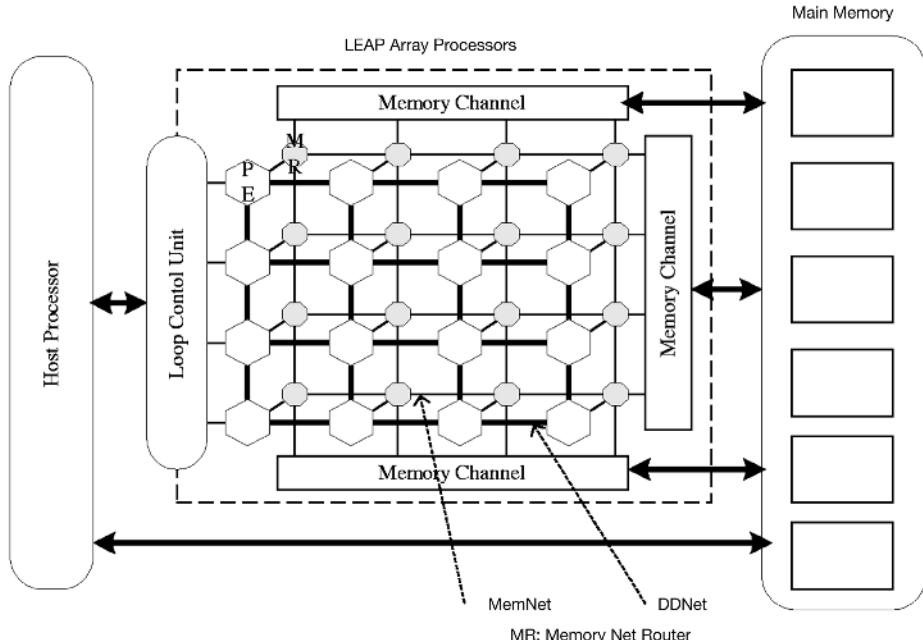


Fig. 1. The architecture of LEAP array processors

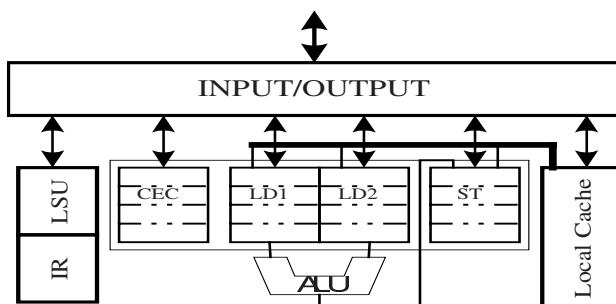


Fig. 2. The architecture of PE

(a) CEC:	prediction variable	To-PE number	field number	dependence type							
LD1:	op-code	base address	offset	To-PE number	field number	dependence type					
LD2:	op-code	base address	offset	To-PE number	field number	dependence type					
ST:	op-code	base address	offset	To-PE number	field number	dependence type					
COMPUTE:	op-code										
(b)	LD1:	load state	base address	offset value	v	address value	v	load value	v	synch value	v
	LD2:	load state	base address	offset value	v	address value	v	load value	v	synch value	v
	ST:	store state	base address	offset value	v	address value	v	store value	v	synch value	v
	CEC:	condition result	v	prediction value		v					
	SI:	index value	work state								

Fig. 3. (a) The format of the long instruction with 5 operation fields; (b)The format of an entry of data slot.

3.2 The Architecture of PE

The PE of the LEAP array consists of an instruction register (IR), a multiple entries data slot (DS), an ALU, an Input/Output Unit, a local data cache (LDC), and a loop stepping unit (LSU), as shown in Figure 2.

IR is used to store the instructions, which corresponds to expression statements of data dependence graph. In addition to the 5 operation fields described in the above section, CEC, LD1, LD2, ST and COMPUTE, information for data dependence edges are presented in each operation field. A detail format is given in the Figure3-a.

Data slot, DS stores the context of instruction execution for a given value of index variable. A statement slice is issued whenever there is a free entry in the data slot. The entries of data slots with the same value of the index variable from all PEs form a frame, which like a virtual array hold a loop slice.

An entry of DS includes five segments corresponding to the instruction format, as shown in Figure 3-b. Global information segment, SI stores the value of index variable and working state of the entry. The prediction segment, CEC stores the values of prediction variables, the valid bit that indicates the availability of prediction condition, and the result of condition calculation. Other segments for LD1, LD2 and ST contain information about the address, value and available state.

The data-driven ALU checks states of operands in the data slot to select ready operands to perform computation. When the result is created, it is temporarily stored

in ST segment, then will be written back to memory whenever the synchronization condition is met.

Loop stepping unit (LSU) is responsible for updating the committed data slots and stepping the index variable to empty slots. Because every PE has an identical LSU for index variable, the control of the loop execution is distributed to facilitate the concurrency between loop slices. A flow-control mechanism has to be adopted to avoid the hazard incurred by the distributed control of loop stepping.

The local cache is used to buffer and prefetch data from memory, it differs from usual data cache in that the address range of each local cache is fixed and not overlapped with each other. This guarantees no data consistency problems exist between local caches. The consistency between the local cache and memory is preserved by the write-through protocol.

4 Execution and Performance Analysis of an Example Program

4.1 Execution and Optimization of the FIR Filter Program

An example program of FIR filter, as shown in Figure 4-a, is given to illustrate the execution process of the innermost loop. Figure 4-b and 4-c depict the host program and slave program. In the host program, the innermost loop is replaced by a subroutine, call-leap. In the slave program, the innermost loop is packaged into an independent subroutine, which will be converted into a data dependence graph shown in Figure 4-d, then transformed to a segment of instructions as shown in Figure 4-f.

When the subroutine, call-leap is called from host, a message, which contains information of start operation, the initial value, boundary value and step value of index variable, and instructions, is sent to target PEs. After each PE is started, the LSU in each PE fills empty data slots with statement slices. Three PEs are involved in the computation, as shown in the Figure 4-d. PE0 is used to produce address offset for the array access $S[i+j]$ in PE1. By adds the value created in PE1 and the value produced by itself in the last slot, PE2 produces $D[j]$ in a pipeline mode. For variable $D[j]$, there is a circle edge from its define to use with dependence distance 1. Except loading $D[j]$ from memory at the first time, the succeeding loads to $D[j]$ need not be sent out, as the value will be write back from results of the last iteration. This is implemented by operation $ld\&ldwait$.

Figure 4-g and 4-h give two samples of unroll transformation for optimizing the performance of FIR program on LEAP, and Figure 4-e shows the data-flow graph of 2 fold unrolling. Two or four loop slices can be performed per clock cycle with unroll fold by 2 or 4 to out loop index variable j . The approach of unrolling the outer loop preserves the benefit of completing more loop slices per cycle, while keep the length of inner loop. It is crucial for pipeline execution.

4.2 Performance Analysis to the Example Program

To verify the correctness of the LEAP architecture, we develop a functional simulator in SystemC language. The simulator contains 64 PEs. The arithmetic function units in each PE work in pipeline mode. The average delay of the arithmetic function is 2

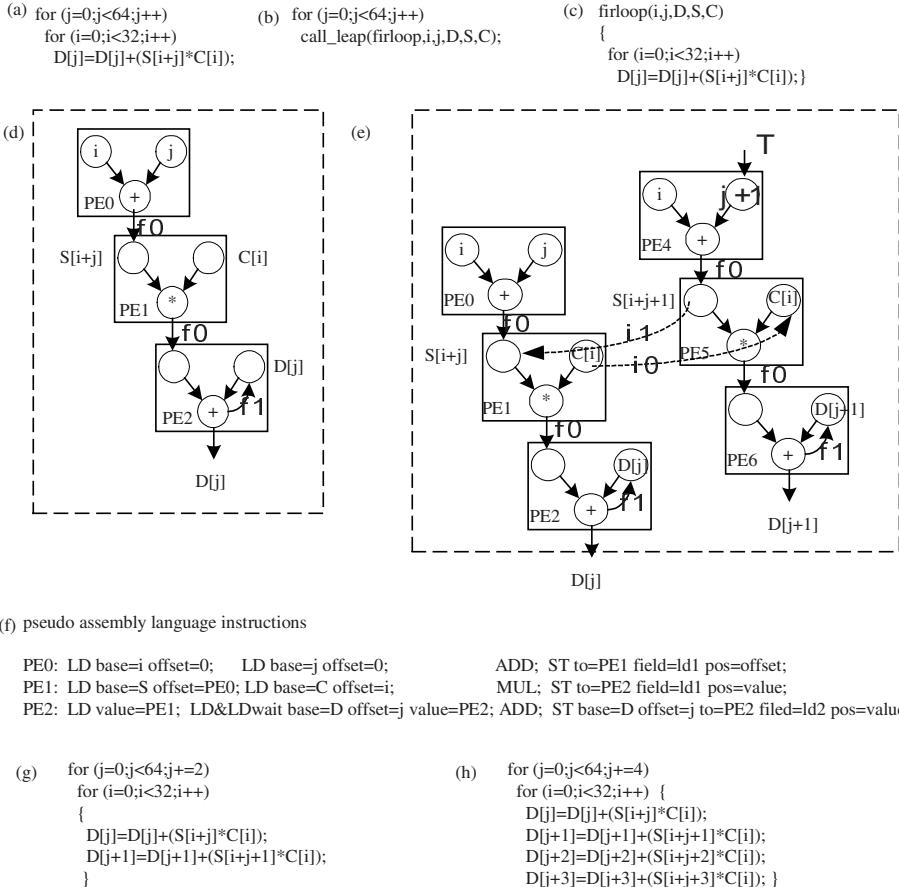


Fig. 4. The example program. (a) the source codes of FIR filter program; (b) the host program for execution on LEAP; (c) the slave program running on LEAP array; (d) the data-flow graph for the innermost loop; (e) the data-flow graph for the innermost loop by 2 fold unrolling on index j; (f) the pseudo instructions for the innermost loop; (g) the source codes for optimization by 2 fold unrolling on index j; (h) the source codes for optimization by 4 fold unrolling on index j.

cycles. There are 2 read channels and 1 store channel between each PE and its local cache. The local memory units are connected to external main memory by 8 buses. The topology of Data Exchange Net is 2 dimensions mesh. The router is a statically configuring cross-point unit with 1 cycle delay.

The reference processor architectures are R10K and Pentium III, which stand for RISC architecture and CISC architecture respectively. We suppose that no miss will occur in data cache and instruction cache and the efficiency of instruction issue is 100%. With these ideal conditions, maximum 4 instructions per clock cycle are issued in R10k, and Max. IPC(Instructions Per Cycle) in PIII is 3. (In fact, the efficiency of instruction issue is lower than 100%, due to the conflicts in function units, in registers and in memory ports).

Table 1. Configurations and performance parameters of three architectures.

Architecture	The number of PEs	loop unroll	Average delay for memory access	Execution time
LEAP	3	1	20	5056
			40	7616
	6	2	20	2528
			40	3808
	12	4	20	1264
			40	1904
Architecture	Processors/ function units	Parallel tasks	Instructions in loop body	Execution time
RISC	1 / 4	1	44	22528
	2 / 8	2	44	11264
	4 / 16	4	44	5632
CISC	1 / 3	1	18	12288
	2 / 6	2	18	6144
	4 / 12	4	18	3072

The methodology of the performance analysis is to compare execution times of FIR loops in clock cycles for three architectures with different parallelism. Assuming under the same semiconductor technology, three architectures will have the same clock frequency. In fact, we hope that the LEAP architecture is more competitive in increasing clock frequency with modularity and simple PE structure.

The pipeline of each PE in the LEAP consists of three parts, memory access, computing and data transferring in the network. To calculate the startup time of the PE pipeline, we have following formula, $T_s = T_m + T_c + T_n$, where T_m is the average delay for memory accesses, T_c is the average delay for function computation and T_n is the average network delay. To sum the total time of executing nested loops of FIR, $T = (T_s+L_i-1)*L_j$, where L_i , L_j are lengths of internal loop and external loop respectively.

To estimate the optimum performance of two reference architectures, we get the execution time from the instruction length of the loop body, L_i , L_j and the max instruction issue rate. To get the number of instruction length of the loop body, the FIR program is compiled and disassembled under R10k and PIII environments. Parallel execution of FIR loops on two reference architectures is under the model of SMP (Symmetric Multiple Processors). Parallel tasks are divided according to the external index variable, j .

Table 1 shows the comparison between the three architectures. Column two shows the usage of the hardware resources. For LEAP architecture, the number of PEs represents the hardware resource, for the other architectures, the resource is the number of processors and the number of arithmetic function units. The factor reflecting the parallelism is shown in column three. The analysis also includes the impacts of different memory delays.

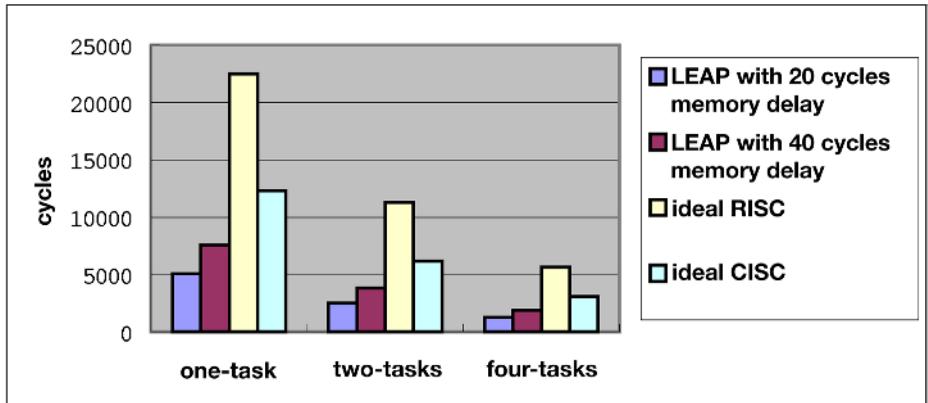


Fig. 5. The performance comparison between 3 architectures with different parallel tasks

As shown in figure 5, under all conditions, LEAP shows its performance potentials compared with other two architectures. When there is only one task, LEAP uses 3 PEs, and RISC and CISC use one processor with 4 to 3 arithmetic units respectively. But, the performance of LEAP is as much 4 to 3 times as that of RISC, and 2.5 to 1.7 times as that of CISC. With the increase of parallel tasks, RISC and CISC architectures need multiple units of logic for instruction decoding, instruction buffering, and instruction scheduling, in addition to the increase of function units. But the LEAP architecture still keeps superior to RISC and CISC in execution time.

During the above comparison, the referenced architectures are under the ideal environments without conflicts of memory, schedule and data use, but the factor of the memory delay is considered to estimate the execution time of FIR on LEAP architecture. Even from the view of the resource usage, LEAP is competitive.

5 Conclusion

This paper presents a framework of a loop engine on array processors. The LEAP array supports directly mapping expression statements to PE, based on transformation from control dependence graph to data dependence graph. Data are streamed into the array. The loop slices are processed in pipeline. The simulation shows the performance of LEAP is nearly one slice per clock cycle. The performance potential is obvious, compared with RISC and CISC.

The framework of LEAP array can be further treated as a general-purposed coarse grain reconfigurable computing architecture, which is specifically designed for innermost loops. It preserves the benefits of speeding up loops while the configurations for the execution are dynamically loaded at runtime. The time for reconfiguration is very short, because only instructions and network configurations need to be loaded, and this information can be buffered as instructions.

References

1. V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, “Clock rate versus IPC: The end of the road for conventional microarchitectures.” In Proceedings of the 27th Annual International Symposium on Computer Architecture, pages 248–259, June 2000.
2. Tullsen D, Eggers S, Emer J, Levy H, Lo J, Stamm R, “Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor” In Proceedings of the 23nd Annual International Symposium on Computer Architecture, pages191–202, 1996.
3. J. R. Allen, K. Kennedy, C. Porterfield, and J. Warren, “Conversion of control dependence to data dependence” In Proceedings of the 10th ACM Symposium on Principles of Programming Languages, pages. 177–189, January 1983.
4. Michael Wolfe, High Performance Compilers for Parallel Computing Addison-Wesley Publishing Company ISBN 0-8053-2730-4 1996

A New Architecture of a Fast Floating-Point Multiplier

Xu Zhou and Zhimin Tang

Institute of Computing Technology,
Chinese Academy of Sciences
Beijing 100080

Abstract. In this paper, a Floating-Point multiplier, supporting IEEE 754 Standard and full pipelined, is presented. A new architecture of dual-Way Multiplier is proposed. It can reduce 13.6% delay of critical path of adder tree according to full size multiplier without increasing area. The overall multiplier has a latency of 3 cycles and a throughput of 1 cycle for a single-precision or double-precision floating-point instruction. This multiplier has been verified on a FPGA and implemented in 0.18 Micron Standard Cell technology, with its frequency 384MHz and area 732902.25um². This architecture is compared with other architectures under the same technique, and the result shows it is effective and efficient.

Keywords: Floating-point Multiplier, Processor

1 Introduction

Floating-point Multiplier Units are commonly found in digital signal processors, and more recently in general-purpose processors. Floating-point multiplication mainly involves single-precision and double-precision operations, which add several partial products together to get the final product. In addition, rounding of the product, adjusting the exponent and generating correct exception flags should be included in floating-point multiplication, if it is compliant to IEEE Standard 754. Floating-point Multiplier Units embedded in modern processors are usually required to be pipelined, small and fast.

Essentially in floating-point multiplication, a large number of additions are used to compress many partial products into the final product. Current floating-point units use Array or Tree circuit architectures to implement the floating-point multiplier. In these designs, the tradeoffs between complexity and circuit style, or speed and area are considered. Some designs use simple arrays with regular connectivity[5][6], however some compressors and serial structures of addition of partial products are brought in. So it leads to the increment of area and delay. Compared with array, tree architectures use parallel mechanism to decrease partial products[7][8][9]. Although it brings extra delay, and difficulty to layout, it is still regarded as more effective and fast[11]. In this paper a fast architecture is proposed, that is, the architecture of dual-way floating-point multiplier, then verified by FPGA, and implemented by standard cell technology. This architecture is compared with other architectures under the same technique, and the result shows it is effective and efficient.

In this paper, Section II simply introduces the process of floating-point multiplication. In section III, a new architecture of floating-point multiplier is proposed. Section IV describes the simulation and verification of this floating-point multiplier, and comparing it with other architectures under the same technique. Section V draws a conclusion.

2 Related Work

There are two people contributing great to fast multiplication. In 1951, Booth discovered that signed numbers could be multiplied using a recoding technique that resulted in $N/2$ additions or subtractions to multiply two N -bit two's-complement numbers[1]. Booth's method was extended in 1961 by MacSorley, and in 1964, at the University of Illinois. Wallace showed that multiplication could be performed in $\log_{(23)}N$ plus an addition of two N -bit numbers[10], using a construct called a "Wallace tree".

Compared with fix point multiplier, IEEE Standard 754-compliant floating-point multiplier needs computing exponent, and it is different to calculate the mantissa. The mantissa part of single-precision of IEEE floating-point multiplication takes two 24-bit unsigned binary numbers to produce a 24-bit product after rounding and normalizing. However, this part of double precision is more complex for two 53-bit unsigned binary numbers are multiplied to gain a 53-bit product. Using the simplest method of shift-and-add, 53 partial products will be generated. But the modified Booth encoding method can reduce the number of partial products to 27 with a little logic, each 54-bit wide (53+sign extension bit due to possible negation after Booth encoding[4]). Then these partial products are added together to get the final result. The process of addition can be divided into two styles: Array and Tree. There is an article[11] which studies these two architectures, drawing a conclusion that tree architecture is better than array one in aspects of delay, area and layout.

Tree architecture has many types, including Wallace tree, Dadda tree, 4-2 counter tree, balance delay tree and overturned stairs tree etc. IEEE-compliant floating-point multiplier can be divided into two types according to the size of tree architecture, full-size and half-size. The full-size multiplier uses 53×53 adder-tree, dealing with 27 54-bit (the result of Modified Booth encoding) partial products, such as UltraSPARC and S390[3]. However the half-size uses 53×27 adder-tree, dealing with 14 54-bit partial products, such as IBM PowerPC 603e[2]. Single-precision multiplication can get through the adder-tree directly to generate the result. But for double-precision floating-point multiplication, some more works need to be done. For an example, 603e requires double pumping through the multiplier array. Another way is to occupy the adder-tree in next cycle. These two ways both have strongpoint and weakness. Double pumping is hard to implement. It needs support from back-end. And the throughput of floating-point multiplication instruction is reduced by occupying the adder-tree two cycles.

3 New Architecture of Floating-Point Multiplier

The conclusion can be educed from the analysis of the second part; the half-size multiplier tree has some strongpoint comparing with the full-size multiplier tree, such as small area and fast speed. But the full-size multiplier tree can execute instructions with high efficiency. A better architecture will be generated by combining the advantages of these two kinds. A new architecture of floating-point multiplication is proposed based on this. It has the character of the high efficiency of the full-size multiplier tree, as well as the character of fast speed of the half-size multiplier tree. And it was implemented in an actual processor. The result shows this architecture can really increase the efficiency of floating-point multiplication.

Indeed, some techniques can increase the frequency of floating-point multiplier. For example, it can increase speed by inserting pipeline stage into multiplier and dividing the critical path, adder tree, into several pipeline stages. But the disadvantage is obvious. Area is increased because adding pipeline stages brings a great number of registers. And the increasing of pipeline stage will also reduce the efficiency of instruction. How to shorten the critical path of multiplier tree and how to find a better tradeoff between area and delay are the key questions to improve the performance of the floating-point multiplier. To achieve those goals, our main idea is to partition the multiplier tree. In brief, partial products are reduced firstly by modified Booth encoding, then divided into two parts, and calculated in separate multiplier tree. The results will be merged in next pipeline stage. In the end the final result is gained. In another words, by the technique of partition, the multiplier is divided into two parts according to high and low bits, and then calculated in parallel. Thus not only delay can be reduced, but also the influence of interconnect wiring can be weakened.

Of cause, multiplier tree can be partitioned into more pieces to reduce delay. For example, the whole tree can be divided into four sub-trees. However it leads to increasing the complexity of algorithm of merging, which unfortunately results in slowing down the next pipeline stage. It is the most reasonable choice to divide multiplier tree into two sub-trees after analysis and experiment. So this architecture is named the dual-way floating-point multiplier.

3.1 Architecture

Figure 1 shows the overview of dual-way architecture. It is composed of two 53*27-size sub-trees. The mantissa of the multiplicator is divided into high and low two parts. When single-precision floating-point multiplication instruction is coming, the mantissa is aligned left, and multiplied by the multiplicand. The data flows through #1 multiplier tree, and the input of #0 multiplier is kept steady. Thus, it can reduce activity of gates, leading to reducing the dynamic power consumption.

When double-precision floating-point multiplication instruction is coming, two sub-trees are both utilized. The high part and the low part flow through #0 and #1 multiplier tree separately. Compressor1 and Compressor2 calculate the outputs of two multiplier trees to generate a couple of carry and sum. Compressor is mainly composed of 4:2 compressors.

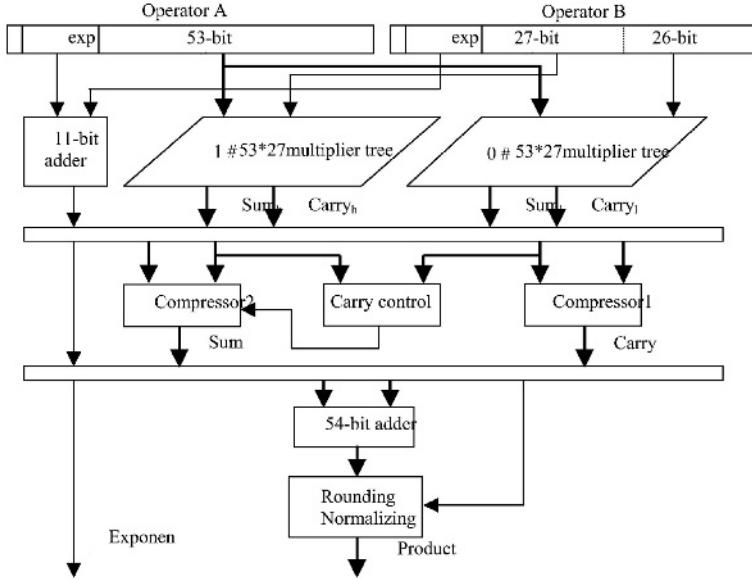


Fig. 1. Architecture of the dual-way floating-point multiplier

3.2 Algorithm

The basic algorithm of the dual-way floating-point multiplier is described as following.

If $X = [x_{n-1}, x_{n-2}, \dots, x_1, x_0]$, $Y = [y_{m-1}, y_{m-2}, \dots, y_1, y_0]$,

where $n = m = \begin{cases} 24, & \text{single} \\ 53, & \text{double} \end{cases}$, so:

$$X \bullet Y = [x_{n-1}, x_{n-2}, \dots, x_1, x_0] \bullet [y_{m-1}, y_{m-2}, \dots, y_1, y_0] \quad (1)$$

$$= [x_{n-1}, x_{n-2}, \dots, x_1, x_0] \bullet [y_{m-1}, y_{m-2}, \dots, y_j] \bullet 2^j + [x_{n-1}, x_{n-2}, \dots, x_1, x_0] \bullet [y_{j-1}, y_{j-2}, \dots, y_1, y_0]$$

where $j = \begin{cases} 24, & \text{single} \\ 26, & \text{double} \end{cases}$.

The result of multiplier trees is two carry-saved binary numbers. Two compressors are needed to merge this two Sum and Carry in order to generate a couple of true Sum and Carry.

$$\begin{aligned} X \bullet Y &= (Sum_h + Carry_h) \bullet 2^j + (Sum_l + Carry_l) \\ &= (Sum_h \bullet 2^j + Sum_l) + (Carry_h \bullet 2^j + Carry_l) \\ &= Sum + Carry \end{aligned} \quad (2)$$

Sum_h , $Carry_h$, Sum_l , $Carry_l$ are the results of #1 multiplier tree and #0 multiplier tree. Sum and $Carry$ are the expectant results of the whole multiplier tree.

Thus, the input of a single multiplier tree is reduced from 27 to 14. In another words, only 14 partial products need to be added. If multiplier tree is built up with 3-2 counter, the full-size tree has 7-level-counter delay. But the dual-way multiplier only needs 6-level-counter. One level counter is saved by the new architecture. And the dynamic power consumption is saved too for single-precision multiplication.

3.3 Process

The basic process of the dual-way floating-point multiplier is shown as following.

Step 1: Preprocess. Including judging the operator for 0, $\pm\infty$ or denormalized etc, calculating the exponent and generating exceptions.

Step 2: The first stage of multiplication. On the assumption that A and B are both normalized mantissa and kept in register, the low 26-bit of A and B multiplies to generate Sum_l and $Carry_l$. At the same time, the high 27-bit of A and B multiplies to generate Sum_h and $Carry_h$.

A and B both have normalized 24-bit mantissa for single-precision. Move A rightward 29-bit, so the least significant bit of single-precision is aligned to the least significant bit of double-precision. B is left aligned, and the least three bits is set to zero. So it has 27-bit and can be consistent with double-precision. Because the rounding positions of two precisions are the same, it can use the same logic to round.

Step 3: The second stage. Sum_l , Sum_h , $Carry_l$ and $Carry_h$ are compressed to generate Sum and $Carry$ by the Compressors.

Step 4: The third stage. The low 26-bit of Sum and $Carry$, and the low 26-bit of Sum_l and $Carry_l$ will be used to calculate the round-bit, carry-bit and sticky-bit.

In this step, the high 54-bit of Sum and $Carry$ are added by a compounded adder to generate Sum , $Sum+1$ and $Sum+2$. In another words, Sum , $Sum+2^{-52}$ and $Sum+2^{-51}$. The rounded product is controlled by the carry-bit, the most significant bit and relevant judge-bit.

The only difference of rounding between single-precision and double-precision is the MSB (the most significant bit).

4 Implementation

4.1 FPGA Verification

First, this floating-point multiplier is verified by FPGA. It is implemented on a Altera chip APEX20KE: EP20K1500EBC652-1X using FPGA Express V2000.11 of

Synopsys and Quartus V2.0 of Altera. The total gate number of the multiplier is 8780 cells with 1370 cells of register in it. Under the frequency of 12.5MHz, the floating-point performance of the FPGA has exceeded the performance of the real chip of Intel486DX50.

The result of test bench programs under following conditions as table 1 is given. The dual-way floating-point multiplier is implemented in a general-purpose processor by FPGA, while the else two are real chips. Note that a design implemented by ASIC is usually faster than it implemented by FPGA.

A special floating-point test program is tested, Paranoia. This program can test whether the floating-point unit is compliant to IEEE standard 754 completely. The result of our unit is “Perfect” that is given by the Paranoia program. When the same program is used to test floating-point unit of Intel processor, the result is that single-precision is “Perfect” but double-precision has a “Defect” and a “Flaw”. (In Paranoia, there are four ranks, which are Flaw, Defect, Serious defect and Failure.)

When it is implemented in 0.18 Micron Standard Cell technology, the floating-point multiplier has 3 pipeline stage, each 2.60ns, with its frequency 384MHz and area 732902.25um².

4.2 Comparing of Several Multiplier Architectures

The dual-way floating-point multiplier is compared with full-size and half-size floating-point multiplier under the same technology and implementation condition. To compare appropriately, the multiplier tree is synthesized separately by Design Compiler Version 2001.8 of Synopsys. And technology library is 0.18 Micron of Artisan Components with vision of 2001.1.1700. The result is shown as table 3. This table is adapted for single-precision multiplication. For double-precision multiplication, the delay for half size architecture needs to be doubled while the else two remain unchanged.

Table 1. System Hardware Parameters

System Parameter	A General Purpose Processor*	486**	IDT64474***
Frequency of Main Board (MHz)	50MHz	25MHz	50MHz
Frequency of CPU (MHz)	12.5MHz	50MHz	100MHz
One level CACHE (KB)	ICACHE and DCACHE both 4K, direct-mapped	CACHE 8K, Unified, 4 set-associative	ICACHE and DCACHE both 8K, 2 set-associative

* A General Purpose Processor is a kind of processor based on MIPS instruction set implemented in FPGA.

** 486 is a real chip produced by Intel Corp.

*** IDT64474 is a processor of MIPS architecture.

Table 2. Running time of test bench

Test Bench	Running Time (Sec)			Relative Time		
	A general purposed processor	486	IDT644 74	A general purposed processor	486	IDT64 474
Floating-point matrix multiplication	2.70	2.68	0.37	1.00	0.99	0.14
FFT	52.89	56.02	7.97	1.00	1.06	0.15
SOR	5.59	5.60	0.91	1.00	1.00	0.16
Computing π	58.03	164.71	11.57	1.00	2.84	0.20
Whetstone	9.11	7.15	1.07	1.00	0.78	0.12
				1.00	1.33	0.15

Table 3. Area and Delay of Several Architectures Under The Same Condition

	Full-size	Half-size	Dual-way
Delay (ns)	2.21	1.96	1.91
Relative Time	115.7%	102.6%	100%
Area (μm^2)	565136.000	265954.50	530671.125
Relative Area	106.7%	50.1%	100%

Following conclusions can be drawn from above results.

1. Delay of dual-way multiplier is shorter than that of full-size multiplier, even shorter than that of half-size multiplier a little.
2. Area of dual-way multiplier is smaller than that of full-size multiplier, and is close to that of double half-size multiplier.

5 Conclusion

A new architecture of parallel floating-point multiplier is presented, dividing the multiplier tree into sub-trees, and computing in parallel, then merging finally. This architecture can reduce delay of multiplier tree without increasing area, and dynamic power consumption can be held because the input pattern of single-precision and double-precision can be controlled separately. This floating-point multiplier reduces 13.6% delay of addition of partial products according to full-size multiplier. It is suitable for the design of high-speed processor.

References

- [1] A. D. Booth, “A Signed Multiplication Technique”, Quart. J. Mech. & Appl. Math. 4, 1951, pp. 236–240
- [2] R. M. Jessani, C. H. Olson, “The floating-point unit of the PowerPC 603e microprocessor”, IBM Journal of Research and Development, Volume 40, Number 5, Sep 1996, pp. 559–566
- [3] E. M. Schwarz, C. A. Krygowski, “The S/390 G5 floating-point unit”, IBM Journal of Research and Development, Volume 43, Number 5/6, Sep/Nov 1999, pp. 707–721
- [4] Hema Dhanesha, Katayoun Falakshahi, Mark Horowitz, “Array-of-arrays Architecture for Parallel Floating-point Multiplication”, 1995 Conference on Advanced Research in VLSI, pp. 150–157
- [5] Craig Heikes, “A 4.5mm² Multiplier Array for a 200MFLOP Pipelined Coprocessor”, 1994 IEEE ISSCC digest of technical papers, pp. 292–293
- [6] Junji Mori et al., “A 10-ns 54*54-b Parallel Structured Full Array Multiplier with 0.5- μ m CMOS Technology”, IEEE J. Solid-State Circuits, vol. 26, No. 4, April 1991, pp. 600–606
- [7] Gensuke Goto et al. “A 54*54-b Regularly Structured Tree Multiplier”, IEEE J. Solid-State Circuits, vol. 27, No. 9, Sept 1989, pp. 1229–1236
- [8] Scott Hilker, Nghia Phan and Dan Rainey, “A 3.4ns 0.8um BiCMOS 53*53b Multiplier Tree”, 1994 IEEE ISSCC digest of technical papers, pp. 292–293
- [9] Leslie Kohn and Sai-Wai Fu, “A 1,000,000 Transistor Microprocessor”, 1989 IEEE ISSCC digest of technical papers, pp. 54–55
- [10] C.S. Wallace, “A Suggestion for a Fast Multiplier,” 1964 IEEE Trans. Electron. Computers, vol EC-13, pp. 14–17
- [11] Pascal C. H. Meier, Rob A. Rutenbar, L. Richard Carley, “Exploring Multiplier Architecture and Layout for Low Power”, 1996 Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 513–516

A Highly Efficient FC-SAN Based on Load Stream

Jiwu Shu, Jun Yao, Changdong Fu, and Weimin Zheng

Dept. of Computer Science and Technology,
Tsinghua University,
Beijing 100084, P.R.C.

{shujw, zwm-dcs}@mail.tsinghua.edu.cn
{yaojun01, fcd01}@mails.tsinghua.edu.cn

Abstract. The speed of storing and fetching data on SCSI disks has a great restriction on the efficiency of SAN based on Fiber Channel Network. In this paper, a high-efficient FC-SAN storage method attributed to the statistics conclusions of the file system workload is designed and implemented. By evaluating the load heavy or light, the system selects DISK or RAM devices as the main storage media according to the different I/O operations, i.e., file operations of larger size will be done on disk devices while reading/writing smaller size or file attribution operations will be done in RAMDISK devices. This method reduces the operations that store and fetch data on physical disks, adjusts the speed difference between CPU and DISK, and increases the efficiency of the system as a whole. Experimental testing proves good results and demonstrates that the application of RAM devices for I/O operation could increase the read efficiency of FC-SAN at about 50% to 100% and write performance at about 10% to 40%.

Keywords: storage network system, RAMDISK, workload, I/O route, FC-SAN

1 Introduction

There are two ways adopted by storage devices in Storage Area Networks (SAN), one through Fiber Channel array controller and the other through SAN storage subsystem. The back end of the former solution is connected to SCSI disk or fiber channel disk via SCSI interface or FC-AL interface respectively, with high integrity, reliability but is lack of flexibility and hard to expand. The latter one is composed of multiple interconnected components and I/O operational nodes, which, in turn, is generally composed of commercialized SMP main board, fiber channel interface adapter, SCSI-RAID adapter or FC-RAID adapter, PCI bus and embedded operating system[1]. The latter is thus highly extensible and flexible, as well as convenient to implement, higher efficiency-price ratio and other features.

We use FCP (Fibre Channel Protocol) to serve as the SCSI protocol of FC-SAN storage system, for FCP is such a protocol that adopts many advanced techniques to introduce the least transport cost into the system, compared with other communication protocol[2]. FC-SAN storage system is commonly connected to SCSI disk array via SCSI bus, or directly to SCSI disk via SCSI-RAID adapter. Storage controller of FC-SAN sends I/O requests directly to physical disks. What's more, although,

theoretically, SCSI bus is thus far estimated to accommodate a transportation speed of 160 MB/S, the maximum reachable transportation speed of a single disk is approximately 30MB/S, which, even with the application of RAID Technology, can hardly match the speed of Fiber Channel. So the bandwidth of disks becomes the bottleneck of the FC-SAN storage system. Although CACHE Technology on the disk and on the server file system reduces the number of I/O requests sent to the physical disks[3], yet this advantage is counteracted because of the direct way of each sending applied by the FC-SAN storage system at present. The data storing and fetching method and speed of disk decide the performance of the I/O sub-system, and thus is the main factor of the efficiency of SAN storage system as a whole. Since visiting to physical disks is tightly correlated with file operation on application system, we could focus the file I/O operation analysis to decrease the workload of physical disks.

The results of file system work load research demonstrate the difference among various application systems with different work load, yet there lies great consistency in big-scale statistics[4-7]. The following is most obvious:

Generally, only 50% of the whole space is taken by the file system consisting of multiple small files, whose way of store and fetch is hardly sequential. On the other hand, large files, though smaller in number, are mostly sequential and consume most of the space taken by the file system. Files are time-bounded and area-bounded. Most of file operations are file property operations. Reading operation far surpasses writing operation in number of being called.

In reference [8], a file system is modified and improved on the conclusions above which develops a miscellaneous DISK file system at the client and applies a perduring RAM as I/O sub-system device, to which most of the file sending operations are passed. However, the system is highly relevant with the file system and operating system of the hosts on which it is running. Use of large-capability and perduring RAM reduces the efficiency-price ratio. In this paper, an improved FC-SAN storage system based on the conclusions of file system workload research is designed and implemented. It adopts both RAMDISK and DISK as the I/O devices of network storage. Two different routes are supplied, file operations of larger data will be done on disk devices, and reading/writing smaller data or file property operations will be done in RAMDISK devices. This method reduces the operations that store and fetch data on physical disks, decreases the speed difference between CPU and DISK, and increases the efficiency of the system as a whole. Experimental testing has proven good results. In the later sections, we first introduce an FC-SAN system with only physical disks as storage media, and then we give the design and implementation based on this FC-SAN system, using both physical disks and RAM disk as the storage media. In the last section, we will give the test results and some analysis.

2 A Basic FC-SAN System

Hardware of SAN storage system includes front-end server system (hosts, initiators), network communication device, SAN storage subsystem (IO nodes, targets) and so on. The relationship of SAN storage system and server system is the relationship between target and initiator. Server system sends I/O requests to SAN storage system and SAN storage device gives response after processing those requests. Server system and SAN storage system communicate via SCSI transport protocol, such as FCP,

which is most commonly used as SCSI transport protocol, and iSCSI or Infiniband[9] protocols can also be selected.

The main obstacle to realize the SAN storage system is the selection and evaluation of the I/O route scheme. The SAN storage system we designed provides SCSI-RAID function, through which many different SCSI hard disks can be integrated into a single one RAID disk, such as RAID 5 or RAID 3, etc. I/O route reflects certain RAID block device with its logical address to corresponding server end. The reflection is transparent to the server, which makes the RAID block logically the same as local disks of the server. FDISK, FORMAT, COPY, READ/WRITE operations and so on can be applied on the system[10]. The I/O route of the highly efficient FC-SAN storage system that already realized is shown in Fig.1.

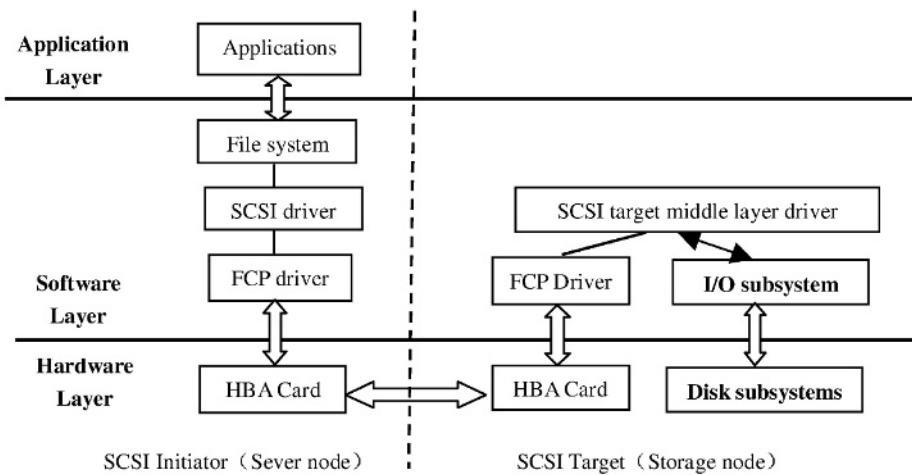


Fig. 1. The I/O route of a FC-SAN storage system

The FC driver and the HBA firmware at the SCSI target are working in target mode, while those of the SCSI initiators (server node) are working in initiator mode. The relationship between initiator and target is similar to that between client and server. Through SAN storage management software, users can configure LUN on the SAN storage system to disks on the server. When LUN on the SAN storage system is reflected as SCSI disk, application programs on the server can then perform reading or writing operations. The general I/O operation is as follows:

- i) The application program at the server sends reading, writing or other kinds of file operation requests, which is translated by the SCSI driver into SCSI commands before passed on to FC initiator driver.
- ii) The FC driver at the initiator locates the corresponding fiber channel device and LUN according to the SCSI four- address identification of the SCSI command (host, bus, target ID, LUN). The SCSI command is translated to IOCB and then passed to FC HBA firmware.
- iii) IOCB is translated to FCP IU by FCP HBA firmware and transported through Fibre Channel network.
- iv) FC HBA firmware in target mode of the SAN storage system receives the FCP IU, translates it into IOCB and passes it on to FC driver, which is in target mode, for procession.

v) FC driver in target mode translates the IOCB, which has been processed, into SCSI command of target mode. The command is then transmitted to SCSI target middle-layer driver via command/message queue.

vi) SCS target middle-layer driver gets SCSI CDB from SCSI command, processes the certain command or data on the I/O sub-system of the SAN storage system and returns the response or result to the initiator.

3 FC-SAN System Design Based on DISK and RAMD

We observed that the FC-SAN storage device above adopted the way of directly sending I/O requests to physical disk. When testing the system with massive data dash and inspecting the throughput of FC switch, we observe that the peak efficiency of 2Gbps semi-duplex FC is between 50% and 60%, which shows that FC-link in target node and the FC-HBA will not become the bottleneck of the system. So, at present, the way of making the store and fetch data on the disk and its corresponding speed are the main factor of the sub-system efficiency as well as efficiency of the whole system. However, the embedded OS of the FC-SAN storage device stated above is based on LINUX subdivision and supports RAMDISK and INITRD mechanism, thereby can be applied as the ultimate storage sub-system based on DISK of RAM. Generally the size of RAMDISK in I/O controller has a capacity between 512MB and 4GB, which makes the way of operating in memory most suitable for small file reading and writing, and file attribution operations. On the other hand, in the suggested FC-SAN system, though two different kinds of storage devices, namely DISK and RAMDISK, are both existent, yet they are managed, stored and fetched by the server file system at the initiator transparently. Therefore, the file system based on RAMDISK miscellaneous device model is still a file system. The I/O sub-system route of the FC-SAN storage system based on RAM/DISK is shown in Fig.2.

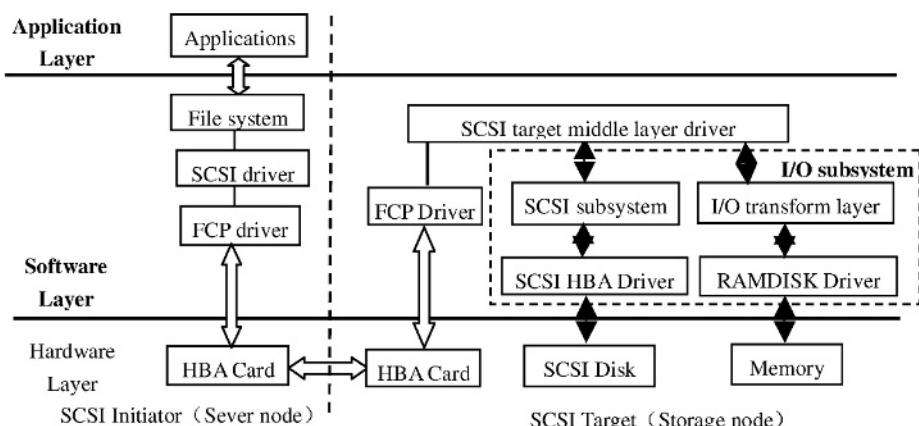


Fig. 2. The I/O route FC-SAN storage system based on RAM and DISK

In Fig.2, the first few steps of the whole I/O process are shown in I~V. Afterwards, SCSI target middle-layer driver distinguishes the certain type of the command which has just been received. If the command is reading or writing operations applied to big file, the original I/O route is taken to read or write SCSI disk. Correspondingly, if the command is any type of operations applied to small files, the new I/O route is taken to read or write memory-based DISK. Because the I/O operations sent by client end to the target middle-layer driver are SCSI commands, which are not able to perform general operations on RAMDISK but need translation from SCSI block commands to the normal I/O requests, a module responsible for SCSI command and general I/O block transformation request is added to the new I/O sub-system route. Because the calling of subroutines in kernel is so rapid that the extra expenditure such as the cost of disk tracing is omitted for RAMDISK, thus greatly increasing the bandwidth of the data channel, therefore the system efficiency is still promoted remarkably though command transformation requests multiply. This has been evidently proved by experiments.

4 Implementation

FC-SAN storage system with RAM/DISK miscellaneous I/O sub-system, via its SCSI target middle-layer, distinguishes and correspondingly processes the SCSI command sent by the client before passing it to the proper device. If the command is sent to memory-based RAMDISK, certain translation is needed. In respect of system reliability, under the conditions of system restart or sudden power failure, data in the memory-based RAMDISK may be lost. Therefore, a method to guarantee the consistency of the file system based on RAMDISK with general hard disk is inevitable.

4.1 Handling SCSI Requests in Target Simulator Module

The architecture and I/O route of SCSI target middle-layer driver is shown in Fig.3. A processing thread in kernel is responsible for reading out SCSI commands and messages from SCSI command queue and message queue respectively. If SCSI command instructs to read or write large data, the processing thread of SCSI target simulator then passes the command to the SCSI sub-system of the immediate lower layer, which fully processes the command, performs SCSI disk operations and waits for the sub-system to return results. As to other commands, the procession thread of SCSI target simulator transfers the command to I/O transferring layer, where the command is translated to block I/O requests and passed to RAMDISK driver for memory procession.

The devices in SCSI target simulator are mapped to SCSI disks (Target: Lun) in initiator one by one. Here one single RAMDISK device is adopted as a whole instead of allocating each SCSI disk device to a certain RAMDISK device. Therefore, RAMDISK is divided into many different sections, and there can be different ways to map RAMDISK section to SCSI disk device, either one of the former to all of the latter or multiple of the former to all of the latter, namely 1:N, M:N, N:N. Here N:N reflection is applied, in which each RAMDISK section is mapped to one small file on

the SCSI disk or file property operation. The asynchronous way applied here to process the SCSI command by the thread greatly promotes efficiency. The concurrent modification of RAMDISK property in the device-mapping table of the SCSI target simulator makes reallocation of the RAMDISK device and section convenient.

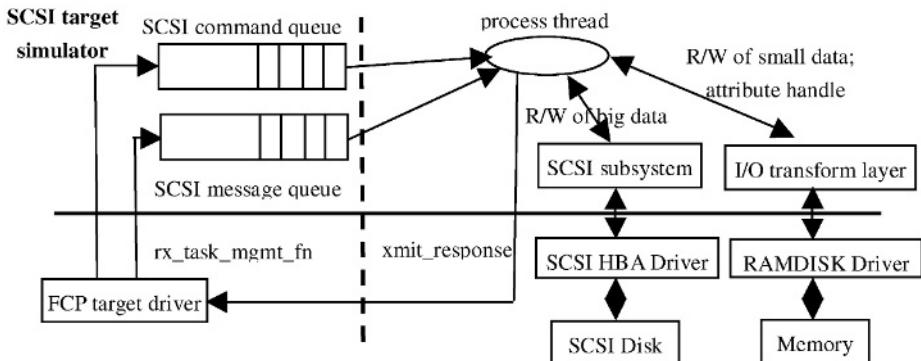


Fig.3. The architecture of SCSI target simulator

On the other hand, since the speed of operations on RAMDISK is very high, synchronous mode can be used to store and fetch data. Thus the sequential operations of the RAMDISK at the client are ensured, as well as the consistency of the RAMDISK with the file system is guaranteed.

In the implementation, the critical sizes for each data block to read and write operations can be flexibly and separately decided corresponding to the statistics attributions of the file system workload in application. Generally the size of RAMDISK is constant and is divided into multiple sections, each corresponding to a small file of SCSI disk or file attribution operation. If the threshold value is set too large, then the RAMDISK area in each SCSI disk will be consumed too quickly. Correspondingly, if it is set too small, most reading and writing requests will then be sent to SCSI disk, affecting the system efficiency as a whole. For an instance, for database and web server application, the size of blocks is generally 8KB, take up to 50% of the whole room, the threshold value can be set to 16KB and RAMDISK can be divided into 8 parts, each with a capacity of 64MB, thus the system gains a quite good efficiency. On the other hand, a larger threshold value can be set when the much more large files are requested, for example, in VOD service system. In future system, not only can RAMDISK be of large capacity, but dynamically auto-adjust threshold method can be applied as well.

4.2 Handling Abnormal Failure and Data Recovery

Since RAM, a volatile media, is one of the main storage devices in such an FC-SAN system based on physical disks and RAMDISK, special process should perform on the RAM-based DISK under circumstances of abnormality, normal logoff and initialization. Therefore, in the real system, a DOM with a capacity of 512M is selected as the perduration storage media for the RAM-based DISK, thus effectively files copy in response to continual I/O requests.

At initialization of the system, program *dd*(a block copy program under the UNIX-like system) is applied to completely contents copy of DOM directly to the RAM-based DISK. When the system is running, three methods can be adopted for the files copy, file compression method, directly complete copy method and LOG file system. In the real system, the second method above is applied, with a speed of one copy per $60*6$ seconds, namely a capacity of 512M from RAM to DOM per $60*6$ seconds. A copying thread running in background manipulates all of the tasks of data copying from RAM to DOM.

In addition, DOM is an IDE DISK based on FLASH, so the copying task of 512M from RAM by using the background copying thread occupies very little CPU resources and therefore has no impact on the system efficiency. In actual application, there is tradeoff between copy window and process ability because data inconsistency may occur when interval between two consecutive tasks of copying is large.

5 Evaluations and Analysis

5.1 Testing Environment

The testing system is made up in laboratory environment. 7 Xeon nodes are used as server nodes which are the clients of the SAN visiting the storage resources. Each node has 4CPUs, SMP mode, 1G DDR RAM and 36G Disk. The operating system is RedHat Linux 7.3. FC network with 2Gbps bandwidth is adopted as resource data transfer network. There is one storage node in the system, which has 2 Xeon 2.4G CPUs, 1G memory and 7*73G disk array. Single SCSI bus connects the disks and the storage node. Special embedded operating system is used in the storage node.

The testing hardware environment is shown in Fig 4.

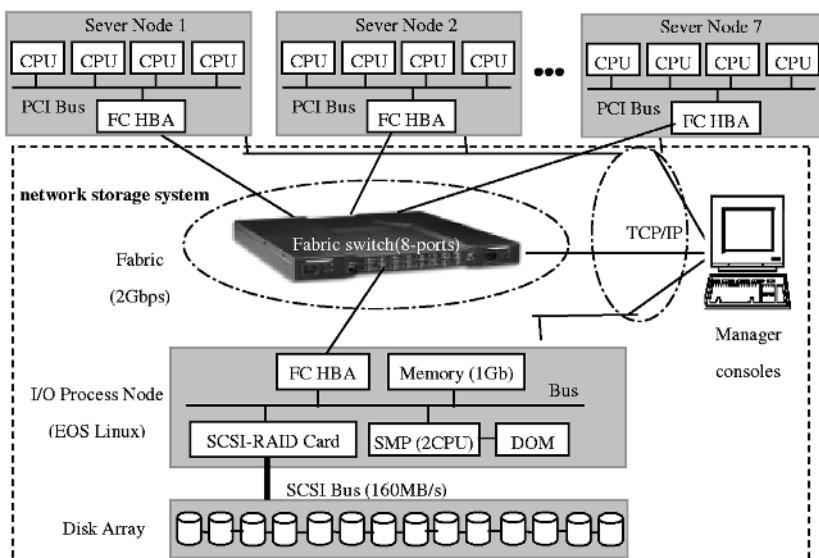
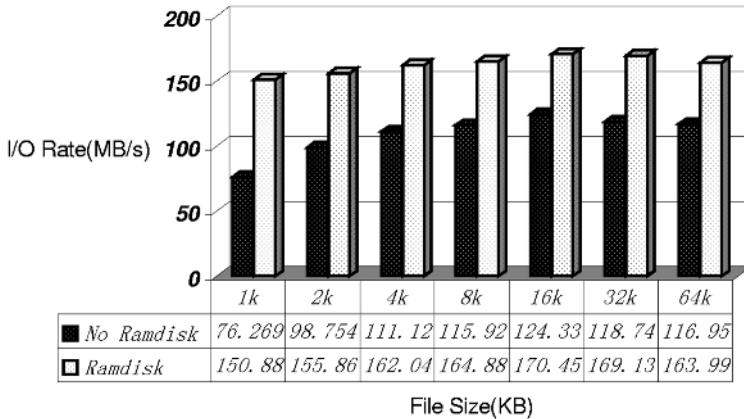


Fig. 4. Architecture of storage area network system

**Fig. 5.** Result of read performance test

5.2 Analysis of the Testing Results

The 7 physical disks in disk array are mapped to front host nodes respectively through storage network, that is to say, every host has one network disk to operate. This mapping procedure is transparent to the host file system, and the network disk operates the same with local disk logically. IOMETER is used as testing software. There are two testing group, one tests the system without RAMDISK technique; the other tests the system with the RAMDISK technique.

5.2.1 Testing of Reading Small Files

The threshold size of files is set to 64KB. We suppose that RAMDISK has been filled with the files to read before the testing starts. All the file requests to the data smaller than 64KB are done by RAMDISK. The results are shown in Fig.5.

We can see from the results that the smaller the files are, the worse the system performance is without RAMDISK. The performance is increased much after adopting RAMDISK technique. It is because that, large quantity of small file operations consumes quite much time on seeking. Especially when the files are very small and distributed discretely, there is bigger expense in disk read performance.

5.2.2 Testing of Writing Small Files

The size of files is still set to 64KB, and sequential mode of writing is used. The results are shown in Fig.6.

The results show that, writing performance is obviously increased after using RAMDISK. The increasing is similar in the area of 2KB to 64KB, but not very obviously. The reason is that, the RAID subsystem in storage node use cache write-back technique. By such technique, an I/O operation is completed when the data have been written into the memory of SCSI-RAID adapter, and the firmware of SCSI-RAID adapter will move the data to physical disk later to get better write performance. Generally, the size of SCSI-RAID memory assigned to one disk is 64KB, so write operations about the data smaller than 64KB can be done in the cache.

That is why the write performance is better than read without RAMDISK. On the other hand, 200MB/s bandwidth of FC also limits the improvement of write performance. With the RAMDISK, the performance is improved 10%~40%, and reaches 185MB/s, 92% of FC bandwidth. It is near the theoretic value.

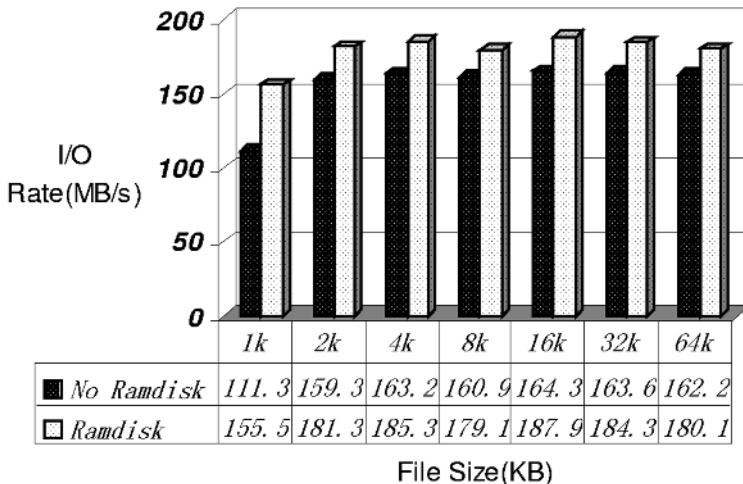


Fig. 6. Result of write performance test

6 Conclusions

An improved FC-SAN based on the rules of file system workload is designed and implemented which uses both RAMDISK devices and DISK devices as the storage devices in network storage system. By evaluating the load heavy or light, the system selects DISK or RAM devices as the main media for I/O operation, file operations of larger data will be done on disk devices, and reading/writing smaller data or file property operations will be done in RAMDISK devices. The evaluation results reveal that, without RAMDISK, the smaller the file operation is, the worse the performance is. To the system above, the performance of read is 124MB/s and the one of write is 164MB/s. after using RAMDISK, the performance of read is improved to 170MB/s and the one of write is improved to 188MB/s. So, we can conclude that:

After using RAMDISK, read performance is improved 50% to 100%, and write performance is improved 10% to 40%. Write operations can make use of 92% bandwidth of Fibre Channel.

The more read requests in mixed requests and the smaller the blocks are, the more the performance of the system can be improved.

This system reduces the time in seeking in disk and operation times of physical disks, decreases the speed gap between CPU and disks and improves the performance of the whole system.

Acknowledgement. The works described in this paper are supported by National High-Tech Research and Development Plan of China under Grant No.2001AA111110.

References

1. Jae-Chang Namgoong, Chan-Ik Park,: Design and Implementation of a Fibre Channel Network Driver for SAN-Attached RAID Controllers. In: Michael L (ed), Eighth International Conference on Parallel and Distributed Systems (ICPADS). IEEE Computer Society, KyongJu City, Korea (2001): 477–483.
2. Kaladhar Voruganti, Prasenjit Sarkar: An Analysis of Three Gigabit Networking Protocols for Storage Area Networks. In: Horst Clausen, Mathew A. Diethelm, Abdelsalam (Sumi) Helal (eds.) 2001 IEEE International Performance Computing, and Communications Conference (IPCCC). IEEE Computer Society, Phoenix, Arizona, U.S.A. (2001): 259–265.
3. J. Griffioen and R. Appleton. The design, implementation, and evaluation of a predictive caching file system. Technical Report CS-264-96, Department of Computer Science, University of Kentucky, June 1996.
4. D. Roselli, J. R. Lorch, and T. E. Anderson. A comparison of file system workloads. In: Proceedings of the 2000 Annual Technical USENIX Conference (2000): 41–54. <http://www.usenix.org>.
5. C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. IEEE Computer, 27(3):17, 1994.
6. Ray Bryant, Dave Raddatz, and Roger Sunshine, PenguinMeter: A new file I/O Benchmark for Linux. In: Proceedings of 5th Annual Linux Showcase &Conference, USENIX, Oakland, California, USA (2001) <http://www.usenix.org>.
7. An-I A.Wang, Peter L. Reiher, Gerald J. Popek, et al. Conquest: Better performance through a disk/persistent-RAM hybrid File system, Carla S.E.ed., USENIX Monterey, California, USA., (2002): 15–28. <http://www.usenix.org>.
8. InfiniBand: Next Generation I/O, <http://sysopt.earthweb.com/articles/infiniband/>.
9. Ashish Paleker, Design and implementation of a linux scsi target for storage area networks. In: Proceedings of 5th Annual Linux Showcase & Conference, USENIX, Oakland, California, USA (2001).<http://www.usenix.org>
10. Y. Zhu and Y. Hu. Can large disk built-in caches really improve system performance. Technical Report TR259/03/02ECECS, Department of Electrical & Computer Engineering and Computer Science, University of Cincinnati (2002).

A New High-Performance Distributed Shared I/O System

Qiong Li, Guangming Liu, Yufeng Guo, and Hengzhu Liu

Parallel and Distributed Processing National Laboratory,
Changsha 410073, China
guo_yufeng21@hotmail.com

Abstract. Distributed Shared I/O System is an important research field of multiprocessor system. HyperTransport is an emerging interconnection standard that offers high-speed, high-performance, point-to-point packet-based link between integrated circuits on a board, particularly between a high-performance processor and peripheral devices. In this paper we introduce a new high-performance distributed shared I/O system developed by ourselves, which uses HyperTransport technology to improve I/O performance and scalability. It attempts to extend the advantages of SMP with the benefits of MPPs and clusters, and overcome the distant I/O problem of distributed non-shared I/O system. To concentrate on a study of the I/O architectural design, some key technologies implemented in the system are discussed.

1 Introduction

The computational performance of multiprocessor systems has been continually improved by leaps, fueled in part by rapid improvements in processor and interconnection technology. When the applications shifted from being CPU-bound to I/O bound in high performance computing, performance cannot be scaled up by increasing the number of CPUs any more, but by increasing the bandwidth of the I/O subsystem. Therefore, to avoid becoming the bottleneck of system performance, the I/O performance becomes ever more critical.

In distributed I/O system, I/O resources are distributed across all processing nodes. The I/O resources on the same node as the processors are called local I/O resources, while those on other nodes are called remote I/O resources.

In distributed non-shared I/O system, each processor is only able to directly access local I/O resources, and each I/O device is only able to DMA local memory on the same node as the I/O device. If a processor wants to access a remote disk, both CPU in Node A on which the application is running and CPU in Node B to which the I/O resource is attached must take part in the distant I/O operation, and will be interrupt when the DMA operation is finished. Furthermore, the requested data must be transported not only from the disk to local memory and vice versa, but also have to be migrated into destination memory. Therefore, distant I/O operation aggravates the overhead of CPU and the load of interconnect fabric, and reduces the performance of entire system.

To overcome the distant I/O problem of distributed non-shared I/O system, we introduce a new high-performance distributed shared I/O system developed by

ourselves. In the system, every processor is able to directly access all I/O resources. Furthermore, any I/O device is able to DMA to and from all memories in the system, not just the local memory. In distant I/O cases, data may be transferred in parallel directly from source to the destination, without interrupting CPUs in the node to which I/O device is attached. In addition, efficient hardware support is provided for high-performance distant I/O, internode communication, and synchronization.

AMD Inc. and its technology partners developed the HyperTransport I/O bus structure to solve the I/O bottleneck in high performance 32- and 64-bit processor based systems. HyperTransport(HT) technology, formerly named Lightning Data Transport (LDT), offers high-performance, Point-to-Point packet-based link between integrated circuits on a board, particularly between high-performance processors and peripheral devices. HyperTransport technology delivers a scalable architecture that provides better than an order of magnitude increase in bus transaction throughput over existing I/O bus architectures such as PCI, PCI-X and AGP and compares favorably with newly proposed I/O bus structures such as RapidIO and Infiniband. Best of all, it provides high performance throughput while maintaining backward compatibility with existing software developed for the PCI bus. With the ability to scale from narrow configurations (such as 8-bit wide) with relatively low speed (200 MHz) clock rates to upwards of 32-bit wide, high speed (800 MHz) clock rates, the HT I/O bus architecture is the ideal platform for implementing the next generation of I/O systems. Therefore, we use HyperTransport technology in the new distributed shared I/O system to improve I/O performance and scalability.

2 Distributed Shared I/O System Architecture

2.1 System Architecture

As shown in figure 1, the system employs CC-NUMA architecture to extend SMP by connecting several SMP nodes into a larger system. While maintaining the advantages of the SMP architecture, the CC-NUMA system alleviates the scalability and availability problem of conventional SMP.

The system has a modular physical structure, which enhances scalability. Not only processors, but also memory capacity and I/O capabilities can be increased by adding more nodes. The system is scalable from 2 to 256 processors, while maintaining proportional data storage, memory access, communication, synchronization, and I/O capabilities. The problems of contention and bandwidth are alleviated because an application can access multiple local memories simultaneously most of time, taking advantage of data locality.

A notable advantage of the system over multicomputers is that the programmer does not have to explicitly distribute data structures over the nodes. The system hardware and software will automatically distribute data among the nodes initially. During runtime of the application, the cache coherency hardware will automatically move the data to the nodes where they are referenced.

As shown in figure 1, the system uses a crossbar ASIC, called CNC (Centralized Node Controller), to connect the processors, memory, interconnection fabric, and the I/O subsystem. CNC is responsible for maintaining Cache Coherence protocol, converting internal packets format to/form the external format used by the

Hypertransport fabric or the interconnect network, supporting PIO and DMA transactions between processors, memory and I/O device, and supervising all I/O transactions for the HT fabric.

NIC (Network Interface Circuitry) is responsible for routing internode communication. Fabric switch technology is used to provide a high-performance interconnection network between individual nodes. Therefore, accesses to remote memory or I/O devices have low latency and high bandwidth, and the aggregate bandwidth grows linearly with the system size.

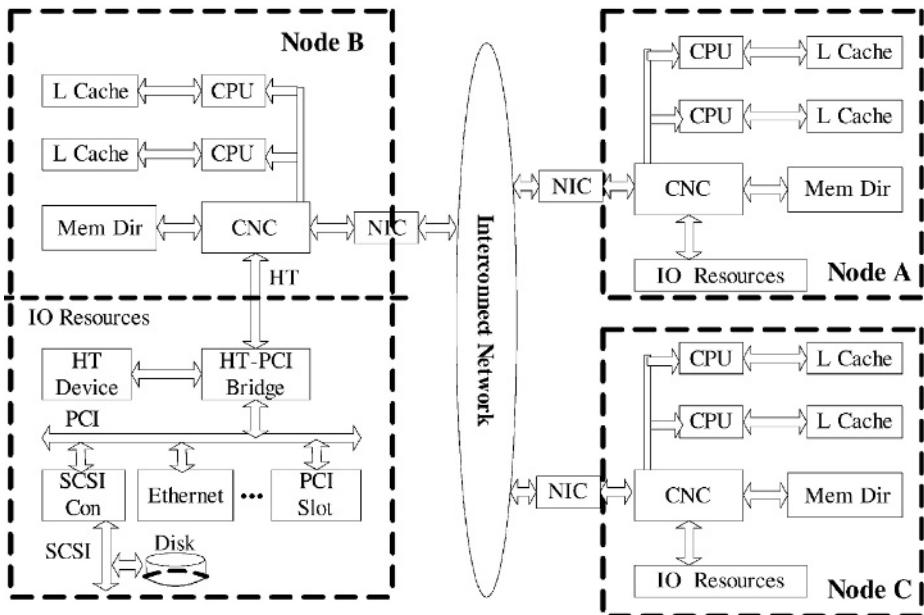


Fig. 1. Architecture Diagram of a new Distributed Shared I/O System

2.2 I/O Subsystem

As shown in figure 1, the I/O subsystem consists of all I/O resources distributed across all processing nodes. I/O resources are accessible from any processor and shared equally.

Each processing node has following primary I/O components:

- 1) **HT-PCI Bridge:** It interfaces the new generation of HT based hosts or HT based peripherals to PCI based peripherals, and provides a high-speed HT tunnel to allow downstream connection to other HT devices.
- 2) **HT devices:** HT technology-enabled devices include HT tunnels and HT cave. A HT tunnel implements two link interfaces and is capable of forwarding traffic between other devices within a chain. A HT Cave implements a single primary link interface, which must always sit on the end of the chain. The system supports HT link widths of 8, 16, and 32 bits. Each HT link supports clock

frequency of 200/400/600/800MHz, and maximum bandwidth of 12.8 GB/s (with 16bits link width) or 25.6 GB/s (with 32bits link width).

- 3) *PCI devices*: Through the 64bit 33MHz/66MHz PCI buses or 64bit 100MHz/133MHz PCI-X buses, different types of I/O devices can be connected to the system, such as SCSI controller, Ethernet controller, Fiber Channel, ATM, extension, etc.

There are many kinds of peripheral devices, as well as different I/O specifications supported in the I/O system, such as HyperTransport, PCI-X, PCI 2.2, SCSI Ultra320, SCSI Ultra160, Giga EtherNet, Fiber Channel, ATM, IDE, LPC, USB, etc.

HyperTransport has two kinds of topologies—chain and tree. HT I/O fabrics are implemented as one or more daisy chains of HT technology-enabled devices, with a bridge to the host system at one end. Devices can implement either one or two links. A single HT I/O chain does not contain HT-to-HT bridge devices. It can contain native HT peripheral devices (such as an Ethernet controller) and can also contain bridges to other interconnects (such as PCI-X bus). A HT tree contains one or more HT-to-HT bridge devices. HT-to-HT bridge device operates as a host bridge for devices on its secondary chain. Every processing node can contain multiple bridges, each supporting either a single HT I/O chain or a tree of HT I/O chains. For convenience in integrating multiple functions onto a single chip, or to allow parallelism between independent request streams, individual HT devices can use multiple UnitID values. There is no specific limit on the number of physical devices. However, there are only 31 UnitIDs available to each chain.

2.3 Parallel I/O Solution

Parallel I/O accessing and concurrent data transfer are supported in the system in multiple hierarchies, such as nodes, Hypertransport fabric, PCI-X buses, PCI buses, SCSI buses, LPC buses, etc., to make it easy to parallelize and load balance.

The interconnect network allows multiple nodes to perform I/O operations simultaneously, without affecting each other's performance or data integrity. In addition, some parallel I/O methods and techniques are used in the I/O system to accomplish the following:

- 1) Furthest use available parallel I/O devices to increase the bandwidth.
- 2) Minimize the number of disk read and write operations per device. It is better to read the requested data by small quantity of larger chunks instead of large quantity but small ones, which avoids disk specific latency time for moving the read/write arm and waiting for the positioning of the spinning disk.
- 3) Minimize the number of I/O specific messages between processes to avoid unnecessary costly communication.
- 4) Maximize the hit ratio to avoid unnecessary data accesses. One of the key techniques is to read a large sequential block of the disk at once into main memory and then to extract smaller data blocks according to the application requests.

2.4 I/O System Characteristics

Other than the features introduced above, there are several primary characteristics of the distributed shared I/O system:

- 1) *Shared I/O solution*: Any processor is able to directly access all I/O resources. Furthermore, any I/O device is able to DMA to and from not only the local memory but also all memories in the system. Hardware supports DMA transactions to read, write global shared memory space, and PIO transactions to read, write global shared I/O space. OS manages global memory and I/O resources, to provide Single Address Space and Single storage Space. Distribution is transparent to applications.
- 2) *High performance*: The aggregate I/O bandwidth (as well as memory and interconnect) increases almost linearly with the numbers of processor, while the latency only grows logarithmically. The system's internal bandwidth matches the I/O rates of peripheral devices attached to the system.
- 3) *Scalability*: The I/O system uses a modular physical architecture, which enhances scalability. The architectural design pays great attention to having a balanced system, so that memory, I/O, and interconnect capabilities all proportionally scales up as the machine size increases.
- 4) *Availability*: Multiple copies of a portion of the operation system can run on multiple nodes, so that the failure of one will not disrupt the entire system.

3 Distant I/O Operation

There are two types of I/O operations: PIO operation and DMA operation. PIO operation provides a method by which the processors in the system can control I/O devices, while a processor in the system is the initiator of the PIO request. When CNC receives the PIO request, it converts the packet to HT packet first, and issues a PIO request downstream to the HT device. DMA operation provides a method by which I/O devices can read and write global memory, while I/O device is the initiator of a DMA request. When CNC receives the DMA request, it sends the request to memory in destination node, through the high-performance interconnection network between individual nodes.

In the Distributed Shared I/O system, any processor in the system can control all the I/O devices by PIO operation. At the same time, any device in the system can access global memory by DMA operation. The I/O data is organized as packets to be passed to various devices. When an application program requires I/O services, the node on which the application is running initiates the file transfer with an appropriate Read or Write command. In distant I/O operation, data may be transferred in parallel directly from source to the destination, without interrupting CPUs of node to which I/O device is attached. Following are the demonstration of the process in accessing distributed shared I/O resources.

For instance, there are four nodes possibly involved in the distant I/O operation, as shown in figure 1:

- 1) Node A, in which the application is running.
- 2) Node B, to which the I/O resource is attached.

- 3) Home Node C, in which there is the requested Cache line in its memory.
- 4) Node D, in which there is the newest copy in Cache.

The principal events involved in writing a file to a distant disk are summarized below. Reading a file in a distant disk into global memory is very similar but the flow of data is reversed.

- 1) CPU of Node A writes DSA and DSP registers of SCSI controller in Node B. When CNC in Node B receives the PIO request, it issues a HT writing request downstream to the HT-PCI Bridge. The HT-PCI Bridge converts the HT packet into a writing transaction targeted to SCSI controller on the PCI bus.
- 2) The Scripts processor within SCSI controller prefetches Scripts instructions.
- 3) The Scripts processor performs Scripts instructions.
- 4) The SCSI controller read Command Buffer, gets SCSI disk writing command.
- 5) Once the DMA FIFO within the SCSI controller is empty, a Read request will be initiated by the SCSI controller. After receiving the PCI Read request, the HT-PCI Bridge sends the memory read request packet to Home node C. The request arrives to CNC in Home Node C via system interconnect network, and will be handled by CNC according to cache coherency protocol. The newest data will be read, either from the memory of Home Node C or from the Cache of Node D decided by who has the newest copy of the requested Cache line, and sent to the node B, to which the disk is attached.
- 6) Once there are data in DMA FIFO, the SCSI controller will initiate SCSI Write transaction to the disk on the SCSI bus.
- 7) To respond the SCSI Write command, the disk starts to receive the data.
- 8) Step 5, 6 and 7 are circled until all requested Blocks (each 512Bytes) have been transported.
- 9) Once the DMA operation is finished, a PCI interruption is generated by the SCSI Controller. After receiving the interruption, the HT-PCI Bridge sends an Interrupt request packet to CNC in Node A.
- 10) Being interrupted, the initiator CPU of Node A will recognize that it is the SCSI controller in the Node B who generated the interrupt, and handle the interrupt according to interrupt cause, which is the completion of the disk writing.

4 Design of HT-Host Bridge

HT-Host Bridge is an important function component of CNC developed by ourselves. HT-Host Bridge is the interface from the host to the HT fabric, responsible for managing all I/O transactions on the HT fabric.

As shown in figure 2, The HT-Host Bridge within CNC consists of following primary function blocks:

- 1) *A bridge register set:* Being a HT-Host bridge, CNC contains a bridge register set to control mapping of transactions between Host and HT fabric. Configuration accesses of type 1 are issued by CNC, which forwards type 1 accesses through the bus hierarchy and translates them to type 0 accesses when driving them onto their final destination bus.
- 2) *Packet translation:* Being a HT-Host bridge, CNC is responsible for packet translation between internal format and HT format.

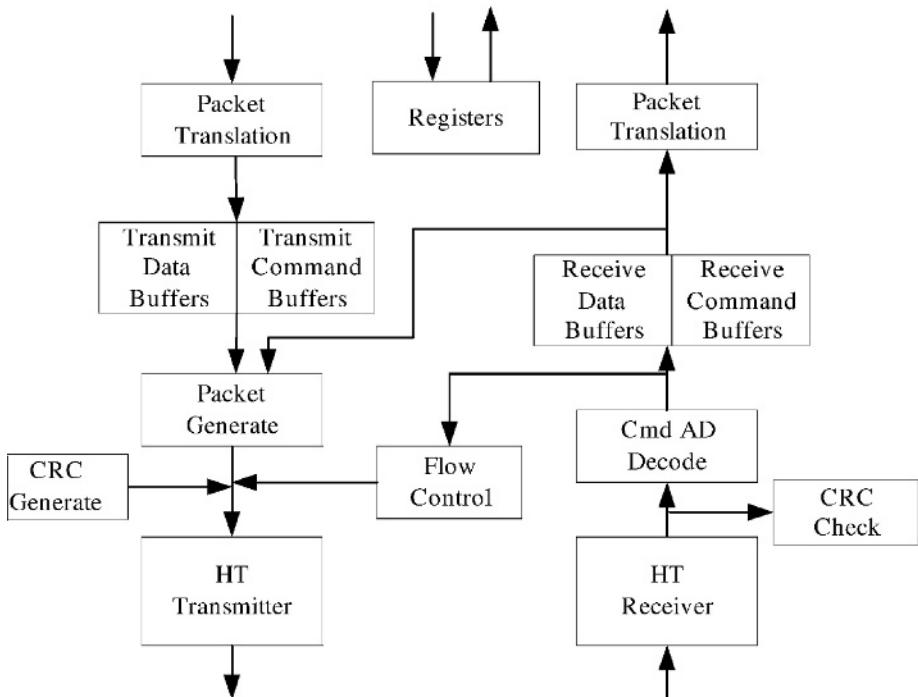


Fig. 2. Block Diagram of HT-Host Bridge

- 3) *Receive command buffers and Receive data buffers:* CNC supports three virtual channels of receive command buffers and receive data buffers, i.e. Posted Requests, Non-posted Requests and Responses. Being a HT-Host bridge, CNC must guarantee that the three virtual channels won't blocking each other due to buffer management and routing issues, which is why each channel has command and data buffer space separate from the other two.
- 4) *Transmit command buffers and Transmit data buffers:* CNC supports three virtual channels of transmit command buffers and transmit data buffers, i.e. Posted Requests, Non-posted Requests and Responses.
- 5) *Buffer manager:* Buffer manager is responsible for managing interactions between I/O streams, managing HT virtual channel, and properly maintaining the order of downstream I/O transactions and the host domain I/O transactions.
- 6) *Packet generation:* Packet generation is responsible for generating HT packet.
- 7) *Command and Address decoder:* Command and Address decoder is responsible for decoding command and address.
- 8) *Flow controller:* HT buffers are flow-controlled at the link level using a coupon-based scheme, in which the transmitter contains a counter for each type of buffer at the receiver. The receiver sends NOP packets to indicate how many buffers of each type being available.

- 9) *HT transmitter and receiver*: HT transmitter and receiver are responsible for managing HT link, transmitting and receiving packet.
- 10) *CRC Generation and CRC Check*: CRC Generation and CRC Check are responsible for CRC based link error detection to ensure data integrity and reliability.

5 Interrupt Handling

Just as mentioned in section 3 “Distant I/O Operation”, after writing disk task being sent to Node B, both CPUs in Node A and Node B will not bother about the DMA operation until it is completed, at that time CPU in the Node A will receive the interruption from distant Node B. To support the Distributed Shared I/O architecture, individual I/O device, such as HT device and PCI device, should be able to send interruption to any node in the system.

There are several kinds of interrupt sources in the system, such as: interruption for the errors within CNC, interruption of HT devices, PCI interruption, etc.

Error interruption can be sent to the local CPU or to the remote CPU pre-designated, which may be the Default Error Processor. For the benefit of the system debugging, the Default Error Processor need not be set in the same node in which the error occurs, in case the CPUs in the node have failure, the error interruption can be sent to another CPU in another node.

I/O Advanced Programmable Interrupt Controller (IOAPIC) provides multi-processor interrupt management. In systems with multiple I/O subsystems, each subsystem can have its own set of interrupts. The interrupt vector and interrupt steering information can be specified per interrupt in corresponding Interrupt Redirection Register (RDR).

Every PCI device has a set of interrupt status registers, which collect the interrupts both for the completion of DMA operation and the errors of PCI device. Once a PCI interruption occurs, it can be sent to the IOAPIC within the HT-PCI bridge by PCI INT pin. Then the HT interrupt request packet will be generated according to corresponding RDR register within the IOAPIC. In the RDR register there are interrupt destination node ID and interrupt vector information used to route and identify interrupt.

All HT interrupt requests, regardless of interrupt class, are sent to the HT host bridge (within CNC) using posted write request packets. CNC is then responsible for delivery the interrupt request to destination node via system interconnect network. According to the interrupt vector, the target CPU can identify which I/O device in which node generated the interrupt, and handle the interrupt according to interrupt cause.

6 Conclusion

In this paper we have introduced the architecture of a new high-performance distributed shared I/O system developed by ourselves. It attempts to extend the advantages of SMP (e.g., single system image, easy to manage) with the benefits of

MPPs and clusters (e.g., scalability), and overcome the distant I/O problem of distributed non-shared I/O system. It's our goal to use the system not only as a supercomputer to satisfy the I/O needs of parallel scientific applications, but also as a high-performance network server to satisfy a large number of simultaneous user I/O requests. Because of its severity, the I/O bottleneck problem deserves systematic attention in every respect of the system design, such as: applications, algorithms, compilers, operating systems and architecture.

References

1. Kai Hwang, Zhiwei Xu,: Scalable parallel computing technology, architecture, programming.
2. Rajkumar Buyya : High performance Cluster computing: Architectures and systems.
3. HyperTransport Technology consortium.: HyperTransport I/O Link Specification 1.05.(2003)
4. API NetWorks, Inc : HyperTransport Technology I/O White Paper,(2003)

IA64 Oriented OpenMP Compiler: Design and Implementation of Fortran Front End

Hongshan Jiang, Suqin Zhang, and Jinlan Tian

Department of CS&T,
Tsinghua University,
Beijing, 100084, China

Abstract. This paper presents an OpenMP compiler Fortran Front End. It introduces principles and algorithms to deal with the implicit data parallelism, which is directed by WORKSHARE directive in OpenMP Fortran API V2.0. For implementation, automatic parallel computation division by compiler is achieved by front end's converting implicit data parallelism to explicit one in Very High WHIRL. In addition, this paper presents some optimization techniques to handle compiler-generated redundant synchronization and consistent DO loop. At the end, a performance experiment is given to prove the effectiveness of mentioned strategy.

1 Introduction

IA64 (Itanium Architecture)[1] is a 64-bit architecture published recently by the Intel Corporation. The main characteristics of this architecture include the following. The instruction level parallelism is enhanced by more structural components such as bigger register heap and instruction bundles that are able to accommodate up to three instructions. By using the branch registers, the conditional registers and the conditional execution instructions, branches have been reduced to the least. By allowing compilers to schedule load operation beforehand and enabling cache management of memory hierarchy, the storage latency is concealed. Finally, the special hardware enabled function returning and invoking mechanism supports those modular codes generated by compilers.

OpenMP is a shared-memory parallel programming language initiated by DEC, IBM, Intel and SGI in Oct. 1997. OpenMP inherits a number of features from ANSI X3H5 standard and Pthreads of IEEE. Its most prominent features[2,3] are its support of incremental parallelism and its suitability for developing coarse-grain parallelism in the thread level. Because of its easy learning and usage together with the wide support from several famous parallel machine venders, it has become the standard of shared-memory parallel programming language.

Using IA64 processors to construct SMP and adopting OpenMP as the parallel programming environment has many advantages due to its effective utilization of IA64's powerful instruction level parallelism capability in conjunction with OpenMP's advantage in thread level parallelism. ORC (Open Research Compiler) is a public compiler research infrastructure aimed for the IA64 architecture. Based on it,

we have developed an OpenMP compiler to support OpenMP Fortran API V2.0. This paper mainly discusses the design and implementation of the Fortran front end of this OpenMP compiler.

2 Design and Implementation

The front end of ORC includes a C/C++ front end and a Fortran front end. It performs lexical analysis, syntax analysis and semantic analysis on source code and finally generates its corresponding Very High WHIRL intermediate presentation. The Fortran front end descends from Cray F90 front end. Its architecture is shown in figure 1.

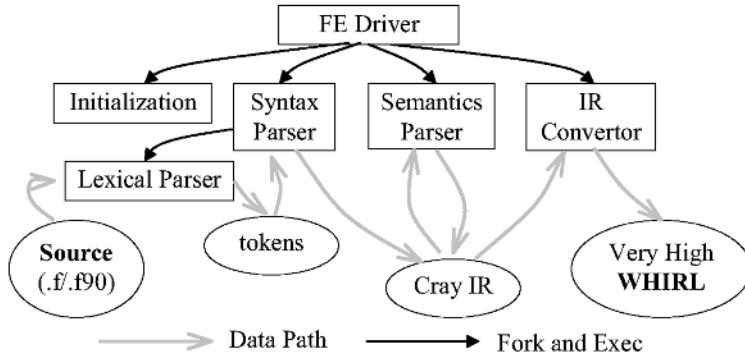


Fig. 1. Control flow and data flow of Fortran FE.

OpenMP is not an independent programming language. It can be considered as a parallel extension to current sequential high-level language such as Fortran or C/C++. The extensions can be compiling directives in source code, functions in programming libraries or OS environment variables. Each front-end processes corresponding version of OpenMP separately.

Current version of Fortran front end can only handle OpenMP Fortran API V1.0 directives. Based on it, we added the processing mechanism for new features in OpenMP Fortran API V2.0, such as the support for NUM_THREADS, COPYPRIVATE clause and WORKSHARE[3] directive etc. Moreover, some semantic processing needs modification, such as the processing for REDUCTION clause. The processing of OpenMP directive spreads over every phase of the front end from lexical analysis to semantic analysis. After being translated to VH WHIRL, OpenMP directives are changed to pragma statements or WHIRL regions with pragma labels.

The handling of WORKSHARE directive, which occupies a large amount of workload for the compiler implementation, is difficult for Fortran front end. WORKSHARE directive is a means for OpenMP Fortran API V2.0 to support implicit data parallelism in Fortran90 i.e. it does not specify the parallel model explicitly. For instance, in the array assignment operation of Fortran90, assignment of different elements of arrays can be executed in parallel, but how these assignment operations are dispatched to different processors is not specified distinctly. The

implicit data parallelism can be regarded as a simple way for programmers to specify parallel behaviors so as to free programmers from some trivial issues such as computation division of parallelism. The only thing a programmer should do is to supply some necessary simple information, the remainder is taken care of by the compiler.

Our solution for implicit data parallelism is to transform WORKSHARE region to WHIRL region with WORKWHARE pragma by front end, and then translate implicit data parallelism in WORKSHARE region to explicit form. This involves the division of computation. In the next section, we will elaborate computation division principles, algorithms and implementations. Furthermore, we will introduce some optimization techniques for the special features of OpenMP. The IA64 related optimization is placed in the back end.

3 Process of Implicit Data Parallelism

A basic block that can be executed in data parallelism is specified by WORKSHARE directive in OpenMP. The data parallelism manner of such basic block is not specified explicitly but handled by the compiler. Syntax of WORKSHARE directive is shown in figure 2.

```
!$OMP WORKSHARE
  block
!$OMP END WORKSHARE [NOWAIT]
```

Fig. 2. Syntax of WORKSHARE directive.

In the figure, statements in the block are the array assignment of Fortran90. WORKSHARE directive indicates that work of each statement in *block* is shared i.e. they are divided into separate units of work and are executed by a group of threads in parallel. If it is guaranteed that each unit will be executed exactly once, the units of work may be assigned to threads of the PARALLEL region[3] in any manner.

3.1 Computation Division Principle

Array operations of Fortran90 can be classified into four categories: elemental operations, query operations, reduction operations and complex operations. According to the semantics of WORKSHARE directive, we introduce the following computation division principle of array operations in the WORKSHARE region.

1. In the case of elemental operations, the evaluation for each element of the array is counted as one unit of computation.
2. In the case of the reduction operation, current work is divided into several units with each unit being executed in one thread. Each unit is to evaluate the local reductive result, and then an additional work unit is added to deduce local results to global reductive results. Necessary synchronization is added at the end of each local reduction.

3. In the case of complex array function operations, the computation division depends on semantics. The principle is to assign irrelevant operations to separate threads. For operations that have strong internal data dependency such as PACK[4] and UNPACK[4], the whole operation forms a single unit of work.
4. If none of the above rules applied, that operation is treated as a single unit of work.

3.2 Computation Division Algorithm

According to semantic requirement, statements in the WORKSHARE region are array assignments, scalar assignments, FORALL statements or WHERE statements[4]. Actually, the last two are conditional assignments, so the main issue of parallel compiling algorithm is the assignment division and the expression division.

Assignment division belongs to the elemental computation division, which can be processed according to the aforementioned principles. Solution of expression division is discussed as follows. The evaluation of expression can be treated as the post-order traversal of an expression binary tree. Every internal node of the tree corresponds to a single array operation while every leaf node corresponds to an array or scalar. Therefore, expression can be mapped to a series of single evaluation statements by post-order traversing an expression tree. For every single evaluation statement, its right value is an array expression that has no more than one operator or function, while its left value is a temporary variable of an array or a scalar. We divide every single evaluation statements into units of work by the rule of aforementioned principles, combine consistent work units, eliminate redundant temporary variables, and thus generate statement series that have been divided into work units. Consistency of work units herein means to compute the elements in the same position of two arrays of the same shape.

For instance, suppose the number of current threads is 2, A and B are one-dimensional arrays of length N . According to statement WHERE($B < 0$) $B = A - \text{MIN}(A)$, its computation division is illustrated in Figure 3.

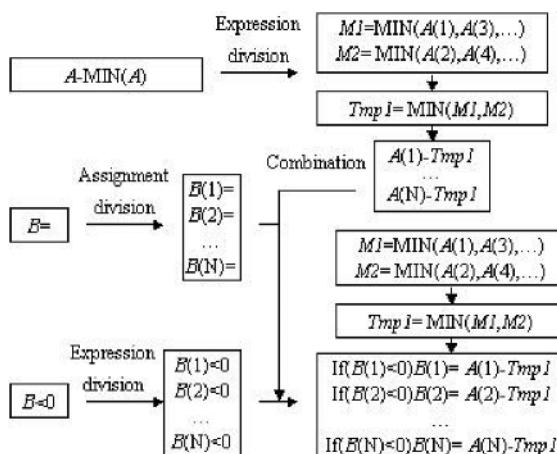


Fig. 3. Example of computation division.

3.3 Implementation

Our implementation is to convert implicit parallelism to explicit one in VH WHIRL. For each array operation in Fortran90, we give its corresponding explicit form of operation. For instance, if A is a two dimensional array of shape N×M, B=TRANSPOSE(A) can be translated to the explicit parallel form as in Figure 4.

```
!$OMP DO PRIVATE(I,J)
DO I=1,M
    DO J=1,N
        B(I,J) = A(J,I)
    END DO
END DO
```

Fig. 4. Explicit form of parallelism of B=TRANSPOSE(A).

After a series of processing, this explicit form will be compiled to parallel codes by the back end. The particular way of computation division is decided by the number of threads at the time and environment variables such as OMP_SCHEDULE.

4 Compilation Optimization

The compilation optimizations are mainly focused on the optimization in VH WHIRL for features of OpenMP directive. Synchronization exists implicitly in many OpenMP directives and may leads to additional cost. How to reduce the synchronization cost to minimum is the main problem that the compilation optimization should care about. Moreover, most parallel computations of OpenMP program exist in loop statements. Therefore, the optimization of loop statement of OpenMP is also a problem needs consideration. The following are the optimization techniques we adopt.

1) Removal of Redundant Synchronization

Synchronization is implied at the end of DO loop directive of OpenMP. If successive loop directives are not data dependent, their redundant synchronization can be removed in order to reduce the waiting time and improve the parallel efficiency. As Figure 5 illustrates, A and B are two successive tasks which have no data dependency. These two tasks are executed by 2 threads in parallel. A synchronization operation occurs between them. Because of the workload imbalance, after thread 1 finishes A2 at time t1, it will wait till thread 0 finishes A1. At time t2, these 2 threads start to execute B. This will bring unnecessary extra execution time. We can see that after the removal of redundant synchronization between A and B, unnecessary waiting time is saved and parallel efficiency is improved.

The time saved by the removal of redundant synchronization equals to the minimum waiting time of all threads, i.e. $T_{\text{saved}} = \text{MIN}(T_{\text{waiting}1}, T_{\text{waiting}2}, \dots, T_{\text{waiting}n})$. In fact

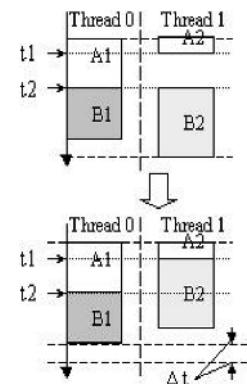


Fig. 5. Case of removal of redundant synchronization.

synchronization always brings additional cost, so $T_{\text{waiting},1}, T_{\text{waiting},2}, \dots, T_{\text{waiting},n} > 0$, thus $T_{\text{saved}} > 0$.

2) Combination of consistent loop

For successive loop statements, if their lower bound, upper bound and step length are all identical, and no data dependency occurs between loop bodies of different subscript, their loop bodies can be combined into one loop body. The execution order of these statements must be kept same as before. This can reduce iteration times, thus enhance the efficiency. Such cases usually exist in statement series converted from implicit parallel statements in Fortran90.

3) Optimization for Orphan Directives[5]

An orphan directive is an OpenMP directive that does not appear in the lexical extent of a PARALLEL construct, but may lie in its dynamic extent. If this directive does not lie in the dynamic extent of a PARALLEL construct, executing the sequential codes could gain higher performance. Therefore, for each orphan directive, whether to execute its parallel codes or to execute its sequential codes is dynamically determined by its context. Our solution is to add a statement, i.e. IF(OMP_IN_PARALLEL()), to tell if the directive is in parallel context or in sequential context. If in parallel context, the parallel version is executed, otherwise the sequential version is executed.

4) Parallelism of data independent Operations

Some operations inside WORKSHARE directive cannot be divided into parallel work units for their tightly coupling. Yet there is no data dependency among those operations. Hence it might be more efficient to execute them in parallel. Our solution is to put data independent operations into different SECTION blocks in one SECTIONS region[3] to execute them in parallel.

Removal of redundant synchronization has little influence on efficiency in balanced workload cases, meanwhile in imbalanced workload cases, efficiency improvement is proportional to the minimum waiting time of all threads. Combination of loops brings about 7% enhancements in efficiency depending on loop body sizes. Optimization for orphan directives can enhance efficiency in sequential cases only if the decision cost is less than that of parallel codes.

5 Test Result

On an Itanium2 parallel machine with four CPUs of 900MHz, 4G bytes memory, we tested our compiler which adopting the above compilation strategy. The tested programs use WORKSHARE directive and Fortran90 array operation statements. They are numeric computation programs such as array addition, multiplication and Jacobi Iteration Method. Testing data includes an integer array of 1 million elements. Table 1 shows the result of addition and multiplication of arrays: $D=A+B*C$, where A, B, C, D are integer arrays of 1 million elements. Table 2 shows the result of using Jacobi Iteration Method to solve the equation: $(d^2/dx^2)u + (d^2/dy^2)u - u = f$, where $u(x,y) = (1-x^2)(1-y^2)$ ($-1 < x < 1, -1 < y < 1$), $f(x,y) = -2(1-y^2)-2(1-x^2)-(1-x^2)(1-y^2)$, with grid size 100 * 100, iterative parameter 0.1, maximum tolerant error 10^{-7} , iterative number 61309.

Table 1. Addition and multiplication

Thread Number	Execution Time (s)	Speed-up	Eff. (%)
1	1.323	1	
2	0.673	1.97	98.5
3	0.462	2.86	95.3
4	0.414	3.20	80

Table 2. Jacobi Iteration Method

Thread Number	Execution Time (s)	Speed-up	Eff. (%)
1	190.02	1	
2	109.25	1.74	87
3	74.12	2.56	85.3
4	56.37	3.37	84.3

The results show that our parallel compilation strategy has gained considerable efficiency.

6 Conclusion

How to take advantage of new features in architecture development and to actively make progress in bridging the gap between architecture and programming interface has long been a hotspot of IT community. This paper introduces the design and implementation of Fortran front end of an IA64 oriented OpenMP compiler. It mainly discusses the processing of implicit data parallelism supported by OpenMP, and elaborates some optimization techniques related to OpenMP programming features. The proposed algorithm and methodology have general meaning to the design of friendly parallel programming interface and to the dispatching of many detail issues, like computation division of parallelism, to compiler for automatic processing so as to improve the programming efficiency. In addition, transformation between control dependency and data dependency has to be further studied to take advantage of new features of conditional instruction of IA64.

References

- [1] Intel Cop., Intel@IA-64 Architecture Software Developer's Manual[OL]. <http://segfault.net/~scut/cpu/ia64/>, July 2000.
- [2] CHEN Guoliang. Parallel Computing : Architecture, Algorithm and Programming[M]. Beijing:Higher Education Press, 1999. (In Chinese).
- [3] The OpenMP ARB, OpenMP Fortran Application Program Interface Version 2.0[OL]. <http://www.openmp.org/specs/>, November 2000.
- [4] Michael Metcalf, John Reid. Fortran 90 Explained[M]. London.:Oxford University Press,1990
- [5] Matthias Müller. Some Simple OpenMP Optimization Techniques[A]. Rudolf Eigenmann, Michael J. Voss. OpenMP Shared Memory Parallel Programming[C]. Berlin:Springer, 2001. 31–39.

An Alternative Superscalar Architecture with Integer Execution Units Only

Kenneth K.C. Lee¹, K.-E. Grosspietsch², and Y.K. Chan¹

¹Department of Computer Science,
City University of Hong Kong,
Hong Kong, China

kclee@cs.cityu.edu.hk
csykchan@cityu.edu.hk

²Fraunhofer-Gesellschaft,
Institute of Autonomous Intelligent Systems,
Sankt Augustin, Germany
grosspietsch@ais.fraunhofer.de

Abstract. Superscalar architecture resulting in aggressive performance is a proven architecture for general purpose computation. The negative side effect of aggressive performance is the need for higher number of register read/write ports to supply operands to multiple execution units; the need to resolve false data dependence and true data dependence; the need to dispatch operand ready instructions to execution units and finally retire out of order executed instructions to program order. A processor architecture is proposed in here to address at least some of the above negative side effects. This processor architecture is call LITERAL QUEUE ARCHITECTURE(LQA). In here, LITERAL has the meaning of immediate data. In LQA, opcode and operands in an instruction are treated as a self contained structured element and forms the necessary and sufficient condition for instruction execution. Sequence of instructions embedded with LITERALS are treated as elements in a QUEUE. The elements are executed with respect to time and rotated out of the QUEUE while new elements are rotated into the QUEUE.

1 Introduction

Currently, the main stream architecture is still the superscalar architecture as it is more general purpose. It is always known that main stream superscalar architecture is bottle-necked at the register file with many read/write ports in order to satisfy the desire for wide issue superscalar performance. However, a multi-ported register file with many registers is very expensive and will increase processor cycle time due to complex circuitry. For example, the area required to build read/write ports is proportional to the square of the number of ports[1],[2]. Many strategies have been proposed to reduce the complexity caused by the register file ranging from different register organization to different architectural choices such multiple cluster register files[2]. The strategy is a variation of general purpose registers and cannot remove the drawback of register files completely. The organization of multi-ported register

file is further complicated that operands are not fed to instructions directly for execution but fed to register renaming circuit. Specifically, register based superscalar micro-architecture has the following limitations:

- 1) The register renaming bandwidth has to grow proportionally with the issue bandwidth. The register renaming process has a sequential component [3]. It involves the determining of data dependencies amongst the instructions that are being simultaneously renamed [4]. The rename stage latency is limited by the number of read ports needed for the register map table, which is proportional to the number of operands simultaneously renamed. These burdens make renaming more than a few instructions per clock cycle difficult.
- 2) The instruction window in the reservation station could grow very large if the sustained instruction dispatch width does not keep up with instruction issue width. A study [5] showed that the window size grows large with fetch bandwidth as performance increases even assuming unlimited functional units and unlimited instruction dispatch width. Furthermore, a larger instruction window can increase the time needed to feed a result and the result tag to all waiting instructions in the instruction window.
- 3) A large instruction window needs a large number of functional units to exploit instruction level parallelism. The complexity of full bypassing of results is quadratic growth with respect to the number of functional units.

Another negative side effect of attempting to achieve high degree of Instruction Level Parallelism by current superscalar architecture is known as data hazards.

The 3 kinds of data hazards are respectively as follows:

RAW (Read After Write) data hazard is also known as pipeline stall;

Earlier Instruction:	Write to a Register say R10
Later Instruction:	Read Register R10

WAR (Write After Read) data hazard is also known as antidependence;

Earlier Instruction:	Read a Register say R11
Later Instruction:	Write to R11

WAW (Write After Write) data hazard is also known as output dependence;

Earlier Instruction:	Write to a Register say R12
Later Instruction:	Also write to Register R12

Current microarchitecture deals with above data hazards through “register bypass circuit to deal with RAW”, “register renaming circuit for renaming logical registers into physical registers”, “reservation station circuit for storing instructions with operands in physical registers”, “out of order dispatch mechanism for the instructions in reservation station to execution units” and finally “reorder buffer circuit for retiring physical registers into logical registers in the correct program sequence”. The combined circuitry enables instruction parallelism to be achieved through simultaneous execution in multiple function units. The undesirable side effect is the increases of circuit complex circuitry and slower clock speed that is typically associated with complex circuits.

This paper attempts to show that the proposed LQA is an interesting alternative to current mainstream superscalar architecture in that the concept of register file does not apply and LQA does not require complex circuitry to deal with data hazard.

1.1 Scope and Layout of This Paper

Due to limitation of length, it is not possible to describe the full feature LQA. A subset of LQA is described that is limited to Arithmetic and Logical execution of in line instructions in a basic block. All the operands are assumed to be supplied internally and need not reference data cache or external memory. The way in which LQA deals with control of flow and associated control of flow dependence is not described in here because of length limitation.

The rest of this paper is organized as follows:

- In section two, the motivation, feature and run time behavior of LQA will be described.
- In section three, the ability of LQA to deal with data dependency will be illustrated.
- In section four, an extension of the basic LQA for single ALU instruction issue to cover dual ALU instructions issue will be described.
- In section five, the way in which LQA can retire out-of-order instruction execution into program order retirement will be described.
- In section six, a loose comparison with the MIPS R1000 in the context of the dual ALU execution units will be described.
- In section seven, other aspects of the LQA are illustrated to show that LQA can be a full feature superscalar processor.
- Section eight is usage of LQA and conclusion.

2 Motivation and Feature of the Literal Queue Architecture

Our motivation is that there must be better and simpler ways to perform instruction execution efficiently without the complexity typically associated with current superscalar architecture. A new processor architecture called LQA is proposed in here. Its main features are:

- Operands are stored closely coupled with the opcode; access to them is organized via a pointer-like access mechanism considerably reducing the normal address decoding hardware overhead and deal with data hazard effectively.
- The accessing scheme needs a lower amount of bits to encode the pointer values when compared with the accessing scheme for accessing registers in a register file.
- Statistic investigations have shown that the proposed mechanism on average needs less bus switching events compared to register file architectures.

2.1 Feature of the Literal Queue Architecture

Specifically, the proposed Literal Queue Architecture has the following unique features:

1. In register architecture ,WAW and WAR data hazards are dealt with by register renaming. In our LQA, operand is always directly fed to the instruction which requires that particular operand. As such, our LQA architecture does not require register renaming circuit. This will reduce die area, power consumption and improve performance.
2. The LQA feeds an operand to the exact instruction which requires that operand. When compared with current superscalar architecture, this desirable characteristics reduces power consumption because the results only go to the selected instruction instead of being broadcasting to all instructions waiting for execution that may require that operand. This run time characteristics is also desirable for bypassing of results from an earlier instruction to the following instruction.

2.2 Run Time Behavior of the Literal Queue Architecture

In terms of run time behaviour, the LQA is illustrated to show how an operand is fed to the instruction to form a self contained package. The illustration is by an example showing the “execution of the 0th instruction and passing the computed result to the 4th instruction” as shown in Figure 1.

Literal Queue Structure				
Literal Queue Architecture Opcode	Left Literal Valid Tag (1=Literal Valid)	Right Literal Valid Tag (1=Literal Valid)	Left Literal Operand	Right Literal Operand
Instruction 0				
Operation Add Left & Right Literals				
Write Literal Result send to Right Literal in Current Instruction +4	1	1	Left literal input	Right literal input
Instruction 1				
Instruction 2				
Instruction 3				
Instruction 4	0 or 1	Set to 1		ALU output to right literal

The diagram illustrates the data flow in the Literal Queue Structure. The 0th instruction adds its left and right literals. The result is written to the 4th instruction's right literal input. The 4th instruction's ALU output goes to its right literal operand.

Fig.1. Right Literal of 4th Instruction ← “Left + Right Literals” of current instruction

The structural element of our proposed LQA comprises of five fields. These are respectively opcode, left literal operand valid tag, right literal operand valid tag, left literal operand and right literal operand. When both Left/Right literal valid tags = 1, the instruction is ready for execution. In the above example of Figure 1, it is shown that the ALU output of instruction 0 is fed to the right operand in instruction 4. In addition, the right operand tag in instruction 4 is set to valid. If the left literal valid tag is zero then instruction 4 is not ready for execution. Otherwise, instruction 4 is ready and directly fed to an ALU for execution.

3 Illustration of How LQA Deals with Data Dependency in a Natural Way

3.1 Elimination of RAW Pipeline Hazard

In Figure 1, there are only two possible situations.

- Either (1) When the left operand tag of instruction 4 is invalid, the ALU output of instruction 0 feeding to the right operand of instruction 4 will be stored in the right literal operand field,
- or (2) When the left operand tag of instruction 4 is valid, it means that instruction 4 is ready for execution. As such, the left operand of instruction 4 and the ALU output of instruction 0 is immediately data forwarded to both the left and right ALU input for execution. In other words, the situation of ‘Pipeline Stall’ can never exist in our architecture as we never need to pass “output” from source instruction as “input” to destination instruction via another medium in the form of register.

3.2 Dealing with Antidependence and Output Dependence

Brief examination of our proposed LQA architecture as illustrated in Figure 1 also shows that it can deal with antidependence data hazard (Write After Read (WAR)) and output dependence data hazard (Write After Write (WAW)). Both above conflicts arise because of two(or more) instructions pointing to the same register which could result in having different operand values at different time. In our case, every instruction has its own unique operand value that is independent of time. The only worry we have is that the operand is not yet ready (tag not valid). The most obvious situation are repetition of basic blocks in a tight loop. In superscalar processing, speculative execution has the effect of unrolling the loop into multiple instances of basic blocks using exactly the same registers in every instance. In the propose LQA, every basic block that has been unrolled will have its own Literal Queue elements and operands are never duplicated.

4 Dual ALU Instructions Issue as an Extension to the Basic LQA

4.1 Managing Literal Operands to Support Dual ALU Execution Units

From Figure 2, it can be seen that it is relatively easy for us to support processing with 2 independent ALUs as it is a simple extension of single ALU execution.

Literal Queue Structure				
Literal Queue Architecture Opcode	Left Literal Valid Tag (1=Literal Valid)	Right Literal Valid Tag (1=Literal Valid)	Left Literal Operand	Right Literal Operand
Instruction 0	1	1	Left literal input	Right literal input
Instruction 1	Set to 1	0 or 1	ALU output to Left literal	
Instruction 2				
Instruction 3	1	1	Left literal Input	Right literal input
Instruction 4				
Instruction 5				
Instruction 6				
Instruction 7				
Instruction 8	0 or 1	Set to 1		ALU output to Right literal
.....				
.....				
.....				
.....				
Instruction 31				

Fig. 2. Literal Queue Architecture supporting multiple execution units

4.2 Dispatching of Operand-Ready-Instructions to ALUs

In every clock cycle, up to two instructions are capable of generating operands for later instructions. The positions for these two issuing instructions within the Literal Queue and the destination instructions receiving the two operands are also known to us. Therefore, only the tags for the two recipient instructions need to be checked for dispatching to ALUs (i.e. instruction 1 and 8). The logic for instruction dispatch(with operand ready) is thus simple and requires no associative logic.

5 Retiring Instructions in Program Order in a 32 Instruction Window

As an illustration, our proposed LQA execution initially starts with instruction 0 to instruction 31 with 64 literal queue operands. As time goes by(when referring to Figure 2), the first 9 instructions(0th to 8th instruction) would have been finished at some time. Therefore, in this example, the first 9 instructions can be retired as a batch of instructions. Subsequent to the orderly retirement of the 9 instructions, 32nd to 40th instructions can now be fetched for execution within our LQA instruction window width of 32 instructions. There is no hard and fast rule on retirement granularity. One reasonable retirement scheme is a granularity of 8 instructions. Our retirement scheme is done in a simple but orderly manner without the need of reorder buffer as required by register architecture for retiring instructions in program order.

6 A Quantitative Example Based on MIPS R10000

Resources needed on registers and buses to achieve instruction level parallelism is best illustrated by the MIPS R10000[6] implementation. The MIPS 10000 has floating point execution units, integer execution units and an address generation unit(Load/Store). For simplicity, we shall only illustrate the two integer execution units in R10000 and to observe hardware complexity in achieving instruction parallelism for these 2 integer execution units.

- (1) Register renaming is effectively performed via a Rename Mapping Table with 32 entries accessed through 12 read ports and 4 write ports.
- (2) In addition to the rename table, there are 64 combined logical and renamed registers in a register file with 7 read ports and 2 write ports. It should be noted that the mechanism of renaming requires associative access during renaming and has a high circuit complexity.
- (3) The Integer Reservation Station consists of 16 instructions and can feed 2 instructions to 2 Integer Execution Units(IEU) in an optimal scenario. Result/results from IEU/IEUs will be sent to both register file and the Reservation Station for updating which requires associative search. Again, associative search has a high circuit complexity.
- (4) In addition to the 32 entries Rename Mapping Table and the 64 registers, there are another 32 registers in the Reorder Buffer.

When summing (1), (2), (3) and (4) we are talking about 128 registers/table_entries in different hardware blocks. There are also 19 read ports and 6 write ports with spaghetti like interconnecting read/write ports between them with very negative consequences in terms of speed and power consumption. And yet, the instruction window in the Reservation Station has only 16 instructions.

In our proposed LQA example, we require 64 literals for storing/retrieving operands. This is halve of the R10000 register number. With 64 literals, we can support 32 instruction window which is twice the number of R10000. Based on our

examples, we only require 4 read ports and 2 write ports which again is significantly less than the R10000. Our access mechanism is explicit and does not require associative search/access that is typical of current superscalar architecture that requires complex circuit.

7 Other Essential Elements for Full Feature LQA

In previous description of LQA, the need for external memory access via Bus Interface Unit has not been factored in. However, this is a relative simple issue. The other more important issue is that the current LQA operands are of used once only and then discard operation. If an operand needed to be used twice, then this needed to be taken care of. This particular issue has already been investigated by Cruz[7] and proposed a scheme as register caching. According to our very preliminary sizing result, about 65% of operands are used once only and the remaining are typically used either twice or three times. In other words, the register caching scheme as proposed by Cruz is effective in dealing with our situation of “multiple use of the same operand” and will not cause a significant negative effect either in terms of clock speed nor power consumption. Our description of the LQA is confined to execution of a basic block. For speculative execution and dynamic branch prediction[8] that can be integrated within the LQA, the overhead and complexity will certainly no worse than current implementation of dynamic branch prediction.

8 Envisaged Usage and Conclusion

Because of the LQA simplicity, it is now possible to design-in an application dependent LQA core into a simple[9] semicustom ASIC. The other envisaged applications are typically multimedia applications requiring high performance and quick engineering turnaround time in order to capture ever decreasing market windows. The variety of usage of the proposed architecture can be used as an engine for video and audio applications such as [10], [11], [12], [13],[14],[15], [16].

The proposed alternative superscalar scheme is not full featured as the two issues of “multiple use of the same operand” and “dynamic branching” has not been fully described. However, there has been good work done in these two areas and it is possible to integrate the two features into the basic LQA without significant negative effect on clock speed and silicon real estate. The other remaining issue on accessing memory via Bus Interface Unit is similar to other generic architectural interface issue and has no negative impact to the basic LQA.

References

1. M. Franklin and G. S. Sohi, "Register Traffic analysis for streamlining inter-operation communication in fine-grain parallel processors". MICRO 25, page 236–245, Dec 1992.
2. Andre Seznec, "A Path to Complexity-Effective Wide-Issue Superscalar Processors", INRIA Research Report N 4242. ISSN 0249-6339, September 2001
3. Shlomo Weiss, " Implementing Register Interlocks in Parallel-Pipeline Multiple Instruction Queue, Superscalar Processors." Proc. First Int'l Symposium on HPCA 1995: 14–21.
4. G. F. Grohoski, "Machine Organization of the IBM RISC System/6000 Processor", IBM Journal of Research and Development, vol. 34, pp. 37–58, Jan. 1990.
5. Sriram Vajapeyam and Tulika Mitra, "Improving Superscalar Instruction Dispatch and Issue by Exploiting Dynamic Code Sequences". 24th Annual Int'l Symposium on Computer Architecture 1997: 1–12
6. L. Gwennap, "MIPS R10000 uses decoupled architecture", Microprocessor Report, 1994. 8(14), 18–22.
7. Jose-Lorenzo Cruz, Antonio Gonzalez, Mateo Valero, and Nigel Topham, "Multiple-banked register file architectures," Proc. of the 27th International Symposium on Computer Architecture, June 2000.
8. Liangxiao Hu and Y. K. Chan, "A proposal for an Integrated Cost Effective Branch Strategy," Computer Architecture 97 – Selected Papers of the 2nd Australasian Conference, Ronald Pose ED, pp69–80
9. D. Avresky, K. E. Grosspietsch, B. Johnson and F. Limbardi, "Embedded Fault-Tolerant Systems", IEEE Micro, SEPT/OCT 1998. pp8–11.
10. Debin Zhao, M. L. Tai, W. K. Ho and Y. K. Chan, "VLSI Design of a Programmable DCT Engine for Digital Camera", SPIE Proceedings – vol. 3650, pp78–86, 1999.
11. Y. Lu, Debin Zhao and Y. K. Chan, "Design of a wavelet slave processor for audio and video decompression", SPIE Proceedings – vol. 4313, pp86–95, 2001.
12. Debin Zhao, Wen Gao and Y. K. Chan, "Morphological Representation of DCT Coefficients for Image Compression", IEEE Trans. Circuits Syst. Video Technol., vol. 12, no. 9, pp. 819–823, 2002.
13. Debin Zhao, Y. K. Chan and Wen Gao, "Low-Complexity and Low-Memory Entropy Coder for Image Compression", IEEE Trans. Circuits Syst. Video Technol., vol. 11, no. 10, pp. 1140–1145, 2001.
14. Debin Zhao, Y. K. Chan and Gao Wen, "Image Compression", Hong Kong Short Term Patent no. HK1037462 4th January, 2002.
15. Debin Zhao, Y. K. Chan and Gao Wen, "Image Compression", US Patent Application No. 10/161,543 filed on June 3, 2002.
16. Kenneth K. C. Lee, Debin Zhao and Y. K. Chan, "Extending Low Complexity and Low Memory Entropy Coder to Motion Compensated Frames", International Workshop on Advanced Imaging Technology, January, 21–22, 2003. Nagasaki, Japan. Proceeding IWAIT 2003, pp425–428.

A High Efficiency Distributed Mutual Exclusion Algorithm

Dan Liu, Xinsong Liu, Zhijie Qiu, and Gongjun Yan

8010 Division

Department of Computer Science

University of Electronic Science and Technology

Chengdu, Sichuan 610054, P.R.China

liudan@uestc.edu.cn

<http://teacher.uestc.edu.cn/teacher/teacher.jsp?TID=yrewy>

Abstract. A high efficiency Distributed Mutual Exclusion (DMX) algorithm based on RA algorithm, is presented. It puts different mutual exclusion operations for reading and writing requests. The algorithm, belonging to nontoken-based type, saves the message complexity while has T synchronization delay. A read/write globe clock stamp which based on the Lamport clock stamp is put forward for the read/write operations. Using the read/write globe clock stamp, reading and writing requests can access Critical Sections (CS) with fairness. Furthermore, a dynamic detection mechanism is adopted in the algorithm to realize self-stability.

Keywords: distributed mutual exclusion, read/write clock stamp, self-stability

1 Introduction

DMX algorithms have been studied intensively in the last 20 years. Several taxonomic research papers of the algorithms have been published[1,2,3,11,12], which can be grouped into two classes, nontoken-based and token-based. The performance of DMX algorithms is generally measured by two metrics: first, the message complexity, which is the number of messages necessary per CS invocation; second, the synchronization delay, which is the time required after a process leaves the CS and before the next process enters the CS. The token-based algorithms can reduce a message complexity to $O(\log N)$, and their synchronization delay can reduce to $O(\log N)T$. The nontoken-based algorithms generally have a message complexity between N and $3(N-1)$, and have a synchronization delay between T and $2T$.

The proposed algorithm belongs to nontoken-based type. Based on Ricart-Agrawala(RA) algorithm[8], it reduces the message complexity and keeps the same synchronization delay as RA algorithm. The algorithm provides different methods for reading mutual exclusion and writing mutual exclusion to reduce the message complexity. According to the Lamport globe clock stamp, a read/write

globe clock stamp suitable for read/write mutual exclusion, is presented. Sub-numbering for reading requests, is brought in to avoid deadlock and starvation.

Many DMX algorithms try to reduce message complexity [6,7,8], that, maybe lead to a less stable system. So these algorithms are always under the assumption: the processes communicate through the asynchronous message passing over an error-free underlying communication network, while the message transit times may vary. Adopting a reply and timeout mechanism, the proposed algorithm is self-stability[4,5,10]. It can tolerate transient failures of communication network, and be more reliable and practical.

The structure of the paper is arranged as following: Section 2 introduces the system model of *RA* algorithm; Section 3, the proposed algorithm is presented in details. The performance analysis of the algorithm is given in Section 4, and in Section 5 conclusions of the algorithm are provided.

2 RA Algorithms Review

In this section, the general system model of *RA* algorithm, which is the basis of the proposed algorithm, is described.

2.1 System Model

The *RA* algorithm runs under the following system model. There are N nodes in the system, which are connected with a communication network. The system has no shared memory, that, the nodes exchange information only via message transfer. The network guarantees error-free FIFO delivery with bounded message delay.

Two types of messages are exchanged among nodes: *REQUEST* and *REPLY*. While wanting to enter *CS*, a node sends the *REQUEST* messages to other nodes and waits for their *REPLY*. Each *REQUEST* for per *CS* invocation is assigned a priority p , which is implemented by the Lamport globe clock stamp[7]. To achieve fairness and to prevent deadlock and starvation, the *REQUESTs* should be ordered by priority for per *CS* invocation.

Define p as $p=(SN, SiteID)$, where SN is a unique locally assigned sequence number to the request and $SiteID$ is the process identifier. SN is determined as follows. Every node maintains the highest sequence number seen so far in a local variable *Highest-M-Seen*. While making a request, a node uses a sequence number which is one larger than the value of *Highest-M-Seen*. When a *REQUEST* is received, *Highest-M-Seen* is updated as follows.

Highest-M-Seen=Maximum (*Highest-M-Seen*, sequence number in the *REQUEST*)

Priorities of two *REQUESTs*, $p_1=(SN_1, SiteID_1)$ and $p_2=(SN_2, SiteID_2)$ are compared as follows. Priority of p_1 is greater than priority of p_2 if $SN_1 < SN_2$ or ($SN_1 = SN_2$ and $SiteID_1 < SiteID_2$).

Definition 1. S_i and S_j are concurrent if S_i 's *REQUEST* is received by S_j after S_j has made its *REQUEST*, and S_j 's *REQUEST* is received by S_i after S_i has made its *REQUEST*. Where S_i represents node i.

2.2 Ricart-Agrawala Algorithm

Each node S_i uses the following local integer variables: my-seq-m_i , replycount_i , highest-m-seen_i , and also uses the following vectors:

$RD_i[1:N]$ of Boolean. $RD_i[j]$ indicates if S_i has deferred the *REQUEST* sent by S_j

The *RA* algorithm is outlined in Fig.1. The *REPLY* messages sent by a process are blocked only by processes which request the *CS* with higher priority. Thus, when a process sends *REPLY* messages to all deferred requests, the process with the next highest priority request receives the last needed *REPLY* message and enters the *CS*. The execution of *CS* requests in this algorithm is always ordered by decreasing priority. For each *CS* invocation, there are exactly $2(N-1)$ messages: $(N-1)$ *REQUEST*s and $(N-1)$ *REPLY*s.

- 1) Initial local state for node S_i
 - $\text{int my-seq-m}_i = 0$
 - $\text{int replycount}_i = 0$
 - $\text{int array of Boolean } RD_i[j] = 0, \forall j \in \{1\dots M\}$
 - $\text{int highest-m-seen}_i = 0$
- 2) *InvMulEx*: S_i executes the following to invoke mutual exclusion
 - $\text{my-seq-m}_i = \text{highest-m-seen}_i + 1$
 - Make a *REQUEST*(P_i) message: where $P_i = (\text{my-seq-m}_i, i)$
 - Send *REQUEST*(P_i) message to all the other processes
 - $\text{replycount}_i = 0$
 - $RD_i[k] = 0, \forall k \in \{1\dots N\}$
- 3) *RcvReq*: node S_i receives message *REQUEST*(P_j), where $P_j = (SN_j, j)$, from node S_j
 - a. If S_i is requesting then there are two cases
 - S_i 's *REQUEST* has a higher priority than S_j 's *REQUEST*. In this case, S_i sets $RD_i[j] = 1$ and $\text{highest-m-seen}_i = \max(\text{highest-m-seen}_i, SN_j)$
 - S_i 's *REQUEST* has a lower priority than S_j 's *REQUEST*. In this case, S_i sends *REPLY* to S_j
 - b. If S_i is not requesting then send *REPLY* to S_j
- 4) *RcvReply*: node S_i receive *REPLY* from node S_j
 - $\text{replycount}_i = \text{replycount}_i + 1$
 - if (*CheckExecuteCS*) then execute *CS*
- 5) *FinCS*: node S_i finish executing *CS*
 - Send *REPLY* to all nodes S_k , such that $RD_i[k]=1$
- 6) *CheckexecuteCS*
 - If ($\text{replycount}_i = N-1$), then return true else return false

Fig 1. Ricart-Agrawala algorithm

3 The Proposed DMX Algorithm

3.1 Basic Idea

The system model that the proposed algorithm requires is almost the same as the section 2. But it is self-stability, and is able to recover from transient errors by itself. It is realized by lock mechanism. A process must get the lock of a CS before entering it, and release the lock after finishing to execute the CS. Each node S_i runs a lock module which is realized by the proposed algorithm. The lock module has no independent threads, and some skills such as sleeping processes and waking up processes are used in it. The module has two queues. One is Request Queue(RQ), which maintains all the requests that have not been finished. Another is Lock Queue(LQ), which maintains all the locks in the system.

There are two types of net locks, Reading(R) lock and Writing(W) lock. A process requests a R -lock when only executing reading operations in CS, while it requests a W -lock when executing writing operations in CS. No mutual exclusion requirements are required between R -locks. But R -lock and W -lock, W -lock and W -lock need do mutual exclusion operations. A R -lock is recorded at local while a W -lock is recorded at all the nodes in the system.

Four types of messages are used in the proposed algorithm: *REQUEST*, *REPLY*, *CREPLY* and *RELEASE*. The functions of *REQUEST* and *REPLY* are the same as their functions in *RA* algorithm. *CREPLY* serves as a collective reply and *RELEASE* serves as a release request message. Their functions will describe in detail in the following context.

The proposed algorithm refers to the idea of the Lodha, S. and Kshemkalyani, A. (*LK*) algorithm[9] to reduce the message complexity. But *LK* algorithm does not consider that reading and writing requests have different requirements for mutual exclusion. In the proposed algorithm, the idea is as follows. Without loss of generality, S_i represents site i , and P_i represents a process which is running at S_i , and R_i represents a *REQUEST* message which is sent by P_i .

When P_j receives R_i for a lock, at the same time P_j wants the same lock and sends R_j . If R_j has a higher priority, then R_i serves as a reply to P_j and P_i need not send *REPLY* to P_j . If P_j has a lower priority, then R_j serves as a reply to P_i . So, when there are concurrent requests, *REQUEST* messages can serve as *REPLY* to reduce message complexity.

When P_i receives R_j for a lock while it does not want the lock, it sends *REPLY* to P_j . When exiting CS and releasing the lock it has got, P_i will select the next highest priority R_k , which is not replied, from the requests set, and send *CREPLY* to P_k as a collective reply from all processes that had made higher priority requests than R_k . When receiving *CREPLY*, that means all the requests, which with higher priorities than R_k , have finished to access CS, and P_k can delete the requests. From this point, the number of *REPLY* is reduced.

To reduce the message complexity and design complexity, the reading requests are not sent to other nodes. But that will bring some problems as follows. If P_i has only reading requests, then there are only local reading requests and

remote writing requests in RQ_i . Because other nodes have not P'_i 's reading requests, no node will send *CREPLY* to P_i when it releases lock, and the other nodes' requests will exist in RQ_i forever. So these requests in RQ_i will not be processed and their owner will wait for the reply forever. To avoid the case, while exiting a *CS*, P_i sends *RELEASE* messages to the nodes which have not messages in RQ_i , that, means the nodes have not writing request. When receiving the *RELEASE* messages, the nodes can delete corresponding requests R_i from local RQ .

3.2 Read/Write Globe Clock Stamp

When generating a request, the priority of it is also generated. *RA* algorithm uses the Lamport clock stamp to define the priority of a request. For the proposed algorithm, while the reading requests not being sent to all the nodes, some abnormal cases as follows will happen if using the Lamport clock stamp.

Assuming P_i has many continuous reading requests but no writing requests, and other nodes have a few requests. By reading requests not spreading to net, while using Lamport clock stamp, the SN of P_i is large but the other processes in other nodes have small SN . When a writing request is generated on other nodes after P'_i 's reading requests, it's priority may be higher than P'_i 's reading request which is generated earlier. To avoid this, the priority of a request is implemented by the read/write globe clock stamp.

Definition 2. read/write globe clock stamp is defined as $(SN_w, SN_r, SiteID)$, the SN_w is the same as SN in Lamport clock stamp. SN_r is the supplement serial number. All the continuous reading requests between two writing requests have the same SN_w , and they use different SN_r to represent their different priorities.

Priorities of two requests

$p_1=(SN_w1, SN_r1, SiteID_1)$ and $p_2=(SN_w2, SN_r2, SiteID_2)$ are compared as follows.

If $SN_w1 < SN_w2$ or ($SN_w1 = SN_w2$ and $SiteID_1 < SiteID_2$) or ($SN_w1 = SN_w2$ and $SiteID_1 = SiteID_2$ and $SN_r1 < SN_r2$) then $p_1 > p_2$.

3.3 Realization of a Self-Stability System

In order to realize DMX, all the nodes must negotiate through communication network. If a message is abnormal when negotiating, the request process may wait forever and deadlock happens. So, in the proposed algorithm, a reply and timeout mechanism is used to implement the reliability and avoid deadlock.

When sending a message, two timeout clocks are generated, $t_1 = \tau, t_2 = k\tau$.

If a message is sent but does not receives its reply after t_1 , it is considered as lost and will be sent again per t_1 interval, until getting the reply of it or sent for k times. If it has been sent for k times while not getting the reply, the destination node is considered as failed node, and will be deleted from the group. So, the source node will not wait a reply from it. Apparently, transient failures not exceeding t_2 can be tolerated by the mechanism.

When recovering, a node sends initial messages to all other nodes, so the other nodes can add it into the group. Using this mechanism, the system can tolerate transient failures in the communication network and implementation automatic recovering.

3.4 Implementation of the Proposed Algorithm

Declaration and Data Structure. The lock of a CS is marked as $L_{(j,t,m)}$, $t \in \{r,w\}$ is the state of a lock, $t=r$ represents a R -lock which is hold by a reading process, and $t=w$ represents a W -lock which is hold by a writing process. For R -lock, m represents the share number.

To realize fairness, each process has a priority p which is implemented by read/write clock stamp. Assuming P_i wants to access CS_j , it will generate a request R_i , marked as $R_{(i,j,p,t,s)}$, where $t \in \{r,w\}$ and $s \in \{l,n\}$. For a reading request $t=r$ and for a writing request $t=w$. The symbol s represents the state of a request. It may be local block state(l) or net block state(n). L state represents the request having the qualification to get the lock but it will wait until the other local process, which has hold the lock, to release it, and n state represents that the request is waiting for the nodes negotiating to permit the qualification to get the lock.

A reading request $R_{(i,j,p,r,l)}$ is not sent to net, so $s=l$. $R_{(i,j,p,r,l)}$ is inserted into RQ_i . If there is no other concurrent request for the lock of CS_j , the requester can hold the lock. Otherwise while the lock released by other process and $R_{(i,j,p,r,l)}$ becoming the highest priority request, the requester will hold the lock of CS_j .

A writing request $R_{(i,j,p,w,n)}$ must be sent to all the nodes in the system. Firstly, $R_{(i,j,p,w,n)}$ is inserted into RQ_i and waits for net negotiating to permit the qualification to get the lock. When all the nodes approve the requester to get the lock, the requester can get the lock of CS_j if there is no other higher priorities requests for the lock, otherwise changing $R_{(i,j,p,w,n)}$ to $R_{(i,j,p,w,l)}$ and waiting for the lock being freed. When the lock is released and $R_{(i,j,p,w,l)}$ becomes the highest request in RQ_i , the requester will hold the lock of CS_j .

For $REPLY$ or $CREPLY$, marked as $REP_{(i,j,p)}$ or $CREP_{(j,p)}$, where i represents the source process, and j represents to request the lock of CS_j , and p represents the priority of corresponding request being replied. When receiving $REP_{(i,j,p)}$, the receiver process checks whether all the replies of the corresponding requests are received. If so, it changes the corresponding request's state from n to l . When receiving $CREP_{(j,p)}$, the receiver deletes all the requests whose priorities are higher than p from RQ and changes the corresponding request's state from n to l .

$RELEASE$ is marked as $REL_{(i,p)}$, where i represents the source process, p represents the priority of corresponding request. When receiving $REL_{(i,p)}$, a process deletes the corresponding request.

Algorithm Implementation. The proposed algorithm is composed of applying R -lock, releasing R -lock, applying W -lock, releasing W -lock, receiving

REQUEST, receiving *REPLY*, receiving *CREPLY*, receiving *RELEASE* procedures.

Define the requests set for CS_j in RQ_i as $\Omega_j = \{R \mid R \in R_{(k,j,p,t,s)}, k \in [1\dots N]\}$.

Define the nodes set $\omega_j = \{S_k \mid S_k \text{ is active}, k \in [1\dots N]\}$.

(1) Applying R-lock. While wanting to enter CS_j and doing reading operations, P_i applies *R-lock*.

- 1. Generates $R_{(i,j,p,r,l)}$ and inserts it into RQ_i .
- 2. If $L_{(j,t,m)}$ exists in LQ_i goes to 3, else goes to 6.
- 3. If $L_{(j,t,m)}$ is *W-lock*, blocks P_i on $R_{(i,j,p,r,l)}$, when it is waked up, goes to 2.
- 4. If a request $R_{(k,j,p',w,l)}$ exists in RQ_i where $\forall k \in [1\dots N]$, and $p' > p$, then blocks P_i on $R_{(i,j,p,r,l)}$, when it is waked up, goes to 4.
- 5. $m=m+1$, P_i holds the *R-lock*, return.
- 6. If a request $R_{(k,j,p',w,l)}$ exists in RQ_i , where $k \in [1\dots N]$, and $p' > p$, then blocks P_i on $R_{(i,j,p,r,l)}$, when it is waked up, goes to 6.
- 7. If there is no lock $L_{(j,r,m)}$ in LQ_i , generates $L_{(j,r,m)}$, P_i holds $L_{(j,r,m)}$, return.

(2) Releasing R-lock. While leaving CS_j which is entered by reading mode, P_i will release *R-lock*

- 1. When P_i releases $L_{(j,r,m)}$, deleting the corresponding request $R_{(i,j,p,r,l)}$ from RQ_i , $m=m-1$, if $m \neq 0$ then return.
- 2. If $\Omega_j = \phi$ then deletes $L_{(j,r,m)}$ from LQ_i , return.
- 3. Get the highest priority $R_{(k,j,p',w,s)}$ from Ω_j , while it is a writing request, if $k=i$ then goes to 4 else goes to 5.
- 4. Changes $L_{(j,r,m)}$ to $L_{(j,w,m)}$, and wakes up the blocked process on it, return.
- 5. Changes $L_{(j,r,m)}$ to $L_{(j,w,m)}$ and sends $CREP_{(i,p')}$ as the reply of $R_{(k,j,p',w,n)}$ to P_k .

(3) Applying W-lock. While wanting to enter CS_j and doing writing operations, P_i will apply *W-lock*

- 1. Generates $R_{(i,j,p,w,n)}$.
- 2. Sends $R_{(i,j,p,w,n)}$ to the nodes set ω_j except local node, and inserts it into RQ_i , blocked P_i on $R_{(i,j,p,w,n)}$. When it is waked up, goes to 3.
- 3. P_i gets $L_{(j,w,m)}$, return.

(4) Releasing W-lock. While leaving CS_j which is entered by writing mode, P_i will release *R-lock*

- 1. When P_i releases $L_{(j,w,m)}$, the corresponding $R_{(i,j,p,w,l)}$ is deleted from RQ_i . If $\Omega_j = \phi$ then deletes $L_{(j,w,m)}$ from LQ_i and goes to 6, else goes to 2.
- 2. Getting the highest priority request $R_{(k,j,p',t,s)}$ from Ω_j , if $k = i$ then goes to 3, else goes to 5.
- 3. If it is writing request $R_{(k,j,p',w,l)}$, wakes up the process blocked on it, return, else goes to 4.

- 4. If no writing request in Ω_j , changes $L_{(j,w,m)}$ to $L_{(j,r,m)}$, wakes up all the blocked processes blocked on the reading requests, goes to 6. If there is writing request in Ω_j , supposing the highest priority writing request is $R_{(k',j,p'',w,s)}$, then changes $L_{(j,w,m)}$ to $L_{(j,r,m)}$, and wakes up all the processes which are blocked on the reading requests whose priority $p > p''$, goes to 6.
- 5. Sends $CREP_{(i,p')}$ as the reply of $R_{(k,j,p',w,n)}$ to P_k
- 6. Selects the nodes $S_k (k \neq i)$ who has no request in RQ_i .
- 7. Sends $REL_{(i,p)}$ to S_k , return.

(5) Receiving REQUEST

- 1. When P_j receives $R_{(i,j,p,w,n)}$, inserts it into RQ_j .
- 2. If there is no local request $R_{(j,j,p',t,s)}$ in RQ_j , P_j sends $REP_{(j,p)}$ to P_i as a reply to $R_{(i,j,p,w,n)}$, return.
- 3. Assuming $R_{(j,j,p',t,s)}$ is the highest priority request in Ω_j , if $p' < p$ then goes to 4 else goes to 5.
- 4. If $t=w$, it means that $R_{(j,j,p',t,s)}$ has been sent to P_i , this message should be regard as the reply, return. If $t=r$, sends $REP_{(j,p)}$ message to P_i as the reply to $R_{(i,j,p,w,n)}$, return.
- 5. If $t=w$, $R_{(i,j,p,w,n)}$ will be regarded as the reply from P_i to $R_{(j,j,p',t,s)}$, goes to (6). If $t=r$, return.

(6) Receiving REPLY or CREPLY

- P_i receives $CREP_{(j,p)}$
 - 1. Sets the receiving mark that $REP_{(i,j,p)}$ has been received.
 - 2. If P_i receives $REP_{(k,j,p)} (1 \leq k \leq N)$ from all other nodes then changes $R_{(i,j,p,w,n)}$ to $R_{(i,j,p,w,l)}$, goes to next step, otherwise return.
 - 3. If $R_{(i,j,p,w,l)}$ is not the highest priority request in Ω_j , return.
 - 4. Generates $L_{(j,w,m)}$, inserts it into LQ_i , wakes up the processes blocked on $R_{(i,j,p,w,l)}$, return.
- P_i receives $CREP_{(j,p)}$
 - 1. Removes $R_{(k,j,p',w,l)}$ from Ω_j whose priority $p' > p, k \in [1...N]$.
 - 2. Changes $R_{(i,j,p,w,n)}$ to $R_{(i,j,p,w,l)}$.
 - 3. If $R_{(i,j,p,w,l)}$ is the highest priority request in Ω_j , then generates $L_{(j,w,m)}$, inserts it into LQ_i , wakes up processes blocked on $R_{(i,j,p,w,l)}$, return.
 - 4. Generates $L_{(j,r,m)}, m=0$, inserts it into LQ_i , wakes up the processes blocked on the $R_{(i,j,p',r,l)}$ whose priority $p' > p$, return.

(7) Receiving RELEASE

- 1. While receiving $REL_{(j,p)}$, P_i removes $R_{(i,j,p,w,n)}$ from RQ_i .
- 2. If $\Omega_j = \emptyset$, deletes $L_{(j,t,m)}$ from LQ_i , return, else gets the highest priority request $R_{(k,j,p',t,s)}$ from Ω_j , if $k=i$ goes to next step, else return.
- 3. As assumption, $R_{(k,j,p',t,s)}$ must be reading request, so generates $L_{(j,r,m)}, m=0$, inserts it into LQ_i . If no writing request in Ω_j , wakes up all the processes blocked on the messages in RQ_i , return. Otherwise gets the highest priority write request $R_{(k',j,p'',w,s)}$ from Ω_j , wakes up all the processes blocked on the messages whose priority is higher than $R_{(k',j,p'',w,s)}$, return.

Fig 2. The proposed algorithm

4 A Performance Analysis

Definition 3. P_i and P_j are concurrent if R_i is received by P_j after P_j has made R_j , and R_j is received by P_i after P_i has made R_i . $CSet_i = \{R_j \mid R_j \text{ is concurrent with } R_i\} \cup \{R_i\}$.

Traditionally, the performance of DMX algorithms is compared on the basis of synchronization delay and the message complexity. In the proposed algorithm, the synchronization delay is an average message delay which is the same as that of RA algorithm, while message complexity is reduced.

The proposed algorithm needs additional communication spending to realize self-stabilizing. In order to compare the performance at the same condition, the proposed algorithm's performance is evaluated without considering the communication spending to realize self-stabilizing.

1) For writing requests

The request process P_i will send $(N-1)$ requests $R_{(i,j,p,w,n)}$, and receives $(N - |CSet_i|)$ replies.

– $|CSet_i| \geq 2$, there are two cases here

(1) There is at least one writing REQUEST whose $p' < p$ in $CSet_i$, and the number of nodes who only has reading REQUEST is k . So, P_i will send one CREPLY and k RELEASE messages. The message complexity is $2N - |CSet_i| + k$. When all REQUESTs are concurrent, there are $N + k (0 \leq k < N)$ messages.

(2) There is no writing request whose $p' < p$ in $CSet_i$, and the number of nodes which only have reading request is k . So, P_i will not send CREPLY. The message complexity is $2N - 1 - |CSet_i| + k$. When all requests are concurrent, there are $N + k - 1 (0 \leq k < N)$ messages.

– $|CSet_i| = 1$. This is the worst case, implying that all requests are serialized. In this case the message complexity is $2(N-1)$, same as RA algorithm.

2) For reading requests

The requester will generate reading request R_i but not send it to other nodes, the message complexity is as follows.

– $|CSet_i| \geq 2$, there are two cases here

(1) There is at least one writing request whose $p' < p$ in $CSet_i$. So, P_i will send one CREPLY. The message complexity is 1.

(2) There is no writing request whose $p' < p$ in $CSet_i$. So P_i will not send CREPLY. The message complexity is 0.

– $|CSet_i| = 1$. Imply that all requests are serialized. In this case the message complexity is 0.

5 Conclusions

In this paper, a high efficiency and self-stabilized DMX lock algorithm is presented. It can recover from transient failures of the system. Unlike most DMX

algorithms, different DMX methods for reading and writing operations, is used to reduce the message complexity and system design complexity. A new globe clock stamp—read/write clock stamp, which is suitable for the proposed algorithm, is presented. The algorithm is proved to be high-performance by performance analysis.

References

1. Y.-I. Chang, "A Simulation Study on Distributed Mutual Exclusion" J. Parallel and Distributed Computing, vol. 33,pp. 107–121, 1996
2. M.Singhal, "A taxonomy of Distributed Mutual Exclusion" J.Parallel and Distributed Computing, vol. 18, no.1, pp.94–101, May 1993
3. Raudal C.Burns. "Semi-Preemptible Locks for a Distributed File System" Performance, Computing, and Communications Conference, 2000. IPCCC'00. Conference Proceeding of the IEEE International , 2000 pp. 397–404
4. Mizuno, M.; Nesterenko, M.; Kakugawa, H. "Lock-based self-stabilizing distributed mutual exclusion algorithms" Distributed Computing Systems, 1996, Proceedings of the 16th International Conference on, 1996, pp. 708–716
5. Dijkstra, E.W. "Self-stabilizing systems in spite of distributed control" Communications of the ACM, 17(11),pp. 643–644, November 1974.
6. O. Carvalho and G. Roucairol, " On Mutual Exclusion in Computer Networks" Technical Correspondence, Comm. ACM, vol. 26, no. 2, pp. 146–147, Feb. 1983.
7. L. Lamport, Time, "Clocks and the Ordering of Events in Distributed Systems" Comm. ACM, vol. 21, no. 7, pp. 558–565, July 1978.
8. G. Ricart and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks" Comm. ACM, vol. 24, no. 1, pp. 9–17, Jan. 1981.
9. Lodha, S.; Kshemkalyani, A. "A fair distributed mutual exclusion algorithm" Parallel and Distributed Systems, IEEE Transactions on , Volume. 11, Issue. 6 , June 2000, pp. 537–549
10. Gouda, M.G.; Multari, N.J. "Stabilizing communication protocols" Computers, IEEE Transactions on , Volume.40, Issue.4 , April 1991, pp. 448–458
11. Sanders, B. "The information structure of distributed mutual exclusion algorithms" ACM Transactions on Computer Systems, 5(3),pp.284–299, 1987
12. Raynal,M. "A simple taxonomy for distributed mutual exclusion algorithm." ACM Operating Systems Review, 25(2),pp.47–50, 1991

The Security Architecture of the Java Operating System JX – A Security Architecture for Distributed Parallel Computing

Christian Wawersich¹, Meik Felser¹, Michael Golm², and Jürgen Kleinöder¹

¹ Dept. of Computer Science 4 (Distributed Systems and Operating Systems)
Martensstr.1, 91058 Erlangen, Germany

{felser,wawersich,kleinoder}@informatik.uni-erlangen.de

² Siemens AG - Corporate Technology (CT SE 2)
Otto-Hahn-Ring 6, D-81730 Munich, Germany
michael.golm@siemens.com

Abstract. Using the unneeded computation power in the internet for distributed computing is getting more and more eligible. To increase the willingness to provide unneeded computing power, a secure platform is needed for the execution of untrusted code. We present the architecture of the JX operating system, which can be used to safely execute untrusted code. The problem of erroneous agents crashing the system is solved by using Java – a typesafe language – as implementation language. The resource consumption of the agents is controlled by a security manager, that inspects every interaction between an agent and a system service. If the security policy does not approve the use of a system service, the access can be denied. An agent execution system build upon JX is presented to illustrate the security problems occurring and the solutions provided by the operating system JX.

1 Introduction

More and more workstations are connected to the internet. The typical applications do not use the full capacity of these machines. This effect is increased by the improvements in computer hardware. Some projects (e.g. seti@home) try to use these unneeded computing power for distributed computing. In a generalized way a personal computer may provide its unused computing power to the net. Everybody who wants to use it can assign a work package (called *agent*) to this computer. The owner or administrator of the machine does not need to trust the agent and must, nevertheless, be sure that the agent does not interfere with the normal operation of the system. For wider acceptance of this model of distributed computing we need a secure execution platform. This platform must assure that erroneous or malicious agents do not crash the system or consume all available resources. If this can be guaranteed the field of application can be even extended to more critical machines in the net, e.g. web servers that do not use their full capacity.

We provide JX a java-based operating system as an execution platform for untrusted code. Java allows developing applications using a modern object-oriented style, emphasizing abstraction and reusability. On the other hand many security problems have been detected in Java systems in the past [7]. The main contribution of this paper is an architecture for a secure Java operating system that avoids Java security problems, such as the huge trusted class library and reliance on stack inspection.

We follow Rushby [23] in his reasoning that a secure system should be structured as if it were a distributed system. With such an architecture a security problem in one part of the system does not automatically lead to a collapse of the whole system's security. Microkernel systems, especially systems that adhere to the multi-server approach, such as SawMill [13], and mediate communication between the servers [17] are able to limit the effect of security violations.

The JX system combines the advantages of a multi-server structure with the advantages of type-safety. It uses type-safety to provide an efficient communication mechanism that completely isolates the servers with respect to data access and resource usage. Code of different trustworthiness or code that belongs to different principals can be separated into isolated domains. Sharing of information or resources between domains can be completely controlled by the security kernel.

The paper is structured as follows. Section 2 gives an overview about Java security. The JX system architecture is described in Section 3. Section 4 completes the architectural overview with focus on security relevant aspect. An agent execution server is presented in Section 5 to illustrate the use of the previously presented security mechanisms. Section 6 describes related work and Section 7 concludes the paper.

2 Java Security

Java security is based on the concept of a sandbox, which relies on the type-safety of the executed code. Untrusted but verified code can run in the sandbox and can not leave the sandbox to do any harm. Every sandbox must have a kind of exit or hole, otherwise the code running in the sandbox can not communicate results or interact with the environment in a suitable way. These holes must be clearly defined and thoroughly controlled. The holes of the Java sandbox are the native methods.

To control these holes, the Java runtime system first controls which classes are allowed to load a library that contains native code. These classes must be trusted to guard access to their native methods. The native methods of these classes should be non-public and the public non-native methods are expected to invoke the SecurityManager before invoking a native method. The SecurityManager inspects the runtime call stack and checks whether the caller of the trusted method is trusted. The stack inspection mechanism only is concerned with access control. It completely ignores the availability aspect of security. This lack was addressed in JRes [6]. Also, Java is perceived as inherently insecure due to the complexity of its class libraries and runtime system [9].

JX avoids this problem by not trusting the JDK class library. No user defined native methods are allowed and the system services are implemented in Java

accessible with a fast RMI-like communication mechanism as described in the following Section.

3 JX System Architecture

3.1 Domains

JX is a single address space system and protection is based on the type-safety of the Java bytecode instruction set. A small microkernel contains low-level hardware initialization code and a minimal Java Virtual Machine (JVM).

The JX system is structured into domains (see figure 1). Each domain represents the illusion of an independent JVM. A domain has a unique ID, its own heap including its own garbage collector, and its own threads. Thus domains are isolated with respect to CPU and memory consumption. They can be terminated independently from each other and the memory that is reserved for the heap, the stack and domain control structures can be released immediately when the domain is terminated. The microkernel represents itself also as a domain. This domain has the ID 0 and is called DomainZero. DomainZero contains all C and assembler code that is used in the system, all other domains execute 100% Java code. JX does not support native methods and there is no trusted Java code that must be loaded into a domain. There is no trust boundary within a domain which eases administration. Because the domain contains no trusted code it is a sandbox that is completely closed. For communication with other Domains we create new holes by introducing *portals*.

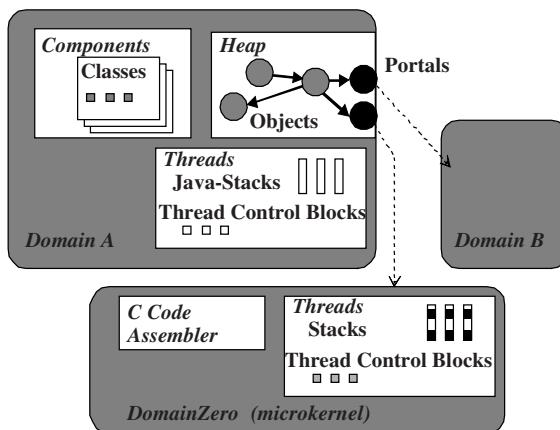


Fig. 1. Structure of the JX system

3.2 Portals

The portal mechanism is used for communication between different Domains, similar to RMI which is used for communication between different JVMs. Portals are proxies

[25] for a service that runs in another domain. Portals look like ordinary objects and are located on a domains heap, but the invocation of a method synchronously transfers control to the service that runs in another domain. Parameters are copied from the client to the server domain.

Portals and services are automatically generated during portal communication. When a domain wants to provide a service it can define a portal interface, which must be a subinterface of `jx.zero.Portal`, and a class that implements this interface. When an instance of such a class is passed to another domain the portal invocation mechanism creates a service in the source domain and a portal in the destination domain. Each domain possesses an initial portal; a portal to a naming service. Using this portal the domain can obtain other portals to access more services. When a domain is created, the creating domain can pass the naming portal as a parameter of the domain creation call.

3.3 Scheduler

CPU scheduling in JX is handled in two levels, which both can be implemented in Java. The first level is the global scheduler that decides which domain is allowed to use the CPU. Since this scheduler controls the CPU, it is a critical part of the system and it must be trusted. When a domain is selected by the global scheduler, a domain-local scheduler is activated. The task of this scheduler is to distribute the allocated CPU time among the threads of its domain. Being only responsible for one domain, the schedulers of the second scheduling level need not to be trusted. If they are malicious, they can only harm their own domain, other domains are uninfluenced concerning CPU time.

4 JX Security Architecture

4.1 JX as a Capability System

The portals, used in JX for inter-domain communication, are capabilities [8]. A domain can only access other domains when it possesses a portal to a service of the other domain. The operations that can be performed with the portal are listed in the portal interface. Although the capability concept is very flexible and solves many security problems, such as the confused deputy [14], in a very natural way, it has well known limitations. The major concern is that a capability can be used to obtain other capabilities, which makes it difficult, if not impossible, to enforce confinement [3].

4.2 The Reference Monitor

JX as described up to now can not enforce confinement. Thus an additional mechanism is needed: a reference monitor that is able to check all portal invocations and can thereby ensure a user defined security policy. A reference monitor must be tamper-proof, mediate all accesses, and be small enough to be verified. A reference monitor for JX must at least control incoming and outgoing portal calls.

We modified the microkernel to invoke a reference monitor when a portal call invokes a service of the monitored domain (inbound) and when a service of another domain is invoked via a portal (outbound). The internal activity of a domain is not controlled. The same reference monitor must control inbound and outbound calls of a domain, but different domains can use different monitors. A monitor is attached to a domain when the domain is created. When a domain creates a new domain, the reference monitor of the creating domain is asked to attach a reference monitor to the created domain. Usually, it will attach itself to the new domain but it can – depending on the security policy – attach another reference monitor or no reference monitor at all.

It must be guaranteed, that while the access check is performed, the state to be checked can only be modified by the reference monitor. For these reasons the access check is performed in a separate domain, not in the caller or callee domain. The reference monitor must have a consistent view of the past parameters. One way is to freeze the whole system by disabling interrupts during the portal call. This would work only on a uniprocessor, would interfere with scheduling, and allow a denial-of-service attack. Therefore, our current implementation copies all parameters from the client domain to the server domain up to a certain per-call quota. These objects are not immediately available to the server domain, but are first checked by the security manager. When the security manager approves the call the normal portal invocation sequence proceeds.

4.3 Access Decision Based on Intercepted IPC

Spencer et al. [26] argue that basing an access decision only on the intercepted IPC between servers forces the security server to duplicate part of the object server's state or functionality. We found two examples of this problem. In UNIX-like systems access to files in a filesystem is checked when the file is opened. The security manager must analyze the file name to make the access decision, which is difficult without knowing details of the filesystem implementation and without information that is only accessible to the filesystem implementation. The problem is even more obvious in a database system that is accessed using SQL statements. To make an access decision the reference monitor must parse the SQL statement. This is inefficient and duplicates functionality of the database server.

There are three solutions for these problems:

1. The reference monitor lets the server proceed and only checks the returned portal (the file portal for example).
2. The server explicitly communicates with the security manager when an access decision is needed.
3. Design a different interface that simplifies the access decision.

Approach (1) may be too late, especially in cases where the call modified the state of the server. Approach (2) is the most flexible solution. It is used in Flask with the intention of separating security policy and enforcement mechanism [26]. The main problem of this solution is, that it pollutes the server implementation with calls to the security manager. This makes approach (3) the most promising approach. Our two example problems would be solved by parsing the path in the client domain. In an analogous manner the SQL parser is located in the client domain and a parsed representation is passed to the server domain and intercepted by the security manager.

This has the additional advantage of moving code to an untrusted client, eliminating the need to verify this code.

4.4 Controlling Portal Propagation

In [18] Lampson envisioned a system in which the client can determine all communication channels that are available to the server *before* talking to the server. In JX this is equivalent to a security manager that knows all portals that are owned by a domain. As we can not enforce a domain to be *memoryless* [18], we must also control the future communication behaviour of a domain to guarantee the confinement of information passed to the domain.

Several alternative implementations can be used to find all portals of a domain or to keep track of them, respectively:

1. A simple approach is to scan the complete heap of the domain for portal objects. Besides the expensive scanning operation, the security manager is not sure, that the domain will not obtain portals in the future.
2. Another approach is to install an outbound interceptor to observe all outgoing communication of the domain. Thus a domain is allowed to possess a critical portal but the reference monitor can reject its use. The performance disadvantage is that the complete communication must be checked, even if the security policy allows unrestricted communication with a subset of all domains.
3. The best approach is a security manager that checks all portals transferred to a domain. This can be achieved by installing an inbound interceptor which inspects all data given to a domain and traverses the parameter object graph to find portals. But this operation is expensive if a parameter object is the root of a large object graph. During copying of the parameters to the destination domain, the microkernel already traverses the whole object graph. Therefore it is easy to find portals during this copying operation and the kernel is able to inform the security manager, that there is a portal passed to the domain. Then the security manager decides whether the portal will be created or not according to the security policy. The security manager must also be informed if a portal is released. This way the reference monitor is able to keep track of the portals a domain actually possesses.

Confinement can now be guaranteed with two mechanisms that can be used separately or in combination: 1. the control of portal communication and 2. the control of portal propagation.

4.5 Principles

A security policy uses the concept of a *principal* [8] to name the subject that is responsible for an operation. The principal concept is not known to the JX microkernel. It is an abstraction that is implemented by the security system outside the microkernel, while the microkernel only operates with domains. Mapping a domain ID to a principal is the responsibility of the security manager. We implemented a security manager which uses a hash table to map the domain ID to the principal object. Once the principal is known, the security manager can use several policies for the access decision, for example based on a simple identity or based on roles [11].

The principal information can also be used for the local scheduling decision of a service domain. In our capability based file system interface, for example, each open file is represented by a portal. Each request (reading from or writing to a file) results in the activation of a service thread in the file system domain. But these threads do not immediately get CPU time. The local scheduler of each service domain decides which thread is allowed to run and thus controls which request is executed first. As described in the previous paragraph, each portal call and therefore also each service thread is associated with the principal information of the calling domain. The scheduling decision can be based on this information.

5 Example of Use: Agent Execution Server

We configure the system to act as an agent execution platform to illustrate how our security architecture works in a real system. A agent execution server should be able to execute code that was previously sent to him by a client. In detail the scenario is shown in figure 2.

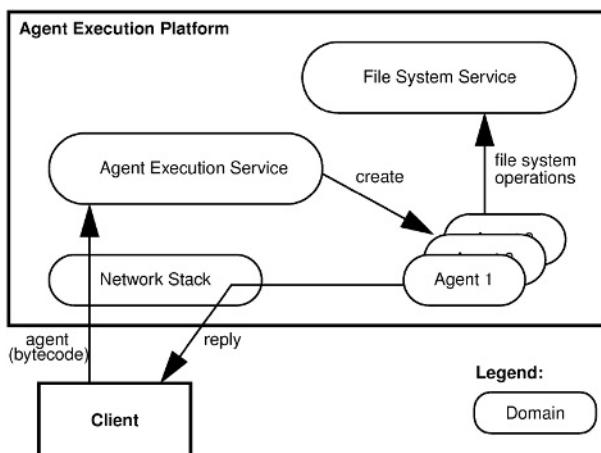


Fig. 2. Agent Execution Platform

A client transfers a component containing bytecode to the server. There the code is verified and compiled to nativecode. Afterwards a new domain is created and the received component is installed inside. Executing the component in a new domain is necessary to protect the system from erroneous or malicious code. Additionally an appropriate security manager must be attached to the domain to restrict the usage of resources and services.

In the following we use an agent implementation, that searches data on the file system. In a remote file system this requires that each file that should be examined must be transferred over the network. Using an agent reduces the network traffic, because it can move to the host that contains the file system and perform local file system operations. In this scenario we have to use a reference monitor that checks

each file system operation. At the same time we have to control the CPU time each client is allowed to use.

Security Manager. We use a security manager, that controls the access to the file system service. Since our file system implementation is accessed using a capability based interface, we only have to check whether a domain (agent) is allowed to get a file capability or not. To accomplish this, the reference monitor is activated when a portal is passed between two domains (see Section 4.4). It reads the principal ID of the receiving domain and determines the portal type to know whether it is a file portal or not. If it is a file portal, the portal type also informs about whether it is for read-only access or for modifying access. On the basis of this information the security policy can be applied. In our example file operations of agents were restricted to a predefined group of files whereas local applications were allowed to access all files. Similar to this scenario any other resource, provided by a service of the agent execution platform, can be controlled.

Scheduling. We also have to restrict the time an agent is allowed to use the CPU. Since we installed each agent in a separate domain this is the task of the global scheduler. We implemented a stride scheduler [28]. The stride scheduler distributes the available CPU time to the domains accordingly to the amount of *tickets* they have. The tickets are described as the right to use a resource. The more tickets a domain has the more often it can use the CPU. Every time a new agent is installed, a predefined amount of tickets is assigned to the agent. If there are no tickets left, we can either stop the execution of new agents or we can redistribute the available tickets between the running agents, thus the maximal CPU time for each agent is reduced.

If an agent uses a service it is desirable that the time a service spends for an agent is charged to the agent's CPU time account. This demands a cooperation between the global scheduler and the scheduler of the service domain hence we have to trust the local scheduler of the service. When the local scheduler is activated, it extracts a list of domains that have called his service from the list of runnable service threads. Then it asks the global scheduler which of these domains should be activated accordingly to the stride scheduling strategy. The service thread which handles the request of this domain is then activated. The time used by this thread can be charged to the client domain's account.

For most services this does not allow an exact accounting of this time because there are often resources that are shared between all service clients. For example the buffer cache management in a file system service. A buffer cache is necessary for an efficient service but the time used to manage it can not be accounted to a single service thread. If we want an exact accounting we have to use a separate file system service instance for each agent, but in most situations it will be enough to have a separate service instances for a groups of agents.

6 Related Work

Capability-Based Systems. Several operating systems are based on capabilities and use three different implementation techniques: partitioned memory, tagged memory

[10], and password capabilities. Early capability systems used a tagged memory architecture (Burroughs 5000 [22], Symbolics Lisp Machine [21]), or partitioned memory in data-containing and capability-containing segments (KeyKOS [12] and EROS [24]). To become hardware-independent, password capabilities [1] have been invented and are used in several systems (Mungi [16], Opal [4], Amoeba [27]).

Type-safe instruction sets, such as the Java intermediate bytecode, are a fourth way of implementing capabilities. The main advantages of this technique are that it is hardware-independent, capability verification is performed at load time, access rights are encoded in the capability type and not stored as a bitmap in the capability, and capabilities can not be transferred over uncontrolled channels.

Virtual Machines. Virtual machines can be used to isolate systems that share the same hardware. The classic architecture is the IBM OS/360 [20]. Virtual machines experienced a recent revival with the VMWare PC emulator [29]. VMs only work at a large granularity. VMWare instances consume a lot of resources to emulate a complete PC which makes it impossible to create fine-grained domains. Most applications require controlled information flow between classification levels; that is between VMWare instances. A virtual machine realizes a sandbox. The holes of the VMWare sandbox are the emulated devices. Thus, communication is rather expensive and stating a security policy in terms of an emulated device may be a difficult task.

Java Security. The J-Kernel [15] implements a capability architecture for Java. It is layered on top of a JVM, with the problems of limited means of resource control. It uses classloaders to separate types. The capability system is not orthogonal to application code which makes reuse in a different context difficult.

The MVM [5], and KaffeOS [2] are systems that isolate applications that run in the same JVM. The MVM is an extension of Sun's HotSpot JVM that allows running many Java applications in one JVM and give the applications the illusion of having a JVM of their own. There are no means for resource control and no fast communication mechanisms for applications inside one MVM. KaffeOS is an extension of the Kaffe JVM. KaffeOS uses a process abstraction that is similar to UNIX, with kernel-mode code and user-mode code, whereas JX is more structured like a multi-server microkernel system. There needs to be no trusted Java code in JX. Communication between processes in KaffeOS is done using a shared heap. Our goal was to avoid sharing between domains as much as possible and we, therefore, use RPC for inter-domain communication.

7 Conclusion

We described the security architecture of the Java operating system JX, which can be used as a secure agent execution platform for distributed computation. The security concept of JX consists of language-based protection and operating system protection. Typical Java security problems, such as native methods, execution of code of different trustworthiness in the same thread, and a huge trusted class library are avoided.

JX provides a number of security mechanisms of different invasiveness. The capability mechanism is inherent in the architecture and guarantees a minimal level of security. On a per-domain basis this mechanism can be supplemented by a monitor that controls propagation of capabilities between domains and, if necessary, a reference monitor that mediates access to these capabilities.

References

1. M. Anderson, R. Pose, C. S. Wallace. A password-capability system. In: *The Computer Journal*, 29, (1986) 1–8.
2. G. Back, W. C. Hsieh, J. Lepreau. Processes in KaffeOS: Isolation, Resource Management, and Sharing in Java. In: *Proc. of 4th Symposium on Operating Systems Design & Implementation*, (2000).
3. W. E. Boebert. On the inability of an unmodified capability machine to enforce the *-property. In: *Proc. of the 7th DoD/NBS Computer Security Conference*, (1984) 291–293.
4. J. S. Chase, H. M. Levy, M. J. Feeley, E. D. Lazowska. Sharing and Protection in a Single Address Space Operating System. In: *ACM Trans. on Computer Systems*, 12(4), (1994) 271–307.
5. G. Czajkowski, L. Daynes. Multitasking without Compromise: A Virtual Machine Evolution. In: *Proc. of the OOPSLA01*, (2001) 125–138.'
6. G. Czajkowski, T. von Eicken. JRes: A Resource Accounting Interface for Java. In: *Proc. of Conference on Object-Oriented Programming Systems, Languages, and Applications* 98, ACMPress, (1998) 21–35.'
7. D. Dean, E. W. Felten, D. S. Wallach, D. Balfanz, P. J. Denning. Java security: Web browsers and beyond. In: D. E. Denning (ed.) *Internet Besieged: Countering Cyberspace Scofflaws*. ACM Press, (1998) 241–269.
8. J. B. Dennis, E. C. Van Horn. Programming Semantics for Multiprogrammed Computations. In: *Communications of the ACM*, 9(3), (1966) 143–155.
9. L. v. Doorn. A Secure Java Virtual Machine . In: *Proc. of the 9th USENIX Security Symposium* , (2000) 19–34.
10. R. S. Fabry. Capability-based addressing . In: *Communications of the ACM*, 17(7), (1974) 403–412 .
11. D. Ferraiolo, R. Kuhn. Role-based access controls. In: *Proc. of the 15th National Computer Security Conference*, (1992) 554–563.
12. B. Frantz. KeyKOS – a secure, high-performance environment for S/370. In: *Proc. of SHARE 70*, (1988) 465–471.
13. Gefflaut, T. Jaeger, Y. Park, J. Liedtke, K. Elphinstone, V. Uhlig, J.E. Tidswell, L. Deller, L. Reuther. The SawMill Multiserver Approach. In: *Proc. of the 9th SIGOPS European Workshop*, (2000).
14. N. Hardy. The confused deputy. In: *Operating Systems Review*, 22(4), (1988) 36–38.
15. C. Hawblitzel, C.-C. Chang, G. Czajkowski, D. Hu, T. v. Eicken. Implementing Multiple Protection Domains in Java. In: *Proc. of the USENIX Annual Technical Conference*, (1998) 259–270.
16. G. Heiser, K. Elphinstone, S. Russel, J. Vochteloo. Mungi: A Distributed Single Address-Space Operating System. In: *17th Australiasian Computer Science Conference*, (1994) 271–280.
17. T. Jaeger, J. Tidswell, A. Gefflaut, Y. Park, J. Liedtke, K. Elphinstone. Synchronous IPC over Transparent Monitors. In: *9th SIGOPS European Workshop*, (2000).
18. B. W. Lampson. A Note on the Confinement Problem. In: *Communications of the ACM*, 16(10), (1973) 613–615.

19. P. Loscocco, S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. In: Usenix 2001 Freenix Track, (2001).
20. G. Mealy, B. Witt, W. Clark. The Functional Structure of OS/360. In: IBM Systems Journal, 5(1), (1966) 3–51.
21. D. A. Moon. Symbolics Architecture. In: IEEE Computer, 20(1), IEEE, (1987) 43–52.
22. E. I. Organick. Computer System Organization: The B5700/B6700 Series. Academic Press, Inc., New York, (1973).
23. J. Rushby. Design and Verification of Secure Systems. In: Proc. of the 8th Symposium on Operating System Principles, (1981) 12–21.
24. J. S. Shapiro, J. M. Smith, D. J. Farber. EROS: a fast capability system. In: Symposium on Operating Systems Principles, (1999) 170–185.
25. M. Shapiro. Structure and Encapsulation in Distributed Systems: The Proxy Principle. In: ICDCS 1986, (1986) 198–204.
26. R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Anderson, J. Lepreau. The Flask Security Architecture: System Support for Diverse Security Policies. In: Proc. of the 8th USENIX Security Symposium , (1999).
27. Tanenbaum. Chapter 7. In: Distributed Operating Systems. Prentice Hall, (1995).
28. A. Waldspurger, W. E. Weihl. Stride Scheduling: Deterministic Proportional-Share Resource Management. Technical Report MIT/LCS/TM-528, Massachusetts Institute of Technology, MIT Laboratory for Computer Science, (1995).
29. Webpage of VMWare, <http://www.vmware.com/>.

Simultaneous Multithreading Trace Processors

K.F. Wang, Zhenzhou Ji, and Mingzeng Hu

Department of Computer Science and Technology of
Harbin Institute of Technology,
Harbin 150001
wkf@pact518.hit.edu.cn

Abstract. In this paper, we propose a new next generation high-performance micro-architecture based on the combination of simultaneous multithreading and trace processor. By exploiting both Instruction-Level Parallelism and Thread-Level Parallelism, Simultaneous Multithreading Trace Processor can be expected to achieve higher performance than SMT or Trace Processor individual. We describe the organization of SMT Trace Processor architecture and some fundamental techniques. A path-based multiple traces prediction mechanism is also described in the paper.

1 Introduction

With the technique progressing and the human's ever increasing requirement for high performance, the issue bandwidth of microprocessor increases every year. Conventional Superscalar uses complex hardware to dynamic schedule instructions which result in hard implementation and validation. Small instruction window also limits effective exploiting ILP (Instruction Level Parallelism) exists in the sequence program. When issue bandwidth of future micro-processor grows beyond 8, pipeline of superscalar faces challenges from front to end. Past research works show that a taken branch exists among every 8 instructions in ordinary application[1]. The conventional superscalar instruction cache can not access two discontinuous cache blocks in one cycle which limits the fetch bandwidth. Complexity of register renaming and dependence checking among all instructions also increase significantly; larger issue bandwidth also imposes more pressure on register file and data cache; and the complexity of result bypassing logic increases extremely. For these problems, conventional superscalar can not satisfy human's requirement for high performance any more. Exploiting next-generation high performance micro-architecture to break superscalar architecture limitations has become research focus in recent year.

Based on absorbing many new proposed architecture benefits, this paper proposes a next generation high performance micro-architecture, Simultaneous Multithreading Trace Processor. The main contribution of this paper is the combination of SMT and Trace Processor which exploiting both ILP and TLP at the same time. We also proposed a new path-based multiple traces prediction mechanism which can provide more traces than the original Trace Processor Model.

2 Previous Work

The idea of this paper grows from the trace processor [2] proposed by Wisconsin university and SMT[3] Architecture proposed by Washington university. Based on these two architectures, we propose an new combination architecture, SMT Trace Processor.

Trace Processor first proposed by Eric Rotenberg, Quinn Jacobson, Yiannakis Sazeides and James E. Smith is a micro-architecture organized around trace, a dynamic instruction execution sequence, which can efficient issue a lot of instructions in a cycle. For the temporal and spatial locality, it is probably that in the near future the same trace will be executed again. A new structure, Trace Cache[4] proposed by Eric Rotenberg, Steve Bennet and James E. Smith is integrated in Trace Processor for exploiting ILP farther. When Trace Processor running, it collects dynamic instruction sequence and stores it into Trace Cache. When the trace is needed again, it can be easily and quickly provided by Trace Cache rather than conventional instruction Cache. For the dynamic sequence character, trace can implicitly include several branch instructions and provide multiple instruction blocks at discontinuous addresses. Fetching at the unit of trace can achieve wider fetch-bandwidth than superscalar. Working around traces, Trace Processor fetches instructions from instruction cache, then constructs them into trace using trace construct unit. After being preprocessed, the trace will be stored into Trace Cache. The centralized hardware mechanism in trace processor allows predict and dispatch next executing trace every cycle. After being preprocessed and predicting the value of live-in registers, this trace can be dispatched to the back-end distributed processing element. For the low ILP within one trace, the processing element can be implemented relative simple, and easy to be validated, just like a small-scale superscalar. Trace Processor allocates resources at the unit of trace, and exploits both control and data prediction at the trace level rather than instruction level. The higher hierarchy avoids logic complexity and architecture limitation exists in the superscalar.

For the increasing difficulty in exploiting ILP in recent year, some research works are turned to TLP. Researchers want to remedy single thread application's insufficient ILP by introducing TLP. The research works in this direction can be divided into three categories: Fain-Grain Multithreading, Single-Chip Multiprocessor and Simultaneous Multithreading. These works want to boost performance by improving system throughput. Fain-grain multithreading hold entire hardware state(PC and registers) for every running thread. Resources are occupied by multiple threads in different clock cycle. In any specific cycle, only one thread can use the data-path resources. When the next cycle coming, processor switches to another thread and keeps running. It solves the inefficient hardware utilization caused by long latency instructions, control hazards, structure hazards and data hazards in superscalar architecture. Only one thread exclusive occupying hardware in one cycle make insufficient use of hardware for the limited ILP. With the increasing of issue-bandwidth, the disadvantages become more clearly. Single-Chip Multiprocessor divides hardware resources on chip into several parts physically. Each part is just a simple superscalar. SCMP allows multiple programs running on different processors, which fully utilize hardware resources by exploiting program or thread parallelism. It boosts performance on most multiple program or multithreading applications. However, the fixed partitioning of hardware and in-flexible resource allocating

mechanism often cause some processor having too many workloads but the others being idle. Especially when there is only one single thread in the application, all processor are being idle except one, which waste hardware extremely. Based on the analysis of benefits and flaws of these two architectures, Dean M. Tullsen and Susan J. Eggers of Washington university proposed Simultaneous Multithreading in 1995. SMT combines hardware features of wide-issue superscalar and multithreaded processors. It inherits the ability to issue multiple instructions each cycle from superscalar and holds hardware state for several threads. The result is that it can issue multiple instructions from multiple threads each cycle, achieving better performance for most applications. It not only solves the horizontal waste caused by the exclusively hardware occupation by single thread, but also tolerates vertical waste caused by long latency instructions, hazards and cache missing in superscalar. It is considered a promise next generation architecture, the most outstanding breakthrough after RISC in micro-architecture research area. However, after introducing TLP, the hardware switching between multiple threads has become a bottleneck to the improvement of performance.

3 SMT Trace Processor Architecture

Exploiting control and data prediction at the trace level can effectively avoids logic complexity of wide-issue superscalar. The improvement of performance comes from parallel execution of different traces on different processing elements. For the probably existence of dependence between traces, whether the multiple traces can be effectively executed in parallel depends on the accuracy of live-in register value prediction. Unfortunately, at present, the accuracy of value prediction can not satisfy our requirement. The more traces being executed in speculation, the more mis-speculation will occur. Executing some useless traces will waste hardware resources and decrease performance/power ratio. If we limit the number of parallel executing traces in a small range, the benefits promised by Trace Processor can not be achieved. Augmenting simultaneous multithreading on Trace Processors allows dispatching multiple traces to distributed processing elements at the back-end of pipeline in one cycle. The number of traces executing in parallel is relative small, which solves the low accuracy problem on value prediction commendably. At the same time, this architecture boosts performance with the improved system throughput. Figure 1 is the structure of SMT Trace Processor.

The figure shows that in order to support multithreading, Program Counter was duplicated to fetch instruction sequence from multiple threads Branch predict unit, Trace Construct unit and Trace Preprocess unit are all duplicated to predict branch instructions in multiple threads, construct traces and preprocess these new traces. At last, the preprocessed traces are stored into Trace Cache which is shared by all the threads.

3.1 Trace Construct

Trace construction method affects performance in a great extent. Trace is a dynamic instruction sequence, and the length of trace is limited by two factors: trace cache line

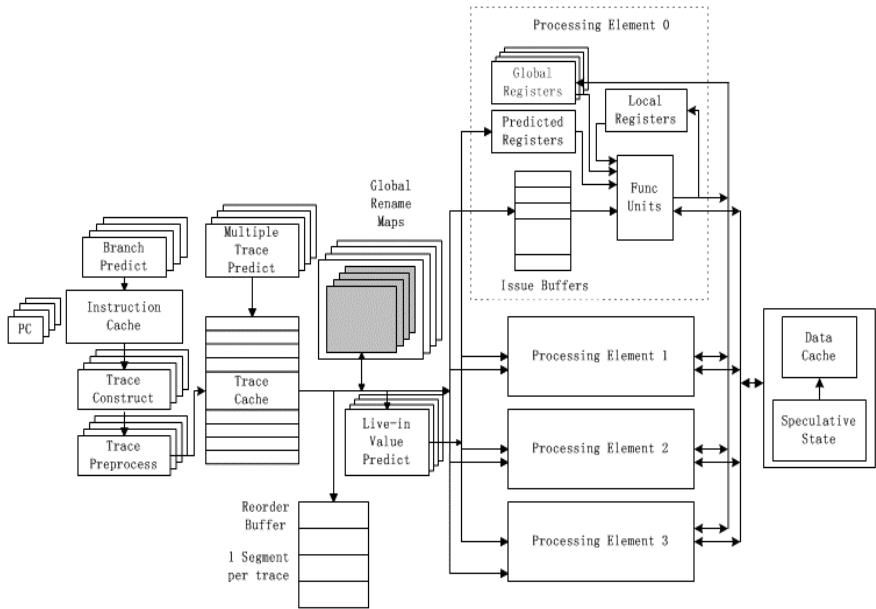


Fig. 1. SMT Trace Processor

size and the number of branch instructions can be stored within one trace. Too short in length imposes negative affect on exploiting ILP within trace, but too long will decrease full-hit rate of traces, which increasing penalty of mis-prediction. There is always some physical limitations when constructing traces, such as the trace length can not larger than a certain threshold and ending trace construction at some call indirect, jump indirect or return instructions. In order to exploit control independence, some mechanisms ends trace construction when meeting loop instructions. We can increase trace length by using branch promotion and trace packing[6] techniques to fully utilize Trace Cache. To support Simultaneous Multithreading, SMT Trace Processor duplicated the Trace Construct unit to create traces for multiple threads.

3.2 Trace Preprocess

In order to eliminate pressure of register renaming unit when traces are being dispatched, we can preprocess these traces in some aspects before they are stored into Trace Cache. Trace Processor divides physical registers into three categories: live-in register, live-out register and local register. The first two types are also called global register. Live-in registers are those registers which value will be used by the instructions in the trace, local registers are entirely used within the trace, the value of live-out registers will be transferred out of the trace and will be used by other traces. At the preprocessing stage, register fields of each instruction will be labeled as local or global. When trace issuing, we can only rename global-registers in global and the renaming of local registers will be processed within Processing Elements. Trace Preprocessing unit can decrease dependence across trace boundary by dynamic re-

scheduling instructions within traces. In order to support SMT, SMT Trace Processor duplicates multiple Trace Preprocess Units to preprocess multiple traces generated by trace constructor unit.

3.3 Trace Predict

Predicting next trace quickly and exactly affects the whole performance in a great degree. Trace can be specified by starting address of the trace and branch outputs of branch instructions within the trace. We can predict next trace based on starting address and outputs of multiple branch instructions, but this mechanism imposes more pressure on fetch-bandwidth of predict unit and the accuracy is not as high as our expectation. In 1997, Quinn Jacobson proposed next path-based trace prediction[5] which use executing histories of previous traces. Each trace is specified by a unique trace id. The last several trace ids are stored in a shift register after being processed by hash logic, and it will be used as history information to predict next trace. To improve the accuracy of prediction, return history stack is added for the special procedure call and return instructions. When meeting a call, the trace history will be pushed into return history stack and popped out when meeting the corresponding return instruction. This method addresses history being missed or polluted by procedure context switching. At the same time, in order to reduce the impact of cold-start and aliasing, a second smaller predictor uses only last trace id is added. The accuracy of this mechanism is higher than multiple branch predict unit, but only one trace can be provided in one cycle. When the number of Processing Elements increases, there is not enough traces to be executed in parallel. So we proposed next path-based multiple traces predict unit, which allows predicting multiple traces in one cycle and these traces will be dispatched to and executed on distributed Processing Elements. The next path-based multiple traces predictor has a two level structure[7] is described in figure 2.

The first trace will be executed in the future is predicted by K bits of the history information, and the second is predicted by K-1 bit plus prediction of the first trace, the rest may be deduced by analogy. This mechanism can provide multiple traces quickly, but the expansibility is a question. SMT Trace Processor duplicates multiple Two-level Adaptive Multiple traces Predictor unit to support SMT. The more traces providing by the predictor, the more utilization of Processing Elements can be achieved.

3.4 Value Predict

To improve performance, we should execute different traces in parallel on processing elements as much as possible. For the probably existence of control and data hazards, the number of traces executing in parallel is limited. Next path-based multiple traces predictor can eliminate control hazards, and the impact of data hazards can be reduced by introducing value prediction. As described above, SMT Trace Processor divides registers into live-in, local and live-out. Live-in registers are those registers that their value will be used in the trace, so they are the only source of true data dependence. When the value of live-in register is unknown, the trace can not be executed. The

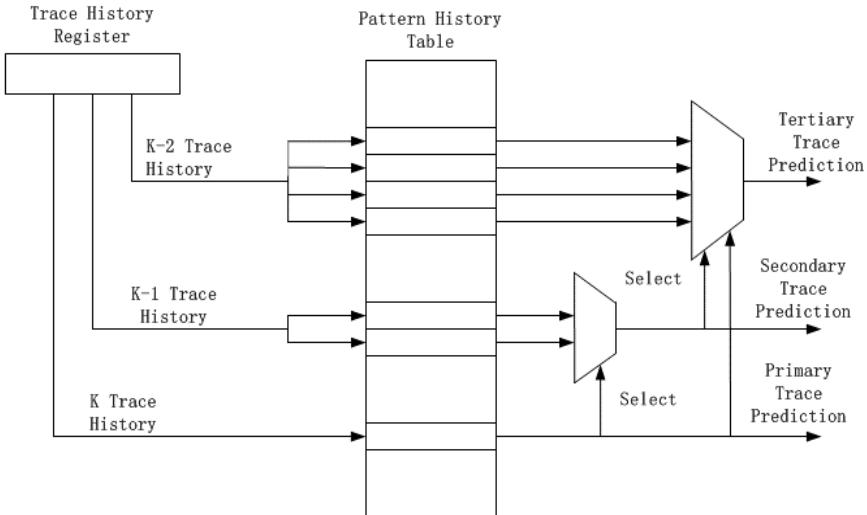


Fig. 2. Two-level Adaptive Multiple traces Prediction

other two type do not hold input value of trace, so we need only predict the values of live-in register and speculative execute the trace. At present, the research works on value prediction are just start, and they are mainly focus on predicting some instructions which behavior show some value locality. For example, some instructions always produce the same result or the results following some patterns. A context-based and organized with a two level table predictor proposed by Y. Sazeides and J. E. Smith was used in the Trace Processor. The first-level table is indexed by prediction id which derived from trace id and register no or predict address. The contents of first-level table are hash values of the last n value in the past, and the content is used to index second-level predict table, which provide the value being predicted.

The value prediction deserves more advance researches. The usage of value prediction in current is limited, and the low accuracy often causes mis-prediction. Introducing Simultaneous multithreading can eliminate this problem in some extent. For multithreads, the number of traces being executed in parallel can be relative small, and the depth of value prediction is reduced, in replaced with lower depth of value prediction from multiple threads. The lower value mis-prediction rate, the more performance can be achieved.

3.5 Register Hierarchy

Dividing registers into three types eases the implementation of register renaming and result bypassing logics. Global registers, not local registers can conduct to the dependence between traces at the time of trace dispatching, so only renaming global registers can we reduce the pressure of register renaming logic. There are only one local register file and copies of multiple global register file which supporting simultaneous multithreading in the Processing Element. When executing a trace, the

destination register of the result may be local or global. For local destination, the result can be write back to local register file because it will be used entirely within the trace, but for global, we must broadcast the result to the others processing elements which executing the same thread's trace after writing it back to local register file. The duplicated multiple global register files are used to support SMT. When broadcasting the global result, only the global register file in those Processing Elements which executing the same thread need to be updated.

3.6 Speculative Memory State

Distributed Processing Elements issue load and store instructions to Data Cache out-of-order and some traces in execution are also being executed in speculative state, so there must be some effective mechanisms to keep the speculative memory states, disambiguate memory addresses, and recover the architecture state when meeting mis-speculation. Address Resolution Buffer[8] proposed by Sohi is a feasible solution. Augmenting an ARB in Data Cache, multiple PE can issue load/store instructions to memory out-of-order. All the loads and stores are buffered in the ARB. Before committing to Data Cache, all these load and store instructions must be reordered and their destination addresses need to be disambiguated. ARB buffers all the stores issued by any PE until it affirms that the store is in the oldest trace and it is the first store in that trace. ARB records all the loads issued from any PE. When a store instruction being issued from a PE, ARB checks all the Processing Elements which are executing latter traces in the logic sequence to determine if any load is related to that store. If there is, these loads need to be re-issued. When a load being issued, ARB checks all store instructions that probably have some dependences with that load. If there is, it must re-direct this load instruction to the corresponding store; otherwise, the load instruction will get a wrong value.

4 Summary

With the increasing difficulty in exploiting instruction level parallelism, turning to thread level parallelism is an effective method to improve performance. This paper proposed Simultaneous Multithreading Trace Processor, based on the research and analysis on trace processor and simultaneous multithreading. Introducing Simultaneous Multithreading on Trace Processor can boost performance by reducing impact of low value prediction accuracy which can increase the recover penalty. Augmenting path-based multiple traces predictor allows multiple traces which can be dispatched to and executed on the distribute Processing Elements in one cycle. For the limited value prediction accuracy, the number of traces in parallel execution can be relative small. Simultaneous multithreading allows multiple threads can be executed in parallel, so high performance can be anticipated. This performance improvement derives from the duplicated hardware, such as: Branch Predict unit, Trace Construct unit, Trace Preprocess Unit, Global Register Maps, Global Register and Live-in Value Predict Unit.

References

- [1] J. L. Hennessy and D. A. Patterson, Computer Architecture A Quantitative Approach, Morgan Kaufmann, 2nd edition, 1996.
- [2] E. Rotenberg, Q. Jacobson, Y. Sazeides and J. Smith, Trace Processors, In Proceedings of the 30th International Symposium on Microarchitecture(MICRO-30), pages 138–148, Dec 1997.
- [3] D. M. Tullsen, S. J. Eggers, and H. M. Levy, Simultaneous multithreading: Maximizing on-chip parallelism. In Proceedings of the 22nd Annual International Symposium on Computer Architecture, pages 392–403, June 1995.
- [4] E. Rotenberg, S. Bennett, and J. E. Smith, Trace cache: a low latency approach to high bandwidth instruction fetching. In Proceedings of the 29th Annual International Symposium on Micro-architecture, pages 24–35, Dec 1996.
- [5] Q. Jacobson, E. Rotenberg and J. E. Smith, Path-Based Next Trace Prediction, In Proceedings of the 30th International Symposium on Microarchitecture, pages 14–23, Dec 1997.
- [6] Sanjay J. Patel, Marius Evers, and Yale N. Patt, Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing, In Proceedings of the 25th International Symposium on Computer Architecture, Barcelona, Spain, June 1998
- [7] T.-Y. Yeh and Y. Patt, Two-level Adaptive Branch Prediction, In proceedings of the 24th International Symposium on Microarchitecture, pages. 51–61, November 1991.
- [8] M. Franklin and G. S. Sohi, ARB: A hardware mechanism for dynamic reordering of memory references. IEEE Transactions on Computers, 45(5): 552–571, May 1996.

A VLSI Architecture Design of 1-D DWT

Zhiyang Cao, Zhenzhou Ji, and Mingzeng Hu

Department of Computer Science and Engineering,
Harbin Institute of Technology,
150001 Harbin, China
czy@pact518.hit.edu.cn

Abstract. In the paper, we design an efficiently, scalable one-dimensional discrete wavelet transform (1-D DWT) architecture. The architecture is constructed with cascaded basic computation units, which implement multiplication, accumulation and down-sampling functions. The architecture is quite regular and scalable so that it fits into a VLSI implementation and can be extended to an arbitrary 1D-DWT with L taps and J octaves.

1 Introduction

Wavelet transforms, in their different guises, have been accepted as a set of tools useful for various applications. In contrary to conventional transform techniques such as Fourier transform, it has inherent time-scale locality characteristics and excellent energy compaction characteristics [1]. These characteristics make it suitable for compressing and analyzing signals and images which involve transient or local parts. The discrete wavelet transform (DWT), as one type of wavelet transforms, has been widely applied in speech analysis, pattern recognition, image processing, and compression etc. Due to its well time-frequency decomposition, the JPEG2000 still image coding and MPEG-4 still texture coding standards have adopted the DWT as their transform coder. DWT has not the blocking effect inherent to discrete cosine transform (DCT) and provides a relatively high compression rate. But it requires much more arithmetic computations and memories. Besides, contrary to the block-based DCT, the DWT is basically frame-based [2].

In order to satisfy the needs of real-time signal processing, the VLSI implementation of DWT has been paid more attention to. A number of architectures have been designed to implement the DWT. The folded structure has two types: intra-octave folding and inter-octave folding, they share some computation resources to improve the hardware utilization efficiency, while they require blocking scheduler and a ping-pong buffer. The lattice and systolic array structure are regular and scalable but requires complex routing networks and scheduling. The SIMD-based structure is very fast, but it has a high circuit complexity which is proportional to the input size. In the paper, we proposed a simple efficient VLSI architecture for 1-D DWT. It requires fewer scheduling and hardware resources.

The paper is organized as follows. Section 2 reviews the concepts and theories of the discrete wavelet transforms. Section 3 introduces the methods of designing the

VLSI parallel architectures and presents the 1-D DWT architectures. Finally, analysis and conclusions are reported in section 4.

2 Theory of Discrete Wavelet Transforms

Wavelet is a finite width wave in time field. The wavelet transforms make use of wavelet as basis functions to transform and represent signal with the basis function at last. These basis functions vary both in frequency and position. A family of wavelet basis functions can be generated by translating and dilating the mother wavelet corresponding to the family.

Three key techniques made DWT develop rapidly, which are the theory of filter family, multi-resolution or multi-scale analysis, sub band coding [3]. The DWT can be looked at as the multi-resolution decomposition of a sequence. The sequence is recursively decomposed into two components in octave bands, which are coarse and detail of the output of the former decomposition. Mallat's pyramid algorithm [4] is considered the most important algorithm to calculate the DWT coefficients and it plays the similarly important role in wavelet transform as fast Fourier transform (FFT) has been in Fourier transform. Mallat's algorithm is basically a collection of cascaded finite impulse response (FIR) filtering operations by a pair of low pass and high pass filters and down-sampling procedure in each scale, which is shown in Fig. 1.

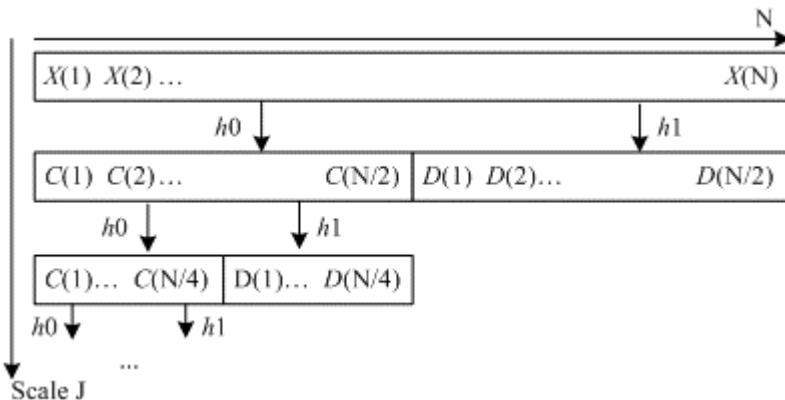


Fig. 1. Mallat's pyramid algorithm

Where $X(n)$ is an input sequence, $n = 1, 2, \dots, N$, which is the length of the sequence, $h0(l)$ is low pass filter and $h1(k)$ is high pass filter, $k = 1, 2, \dots, L$, which is the number of taps of the filters, J is the number of octave. The first $N/2$ signals $D(1) \dots D(N/2)$ are the high half band signals and $C(1) \dots C(N/2)$ are the low half band signals and so on. These signals are down-sampled by a factor of two. The recursively decomposition of input signal is expressed with convolution of input signal sequences and the filtering coefficients as follow:

$$C_{j+1}(n) = \sum_{k=0}^{L-1} h0(k)C_j(2n-k). \quad (1)$$

$$D_{j+1}(n) = \sum_{k=0}^{L-1} h1(k)C_j(2n-k). \quad (2)$$

The inverse discrete wavelet transform (IDWT) does the exactly opposite computation of the DWT, which is expressed as:

$$C_{j-1}(n) = \sum_{k=0}^{L-1} h0'(n-2k)C_j(k) + \sum_{k=0}^{L-1} h1'(n-2k)D_j(k). \quad (3)$$

Similarly to the DWT, the computation of IDWT can be implemented with an up-sampled filter bank. All the coefficients of the filters in DWT and IDWT are relative to the selected wavelets. However, for the decomposed signal to be reconstructed to the original signal stage by stage, the analysis filters and synthesis filters must satisfy to the perfect re-constructible quadrature mirror filter (RQMF) property.

The key step to construct wavelet is the selection of low pass filter. Wavelet can be constructed from it. It must satisfy to the symmetric condition:

$$H0^2(S_N/2 + s) = 1 - H0^2(S_N/2 - s). \quad (4)$$

$H0(s)$ is the transfer function of $h0(t)$. S_N is Nyquist sample frequency. In general, high pass filter is the mirror filter of low pass filter.

$$h1(L-1-k) = (-1)^k h0(k). \quad (5)$$

3 Architectures

3.1 Design of VLSI Parallel Architectures

For an application, the design of VLSI chip involves three aspects: application field, computer architecture and VLSI chip design [5]. These aspects reflect the different layers and stages of design of VLSI parallel architecture.

The first step is expression of application problem. Computation model is abstracted from the application problem. Then select adaptive serial algorithm and find suitable method to describe it. The second step is getting the description of array structure from the description of the serial algorithm. The procedure includes the constructing of PE, the interconnection of PE, the style of data flowing and arrangement of computation time etc. The last step is designing the logical circuit of VLSI chip and layout.

In the second step, parallelism needs being exploited as most as possible so that the problem can be processed in parallel. The basic methods exploiting parallelism

include analysis of variable, substitution of sentence, wave-front, recursion and loop distribution.

3.2 Proposed 1D-DWT Architecture

The main computations of DWT are convolutions. In terms of computer architecture, it can be looked at as specific matrix multiplication computation. The basic computation is inter-product computation step. Convolution computation can be expressed as linear recursive form, so it can be computed in parallel and pipeline in VLSI array.

Equations (1)-(2) can be expressed as recursive forms:

$$\begin{aligned} C_{j+1}^{-1}(n) &= 0 \\ C_{j+1}^k(n) &= C_{j+1}^{k-1}(n) + h_0(k)C_j(2n-k). \quad (k = 0, 1, \dots, L-1) \\ C_{j+1}(n) &= C_{j+1}^{L-1}(n) \end{aligned} \quad (6)$$

The 1-D DWT involves two procedures in each octave: convolution and down-sampling. We design a basic unit that involving the two procedures.

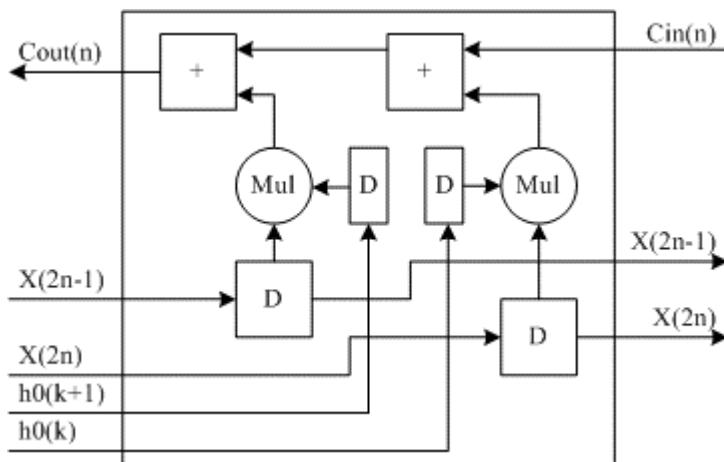
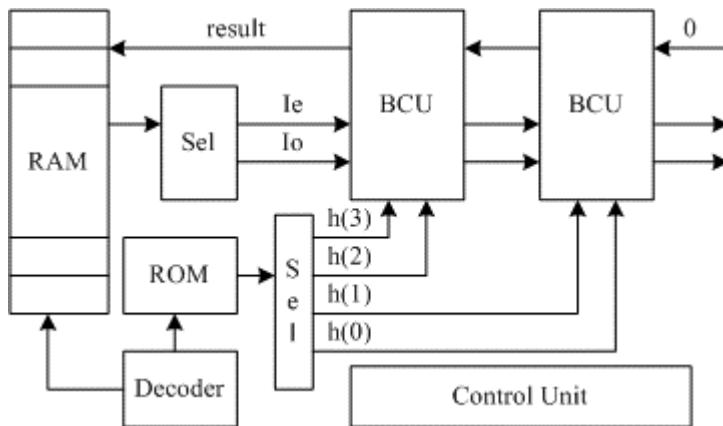


Fig. 2. The basic computation unit

Each basic unit involves two adders, two multipliers and four registers. Two registers stores the coefficients of filter. The other registers store the input data and shift the input data to next unit. $Cout(n)$ is the output data, it flows in reverse direction with input data.

Use the basic unit to construct the 1D-DWT architecture based on Daubechies 4-taps filters. The block graph is shown in Fig.3. Only main function parts are drawn.

RAM stores the input original data and results. ROM stores the coefficients of the filters. Even address data are input to Ie, odd address data are input to Io.

**Fig. 3.** The structure of 1D-DWT

4 Analysis and Conclusion

The proposed 1D-DWT uses one dimensional linear array structure. Data flows from one unit to another unit. Routing and communication between two units is simple. Bus is not used to reduce the global communication. The operation of the basic computation unit is controlled by the Control Unit. The decoder implements the address decoding. Computation and communication can be interleaved to improve the performance of the system.

The structure uses four adders, four multipliers and eight registers. Schedule controlling is not complex. The architecture is quite regular and scalable so that it fits into a VLSI implementation and can be extended to an arbitrary 1D-DWT with L taps and J octaves.

References

1. Chokri Souani and et al., “VLSI design of 1-D DWT architecture with parallel filters”, INTEGRATION, the VLSI journal 29, pp 181–207, 2000
2. N. D. Zervas and et al., “Evaluation of design alternatives for the 2-D-discrete wavelet transform,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 12, pp. 1246–1262, Dec. 2001
3. Kenneth R. Castleman, “Digital Image Processing”, Publishing House of Electronics Industry, 2002
4. Stéphane Mallat, “A wavelet tour of signal processing”, Academic Press, 1999
5. Chen Guoliang, Chen Ling, “VLSI computing theory and parallel algorithm”, University of Science and Technology of China Press, 1991

Overcoming Static Register Pressure for Software Pipelining in the Itanium Architecture*

Haibo Lin, Wenlong Li, and Zhizhong Tang

Tsinghua University,
Beijing 100084, P.R. China
`{linhaibo99, liwenlong00}@mails.tsinghua.edu.cn`
`tzz-dcs@tsinghua.edu.cn`

Abstract. Software pipelining techniques have been shown to significantly improve the performance of loop-intensive scientific programs. The Itanium architecture contains many features to enhance parallel execution, such as support for efficient software pipelining of loops. The drawback of software pipelining is the high register requirements, which may lead to software pipelining failure due to limited static general registers in Itanium. This paper evaluates the register requirements of software-pipelined loops. It then presents a novel register allocation scheme, which allocates stacked registers to serve as static registers. Experimental results show that this method gains an average 2.4% improvement, and a peak 18% improvement in performance on NAS Benchmarks.

1 Introduction

The Itanium architecture contains many features to enhance parallel execution, such as an explicitly parallel (EPIC) instruction set, large register file, etc. It also provides features such as register rotation [1] to support efficient software pipelining without increasing the code size of the loop. Software pipelining [2] tries to improve the performance of a loop by overlapping the execution of several successive iterations. This improves the utilization of available hardware resources by increasing the instruction level parallelism (ILP).

The drawback of aggressive scheduling techniques [3] such as software pipelining is that they increase register requirements [4]. There have been proposals to perform register allocation for software-pipelined loops [5]. If the number of registers required is larger than the available number of registers, spill code has to be introduced to reduce register usage, or software pipelining is given up.

This paper evaluates static register requirements of software-pipelined floating-point intensive loops. A new method to software pipeline loops that require more static general registers than those Itanium provides is also presented. It uses stacked general register to serve as static general register, thus increasing software-pipelined loops that require many static general registers but few rotating general registers. It has been implemented in Open Research Compiler (ORC), and results in an average

* This work was supported by NSFC grant 60173010.

2.4% improvement and a peak 18% improvement in performance on NAS Benchmarks.

2 Itanium Architecture Features

Itanium provides 128 general registers, 128 floating-point registers, and 64 predicate registers. The general registers are partitioned into two subsets. GR0-GR31 is termed the static general register. GR32-GR127 is termed the stacked general register. The stacked registers are made available to a program by allocating a register stack frame consisting of a programmable number of local and output register. The floating-point registers and predicate registers are also partitioned into two subsets respectively. FR0-FR31 (PR0-PR15) is termed the static floating-point (predicate) register. FR32-FR127 (PR16-PR63) is termed the rotating floating-point (predicate) register.

A fixed-size area of the floating-point and predicate register files(FR32-FR127 and PR16-PR63), and a programmable-sized area of the general register file are defined to rotate. The size of the rotating area in the general register file is determined by an immediate in the `alloc` instruction and must be either zero or a multiple of 8, up to a maximum of 96 registers. The lowest numbered rotating register in the general register file is GR32.

3 Register Requirements of Software Pipelining in the Itanium Architecture

This section evaluates static general register requirements of all the innermost loops of the NAS Benchmarks that are suitable for software pipelining in the Itanium architecture. These loops have been obtained with the ORC compiler. A total of 363 loops amenable for software pipelining have been used. This set includes all the innermost loops that do not have subroutine calls or conditional exits. Loops with conditional structures in their bodies have been IF-converted, with the result that the loop now looks like a single basic block.

In the Itanium architecture, three kinds of variables require static general registers. They are:

1. Dedicated variants, which require special general registers such as global pointer (gp), memory stack pointer (sp);
2. Loop invariants, the values of which are repeatedly used by a loop at each iteration, but never written by it;
3. Base update variants, something like induction variables. Consider the following load instruction `ld4 r1=[r2], 4`. After the value consisting of 4 bytes is read from memory starting at the address specified by the value in GR r2, the value in GR r2 is added to 4, and the result is placed back in GR r2. The variable in GR r2 is termed base update variant. Although the value of such a variable varies at each iteration, the def and ref are involved in one instruction, resulting in a live-range of 0 cycle. So we should assign it a static general register instead of rotating general register.

Figure 1 shows the cumulative distribution of the requirements for static general registers for all 363 loops. In general, loops have very few dedicated variants. For instance 96% of the loops have no dedicated variants and only 5 loops have 1 dedicated variant. Loops also have few loop invariants. For instance 65% of the loops have no loop invariants and the other loops have at most 6 invariants. Nevertheless loops have a high number of base update variants. For instance 3 loops have more than 32 base update variants.

Taking into account of several special registers, such as global pointer (gp), memory stack pointer (sp), reserved thread pointer (tp), and 1 or 2 register serving other purposes, there are only 27-28 out of 32 static general registers available for software pipelining. Since 6% of the loops require such many static general registers, these loops will fail in software pipelining phase.

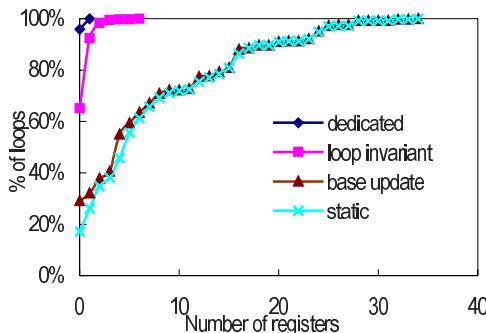


Fig. 1. Cumulative distribution of variants requiring static general registers. Each point (x,y) of the graph represents the percentage y of loops having less than x registers

4 Stacked Register Allocation

General registers GR32-GR127 form a register stack that is automatically managed across procedure calls and returns. Each procedure frame on the register stack is divided into two dynamically sized regions, one for input parameters and local variables, and one for output parameters. The hardware makes no distinction between input and local registers. On a procedure call, the registers are automatically renamed by the hardware so that the caller's output registers form the base of the callee's new register stack frame. On return, the registers are restored to the previous state, so that the input and local registers are preserved across the call.

A subset of the registers in the procedure frame may be designated as rotating registers. The rotating register region always starts with GR32, and may be any multiple of 8 registers in number, up to a maximum of 96 rotating register. The renaming is under control of the Register Rename Base (RRB). If the rotating registers include any or all of the output register, software must be careful when using the output registers for passing parameters, since a non-zero RRB will change the virtual register numbers that are part of the output region. In general, software should either ensure that the rotating region does not overlap the output region, or that the RRB is cleared to zero before setting output parameter registers.

Since the rotating general register requirements of software pipelining are not very high, there are often some unused general registers in the register stack. We propose a novel register allocation scheme for software pipelining called Stacked Register Allocation (SRA). SRA tries to allocate these registers to variants needing static general registers. Actually, SRA allocates stacked non-rotating registers right on top of stacked rotating registers, and put the rest of the register stack frame to higher register regions. SRA realizes dynamic partition between static general registers and rotating general registers to some extent (SRA can not reduce the number of static general registers to less than 32). It improves the utilization efficiency of general registers, and results in more software-pipelined loops. Fig. 2 shows register stack usage model before and after SRA.

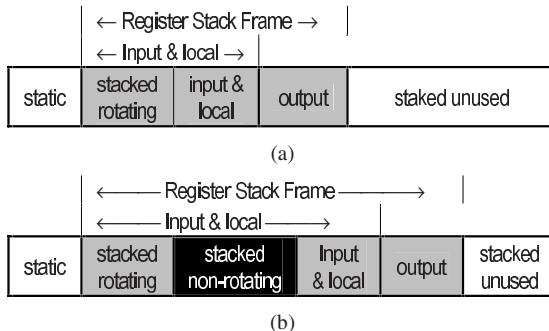


Fig. 2. Register stack usage model: (a) before SRA, (b) after SRA

5 Experimental Results

We have implemented SRA in the ORC compiler for Itanium. In this section, we compare the results of using this technique and original implementation in ORC on the NAS suite of benchmark programs, that is, the speedup of our method. These results were obtained by running the benchmarks with -O3 level optimization. Profile feedback and Inter-Procedural Analysis (IPA) were not used.

Provided with 32 physical general registers, the number of loops that fail in software pipelining caused by non-enough static general registers range from 0 to 11. These loops are all pipelined after SRA is applied. It is reasonable that the number of un-pipelined loops increases when the number of available general registers decreases. SRA can solve the problem of software pipelining failure even in the context of 24 general registers.

Figure 3 shows the percentage improvements of SRA in execution times on an Itanium 733 MHz machine. SRA results in an average 2.4% improvement and a peak 18% improvement in performance with 32 general registers. This result is rather exciting because only 6% of loops are optimized. When the number of available general registers reduces to 24, SRA gains an average speedup of 3.1%.

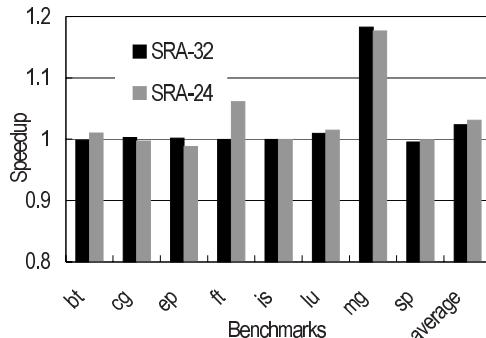


Fig. 3. Performance of SRA. SRA-32 and SRA-24 refer to the number of available static general registers of 32 and 24 respectively

6 Conclusion

In this paper we have evaluated the static general register requirements of software-pipelined loops of NAS Benchmarks on Itanium architecture. We have also shown that some loops with high static general register requirements fail in software pipelining phase. A new method is presented to increase software-pipelined loops on Itanium architecture. It allocates free general registers from register stack for loops that require more static general registers than those Itanium provides. The experimental results show significant improvements in the execution time of NAS Benchmarks.

References

1. Dehnert J. C., Hsu P. Y., Bratt J. P.: Overlapped Loop Support in the Cydra 5. Proceedings of ASPLOS'89. (1989) 26–38
2. Allan V. H., Jones R. B., Lee R. M., Allan S. J.: Software Pipelining. ACM Computing Surveys. **27** (1995) 367–432
3. Huff R. A.: Lifetime-sensitive modulo scheduling. Proceedings of PLDI'93. (1993) 58–267
4. Llosa J.: Reducing the Impact of Register Pressure on Software Pipelining. PhD thesis. Universitat Politècnica de Catalunya (1996)
5. Rau B. R., Lee M., Tirumalai P., Schlansker P.: Register allocation for software pipelined loops. Proceedings of PLDI'92. (1992) 283–299

Separating Data Storage, Data Computation, and Resource Management One from Another in Operating Systems*

Fuyan Liu^{1,2} and Jinyuan You¹

¹ Dept. of Computer Sci. & Eng.,
Shanghai Jiaotong Univ.,
Shanghai 200030, China

² Dept. of Computer Sci. & Tech.,
North China Inst. of Tech.,
Taiyuan 030051, China

Abstract. It's impossible for traditional operating systems to separate the abstract for data storage (process virtual address space), the abstract for data computation (thread), and the abstract for resource management (process itself) one from another. In this paper, firstly we analyze the problems due to not separating these three abstracts. On the basis of the analysis, we propose the idea that these three abstracts should be separated, and on the basis of the idea, we propose Operating Systems Basing Virtual Address Spaces on Files (OS-BVASF). Then we investigate OS-BVASF architecture model, Thread Migration and Instructions Accessing Files Directly that implement the separation. In the end of this paper, we discuss its implementation and performance test.

1 Introduction

Constructing some abstracts, with system calls to operate them, can be thought as operating system's function. Operating Systems should construct at least four types of abstracts: the one for data storage, the one for data computation, the one for resource management, and the one for input and output. In a traditional operating system, the abstract for data storage is process virtual address space; the abstract for data computation is thread; the abstract for resource management is process, through which operating system implements the resource management and allocation; the abstract for input and output is file.

* This work was support by the National Natural Science Foundation of China (60173033, The Architecture Research of Operating System Basing Virtual Address Space on Files); also by the Youth Technology Research Foundation of Shanxi Province (20021016, The Architecture Research of Operating System Basing Virtual Address Space on Files), and by the Defense Science and Technology Key laboratory Foundation (51484030301JW0301, The Architecture Research of Operating System Basing Virtual Address Space on Files for Large Scale and Parallel Computer).

Among the four types of abstracts constructed by operating systems, file is a special one. Only file is persistent, only file belongs to no process. The other three abstracts must belong to some process or even process itself, and cannot be separated one from another:

1. Thread and process cannot be separated: a thread can only use resources that belong to its process; threads that belong to the process can only use the resources of a process.
2. Thread and process virtual address space can not be separated: the virtual address space of a process can only be addressed by threads that belong to the process; threads that belong to a process can only address the virtual address space of the same process.
3. Process virtual address space and process itself cannot be separated: it's impossible to separate process virtual address space and process itself one from the other. Process virtual address space is part of the process itself.

Because data storage (process virtual address space), data computation (thread), and resource management (process) can not be separated one from another, when a thread that belongs to one process access services provided by other process, there exist some problems, for example,

1. Because thread and process cannot be separated, the client thread cannot enter directly into server process. When client thread need call a service in server, it has to employ client/server model and inter-process communication mechanisms, such as Message-Passing, semaphore, pipe, mail-box; socket, the service is called by a proxy thread in server process. Firstly client thread awakes server thread, then client thread get slept. When it finishes its task, the server thread awakes client thread, after that it gets slept. The client thread gets awaked and keeps on executing. For the client thread, it only calls a server service once, but for operating system, it need execute several times thread-switching as well as thread sleeping and waking up. Comparing with that of calling a function in libraries or a system call in operating system kernel, the performance of calling a service in server is poorer. This is one of the reasons why micro-kernel operating system is of poor performance [1].
2. Because thread and process virtual address space cannot be separated, server thread cannot access data in client process virtual address space. Client thread has to pass calling parameter through mechanism such as message passing, pipe, mail-box, socket, etc, so has to server thread return result to client thread. If there needs passing large mount of data, the performance may turn worse. This is another reason why micro-kernel operating is of poor performance [1].
3. Because process virtual address space and process itself can not be separated, there exists the following problems:
 - The problem of data persistency: the non-persistency of a process makes the data in process virtual address space lose its persistency. To implement data persistency, data has to be read from file before threads access them, also it has to be written to file after threads has finished accessing them. Process virtual address space is NOT physical memory, although it is an abstract constructed through memory and PERSIST swapping device (usually one or more disks), it has lost the persistency of swapping device.

- The problem of complex data structure preservation: the meaning of a pointer in high-level languages is essentially an address in process virtual address space. Even if we have saved a pointer persistently, once the process holding the pointed data has exited, the pointer turns out meaningless. This means that complex data structure cannot be persistently preserved in file directly, it need to be translated into another data structure without pointer before it is saved to file.
- The problem of complex data structure share: a pointer belongs to a unique process virtual address space, a pointer defined in one process virtual address space is meaningless to another process, complex data structure that comprise pointers can not be shared among different processes. Although it has implemented memory-sharing and memory mapping file, traditional operating system can only implement the sharing of some sections of process virtual address space, it can not implement the sharing of complex data structure that comprise pointers among different applications.

To solve the problems listed above, we propose and are implementing a new operating system, which constructs abstracts different from the ones in traditional operating systems. In our operating system:

1. Only three abstracts are constructed, the three abstracts include the one for data computation (thread), the one for resource management (process), and the one for input and output (file). The function implemented by data storage abstract (process virtual address space) in traditional operating systems is implemented by files. Similar to memory mapping file in traditional operating systems, in our operating system, instructions access files directly, threads run directly on files.
2. The three abstracts, including the abstract for data computation (thread), abstract for resource management (process), and the abstract for input and output (file), are separated one from another. Operating system kernel provides Thread Migration system call, by calling it, a thread can enter from current process to another process, calling services in server process directly, and the thread can also return to original process and keep on running, just as calling system calls in traditional operating systems.

As our operating system integrates process virtual address space and file together, applications run directly on files, the virtual address spaces accessed by threads are file spaces; we call it Operating Systems Basing Virtual Address Spaces on Files (OS-BVASF). In OS-BVASF, when a thread need access some services implemented by other processes, the thread can get into other process directly by calling thread migration system call implemented by operating system kernel. When completed, the thread can also return to original process and keep on running by calling thread return system call.

Fig. 1 shows the abstracts constructed in OS-BVASF, as well as their relations. OS-BVASF can solve the problems due to not separating the three abstracts.

1. Because thread and process are separated, thread can enter directly into server process by thread migration system call, avoiding operations such as thread-switching, thread-sleeping and thread-waking-up that are necessary in traditional operating systems.

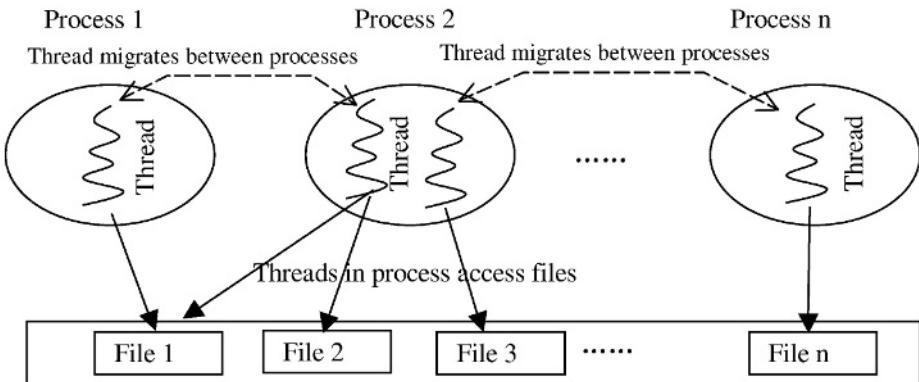


Fig. 1. The abstracts built by operating system and their relations

2. Because process virtual address space and file are integrated, all threads run on files, a thread can access all data in all files provided it has the accessing right. Safety and protection is implemented through file right control. The data that a thread can access is not confined within its process virtual address space. It's unnecessary to copy calling parameters and returning results between client and server when a client calls a service in server. This can help improve the performance of micro-kernel operating systems.
3. Because all threads run on files, the meaning of a pointer in high-level language is an address in file space, not the one in process virtual address space. A pointer is not confined within a process virtual space; it can be saved persistently in files directly, as well as be shared among different applications. This can avoid the problem of complex data structure preservation and the problem of complex data structure share.

2 The Architecture of OS-BVASF

2.1 The Hierarchy of OS-BVASF

Fig. 2 illustrates the hierarchy of OS-BVASF. In Fig. 2, the middle layer is OS-BVASF kernel, which implements thread migration, semaphore management, processor management and memory management. The System Call Interface, Interruption Handler, Address Mapping Component, and Switch Component in Fig. 2 is related to hardware platform, which should be re-implemented when OS-BVASF is migrated to other platforms. Upon the kernel are all sorts of servers, including communication server, file server, directory server, Linux emulator, etc. Applications are also servers existing in the form of process; any application can be called by other applications as a server.

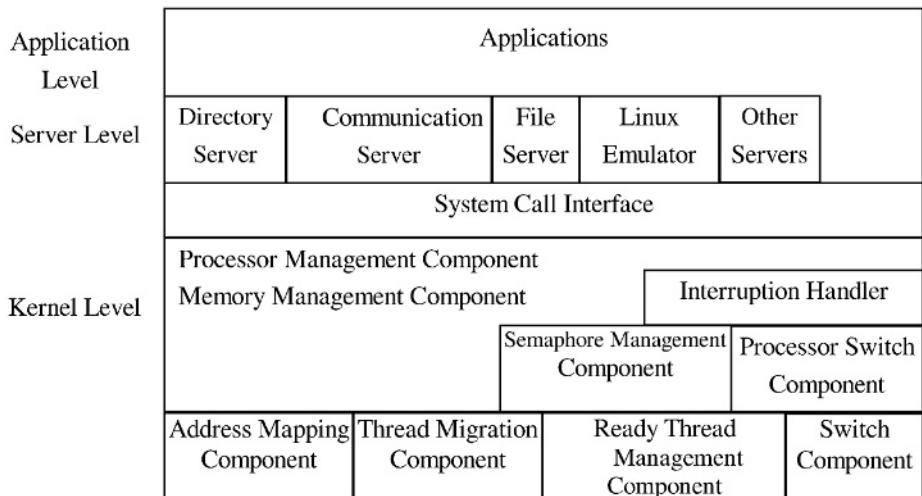


Fig. 2. The Architecture and hierarchy of OS-BVASF

2.2 The Function of OS-BVASF Kernel

In OS-BVASF, the following functions are implemented in the kernel: Thread-migration and thread-return, semaphore management, processor management and memory management. Through Thread-migration and thread-return, a thread can migrate between processes; through semaphore P operation and V operation, the mechanisms such as thread synchronization and thread mutex, are implemented. Processor management (process creation and exit, thread creation and exit, setting thread priority, etc.) and memory management (storage domain creation and exit, file open and close, etc.) are also implemented in kernel. File systems can exist either in the kernel as device drivers or outside the kernel as applications (servers).

2.3 File Servers, Communication Servers, and Applications

In OS-BVASF, all applications exist outside the kernel; they can be treated as virtual file servers, as a program module that can be called by other applications, and also as a program unit that implements application division and protection. Through thread migration, a thread can enter directly into another application, starting from the Initial Execution Point, the thread begins to run within the new process. When completed, the thread can also return to its original process and keep running.

The function of all sorts of applications (also can be treated as virtual file servers) is to manage persistent objects inside and provides virtual file call interfaces to other applications. By treating all applications as virtual file servers, OS-BVASF implements the function of file systems in traditional operating system, as well as the integration of file server and communication servers.

2.4 Linux Emulator

OS-BVASF implements Linux compatibility through Linux emulator as Mach does. A Linux thread calls Linux emulator through thread migration system call. Just as a Linux thread enters Linux operating system kernel, a thread in OS-BVASF enters Linux emulator, calls it execute function of Linux kernel. When completed, it will return to original application.

3 Thread Migration – Separating Data Computation and Resource Management One from the Other

3.1 The Conception of Thread Migration

In OS-BVASF, two processes relate to a thread, one is called owner process, it identifies to which process the thread belongs, the other is called current process, it identifies within which process the thread currently exists, its also identifies which process's resource the thread currently uses. Within its whole lifetime, thread can not change its owner process, whereas at any time, thread can change its current process through thread migration. When it calls thread migration system call, a thread changes its current process and resource environment, enters from one process to another, executes program within the new current process, when completed, it can also return to its original process. Thread migration is one of the main functions implemented within operating system kernel. One of the differences between OS-BVASF and traditional operating system is that through thread migration instead of message-passing or pipe, etc., a thread calls functions within others processes, thus prevents thread switching, thread sleeping and waking up, and improves performance.

Thread migration implements the separation of data computation (thread) and resource management (process).

3.2 Some Techniques Related to Thread Migration

Many operating system designers recognize the worse performance due to client/server model and adopt some techniques to solve this problem. From these techniques and systems, we propose the thread migration:

- Sun's Spring OS introduces a new conception: the Door, through which thread can enter from one domain to another. In Spring OS, Door is defined in IDL language, which makes threads be possible to enter into other process even on different computer or heterogeneous architecture. Sun's Spring OS is one of the operating systems from which we propose OS-BVASF thread migration technique.
- Grasshopper operating system constructs Loci (similar to thread in traditional operating system) which can migrate from one Container (similar to process in traditional operating system) to another. Loci abstract in Grasshopper operating system is persistent. It is from the Loci abstract in Grasshopper operating system that we propose the separation of data computation (thread)

and resource management (process), the implementing technique is thread migration

- MIT's Aegis operating system adopts the idea of Exokernel. Through exception forwarding and upcall, execution in Aegis can go from one process to another. From MIT's Aegis operating system, we propose exception processing in OS-BVASF.

3.3 Thread Initial Executing Context (TIEC) and Thread Initial Executing Point (TIEP)

When a thread enters into target process through thread migration, it starts running with a special CPU context called Thread Initial Executing context (TIEC). TIEC's instruction pointer is address of the first instruction that thread executes within target process, TIEC's registers store calling-parameters from source process.

In traditional operating systems, when a thread enters into kernel, it will start running at a special address, at which operating system kernel does switch and checks if the thread owns the accessing rights. If it does own, operating system kernel executes the called function, if not, operating system kernel refuses. This mechanism ensures the security and safety. Just as in traditional operating systems, in OS-BVASF, when a thread enters target process through thread migration system call, it will start running at Thread Initial Executing Point (TIEP), where target process will do some checks to ensure the thread is not a malicious one, and guarantee security and safety.

3.4 The Returning of Migrated Thread

When a thread has finished its task within target process, it can return to its original process through calling thread return system call. The execution flow of thread migration and thread return is similar to that of system call in traditional operating system. The difference is that in traditional operating system, the called object is operating system kernel, whereas in OS-BVASF, the called objects are processes outside operating system kernel. Instead of client/server model and message passing in traditional micro-kernel operating systems, OS-BVASF employs thread migration and thread return to put some function of traditional operating system outside operating system kernel. This mechanism can improve extensibility and performance of operating system.

4 Instructions Access Files Directly – Separating Data Storage from Data Computation and Resource Management

Similar to Memory Mapping File in traditional operating system [6], in OS-BVASF instructions access files directly, all applications run on files. Through Instructions Access Files Directly, OS-BVASF implements separating data storage (file) from data computation (thread) and resource management (process).

In computer systems, instructions access virtual address space. For example, at the protection mode of X86 processor, when it executes an assembly instruction MOV DS:[2000H],AX, X86 CPU will save the value in register EAX to a virtual address unit whose segment is identified by DS register and offset is 2000H. This unit may be either in physical memory or on exterior storage device. If it is not in physical memory, the executions of the instruction will trigger a memory fault exception; the exception handler will load data of the unit into physical memory, modifies segment table and page table, and restarts the instruction execution. The relation between virtual address space and physical memory as well as exterior storage device is maintained by operating system. If we define it as the relation between file address space and physical memory as well as exterior storage device, operating system will implement Instructions Access Files Directly. When programs are running, it seems that instructions access files directly, it also seems to the users that there exist no process virtual address space; programs are not running in process virtual address space, but in file address space. Instructions Access Files Directly can be implemented in the following method:

1. CPU employs segmentation or segmentation with paging virtual memory management, which a lot of CPU chip support. The virtual address space is two-dimensional, one is segment, and the other is offset within the segment.
2. Logically a segment in virtual address space corresponds to a section of file on file server.
3. Before an application tries to access data, file should first be opened. When open file, operating system constructs segment table and page table, return the segment ID to application as file handler. When application access data through the segment ID (file handler), it will actually access file data.
4. When thread is running and accessing a segment identified by the segment ID (file handler), if data does not exist in physical memory, a memory fault exception occurs, the exception handler will call file server to load data into physical memory, modifies segment table and page table, restart the excepted instruction to keep on running.
5. If memory resource gets scarce, operating system can select some page frames, write data in them to files on file servers, and releases these page frames.
6. In the end, applications close files, write all data in physical memory to file on file server, release memory resource as well as segment table and page table.

By the method described above, operating system implements Instructions Access Files Directly. When instructions access segments, it seems that the instructions access files. To applications, there is only conception of file, without that of process virtual address space. Applications do not run among process virtual address space, but on files.

Because the function of data storage is implemented not through process virtual address space, but through file abstract, and because files belong to no thread (data computation) or process (resource management), OS-BVASF separates data storage from data computation and resource management through Instructions Access Files Directly.

5 OS-BVASF Implementation and Performance Test

Upon X86 PC, we have implemented an OS-BVASF kernel. Its architecture is based upon Thread Migration and Instructions Access Files Directly. Upon the kernel, OS-BVASF applications run directly on files. Our development environment is Redhat Linux 7.1, Linux kernel version is 2.4.2, CPU is Pentium-IV, Clock Frequency is 1.8G, main memory capacity is 512M, and hard disk capacity is 40G.

The development language of OS-BVASF kernel is GCC under Linux environment. We also employ Linux LILO to install OS-BVASF kernel. But OS-BVASF kernel and exterior kernel applications run directly upon hardware, without the support from Linux. Our implement is not an emulator upon Linux environment; Linux is only our development environment.

5.1 Performance Test of Thread Migration

To appraise performance of Thread Migration, we test the time length needed for thread migration between two different processes. On the same hardware platform and Linux operating system, using pipe and message buffer, we also test the time length to do an execution switching between two different Linux processes. Test result is listed in Tab. 1.

OS-BVASF threads can execute at either kernel mode or user mode. If the source process and target process mode are different, so is the test result. That is the reason why Tab. 1 includes four items. But our test result doesn't show the difference too much.

Table1. Execution switching time length in OS-BVASF and Linux (unit: μ s)

In OS-BVASF through Thread Migration			
User-User	User-Kernel	Kernel-User	Kernel-Kernel
3.2	2.9	3.3	3.4
In Linux through Message Buffer and Pipe			
Through Message Buffer		Through Pipe	
6.3		6.4	

From Tab. 1 we can conclude that OS-BVASF execution switching time between two different processes is shorter than that of Linux.

5.2 Performance Test of Instructions Access Files Directly

The mechanism of Instructions Access Files Directly is very similar to that of Memory Mapping Files in traditional operating systems [6]. To appraise the performance of OS-BVASF memory management, we test the performance of Linux Memory Mapping File, and compare the test result to that of Instructions Access Files Directly in OS-BVASF. The test method is:

- Under Linux environment, we map file `/dev/zero` to a process virtual address space. Through repeatedly accessing mapped virtual pages, we test the time length for handling a page fault exception. File `/dev/zero` is a special character device that supports memory mapping file, all data read from it is zero.
- Under OS-BVASF environment, we open a file. Also through repeatedly accessing mapped virtual file pages, we test the time length for handling a page fault exception. In order to make the test comparable, we also set all data in file zero. We do the test when file is both managed by device driver within memory manager in kernel and file system outside kernel.

The time length for handling a page fault exception under Linux environment is 19.8ms under our test platform and environment. The test result under OS-BVASF environment is listed in Tab. 2. Once one memory fault exception occurs, OS-BVASF memory manager can set up address mapping for multi-page frames when handles the exception. The number of frames is identified when file is opened. In Tab. 2, the first row is the number of frames, the second row is test result when device driver within memory manager in kernel manages file, and the third row is test result when file is managed by file system outside kernel.

Table 2. Time length for handling memory fault exception (unit: μs)

Mapped Page Frame Number	File is managed by device driver within memory manager in kernel	File is managed by file system outside kernel
1	15.7	37.6
2	13.3	29.6
4	12.2	28.8
8	11.6	28.7
16	11.5	27.8
32	11.4	27.7
64	11.4	27.7

From Tab. 2, we can conclude that if file is managed by device driver within memory manager in kernel, the performance for handling memory fault exception is better than that of Linux, if by file system outside kernel, the performance is worse.

6 Conclusions

By replacing message-passing and client/server model with Thread Migration, and also by Instructions Access Files Directly, OS-BVASF abandons process virtual address space, it makes all programs run directly on files, implements Separation of data storage, data computation and resource management one from another in operating systems. Comparing with traditional operating systems, OS-BVASF can solve the three problems due to not separating data storage, data computation and resource management. Our implementation shows OS-BVASF feasible and of better performance.

References

1. Wang, Shiyou, Guo, Fushun. The performance effect of Microkernel operating system architecture, Computer Research and Development, Jan, 1999 (Chinese).
王世轴 郭福顺 等, 微核心操作系统的结构对性能的影响, 计算机研究与发展, 1999 年 1 月
2. James G. Mitchell, Jonathan J.Gibbons: An Overview of the Spring System Sun Microsystems Inc. 2550 Garcia Avenue, Mountain View Ca 94303.
3. Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, Francis Vaughan. Grasshopper: An Orthogonally Persistent Operating System. Computing Systems, 7(3), pp 289–312, Summer 1994
4. Alan Dearle, Rex di Bona, James Farrow, Frans Henskens, Anders Lindstrom, John Rosenberg, Francis Vaughan. Grasshopper: An Orthogonally Persistent Operating System. Grasshopper Technical Report GH-10. Department of Computer Science University of Adelaide S.A., 5001, Australia, 1994
5. Dawson R. Engler. The exokernel operating system architecture. Ph.D. thesis, Massachusetts Institute of Technology, October 1998.
6. M. Frans Kaashoek, Dawson R. Engler, Gregory R. Ganger, Héctor M. Briceno, Russell Hunt, David Mazières, Thomas Pinckney, Robert Grimm, John Jannotti, and Kenneth Mackenzie. Application performance and flexibility on exokernel systems. In the Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP '97), Saint-Mal, France, October 1997, pages 52–65.
7. W. Richard Stevens, Advanced Programming in the UNIX Environment, Addison Wesley Publishing Company, 1992, Translate by You, Jinyuan, China Machine Press, 1999.
尤晋元等译, UNIX 环境高级编程, 机械工业出版社, 2000.2, pp.307–311

A High Performance Design and Implementation of the Virtual Interface Architecture

Yu Chen, Zhenqiang Jiao, Jun Xie, Zhihui Du, Peng Liu, and Ziyu Zhu

Department of Computer Science,
Tsinghua University,
Bei Jing, P.R. China
chenyu@hpclab.cs.tsinghua.edu.cn

Abstract. Virtual Interface Architecture (VIA) is a communication protocol with low latency and high bandwidth, and it's a standard of user-level high-performance communication specification in cluster system. This paper analyzes the current development, principle and implementations of VIA, and presents a user-level high-performance VIA implementation - MyVIA based on Myrinet, which is comfortable with VIA specification. The paper first describes the design principle and framework of MyVIA, and then proposes new technologies for MyVIA. Experimental results of performance compare and analysis are presented. The results show the performance of MyVIA surpassed that of other implementations of VIA.

1 Introduction

High-Performance Cluster Computing is becoming more and more important currently. High performance interconnect systems are of key importance for effective cluster computing. However, traditional operating system prevents applications from taking advantage of the performance improvements of Giga-bit-per-second interconnects like Myrinet. Virtual Interface Architecture (VIA) specification is the industry standard for User Level Network communication. As a standard mechanism to deliver high performance directly to applications, and to eliminate overhead and inefficiencies common to operating systems and network stacks, VIA defines generic system architecture that is independent from the physical layer. It specifies a simple set of data structures and operations that move data between network-connected endpoints. It dose not propose the implementation details, so the researchers and developers could design their own implementation of VIA according to their special hardware and software environments.

Currently, two representatives of VIA implementations are Modular VIA [1] from National Energy Research Scientific Computing Center of the National Laboratories Lawrence Berkeley and Berkeley VIA [2] from U.C. Berkeley. Berkeley VIA is based on Myrinet hardware, and it could run on Solaris, Linux 2.2.x and Windows NT. Modular VIA could run on Linux and supported some high speed Ethernet card, such as Packet Engines GNIC-I/II Gigabit Ethernet cards, etc. The other implementations of VIA include GigaNet, IBM FirmVIA [3,4], VI-GM, etc. In this paper, a high-performance VIA implementation – MyVIA is introduced.

2 VIA Specification

VIA is the result of continuing development on user-level communication protocol [6]. It is a connection-oriented and point-to-point protocol. The logical structure of VIA mainly includes four components: VI provider, VI consumer, Virtual Interface (VI) and Complete Queues. The architecture of VIA is shown on Figure 1. VI provider consists of a VI capable NIC and a VI Kernel Agent. The VI capable NIC transfers data between hosts and network. The VI Kernel Agent runs as a device driver, and its main tasks are open/close Myrinet NIC, building/destroying VI, registering memory, connecting/disconnecting VI, managing interrupt events, etc. VI consumer consists of VIA applications and VIA user agent. User agent provides the consistent programming interface to applications (MPI, TCP/IP, etc.) and hides the details of different VIA implementation. The detail specification of VIA is shown in [7,8,9].

The main idea of VIA specification is to bypass OS and access the network interface directly in network communication. It exploits the hardware performance, eliminates the extra software overhead and redundant data copy operation, and improves the scalability of cluster. Furthermore, VIA specification keeps some vagueness and abstraction on VIA implementation, so VIA protocol can be implemented using any hardware, firmware or software based methods [10].

3 Design of MyVIA

3.1 Characteristics of MyVIA

In recent high-performance cluster environment, the performance of the inter-node communication is influenced by many factors, but the most important factor is the communication overhead of software. The overhead of software comes from varied sources, such as the complexity of communication protocol, the interruption count of triggering, the context switch of processes, routing mechanism, aggregation communication, redundant data copies, etc.

When the size of message is small, the average communication overhead of software for each byte is relatively high, and the peak bandwidth of network hardware could not be used effectively. According to the research results of Gusella [11] and our experiments on lots of parallel programs, we found the typical communication pattern in the typical cluster environment: the sizes of 80% messages are between 128 bytes and 4096 bytes and the sizes of other 8% messages are larger than 8192 bytes. Based on this phenomenon, the design of MyVIA not only pursues the peak bandwidth with large messages, but also focuses on the communication bandwidth with the small message size.

Compared to other VIA implementations, MyVIA system has these characteristics:

1. High communication performance and scalability: we proposed several technologies that could reach higher communication performance of VIA: User Translation Lookaside Buffer (TLB) on hosts; Pipeline management based on resources and DMA chain;Physical descriptor ring;Continuous host physical memory and variable NIC buffer management.

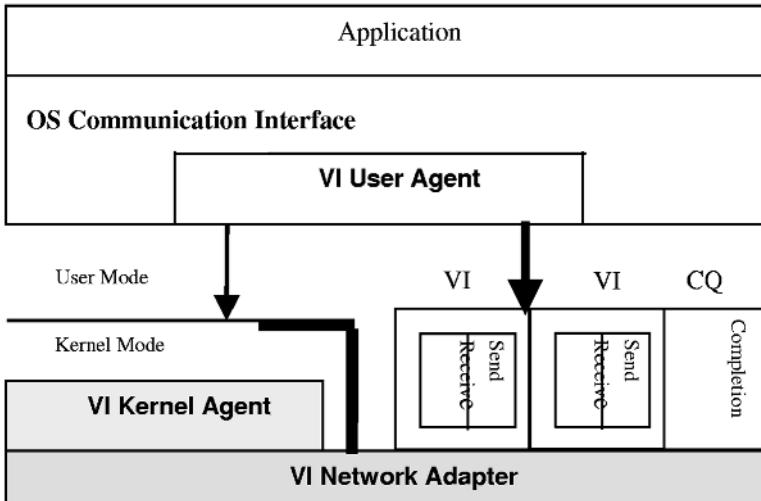


Fig.1. Architecture of VIA

2. Transparency: Although the whole hardware structure of Myrinet [12] LANai9 is not changed, LANai9 is better than LANai4 in every aspect. In order to get each communication hardware peak performance, we designed the NIC firmware agents for LANai9 and LANai4 separately. To decrease the modification on MyVIA software and make the NIC's differences as much as transparent to kernel agent layer and VIA library layer, a unified abstract hardware interface above NIC firmware agent is also built.

3.2 Architecture of MyVIA

MyVIA supports Myrinet hardwares based on LANai4 and LANai9. MyVIA mainly composes of three components and the main functions of each component are illustrated below

1. User Agent (UA) – user-oriented VIA library: It provides VIA program interface for users, including data sending, data receiving, communication polling and so on.
2. NIC Agent (NA) – NIC-oriented agent component: It manages VIA resources in NIC and the DMA sending/receiving, etc. NA is located in Myrinet NIC.
3. Kernel Agent (KA): It runs as a device driver. The main functions that KA provided include:
 - a. VI management
 - b. Memory Register and Protection
 - c. Device initiation and contact
 - d. Network and Connection Management

4 Implementations of MyVIA

The Berkeley VIA is an implementation of VIA specification on Myrinet LANai4/7/9. After analyzing and experiment, we find that Berkeley VIA does not exploit the performance of Myrinet hardware. In Berkeley VIA, many works are done by the Myrinet NIC, but the speed of CPU in Myrinet NIC is much slower than that of host CPU. As a result, the CPU in Myrinet NIC became the new "bottleneck" in the communication process. Furthermore, Berkeley VIA does not fully exploit pipelining parallelism on Myrinet hardware, and results in the relative low performance.

In order to reduce the executive time of instruction flow on the critical path of transferring messages, balance the functions between host CPU and hardware NIC for best pipelining, improve the scalability of VIA, new technologies were brought up. These technologies include User TLB on host, continuous host physical memory and varied NIC buffer management, pipeline management based on resources and DMA chain, physical descriptor ring.

After these technologies have been implemented, the communication performance of MyVIA is greatly enhanced. In our experiments on Myrinet LANai9, the point to point bandwidth for 4K bytes messages could reach 250MB/s, which is 96% peak hardware bandwidth, and the least one way point to point latency is 8.46 us. In the following section, these technologies of MyVIA will be analyzed.

4.1 Virtual-to-Physical Memory Address Translation

When NIC performs DMA operation to transfer data in host memory, NIC needs to know the physical address of data. There are three main virtual-to-physical memory address translation mechanisms :

1. TLB is located in host memory and host CPU executes address translation. This mechanism will add the burden of host CPU, but the NIC will not take care of address translation;
2. TLB is located in NIC and NIC performs address translation. This method needs more buffer in NIC and has more data transfer overheads;
3. TLB is located in host memory and NIC performs address translation. Berkeley VIA chooses the method. In Berkeley VIA, the TLB is located in host memory, and the TLB cache is located in NIC. This method voids the overhead of address translation in the host and is suitable for programs with good locality. However, if TLB cache has no desired address information, then the NIC will interrupt host to ask for new TLB entry. The performance of address translation in Berkeley VIA is limited by the size of the TLB cache in NIC and the locality of application.

According to the analysis and experiment, MyVIA adopted a technology that is similar to the first address translation method. When MyVIA kernel agent is loaded, it creates a Registered Memory Manage Table (RMMT) that is used to keep memory registration information (such as the TLB entries) for VIA applications. Once the VIA-enabled NIC is opened by VIA application, the RMMT is mapped to VIA application's user space by kernel agent, so the user agent can read content of RMMT. When VIA applications register user-level memory, the related TLB entries about the registered memory are stored in RMMT by kernel agent. When VIA applications send/receive messages which is located in registered memory space, the

VIA user agent looks up the physical address of messages from RMMT, then writes the physical addresses to NIC. The memory protection tag mechanism in VIA prevents VIA application from accessing RMMT and other memory arbitrarily.

There are three advantages of User TLB on host method. First, it has the best scalability since host memory is much larger than NIC memory and it can process large parallel applications effectively. Second, it is beyond the limit of program locality, but if the Berkeley VIA applications have poor locality, then there are a lot of MCP (Myrinet Control Program) interrupts to query TLB entries on host, and the performance of applications will be dropped 10%-40% usually. However, MyVIA's MCP in NIC doesn't bring up any interrupt because the TLB entries are located in RMMT. Third, this address translation overhead of communication critical path is smaller than others, since the related TLB entries can be directly visited by user agent and the host CPU is much faster than the NIC CPU.

4.2 Pipelining Communication Based on Resource for Myrinet LANai4

Myrinet LANai provides sufficient support for pipelining communication. Berkeley VIA for Myrinet LANai4 uses a little pipelining mechanism in the MCP program. Berkeley VIA for Myrinet LANai9 doesn't set the pipelining steps properly. Compared to Berkeley VIA, MyVIA achieves remarkable communication performance on Myrinet LANai9 NIC.

Because Myrinet LANai4 NIC does not support hardware DMA chain, MyVIA's pipelining communication for LANai4 adopts technology that uses limited-state automata based on resources (two buffers, three DMA engines and physical descriptors in NIC). The sending and receiving process in NIC are separated logically. With the help of physical descriptor ring mechanism, pipelining communication based on resource improves the communication performance of MyVIA greatly.

According to the internal of VIA specification and Myrinet NIC, the sending communication in NIC mainly includes four aspects: parse physical descriptor, receive data buffer from host, send data buffer to network and write status word to host. The receiving communication also includes four aspects:

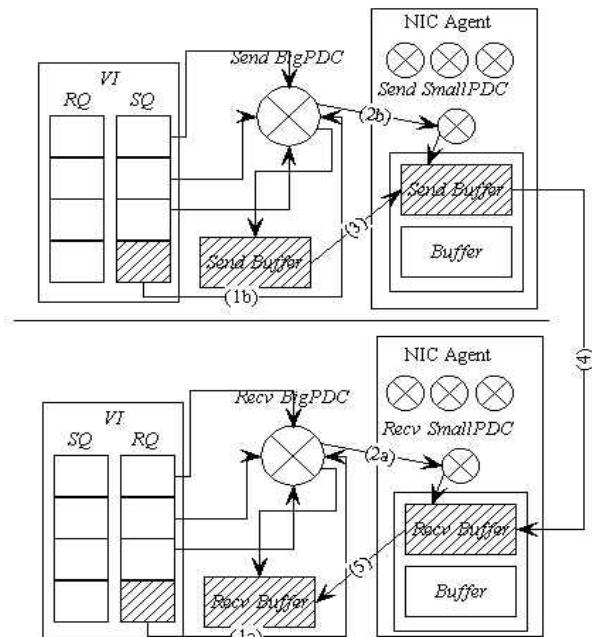


Fig. 2. Logical Communication of MyVIA

receive data from network, parse physical descriptor, transfer received data buffer to host and write status word to host. In order to improve the communication parallelism, MyVIA refines the sending process to seven pipelining stages and receiving process to five pipelining stages according to the NIC resources .

4.3 Pipelining Communication Based on DMA Chain for Myrinet LANai9

Myrinet LANai9 supports hardware DMA chain, so MyVIA can use DMA chain to decrease the DMA startup overheads of transferring data with discontinuous host physical pages in host, and simplify control flow in NIC agent. If MyVIA uses seven or five pipeline stages based on resources to communication in LANai9, then the benefit of DMA chain could not be exploited effectively, so MyVIA for LANai9 refines the sending and receiving processes to two-stage pipeline based on DMA chain, and uses double buffers to store communication data in different stage.

The first stage of sending is to get messages from host and put status word to host using one DMA chain operation, and the second stage is to transfer data to network. The first stage of receiving is to get messages from network, and the second stage is to put messages and status word to host using one DMA chain operation. The experiment results show that the two-stage pipelining communication based on DMA chain with double buffer technology improves bandwidth performance of MyVIA greatly.

Although the NIC agent implementation for LANai9 is different from that for LANai4, the fundamental ideas are identical. Therefore, an abstract interface on the NIC agent is developed, and the different NIC agents are transparent to Kernel Agent and User Agent.

The simplified components and logical communication process of MyVIA are shown in Figure 2. RQ is receiving descriptor queue, SQ is sending descriptor queue, BigPDC is the big ring of physical descriptor in host, SmallPDC is the small ring of physical descriptor in NIC. The 2a step should be prior to 2b in Figure 2, because the receiving operation of VIA must be prior to sending operation.

4.4 Doorbell Based on Physical Descriptor Ring

Doorbell mechanism is used to notify NIC that the VI consumer has sending or receiving requests. In Berkeley VIA, the sending and receiving work queue is located in host's memory, and a block of memory in NIC is reserved as the doorbell area of VIA. When the host send communication request, Berkeley VIA must write 2-bytes request information to NIC doorbell block by PIO (Programmed IO) operation, then the NIC gets 64-bytes VI descriptor from host by DMA operation and do next operation according to the information in VI descriptor. According above processes, the NIC CPU must do many works, alought it is much slower than host CPU.

To decrease communication overheads related with VIA doorbell, MyVIA reduces the size of VI descriptor and the times of data transferring of NIC.

Some information in VI descriptor isn't needed by NIC, so MyVIA convertes VI descriptor (64 bytes) to VI physical descriptor (16 bytes). When MyVIA NIC agent begins, it create N ($N=64\sim256$) NIC VI which consists of M^2 ($M=15\sim255$) VI physical descriptors for one VIA application (at most $K=1\sim4$ application), and kernel

agent remaps these NIC memory to kernel memory space. Because the structure of NIC VI is a ring of VI physical descriptors, We call one NIC VI is a small ring of VI physical descriptors.

When VIA application creates VI, MyVIA kernel agent creates a big ring which consists of L^2 ($L=128$) VI physical descriptors and remap small ring and big ring to user space. The parameters K, N, M and L could be changed according to the memory size of NIC and other factors.

When VIA application send/receive data, MyVIA user agent can directly write VI physical descriptor into NIC's small ring by PIO operation, and then NIC can get/put data from/to host according the information of VI physical descriptor. Therefore, MyVIA only needs one 16-bytes PIO operation to notify NIC, but Berkeley VIA needs one 2-bytes PIO and one 64-bytes DMA operation to do so. The parallel degree between host and NIC can be ensured by adjusting the big/small ring sizes.

Due to the speed of host CPU is much faster than that of NIC, the small ring in NIC could be filled. When this situation occurs, NIC agent sets a flag bit in host to denote that user agent stop to transfer physical descriptor, and the sequential physical descriptor will be saved in big ring temporarily. When the number of unused physical descriptors in small ring reaches a lower limit, and there are unprocessed physical descriptors in host's big ring, then NIC agent will interrupt host and request kernel agent to transfer more physical descriptors in big ring to NIC. The Relationship between big ring and small ring is shown in Figure 3.

The value of lower limit of the unused physical descriptors in small ring should be selected carefully, and the lengths of the big ring and small ring should be set by experiments. This value cannot be set too big, or too many interrupt will be brought up and affect the host greatly. It also can't be set too small, so the kernel agent on host can move physical descriptors from big ring to small ring in time when NIC agent is dealing with the rest physical descriptors in small ring. In a word, the lower limit value should ensure that Myrinet NIC can send/receive data with full speed, and does not affect the host.

From our experiment, when the value of lower limit and the length of the small ring are adjusted to certain values, the interrupts for physical descriptors from NIC have little effect on the performance of communication in Myrinet LANai4, and almost no interrupt will be produced in LANai9. Because of the small size of physical descriptor, MyVIA could support more VIs and support large-scale parallel applications.

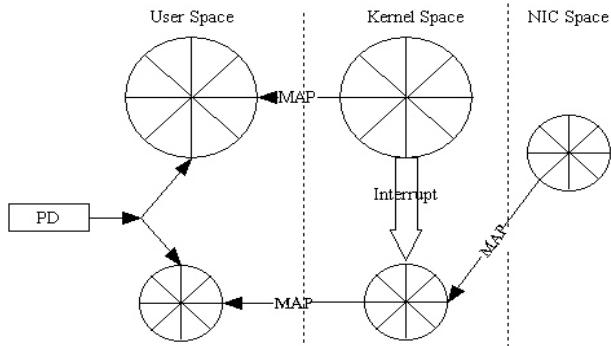


Fig.3. Physical Descriptor ring

4.5 Memory Management

In memory management, Berkeley VIA can only carry 64KB messages at most in one DMA operation with relative more overheads (such as more DMA operations, etc.). While Myrinet NIC doesn't support DMA operation with virtual address, and the user-level applications can only visit virtual memory with possibly discontinuous physical address, Berkeley VIA have to translate virtual address into physical address and perform several DMA operations (LANai4 doesn't support DMA chains). In order to reduce the overheads related memory management, we develop technology of continuous host physical memory allocation in Linux kernel and variable-length buffer management in NIC.

When VIA application ask MyVIA to malloc a buffer to store the sending/receiving data, MyVIA kernel agent mallocs a memory with continuous physical address to VIA application and records related TLB entries to RMMT. Because MyVIA ensure the VIA registered virtual memory in VIA application has continuous physical address space, one DMA operation could transfer the total registered virtual memory space, and leads to lower communication overhead. Furthermore, the variable-length buffer management on NIC Agents supports DMA operation with 120KB data block, and the block size is only limited by NIC memory size.

5 Performance of MyVIA

MyVIA and other VIA implementations have been tested in two environments. In environment A, there are two computing nodes connected with Myrinet LANai4.1 NIC and eight port Myrinet Switch. Each node has two Intel Pentium III 733 MHz, 512MB SDRAM memory, 33MHz/32bit PCI bus and runs Redhat6.2/7.2. In environment B, there are computing nodes connected with Myrinet LANai9.2 M3M-PCI64C-2 NIC and 32 port Myrinet-2000 Switch. Each node has two AMD AthlonMP 1600+ CPU, 512MB SDRAM memory, 66MHz/64bit PCI bus and runs Redhat6.2/7.2.

In environment A, we have tested one-way latency and bandwidth of both Berkeley VIA3.0 and MyVIA. The programs we tested are pingpong and window in Berkeley VIA3.0 package, and the results are shown in Figure 4. The program pingpong transfers the messages from one node to another for 1000 times and computes the average time as the result of one-way latency. From the result, we find that the result curves of Berkeley VIA3.0 and MyVIA are almost superposed. In order to test the bandwidth, the program window sends 2000 messages from one node to another continuously, and doesn't care whether the other side has received these messages. From the results, we could see that the bandwidth of MyVIA is lower than that of Berkeley VIA when the sending message is about less than 2000 bytes. However, when the sending message is more than 2000 bytes, the bandwidth of MyVIA will be higher. From experiment, the bandwidth of Berkeley VIA3.0 is 47.46 MB/s at the most, and that of MyVIA even can reach 76.2 MB/s.

From above, the performance of MyVIA will be lower than that of Berkeley VIA when sizes of messages are less than 2000 bytes. The reason lies in many aspects. The mostly important reason is that MyVIA adopts the pipelining communication based

on resource, and there are many competitions among the resources of NIC in the critical path of data transferring. The 33 MHz NIC CPU runs slowly to control the relative complex control logic. When sending small messages continuously, the parallel degree among host-NIC DMA, NIC-network DMA is not sufficient, so the overhead of control logic in NIC exceeds the benefits of pipeline parallelism. When messages sent are more than 2000 bytes, the advantage of pipelining communication in MyVIA will counteract and overcome the overhead of pipelining control logic in MyVIA, so the bandwidth of MyVIA is higher than that of Berkeley VIA.

In environment B, we tested GM 1.5, Berkeley VIA3.0 and MyVIA. When testing GM, we use program allsize in GM software package, and the results are shown in Figure 5. From the results, we found that because of the high performance of Myrinet LANai9 hardware and 66Mhz/64bit PCI bus, the bandwidth of all three packages can reach 250MB/s when sending 32KB messages. When messages sent are less than 8KB, the performance of MyVIA is two times higher than that of Berkeley VIA. Further more, when sending 4KB messages, MyVIA could reach 250 MB/s bandwidth and 8.46 us least one-way latency. The reason is that all kinds of optimize technology of MyVIA can sufficiently exert hardware performance of Myrinet LANai9 and improve the parallelism of components in LANai9. From the results, the performance of MyVIA is a little lower than that of GM1.5, the main reasons are that GM doesn't satisfy VIA criterion, and GM gets rid of some operations prescribed by

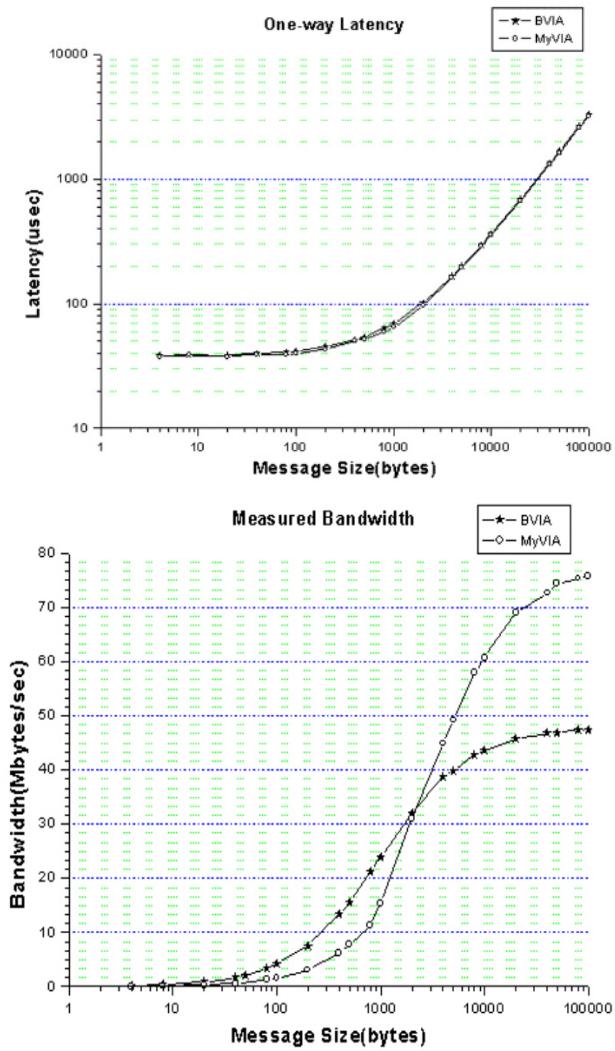


Fig. 4. Performance Analysis on MyVIA and Berkeley VIA in Myrinet LANai4

and 66Mhz/64bit PCI bus, the bandwidth of all three packages can reach 250MB/s when sending 32KB messages. When messages sent are less than 8KB, the performance of MyVIA is two times higher than that of Berkeley VIA. Further more, when sending 4KB messages, MyVIA could reach 250 MB/s bandwidth and 8.46 us least one-way latency. The reason is that all kinds of optimize technology of MyVIA can sufficiently exert hardware performance of Myrinet LANai9 and improve the parallelism of components in LANai9. From the results, the performance of MyVIA is a little lower than that of GM1.5, the main reasons are that GM doesn't satisfy VIA criterion, and GM gets rid of some operations prescribed by

VIA criterion. GM also makes sufficient use of some specialties of LANai9 to improve the communication performance.

We also tested M-VIA with Intel Pro100 Ethernet NIC, and the highest bandwidth we get is 11.6 MB/s and the one-way delay is 22 us. There is little difference on hardware performance between this card and 100Mb/s NIC. According to [1], the highest bandwidth of M-VIA is 59.7MB/s through GNIC-II Gigabit Ethernet NIC, and there is a big gap relative to highest performance of GNIC-II. According to analysis to source code, we found the reason is that M-VIA puts most VIA works on the host computer, and it cannot avoid the interruption while data transmission. These entirely make M-VIA doesn't exert the efficiency of high performance communication hardware.

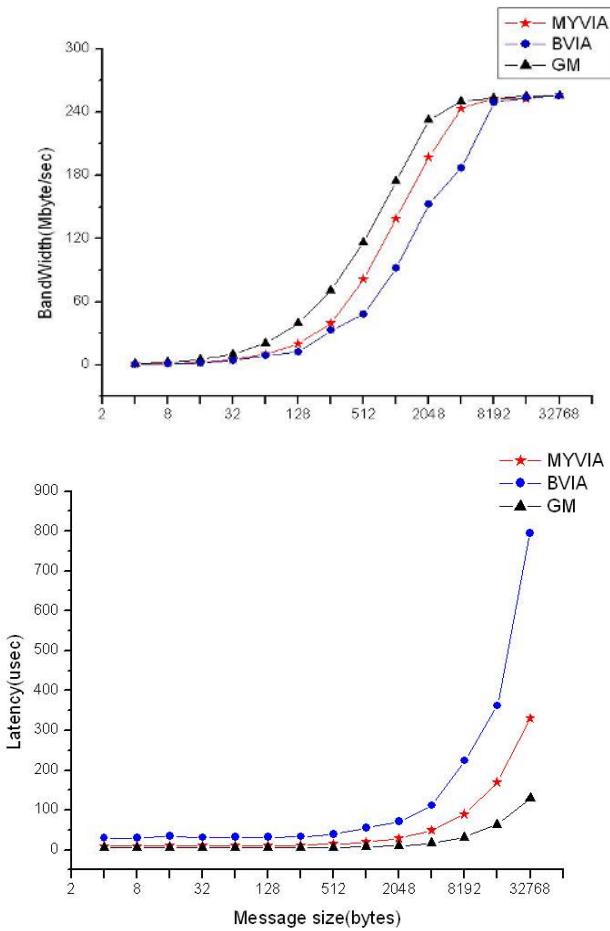


Fig. 5. Performance of MyVIA•BVIA and GM on LANai9

6 Conclusion

We design and implement MyVIA, a high performance and scalable VIA implementation. Our experiments indicate that MyVIA could exploit the high performance advantage of Myrinet hardware and improve the pipeline parallelism of communication effectively. We also developed a MPICH implementation for MyVIA. From the performance test of MPICH implementation for MyVIA, the RDMA

support of MyVIA is very important to improve the performance MPICH. Although the peak performance of MyVIA is very exciting, there are much works (Such as RDMA operations, etc.) still need to be done to improve the scalability and reliability of MyVIA.

Reference

1. P. Bozeman, B. Saphir, "A Modular High Performance Implementation of the Virtual Interface Architecture", Technical Report, Lawrence Berkeley National Lab., USA, 2000
2. Philip Buonadonna, Andrew Geweke, "An Implementation and Analysis of the Virtual Interface Architecture", Proceeding of SuperComputing 98, 1998
3. M. Banikazemi, V. Moorthy, etc, "Efficient Virtual Interface Architecture Support for IBM SP Switch Connected NT Clusters". International Parallel and Distributed Processing Symposium. 2000
4. M. Banikazemi, B. Abali, etc., Design Alternatives for Virtual Interface Architecture (VIA) and an Implementation on IBM Netfinity NT Cluster, Journal of Parallel and Distributed Computing, Special Issue on Clusters, 2000
5. W. Gropp, E. Lusk, etc., A high-performance, portable implementation of the MPI message passing interface standard, Parallel Computing, 22(6), 1996
6. Raoul A.F. Bhoedjang,Tim Rühl, Henri E. Bal , "User-Level Network Interface Protocols", 0018-9162/98/ IEEE 10,1998
7. Compaq, Intel and Microsoft Corporations, "Virtual Interface Architecture Specification. Version 1.0", 10, 1997
8. Intel Corporation, "Intel Virtual Interface (VI) Architecture Developer's Guide revision 1.0", 1998
9. Dave Dunning, Greg Regnier, etc, "The Virtual Interface Architecture", 0272-1732/98, IEEE Micro, 4, 1998
10. M. Banikaze, B. Abali, and D. K. Panda, "Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA)", Fourth Intel Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing, 2000
11. R.Gusella, A Measurement Study of Diskless Workstation Traffic on an Ethernet, IEEE Trans. Communications, Vol.38, No.9, Sept.1990
12. N. J. Boden, D. Cohen, etc,"Myrinet – A Gigabit-per-Second Local-Area Network", IEEE MICRO, Feb 1995

A Portable Debugger for PVM / MPI Programs on IA64 Cluster

Xi Qian, Jian Liu, and Weimin Zheng

Institute of HPC,
Department of Computer Science and Technology,
Tsinghua University
qx02@mails.tsinghua.edu.cn

Abstract. We describe the design and implementation of a portable debugger for PVM / MPI programs on IA64 cluster. The design is based on client-server model and uses existed sequential debugger gdb. Thus we make the debugger portable for most systems supported by gdb. The protocol used in client-server interactions is precisely designed to make it work well on both IA64 and other architecture. The debugger is implemented on IA64 architecture. It meets three general goals of High Performance Debugging Standard. In addition, its interface is easy to learn and to use.

1 Introduction

To achieve both advantages of IA64 and cluster, we can use an IA64 cluster in high performance computing. But we have to deal with a lot of problems when we choose MPI [1] or PVM [2]. Thus a parallel debugger on IA64 is in demand.

Few parallel debuggers can run across multi platform. And their poor usability is often been criticized. The HPD standard [4] is expected to solve this problem.

To achieve all these goals of this standard, we present a design and implementation of a portable debugger supporting PVM / MPI named Buster (**D**ebugger for **C**luster). It is implemented on IA64 cluster. Section 2 describes its well divided level model. Section 3 describes its implementation, especially mentions its protocol. Section 4 compares it with other related work. At last we give some conclusions.

2 Design of Buster (Debugger for Cluster)

There are two kinds of debugging [5]. The first is performance debugging, which is to adapt an algorithm to gain efficiency. The second is to establish the correctness of a program which is called correctness debugging. Buster is a correctness debugger.

To build a parallel debugger, you can use an existed sequential debugger or use own developed sequential debugger. The second way is more difficult. A debugger developed by this way depends on special platform, such as CM-5's Node Prism [6], Dolphinics's TotalView [7] and Intel's IPD [8]. The first way is more popular. P2D2 [3], Mantis [9], and Xmdb [10] are the typical examples using the first method.

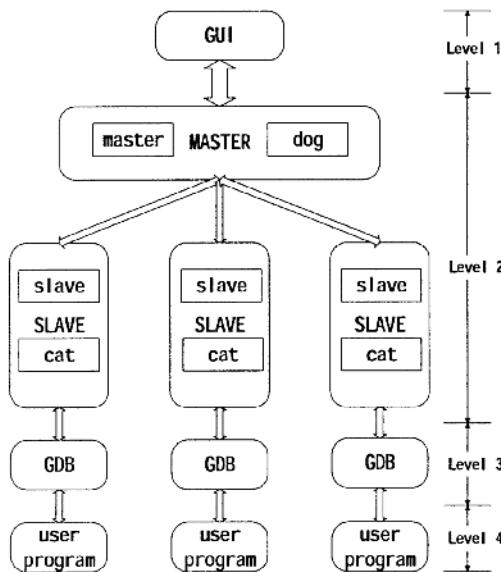


Fig. 1. The Design Level of Buster.

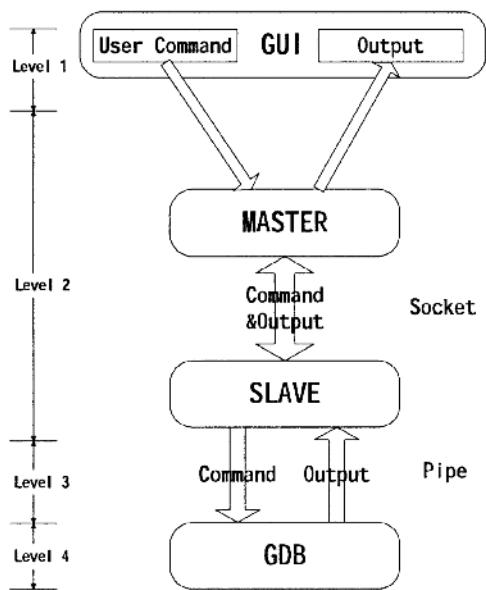


Fig. 2. The Data Flow of Buster

Buster is a parallel debugger of the second type. It is designed to support network computing environment with Linux system. Its debugging function is implemented by extending the functions of gdb. That makes the debugger more practical and robust.

The design model of Buster is divided into four levels. It is described in figure 1.

The first level is GUI level, with a friendly graphic user interface. It receives users command and shows the result. The second level is control and communication level. It is mainly divided into two module group, master and slave. Master is to control all slaves and slave is to control sequential debugger. The third level is sequential debugger level. It is to execute commands forward by slave. The lowest level is user's program to be debugged.

What users can see is only the GUI. The main data flow is described in figure 2. We will discuss the details in section 3.

The design model of Buster has following main characters:

- **Portability.** By using the leveled model and sequential debugger gdb, we achieve portability of Buster. Parallel debugger does not care how the debugger command is executed. We can switch sequential debugger easily.
- **Scalability.** The parallel debugger Buster can automatically detect process created by MPI / PVM. No matter how many processes are used, Buster can show one debugging window for each process and execute command for each process. User can get full scalability just limited by MPI / PVM and their machines.
- **Practicability.** Buster's user interface is very friendly. That can reduce difficulties caused by debugging large amount of processes in MPI / PVM.

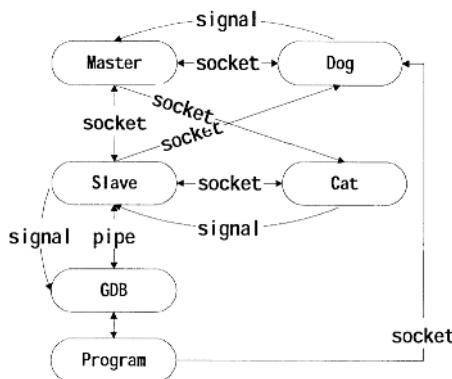


Fig. 3. The Communication between Modules

3 Implementation of Buster

Buster is a portable parallel debugger. Every module can be replaced with another module implemented the same communication protocol. We implemented these modules in ANSI C. It can be compiled and run on any Linux system with gdb.

3.1 Communication Protocol

To make Buster more robust and portable, we implement four modules for communication and control. Figure 3 describes their relationship:

To execute a synchronous command such as to set a breakpoint, master will forward the command to slave and wait until the command is executed by gdb and result is return to master by slave.

To execute an asynchronous command such as to run the program, master will forward the command to a slave and return without waiting. When user wants to make the program pause, master will forward the command to the cat. Then cat uses signal to interrupt the slave and make it execute the command.

When a slave wants to communicate with busy master, it will do the similar things.

3.2 The Support for Debugging PVM and MPI Programs

To use Buster with PVM program, user should insert "#include <buster.h>" into every source file, then compile them with -g option and link object files with libbuster.a. When pvm_spawn() is called by PVM program, it is replaced by trace_spawn(). This function spawns a load module and executes a slave. So the each process can be showed controlled by user.

In order to enable source level debugging for MPI program, user just needs to compile source codes with “-g” option. Master runs such command after a MPI program is chosen to debug: mpirun –buster.slave –np L/R address port program [arg1 arg2 ...]. In this way, all processes created by MPI can be controlled by user.

4 Comparison with Related Work

Three research works are similar with Buster: P2D2 [3], Mantis [9] and DCDB [11]. Mantis only supports debugging Split-C programs. DCDB is more portable. But its way to control slave may make the load of local machine huge. Buster just uses two processes on local machine to control all the processes. That balances the load.

TotalView [7] is only available on homogenous computer systems. Buster can support heterogeneous easily. Data parallel debugger Prism [6] is neither portable nor general-purpose. Buster can run at most platform includes IA64 cluster. Its design model is more general.

We have implemented Buster without gdb support before [11]. Each modification is hard. But now we need even no modification when we try to make it work on different platform.

5 Conclusion

This paper describes a parallel debugger for MPI / PVM implemented on IA64 cluster. It is implemented by extending function of gdb. So we can focus on the parallel control and communication between master and slave. Thus we make it portable, scalable and practical. It is easy to learn and easy to use. We have much to do to make it more practical. Some classic problem in parallel debugging will be considered further.

References

1. William Gropp, Ewing Lusk, and Anthony Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, the MIT Press, 1999.
2. Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek and Vaidy Sunderam, *PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing*, the MIT Press, 1994.
3. Doreen Cheng and Robert Hood, *A Portable Debugger for Parallel and Distributed Programs*, Proc. of Supercomputing 94', Nov. 1994.
4. High Performance Debugger Forum, HPD Version 1 Standard: Command Interface for Parallel Debuggers, <http://www.ptools.org/hpdf/draft/>, Sep. 1998.
5. Liu Jian, Shen Mei-Ming and Zheng Wei-Min, *Research on Perturbation Imposed on Parallel Programs by Debuggers*, CHINESE J. COMPUTERS, Vol. 25 No. 2, Feb. 2002.
6. Think Machines Corporation, *Prism 2.0 Release Notes*, May 1994.
7. Dolphin Interconnect Solutions, Inc., *TotalView Multiprocess Debugger, Release 3.7*, <http://www.dolphinics.com/>
8. Intel Corporation, *iPSC/2 and iPSC/860 Interactive Parallel Debugger Manual*, April 1991.
9. S.S.Lumetta, *Mantis: A Debugger for the Split-C Language*, University of California at Berkley, Tech. Report #CSD-95-865, 1995.
10. Damodaran-Kamal, *Xmdb Version 1.0 User Manual 1.2*, Los Alamos National Laboratory, 1995.
11. Du Shu, Chen Xiao-Pen, Wang Dong-sheng and Zheng Wei-Min, *Buster: A Portable Parallel Debugger, Mini-Micro System*, Vol. 22 No.10, Oct. 2001

Optimization of Asynchronous Volume Replication Protocol

Huanqing Dong and Zhanhuai Li

Dept. of Computer Science and Engineering,
Northwestern Polytechnic University,
No.127 West Youyi Road, Xi'an, Shaanxi, China 710072
hqdong@co-think.com
lizhh@nwpu.edu.cn

Abstract. To keep data consistency on backup volume, traditional asynchronous volume replication protocol has to transmit every block changed on the Primary to the Backup and keep block update order during replication. This paper presents an enhanced asynchronous volume replication protocol, which can decrease the number of blocks required to be transmitted and updated to the Backup during replication. The enhancement is especially useful when network bandwidth is a major consideration. Analysis shows that in most cases the new protocol can keep the same data consistency as traditional protocols.

1 Introduction

A common approach to building fault-tolerant distributed systems is to replicate servers that fail independently. The objective is to give the clients the illusion of service that is provided by a single server. The main approaches for structuring fault-tolerant servers are active (state-machine) replication and passive (primary-backup) replication [1]. In active replication, a group of identical servers maintain copies of the system state. Client update operations are applied atomically to all of the replicas so that after detecting a server failure, the remaining servers can continue the service. Passive replication, on the other hand, distinguishes one replica as the primary server, which handles all client requests. A write operation at the primary server invokes the transmission of an update message to the backup servers. If the primary fails, a failover occurs and one of the backups becomes the new primary.

This paper focuses on the asynchronous primary-backup volume replication approach. Unlike database replication [3] or file replication [4], volume replication operates at block level, so it is unable to guarantee the consistency of volume contents from a database management system or file system viewpoint [6]. Only when a database management system or file system has no transactions outstanding and no updated data in cache that has not yet been written to disk, its on-volume image is said to be internally consistent. Though volume replication protocol is unable to ensure internal consistency on a volume, it must ensure that the Backup volume always represents a state of the primary volume at some previous point by keeping

update order [1][6]. Otherwise, in case of a primary disaster, an application failed over to the Backup may not be able to recover.

Traditional asynchronous volume replication protocol keeps data consistency on the Backup by strictly maintaining block update order, each block update are applied on the Backup in the same order as they are received by the primary.

In this paper, we presented an enhanced asynchronous volume replication protocol (EAVRP), which can decrease the number of blocks required to be transmitted and updated to backup volume during replication. The main points are:

1. On the primary, for those blocks that have been updated more than 1 times in a given window, only the latest block image will be transmitted to the Backup, and all the blocks to be transmitted will be marked as an Atomic Block Group (ABG);
2. On the Backup, the commission of an ABG is carried out as an atomic operation, all blocks in the ABG are committed as a single update.

We compared our protocol with the traditional protocol. The comparison result shows that our protocol is more network bandwidth saving than the traditional one, and in most cases our protocol can maintain consistency as well as the traditional one does.

This paper is organized as the following. The next section discusses related work on asynchronous replication models and consistency metrics. Section 2 describes the EAVRP protocol by highlighting how it reduces the number of blocks to be transmitted during replication on the Primary. Section 3 compares the new protocol with the traditional protocol from the view of network bandwidth consumption and consistency. Section 4 concludes this paper.

2 Related Works

2.1 Asynchronous Replication Models

Many of past work are related to asynchronous replication model. An asynchronous volume replication protocol adopted by VERITAS Volume Replicator was described in [6], and a database replication protocol was described in [7]. Both protocols are focused on primary-backup replication model.

The protocol presented in [6] maintains a log in persistent storage on the Primary, the log contains all the blocks that have been updated to the primary volume and have not been acknowledged by the Backup, ordered by timestamps, which indicates the time when the block request arrived; the protocol presented in [7] maintains a vector on the primary, which contains all the transaction tuples that have been executed on the primary server and have not been acknowledged by the Backup, ordered by the time they are executed.

The protocol presented in this paper is based on the Primary-Backup replication model, and it is similar to the protocol presented in [6] [7] in the way of maintaining a log or queue in the Primary, the contribution of this paper is the method of saving network bandwidth.

2.2 Consistency Metrics

Window-inconsistency [1] is a metric to indicate the staleness of the Backup. Timestamps $\tau^P(t)$ and $\tau^B(t)$ identify successive versions of object O at the primary and backup sites, respectively. At time t , the Primary P has a copy of O written by the application at time $\tau^P(t)$, while the Backup B stores a possibly older, version originally written on P at time $\tau^B(t)$, i.e., at time t , B represents the state of P at $\tau^B(t)$. At time t , a backup copy of object O has window-inconsistency $t - t'$, where t' is the maximum time such that $t' \leq t$ and $\tau^P(t') = \tau^B(t')$. The consistency metric *Staleness* presented in [5] is similar to Window-inconsistency. In this paper, we adopted the *Window-Inconsistency* metric to evaluate our EAVRP protocol.

3 The EAVRP Protocol

Each site (P and B) has two threads related to volume replication, namely *Primary Volume Update Thread* (PVUT), *Primary Transmission Thread* (PTT), *Backup Volume Update Thread* (BVUT) and *Backup Transmission Thread* (BTT). The architecture is shown in Fig. 1.

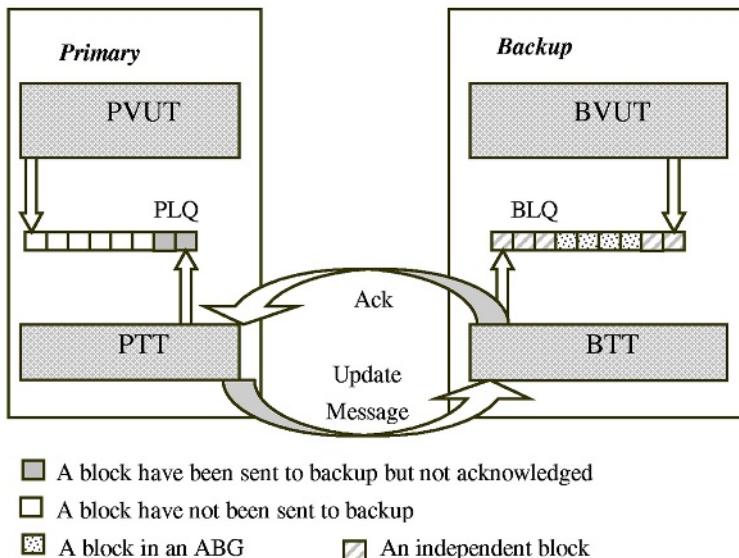


Fig. 1. Architecture of EAVRP protocol

3.1 Operations on the Primary

Definition 1. We define the *Primary Logical Queue* $PLQ = \{b_1, b_2, \dots, b_N\}$ to be the logical queue of blocks to be updated to B, N is the quantity of blocks currently in PLQ, blocks in PLQ are sorted by their timestamps in ascendant order. Physically, the PLQ can be arranged on a fixed length of continuous space.

Let S denote the quantity of blocks currently in PLQ that have been sent to B but not acknowledged, so the first block to be sent to B should be b_{S+1} .

Definition 2. We Define *Current Block Set* $CBS = \{b_i \mid S < i \leq S + M\}$ to be the set of blocks from where we try to find duplicate block updates. The constant M represents the max quantity of block in an CBS, in the following description, we assume $M \leq N - S$ is always TRUE.

Definition 3. Let n_i denote the block number (on volume) of b_i , We define *Current Duplicate Block Set* $CDBS = \{(b_i, b_j) \mid b_i, b_j \in CBS, \text{and } n_i = n_j\}$ to be a set of block pairs which represent duplicate block updates in the CBS.

Since both threads (PVUT and PTT) will operate on the PLQ independently, a lock mechanism is required to avoid conflict, each thread must obtain the lock before writing PLQ and must release it later. The lock mechanism is leaved out in this paper.

Upon accepting a block update request from the upper layer (File System or Database), PVUT will take the following steps:

1. Add the block in the update request to the end of PLQ;
2. Execute the update request (Update the block to logical volume), and return to the upper layer.

PTT operates as the following steps:

1. If exists $j, (b_{S+1}, b_j) \in CDBS$, then let $l = \{\max(w) \mid \exists i, (b_i, b_w) \in CDBS\}$, and $ABGOF = \{b_k \mid S + 1 \leq k \leq l\}$ denote the original field of an ABG, figure out $ABG = \{b_k \mid b_k \in ABGOF, \text{and } \forall i, k < i \leq l, (b_k, b_i) \notin CDBS\}$, which denotes the set of blocks to be sent to B and should be committed on B as an atomic operation. For duplicate block updates of a same block, only the latest block image will be included in the ABG.
2. Otherwise, send b_{S+1} as an independent block;
3. Check for arriving of acknowledge from B, if an acknowledge of an ABG arrives, remove all the blocks in corresponding ABGOF from PLQ; Otherwise, remove the acknowledged block from the head of PLQ;
4. Go to step 1.

3.2 Operations on the Backup

The BTT thread repeatedly receives block updates from P (PTT) and put them in the BLQ, blocks in BLQ are sorted by their timestamps in ascendant order.

To avoid out of order updates, the BVUT thread will commit all the blocks in an ABG as an atomic operation, and acknowledge the ABG as a whole; for independent blocks, BVUT will commit them according to their timestamps and acknowledge them one by one.

3.3 Failure Handling and Recovery

In case of a communication link error, some data (including block updates sent by the Primary and acknowledgments sent by the Backup) may be lost. But since each block will be kept in PLQ until the Backup has acknowledged it, The Primary will retransmit block updates from the first unacknowledged one when link recovers. If a communication link error occurs during replication of an ABG, then all the blocks in the ABG must be transmitted after link recovering.

Handling of other failures including overflow of PLQ, failure of the Primary Volume, etc., which are similar to that of [6], are leaved out in this paper.

4 Comparison of EAVRP with the Traditional Protocol

4.1 Network Bandwidth Consumption Comparison

As shown in Fig2, it is clear that EAVRP can reduce the quantity of blocks to be transmitted to the Backup when an ABG exists. In the example, 8 consecutive block updates (M was set to 8) were made to P, but only 5 blocks (In the form of an ABG) were transmitted to the Backup (B). In contrast, 8 blocks must be transmitted to the Backup (B') when traditional protocol is adopted. It can be deduced that the larger value of M is, the more possible that duplicate block updates would happen, so the fewer blocks are required to be transmitted to the Backup. But on the other hand, larger value of M will cause larger inconsistency window between the Primary and the Backup.

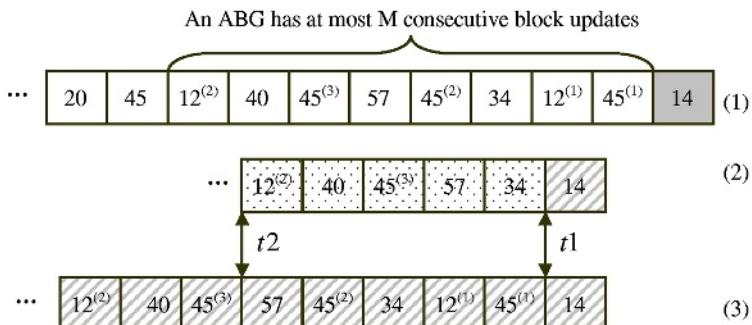


Fig. 2. (1) PLQ on the Primary; (2) BLQ on the Backup (B) when EAVRP protocol is adopted; (3) BLQ on the Backup (B') when traditional protocol is adopted

4.2 Window-Inconsistency Comparison

In this section, we will compare the staleness of the Backup (B) when using EAVRP and the staleness of the Backup (B') when using traditional protocol.

Definition 4. At time t , the backup has window-inconsistency $W(t) = t - t'$, where t' is the maximum time such that $t' \leq t$ and $\tau^P(t') = \tau^B(t)$.

The value of $W(t)$ indicates the staleness of the Backup. Let $W_E(t)$ denote the window-inconsistency of the backup when EAVRP is used, and $W_T(t)$ denote the window-inconsistency of the backup when traditional protocol is used. Let's Assume $W_E(t) = W_T(t)$ before the ABG was transmitted to the Backup (at time $t1$, shown in Fig2).

1. Since B couldn't commit the ABG until all of its blocks have arrived, no update would happen on B during period $(t1, t2)$, while B' may update blocks incrementally. As shown in Fig2, during the period $(t1, t2)$, $W_E(t)$ will be greater than $W_T(t)$, i.e., B is staler than B' during period $(t1, t2)$.
2. Once the atomic update of the ABG is finished, B should have reflected all block updates in the ABG, $W_E(t)$ will be less than $W_T(t)$, i.e., B' is staler than B after $t2$.

The major disadvantage of EAVRP is that, if a primary fails when only part of the blocks in the ABG has been transmitted to the Backup, all the blocks in the ABG can't be committed on the Backup, so $W_E(t)$ will be larger than $W_T(t)$, it means more data loss if a recovery is to be done on the Backup at this time. Clearly, The distance between $W_E(t)$ and $W_T(t)$ is related to M , if M were set to 1, EAVRP would act the same as the traditional protocol do, and $W_E(t)$ would always be equal to $W_T(t)$.

5 Conclusion and Future Work

Traditionally, asynchronous volume replication protocol have to transmit every block changed on the Primary to the Backup, and each block must be updated to the Backup in the same order they are updated to the Primary. For replication in a WAN environment, in which case network bandwidth is a major consideration, it is very meaningful to decrease network traffic of replication procedure. Our protocol meet this goal by omitting some duplicate block updates when spreading updates to the Backup. The advantage of our protocol is that it can save network bandwidth, and the disadvantage is that the Backup may be staler than when traditional protocol is used.

Since the update order among blocks inside an ABG needn't be hold during the atomic commission, it is possible to optimize the atomic operation procedure in order to achieve better performance. We will investigate this issue in the future work.

The other issue we plan to study is how to decide the appropriate value of M . To do this, we must tradeoff between staleness of the Backup and network bandwidth consumption.

References

1. Ashish Mehra and Jennifer Rexford. Design and Evaluation of a Window-Consistent Replication Service[C]. IEEE Transactions on Computer, Vol. 46, NO.9, Sep 1997
2. Hengming Zou and Farnam Jahanian. Real-Time Primary-Backup Replication with Temporal Consistency Guarantees[C], In Proceedings of the IEEE International Conference on Distributed Computing Systems, June 1998
3. Rainer Gallersdorfer and Matthias Nicola. Improving Performance in Replicated Databases through Relaxed Coherency[C]. VLDB 95
4. Hurley, R.T and Soon Aun Yeap. File migration and file replication: a symbiotic relationship[C]. IEEE Transactions on Parallel and Distributed Systems, Volume 7, Issue 6, Jun 1996. Page(s): 578–586
5. Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services[C]. In Proceedings of the 18th ACM symposium on operating systems principles, October 2001. Page(s): 29–42
6. Paul Massiglia. VERITAS Volume Replication and Oracle Databases [OL].
<http://www.biffsocko.com/whitepapers/OracleAndVVR.pdf>, VERITAS Corporation, 2000
7. Yang Zhaohong and Gong Yunzhan. A new primary lazy replica update protocol [J]. Computer Science, Vol29, No.8, Aug 2002.

Predicate Analysis Based on Path Information

Li Shen, Zhiying Wang, and Jianzhuang Lu

Department of Computer Science and Technology,
National University of Defense Technology,
Changsha, 410073 China
`{lishen, zywang}@nudt.edu.cn`

Abstract. Predicated execution leads new challenges to traditional optimizers. Limited by predicate representations, current predicate analysis methods could only perform locally or have low efficiency. A new representation based on path information is proposed in this paper, based on which a global predicate relation query system is constructed. Experiment results indicate that precise and efficient global predicate analysis could be achieved with this system and the performance of optimized codes can also be improved.

1 Introduction

Predicated execution [1] can exploit ILP effectively. Research results indicate that with predicated execution, ILP can be increased by 3 times [3] and 27% branches and 56% misprediction can be eliminated [4,5]. But limited by predicate representations, current methods, such as Predicate Hierarchy Graph [3], P-facts [6] and Predication Partition Graph [7,8], can only perform local analysis because they just keep predicate usage but omit their definitions, which should be rebuilt before analysis so that the complexity of optimization is increased. This paper presents a predicate representation based on path information. It can convert predicate definitions into domain codes, which can be kept during optimization and the reconstruction is avoided. Based on it a global predicate relation query system is constructed. Experiment results show that precise and efficient global predicate analysis can be achieved with this system and the performance of optimization codes can also be improved.

2 Predicate Representation Based on Path Information

Accurate and efficient predicate analysis relies heavily on predicate representations.

Definition1. *Control Flow Graph (CFG)* $G = \langle N, E \rangle$ of a procedure is a directed graph. Each element S in N is a basic block and $E = \{ \langle S_i, S_j \rangle \mid \text{control flow can enter } S_j \text{ from } S_i \text{ in } G \}$.

Each node $S \in N$ has at most 2 successors. If there is no branch between S_i and S_j , then edge $S_i S_j$ is labeled blank. If there is a branch between them and control flow enters S_j from S_i when branch conditional is true, it is labeled T, or its label is F.

Definition2. *Path code* is a string of 0 and 1, which can be defined using induction, where \parallel means string concatenation:

1. The path code for the first code in a path is empty, denote as *nil*.
2. For each node S in the path other than the first, let R be S 's predecessor in the path. If the edge from R to S is unlabeled, then $PC(S) = PC(R)$. If it is labeled F , then $PC(S) = PC(R) \parallel 0$. If the edge is labeled T , then $PC(S) = PC(R) \parallel 1$.

Definition3. For each $S \in N$, the *domain* of S is $D = \{P \mid P \text{ is a path in } G \text{ and } S \text{ lies in } P\}$, and its *domain code* $DC(S) = \{PC(S) \mid P \in D\}$.

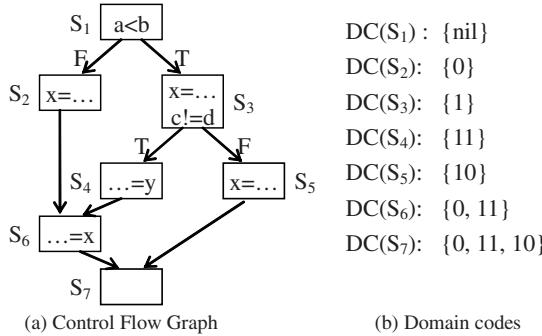


Fig. 1. Control Flow Graph and domain codes of its nodes. (a) Control flow graph. (b) Domain code of each node.

Fig. 1 shows a CFG and domain codes of its nodes. For each node S , each bit of $PC(S)$ indicates a branch condition when control flow reaches S from entry node. $DC(S)$ is {nil} implies that there is no branch from entry to S and the predicate of S is always true. Predicate definitions can be extracted from domain codes of corresponding basic blocks. Encoding can be performed during if-conversion [2]. Following are some important relations among path codes and domain codes.

Definition4. For two path codes PC_1 and PC_2 , $L = \min(\text{length}(PC_1), \text{length}(PC_2))$, $PC_1[i] = PC_2[i] (1 \leq i \leq L)$, if $L = \text{length}(PC_1)$, then PC_1 is included in PC_2 , denoted as $PC_1 \subseteq PC_2$; otherwise PC_1 contains PC_2 , denoted as $PC_1 \supseteq PC_2$.

Definition5. For two domain codes DC_1 and DC_2 , if $\forall PC_1 \in DC_1, \exists PC_2 \in DC_2 \Rightarrow PC_1 \subseteq PC_2$, then DC_1 is inclusive in DC_2 , denoted as $DC_1 \subseteq DC_2$.

Definition6. The intersection of two domain codes DC_1 and DC_2 can be defined as: $DC_1 \cap DC_2 = \{PC_1 \mid PC_1 \in DC_1, \exists PC_2 \in DC_2, PC_1 \subseteq PC_2\} \cup \{PC_2 \mid PC_2 \in DC_2, \exists PC_1 \in DC_1, PC_2 \subseteq PC_1\}$.

Domain codes have some important properties. Property 1 shows how to decide disjoint relation, property 2 and 3 show how to decide imply relation.

Property1. For two nodes S_1 and S_2 , PRE_1 and PRE_2 are their predicates respectively. Both PRE_1 and PRE_2 can be true if and only if the intersection of their domain codes $DC_1 \cap DC_2$ is not empty.

Property2. For two nodes S_1 and S_2 , PRE_1 and PRE_2 are their predicates respectively. Suppose the domain codes of them are DC_1 and DC_2 , if $PRE_2 \Rightarrow PRE_1$, then $DC_1 \subseteq DC_2$.

Property3. For two nodes S_1 and S_2 , PRE_1 and PRE_2 are their predicates respectively. If $DC_1 \subseteq DC_2$, and $|DC_1| \geq |DC_2|$, then $PRE_2 \Rightarrow PRE_1$.

Elements in domain codes can be combined if some conditions are satisfied. For example, domain code of S_7 in Figure 1(a) corresponds to three branch conditions: $(a < b)$, $!(a < b) \wedge (c = d)$ and $(a < b) \wedge (c \neq d)$. $(a < b) \vee (!(a < b) \wedge (c = d)) \vee ((a < b) \wedge (c \neq d)) = \text{true}$, thus the predicate of S_7 is always true and $DC(S_7)$ is $\{\text{nil}\}$.

Theorem1: For two path codes $PC_1, PC_2 \in DC(S)$, if following conditions are true:

1. $\text{length}(PC_1) = \text{length}(PC_2)$
2. $PC_1[\text{length}(PC_1)] = !PC_2[\text{length}(PC_2)]$
3. $PC_1[i] = PC_2[i]$ ($1 \leq i \leq \text{length}(PC_1)-1$)

Lets $PC' = \text{copy}(PC_1, 1, \text{length}(PC_1)-1)$, then DC_S is equivalent to $DC_S' = DC_S - \{PC_1, PC_2\} + \{PC'\}$. Function *copy* returns a substring of *str* from *pos*.

3 Predicate Analysis

In non-predicated codes, CFG specifies the fetch condition of each instruction, equivalent to the execute condition if there is no speculation. Compilers must consider an instruction's position in CFG relative to others to determine if a particular transformation is legal. But in predicated execution, an instruction can be executed only after it was fetched and its predicate is true, where CFG only indicates fetch condition. So both fetch conditions and predicates should be considered in predicate optimization and instruction relations must be revisited according to their predicates. For example, let *fdom*(fetch condition) assume the traditional definition of dominance: $I_1 fdom I_2$ iff each path from entry node to I_2 includes I_1 . But it should be modified to *edom*(execution condition) as following in predicated execution: $I_1 edom I_2$ iff $I_1 fdom I_2$ and $PRE_1 \supseteq PRE_2$.

Two predicate relationships need to note in predicated execution: disjunction and implication. According to section 2, they can be determined by analyzing domain codes relationships. Five interface functions are defined in our predicate query system:

1. **in**(PC_1, PC_2): Both PC_1 and PC_2 are path codes. It decides whether PC_1 is included in PC_2 . It returns true if $PC_1 \subseteq PC_2$, otherwise it returns false.
2. **in**(PC, DC): PC is a path code while DC is a domain code. It decides whether $PC \in DC$ is true. If $\exists PC_2 \in DC$ and $PC \subseteq PC_2$, it returns true, or it returns false.
3. **is_disjoint**(DC_1, DC_2): Both DC_1 and DC_2 are domain codes. It decides whether the intersection of them $DC_1 \cap DC_2$ is Φ . It returns true when $DC_1 \cap DC_2$ is Φ , false otherwise.
4. **subset**(DC_1, DC_2): Both DC_1 and DC_2 are domain codes. It decides whether DC_1 is subset of DC_2 . It will return true if $DC_1 \subseteq DC_2$, false otherwise.
5. **equivalent**(DC_1, DC_2): Both DC_1 and DC_2 are domain codes. It decides whether DC_1 is equivalent to DC_2 . If $DC_1 = DC_2$, it returns true, or it returns false.

Global analysis is necessary to improve the result of optimization. Domain codes should be extended to perform global analysis. A program is divided into hyperblocks after if-conversion [2] and they can also be encoded according to definition 2 and 3. Domain codes of hyperblocks indicate predicate relations among hyperblocks.

Following property can be proven: $DC(H_1) \cap DC(H_2)$ is empty \Leftrightarrow for two predicate P_1 from H_1 and P_2 from H_2 , $P_1 \wedge P_2 = \text{false}$. Tuple $\langle DC(H), DC(S) \rangle$ is used to represent domain code of basic block S in hyperblock H . Global predicate analysis can be performed as following:

1. If two predicates locate in the same hyperblock, interface functions above should be used to judge their relations.
2. Relation among hyperblocks should be judged. If they cannot exist in any path, the two predicates cannot be true at one time. If there is at least one path holding them and $H_1 \text{ edom } H_2$, then $\text{PRE}_1 \text{ edom } \text{PRE}_2$ too.

4 Performance

A set of real programs are selected to evaluate its performance. All benchmarks are compiled by IMPACT2.36, which is run on a RedHat Linux 8.0 system operating at a clock frequency of 1400MHz with 256MB RAM. Three relations, is_disjoint, subset and equivalent, between each pair of predicates in a CFG are tested. There are 837,315 query operations altogether, which are accomplished in 2.17s. About 6.3% of all query operations can lead to performance improvement, and complier could only produce conservative results without these queries.

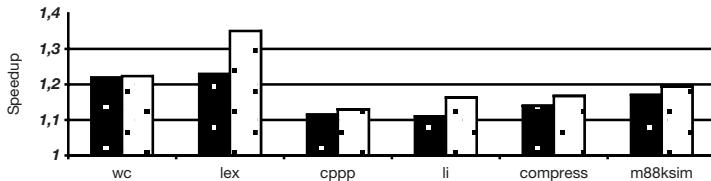


Fig. 2. Speedup for Predicate Analysis based on Path Information.

Figure 2 compares the performance of codes transformed with the described techniques to the performance of baseline codes. The baseline codes are generated by the IMPACT compiler using hyperblock compilation techniques. The transformed codes correspond to the baseline codes after our techniques are used to restructure domain codes and after they are rescheduled. Speedup is derived by computing the ratio of the execution cycle count. Two results are reported. The first is the benchmark speedup on the target architecture (black bar) and the average speedup is 1.176. The other is the speedup after our technique is used and the average speedup is 1.207 (white bar). Code quality for them is improved except *wc*. Although it needs a very complex predicate analysis, it has few predicated instructions, which reduces the potential effect of predicate analysis. This method has little impact on compilation time, as listed in Table 1. Two set of results are compared, including the compilation time of baseline code and that of transformed code.

Table 1. Compilation Time for Predicate Analysis based on Path Information

Benchmarks	wc	lex	cccp	li	compress	m88ksim
Baseline	0.64	0.91	1.04	4.58	3.34	3.67
Transformed	0.71	0.89	1.07	4.03	3.66	3.40

5 Conclusions

This paper proposes a predicate representation based on path information at first, with which predicates can be represented by domain codes of corresponding instructions. This method provides a global predicate representation and avoids predicate definition reconstruction during optimization. Then a global predicate relations query system with this representation is put forward and system interface functions are also provided. It can analyze predicate relations on a global scope. Experiment results show that it can perform global predication relations accurately and efficiently, which can help the performance improvement for optimization compilers.

References

1. U.Kathail, M.Schlansker, B.Rau. HPL PlayDoh Architecture Specification: Version 1.0. Hewlett-Packard Laboratories Technical Report, HPL-93-80, Feb, 1993.
2. J.R. Allen, K. Kennedy, C. Portfield, etc. Conversion of Control Dependence to Data Dependence, In Conf. Record of the 10th ACM Symp. on Principles of Programming Languages, pp.177-189, Jan, 1983.
3. S.A. Mahlke, D.C. Lin, W.Y. Chen, etc. Effective compiler support for predicated execution using the hyperblock. In Proc. of the 25th Int'l Symp. on Microarchitecture, pp.45-54, Dec, 1992.
4. G.S. Tyson. The effects of predicated execution on branch prediction. In Proc. of the 27th Int'l Symp. on Microarchitecture, pp.196-206, Nov.30-Dec.2, 1994.
5. S.A. Mahlke, R.E. Hank, R.A. Bringman, etc. Characterizing the impact of predicated execution on branch prediction. In Proc. of the 27th Int'l Symp. on Microarchitecture, pp.217-227, Nov.30-Dec.2, 1994.
6. A.E. Eichenberger, E.S. Davidson. Register Allocation for Predicated Code. In Proc. of the 28th of Int'l Symp. on Microarchitecture, Nov, 1995.
7. R. Johnson and M. Schlansker. Analysis Techniques for Predicated Code. In Proc. of the 29th Int'l Symp. on Microarchctecture, Dec, 1996.
8. J.W. Sias, W.W. Hsu, and D.I. August. Accurate and Efficient Predicate Analysis with Binary Decision Diagrams. In Proc. of the 33rd of Int'l Symp. on Microarchitecture, pp. 112-123, Dec, 2000.

kd-Clos: New No-Blocking Permutation Network

Mei Ming, Dong Li, and Bo Fu

Institute of Computing Technology,
Chinese Academy of Sciences,
Beijing, China, 100080
mm@ncic.ac.cn
{dongli,fubo}@ict.ac.cn

Abstract. In this paper a new no-blocking network named kd-Clos (k -dimension Clos network) is proposed. For different size of the network, we can select different k , the hardware cost is much lower than Clos networks. The routing algorithm for permutation on kd-Clos network is also proposed and the time complexity of the algorithm is $O(\log^3 N)$ which is same as previous works on Clos network.

1 Introduction

Permutation is one of the most important communication operations and is highly demanded in parallel and distributed applications. Rearrangeable switch networks can perform all permutations and the hardware cost of Rearrangeable switch networks is lower than no-blocking networks [1]. But due to the rearrangement, the time delay of rearrangeable switch network is much high. When the number of the middle-stage switches m satisfied $n \leq m < 2n-1$, the Clos network is also a rearrangeable network. But when m satisfied $m \geq 2n-1$, the Clos network [2] can realize any permutation without any blocking so that it has been investigated broadly and much progress has been made [3,4,5,6,7,8]. The cross-points of a no-blocking Clos network are $O(N^{\frac{3}{2}})$. This hardware cost is too high to be more practical.

In order to find a no-blocking MINs with low hardware cost, a new no-blocking network named kd-Clos (k -dimension Clos network) is proposed. The routing algorithm for permutation on kd-Clos network is also proposed and the time complexity of the algorithm is same as previous works on Clos network.

The rest of this paper is organized as follows: Section 2 provides some aspects about Clos network. Section 3 presents kd-Clos as well as the routing algorithm for permutation on them. Section 4 concludes this paper.

2 Clos Network

The general Clos network may have any stages of odd number and is built in recursive fashion from smaller size networks. In general, a three-stage Clos network with N input ports and N output ports has r switch modules of size $n \times m$ in input-stage,

m switch modules of size $r \times r$ in middle-stage, and r switch modules of size $m \times n$ in output-stage. Such a three-stage network is denoted as $V(m, n, r)$.

3 kd-Clos Network

3.1 1d-Clos Network

A no-blocking Clos network can be considered as a 1d-Clos network. A 1d-Clos network is denoted as $1d-C(d_1, u_1, p_1)$. It means that there are u_1 nodes that have input-ports and output-ports among d_1 nodes in the 1st dimension, the nodes that have input-ports and output-ports is named as **useful-nodes** and the nodes that have not input-ports and output-ports are named as **additory-nodes**. Every useful-node has p_1 input-links and p_1 output-links that connected to Clos network as well as one input-port and output-port. Every additory-node only has a pair of input-link and output-link that connected to Clos network. The cross-points of a $1d-C(d_1, u_1, p_1)$ is $O(d_1^{\frac{3}{2}})$ and the routing algorithms of permutation on Clos network can be used to route permutation on $1d-C(d_1, u_1, p_1)$.

3.2 2d-Clos Network

The general 2d-Clos network can be constructed with Clos network in 2-dimension. A 2d-Clos network is denoted as $2d-C(d_1, u_1, p_1, d_2, u_2, p_2)$ and every node in 2d-Clos network is connected to two Clos network which are denoted as d_1 -Clos and d_2 -Clos. There are u_1 useful-nodes among d_1 nodes in 1st dimension and u_2 useful-nodes among d_2 nodes in 2nd dimension, so there are $u_1 \cdot u_2$ useful-nodes among $d_1 d_2$ nodes in 2d-Clos network and there are $u_1 \cdot u_2$ pairs of input-port and output-port in $2d-C(d_1, u_1, p_1, d_2, u_2, p_2)$. Every useful-node of 1st dimension has p_1 input-links and p_1 output-links that connected to d_1 -Clos and Every useful-nodes of 2nd dimension have p_2 input-links and p_2 output-links that connected to d_2 -Clos. The architecture of $2d-C(d_1, u_1, p_1, d_2, u_2, p_2)$ is shown in Fig.1.

There are $d_1 d_2$ -Clos networks and $d_2 d_1$ -Clos networks in the 2d-Clos network. The size of 2d-Clos network is $N_{2d} \times N_{2d}$, $N_{2d} = u_1 \times u_2$, we can consider that $u_1 = u_2 = O(N_{2d}^{\frac{1}{2}})$. Then the size of d_1 -Clos is $u_1 \times u_1 = N_{2d}^{\frac{1}{2}} \times N_{2d}^{\frac{1}{2}}$ and the size of d_2 -Clos is $(d_2 + u_2) \times (d_2 + u_2) = 3N_{2d}^{\frac{1}{2}} \times 3N_{2d}^{\frac{1}{2}}$. Because the cross-points of $N \times N$ Clos network are $O(N^{\frac{3}{2}})$, the cross-points of 2d-Clos network are $d_2 \times (N_{2d}^{\frac{1}{2}})^{\frac{3}{2}} + d_1 \times (3N_{2d}^{\frac{1}{2}})^{\frac{3}{2}} = O(N_{2d}^{\frac{5}{2}})$. This result is much better than Clos network.

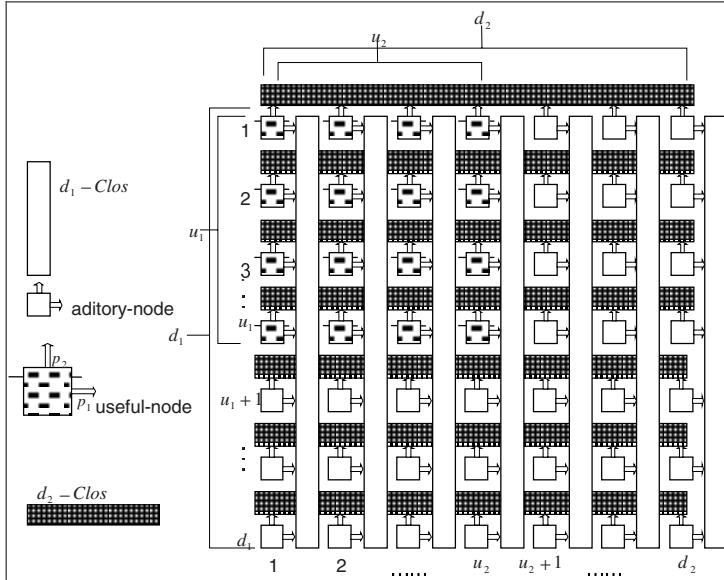


Fig. 1. A schematic of the $2d\text{-}C(d_1, u_1, p_1, d_2, u_2, p_2)$

3.3 The Routing Algorithm of Permutation on 2d-Clos Network

A node in $2d\text{-}C(d_1, u_1, p_1, d_2, u_2, p_2)$ can be donated as (s_1, s_2) $1 \leq s_1 \leq d_1$, $1 \leq s_2 \leq d_2$. If the node (s_1, s_2) is a useful-node and the destination of it is (o_1, o_2) , then the assignments requirement is donated as $(s_1, s_2) - (o_1, o_2)$.

We define a $d_1 \times d_2$ matrix and blank (s_1, s_2) with o_1 , then 2d-Clos routing matrix of $d_1 \times d_2$ can be defined and we named the matrix as 2d-Matrix. There are u_2 1's, u_2 2's... u_2 u_i 's distributed in 2d-Matrix. We say a 2d-Matrix is column-complete if and only if each column contains each element exactly once.

The main idea of our routing algorithm is that: First, reassign the elements in each row and make 2d-matrix become column-complete; Second, according the 2d-Matrix and the column-complete 2d-matrix, perform a permutation on every d_2 -Clos; Third, perform permutation on every d_1 -Clos so that the information from node (s_1, s_2) can be routed to (o_1, s_2) ; Finally perform permutation on every d_2 -Clos so that the information from node (o_1, s_2) can be routed to (o_1, o_2) . The first step is the key part of the algorithm and we name the first step as column-complete algorithm.

Theorem 1: The column-complete algorithm for $2d\text{-}C(d_1, u_1, p_1, d_2, u_2, p_2)$ is equivalent to the routing algorithm for permutation on three-stage Clos network $V(m, n, r)$.

Prove: First we use a $r \times m$ matrix to describe the basic function of the routing algorithm for permutation on three-stage Clos network $V(m, n, r)$ and we named the matrix as $S = (s_{ij})$, where s_{ij} is the j th element in the i th row of S . The rows of S are indexed by input switches and columns by center switches, while entries represent output switches. Thus, $s_{ij} = e$ implies that a connection from i th input switches to the

eth output is routed through j th center switch. Clearly, there are n 1's, n 2's, ..., n r 's distributed in S . The routing represented by S is feasible if and only if each column is complete, containing each output exactly once. S is called complete if every column is complete. The routing algorithm for permutation on three-stage Clos network is to reassign the elements in each row and make S become complete.

When $d_i = u_i$, an 2d-Matrix can be considered as matrix S for $V(m,n,r)$ ($m = d_2$, $n = u_2$, $r = u_1$). Now we can see that the routing algorithm for permutation on three-stage Clos network is equivalent to the column-complete algorithm.

We denote the routing algorithm for permutation on three-stage Clos network as RAPTC and we use RAPTC to replace the column-complete algorithm. Then the routing algorithm for permutation on 2d-Clos network can be shown in Fig.2.

RAP-2d-Clos (Routing Algorithm for Permutation on 2d-Clos network)

Perform RAPTC so that the 2d-Matrix becomes column-complete;

According the 2d-Matrix and the column-complete 2d-matrix, perform a permutation on every d_2 -Clos;

Perform RAPTC on every d_1 -Clos so that the information from node (s_1, s_2) is routed to (o_1, s_2) ;

Perform RAPTC on every d_2 -Clos so that the information from node (o_x, s_y) can be routed to (o_1, o_2) .

Fig. 2. RAP-2d-Clos

In d_1 -Clos, every node performs permutation just once, so $p_1 = 1$. But every node in d_2 -Clos performs permutation twice, so $p_2 = 2$. So the d_1 -Clos is $d_1 \times d_1$ Clos network and the d_2 -Clos is $(d_2 + u_2) \times (d_2 + u_2)$ Clos network. Because the necessary and sufficient condition for no-blocking permutation on Clos network is that $m \geq 2n-1$, we known that when $d_2 \geq 2u_2-1$ the permutation in step1 of RAP-2d-Clos is no-blocking. When d_1 -Clos and d_2 -Clos are no-blocking, RAP-2d-Clos can route permutation on 2d-Clos without any blocking.

The routing time complexity of RAP-2d-Clos is related to the RAPTC, there have been a lot of routing algorithms and we can select the best of them to be used in RAP-2d-Clos. If the time complexity of RAPTC on $N \times N$ Clos network is donated as $T_p(N)$, then the routing time complexity of RAP-2d-Clos T_{2d} can be calculated: $T_{2d} = T_p(u_1 \times u_2) + T_p(d_1) + T_p(d_2 + u_2)$. The 2d-Clos network have $u_1 \times u_2$ pairs of input-ports and output-ports, we say the size of 2d-Clos network is $N_{2d} = u_1 \times u_2$. The necessary and sufficient condition for no-blocking permutation on 2d-Clos are that $d_2 \geq 2u_2-1$ and $d_1 = u_1$, then $T_{2d} = T_p(N_{2d}) + T_p(u_1) + T_p(3u_2)$. We use the graph-coloring algorithm in [5], the time complexity of it is $T_p = O(\log^2 N \log m)$. Then the time complexity of RAP-2d-Clos is $T_{2d} = \log^2 N_{2d} \log d_2 + O(\log^3 u_1) = O(\log^3 N_{2d})$.

The result of 2d-clos can be easily extended to kd-clos. The cross-points of kd-Clos are $O(3^k k N_{kd}^{\frac{2k+1}{2k}})$, the time complexity of the routing algorithm is $O(\log^3 N_{kd})$.

4 Conclusions

In this paper, a new no-blocking network named kd-Clos(k-dimension Clos network) is proposed and the cross-points of kd-Clos are $O(3^k kN_{kd}^{\frac{2k+1}{2k}})$. For different size of the network, we can select different k , the hardware cost is much lower than Clos network. The time complexity of the permutation routing algorithm on kd-Clos network is $O(\log^3 N)$ which is same as previous works on Clos network. So we can say that the kd-Clos network is a useful and practical network.

References

- [1] T. Feng and S. Seo, "A New Routing Algorithm for a Class of Rearrangeable Networks", IEEE Trans.Computers, vol. 43, no. 11, Nov. 1994
- [2] C. Clos, "A Study of Non-blocking Switch Networks", Bell Syst. Tech. J., pp. 406–424, 1953.
- [3] Moo-Kyung Kang, Ju-Hwan Yi, You-Sung Chang and Chong-Min Kyung, "Switch Expansion Architecture Using Local Switching Network", 2000 IEEE International Conference on Communications, Vol. 3, pp. 1426–1429, 18–22 June 2000.
- [4] Y. Yang and G.M. Masson, "The Necessary Condition for Clos-Type Nonblocking Multicast Networks", IEEE Transactions on Computers, vol. 48, pp. 1214–1227, Nov. 1999.
- [5] G.F. Lev, N. Pippenger and L.G. Valiant, "A fast parallel algorithm for routing in permutation networks", IEEE Trans.Computer, vol. C-30, pp. 93–100, Feb.1981.
- [6] Y. Yang and J. Wang, "Wide-Sense No blocking Clos Networks under Packing Strategy", IEEE Trans. Computers, vol. 48, No. 3, pp. 265–284, Mar. 1999
- [7] Y. Yang, "A Class of Interconnection Networks for Multicasting", IEEE Trans. Computers, vol. 47, No. 8, pp. 899–906, Aug. 1998
- [8] A. Itoh et al., "Practical Implementation and Packaging Technologies for a Large-Scale ATM Switching System", IEEE J. Selected Areas in Comm., Vol. 9, No. 8, pp. 1280–1288, Oct. 1991

LilyTask: A Task-Oriented Parallel Computation Model*

Tao Wang and Xiaoming Li

Computer Networks & Distributed Systems Laboratory,
School of Electronics Engineering and Computer Science,
Peking University
100871 Beijing, P.R. China
wangtao@ieee.org
lxm@pku.edu.cn

Abstract. While BSP model is a concise parallel computation model, it has some limitations in functionality and performance. To overcome those limitations, we propose a task-oriented parallel computation model, named LilyTask, while remaining the virtue of conciseness. In LilyTask, *tasks* are taken as scheduling units in parallel computation, which may directly reflect the decomposition process of original problems. Further more, a task in LilyTask is allowed to generate subtasks of various granularities at runtime, which makes LilyTask very suitable for irregular problems. LilyTask model may also be applied to computational Grid to solve coarse-granularity parallel problems and loosely-coupled problems sets.

Keywords: BSP, parallel computation model, task, irregular problem, computational Grid

1 From BSP to LilyTask

1.1 Parallel Computation Models

A parallel computation model should be powerful, natural, and concise. PRAM[1] is the first widely studied one. While it is good for theoretical studies, it does not capture enough characteristics of real computer systems.

In 1990, L.G. Valiant proposed BSP[2]. A BSP algorithm composes several supersteps. In each superstep, multiple computing components proceed in parallel, and the data transfers among them cannot really occur until all other work has been finished. There is a forced, implicit barrier between two successive supersteps. This model is very concise. In 1998, Jonathan suggested an implemental standard of BSP - BSPlib[3], and several library[4, 5] implementations based on BSPlib appeared later.

In 1993, D.E. Culler et al. proposed LogP[6]. Although it has very strong influence in academic community, due to its intricacy, not many real implementations exist.

There exist many literatures that compare BSP and LogP. All suggest that they can simulate each other, and have similar performances. A paper[7] considers “BSP seems

* This paper is funded by Project 69933020 of the National Natural Science Foundation of China.

somewhat preferable due to greater simplicity and portability, and slightly greater power". We adopt this conclusion, so we take BSP model as our port of sailing.

1.2 The Limitations of the BSP and the Emergence of the LilyTask

BSP model does not have mechanism for nested parallelism, and it is hard to deal with some irregular problems. The forced synchronization between two supersteps may decrease the performance. There is also no mechanism in BSP model to encapsulate the action of a meaningful task and the data it works on. So with BSP model, the procedure of decomposing a big problem into some sub-problems cannot directly be mapped to the parallel algorithm; and an algorithm designer has to consider explicitly how to translate the sub-problems into some operation sequences, thus map them onto the computing components. This aggravates the burden of the algorithm designer, and makes more difficult the abstraction, description and analysis of a problem's decomposition. How to solve these limitations of the BSP model, without loss of the virtue of conciseness, is our purpose of proposing LilyTask model.

We can take tasks, which directly map the problems, as the running units. So the recursive decomposition of a problem directly maps to the recursive decomposition of a task. Thus we support nested parallelism. Since we have no limit on dynamically decomposition of a task, that is, we allow a task to be decomposed into various numbers of subtasks with various granularities in the algorithm running time, and the task scheduling can be dynamic, we can perfectly deal with irregular problems.

We can adopt the constraint of data transfer in BSP model, which is the key source of the conciseness. A task obtains its data from other tasks only before its actual work begins, and transfers its data to other tasks (not its subtasks) only after its actual work finishes. We call this property obturation. The algorithm designer may first consider what tasks an algorithm requires, as well as the sequential/data relations among the tasks, and then think of the concrete bodies of the tasks. The procedure of algorithm design will be very clear, and the algorithm will be very concise.

Finally, we obtain a task-oriented parallel computation model. While remains the virtue of conciseness, it solves the limitation of the BSP model. We name this model LilyTask model. In the next section, we begin to discuss the definitions of the LilyTask model.

2 Definitions of LilyTask Parallel Computation Model

2.1 Tasks, Their Sequential/Data Relations, and Task Groups

For a computer system, to solve a problem means to do a sequence of operations on some data, and then to generate some results. We can encapsulate the data, the operations and the results together as a task. Then solving a problem just reflects running a corresponding task. In this paper, the concept "data to be operated on for a task" means the data transferred by some other tasks before the task begins.

When we decompose a big task into some subtasks, we will find sometimes it will be convenient to partition the executions of them into several sequential phrases, in which several tasks run. We define the set of tasks run in the same phrase a task

group. In a task group, some tasks may run in parallel, but some others may have some sequential/data relations, which run in light of such relations.

With the description of 1.2, we can now give the definition of a task in LilyTask model: A task, which has the property of obturation, is an abstract description of a problem, and the running of a task means the solving of the corresponding problem. A task has four elements: 1) Data to be operated on; 2) Operations themselves; 3) Results the operations generate; 4) Sequential/data relations with other tasks.

2.2 The Properties and Requirements of Task and Task Group

We define three kinds of tasks: the chief task, the trivial task and the normal task. The chief task corresponds the whole problem to be solved. A trivial task is a sequence of operation, while a normal task is composed by a InData list of data-to-operate, a sequence of operation, a OutData list of results, and a list of the descriptions of the sequential/data relations with other tasks. A normal task should be declared, created and destroyed. The creation of a normal task doesn't mean the running of the task.

There also exist two kinds of task groups: the trivial group and the normal group. A trivial group contains and only contains one trivial task, while a normal group is constituted by some normal tasks. The running of a task group means the tasks in that group run in light of their sequential/data relations. The scheduling of the normal tasks in a normal group is dynamic. A task group may run several times, while after the execution of one instance, the values of the InData and OutData lists of it's tasks remains.

All the trivial tasks of the same parent task may access each other's data. A trivial task may transfer data to a task in the next-run group instance, and it may obtain data from a task in the previous-run group instant. If two normal group instances have the same parent task, a task in the first-run group instance may transfer data to a task in the next-run group instance. If there are sequential/data relations between two tasks, the previous-run task may transfer to the next-run task the values of the elements in the former's OutData list, and the values will be assigned to the variables in the latter's InData list. We use expression `<Task1.Outdata1, Task2.Indata2>` to denote such a circumstance.

3 Some Typical Parallel Algorithms in LilyTask

In this section, we give the parallel algorithms for several typical problems. Since our aim is to exhibit the approach to design the algorithms in LilyTask model, we do not try to give the best algorithms.

3.1 Solving the Linear Equations Using Gaussian Elimination

Solve $X[1..N]$ from equation $A[1..N][1..N] \cdot X[1..N] = B[1..N]$, where $A[1..N][1..N]$ and $B[1..N]$ are given. For simplicity, we will not find the pivot. The algorithm design process is:

- 1) Define normal group G
- 2) Define N normal tasks M[1]..M[N], put them into group G, which will run N-1 times: each of their InData lists contains variant PivotRow[1..N+1]; each of their OutData lists contains variant MyPos, MyRow[1..N+1], PivotPos; MyPos of task M[i] is initialized as i, PivotPos as 1, MyRow[1..N] as A[i][1..N], MyRow[N+1] as B[i], , where i in Z, and i<=N; the operation sequences of task M[i] are of the eliminating of a single row i.
- 3) Define trivial task P, which will run after each instance of G, except for the last one, finishes, and after which finishes, an instance of G runs; the operation sequences of jth instance of P are of dispatching the results generated by the last instance of G to the next instance.:
- 4) Define trivial task Q, which will run after the last instance of G, GN-1, finishes; the operation sequences of Q are of getting the final result:
- 5) G runs N-1 times, P runs N-2 times, and at last Q runs. When Q finishes, X[1..N] is what we want.

We can manage a task to solve several lines, so as to reduce the number of tasks. We can see that the workloads in different tasks differ. When the number of tasks is larger than the number of computing components, and when the tasks' workloads differ, the property of dynamic task scheduling shows its advantage.

3.2 QuickSort

Sort N integers A[1..N] using QuickSort method. Because this problem has the property of recursion, let's first define a task template TP, which means some tasks may be of the same definition of TP: each of their InData lists contains variant MyNum, MyA[1..N]; each of their OutData lists contains variant MyFinalA[1..N]; the operation sequences are of dividing the numbers into lesser part and greater part, and creating subtask using the same template TP if needed. Then the next process is:

- 1) Define normal group G
- 2) Create task Work using TP, put it into group G
- 3) Run group G, when it finishes, the Value of FinalA in task Work is the sorted integers.

We can see in this algorithm the LilyTask model's support to multi-level task and dynamic task creation.

3.3 BSP on LilyTask

At the end of this section, we would like to mention that it is possible to map an arbitrary algorithm designed for BSP model onto LilyTask model. In fact, we find the mapping is very simple. Since the soundness of BSP model is of common acceptance, this mapping may serve as an indicator of that of LilyTask model.

We can simply define each work a computing component should do in a superstep a task, and put those tasks that map the work done in the same superstep into one task group. We can also define the h-relations between two groups. Thus, the work of migration has been done.

4 Conclusion and Discussion

In this paper, taking BSP as our starting point, we have sketched a task-oriented parallel computation model, named LilyTask. A detailed description can be obtained from reference[8]. This model overcomes some limitations of BSP, while remains the virtue of its conciseness.

We discuss the definitions and properties of task, task sequential/data relations and task groups in LilyTask. Then we give some typical algorithm in LilyTask to exhibit some aspects we should consider in algorithm design. We find that because LilyTask is a task-oriented model, it is very convenient to design algorithms, especially those which aim at irregular problems.

Consider an application suitable for the computational Grid environment. It may be a coarse-granularity parallel problem or a set of coarse-coupled problems. We can decompose the problem recursively into several closed sub problems, then we may define each of them a task and add the sequential/data relations among them. The tasks may run in different parallel levels of the Grid: that of the Grid level, and that of the level of a parallel computer system. We can say, LilyTask model is also suitable to design the algorithms on computational Grid environment.

References

1. Fortune, J. Wyllie: Parallelism in Random Access Machines. In Proc. 10th Ann. ACM Symp. on Theory Computing (1978) 114–118
2. L.G. Valiant: A Bridging Model for Parallel Computation. Communications of the ACM, August 1990
3. J. Hill, B. McColl, D. Stefanescu, M. Goudreau, K. Lang, S. Rao, T. Suel, T. Tsantilas, R. Bisseling: The BSP Programming Library. Parallel Computing 24 (1998) 1947–1980
4. University of Oxford: Oxford BSplib. <http://www.bsp-worldwide.org/implmnts/oxtool/>
5. University of Paderborn: Paderborn BSP-Library. <http://www.uni-paderborn.de/~bsp/>
6. D.E. Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos, R. Subramonian, T.V Eicken: LogP: Towards a Realistic Model of Parallel Computation. In Proc. 4th ACM Symp. on Principles and Practice of Parallel Programming (1993) 1–12
7. G. Bilardi, K.T. Herley, A. Pietracaprina, G. Pucci, P. Spirakis: BSP vs LogP. In Proc. 8th ACM Symp. on Parallel Algorithms and Architectures (1996) 25–32
8. T. Wang, X. Li: LilyTask: A Task-Oriented Parallel Computation Model (Technical Report, PKU_CS_NET_TR2003008). <http://162.105.80.88/crazysite/home/report/>

A Method of Data Assignment on Heterogeneous Disk System*

Jingli Zhou, Dong Xiang, Shengsheng Yu, Lin Zhong, and Jian Gu

National Storage Lab/Computer Sci. & Eng. Department,
HUST, Wuhan, 430074
thincat_china@hotmail.com

Abstract. With the development of applications and storage technology, heterogeneous disk system comes forth. But methods of data assignment on homogeneous disk systems, such as striping, do not adapt to heterogeneous disk systems any more. In this paper, we propose a new method of data assignment on heterogeneous disk system. Firstly, it constructs a logical collection of disk drives from an array of heterogeneous disk drives. Each logical disk has identical bandwidth. Then, these logical disks are reorganized to strip the data. Experimental results show that this method can increase the bandwidth of the heterogeneous disk system effectively.

1 Introduction

A number of recent technological trends have made data intensive applications, such as continuous media servers and digital libraries possible. In order to satisfy the storage needs of data intensive applications. Storage systems are often constructed from disks with same performance. Load balance is easy to achieve in homogeneous systems, and data assignment policies for homogeneous systems are simple and effective in contrast with heterogeneous systems. But these policies such as striping [1], [2], which treat all kinds of systems the same without thinking about the different performance of disks in heterogeneous system. They are not suit to heterogeneous systems.

1.1 Related Work

In homogeneous disk systems, one way to remedy the performance difference between computing elements and storage devices is to stripe data across the disks. This distribution technique is a basic way for RAID systems, and striping file systems. Many people have studied the striping technology, and put forward some useful investigations on disk striping optimization [1], [2], [3].

There is little work over distribution technique of heterogeneous. Two studies have investigated heterogeneous disk system techniques in support of single-user environments under which the data is retrieved at a high bit rate [4], [5].

* Sponsored by Defensive Research Project (413160502) of P. R. China.

2 Analyze Disk Striping on Heterogeneous Disk System

Model of striping in heterogeneous disk system under large data request:

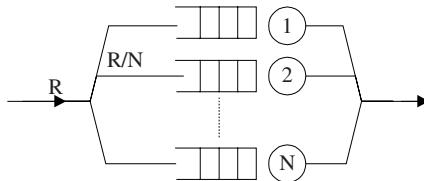


Fig. 1. This shows the model of large data request. Under large data request, the data requested are striping on multiple disks. At the most special situation, the data requested are striping on all the disks in the system. We presume that the system is composed of n disks; the data requested R is striping on n disks, then, the sub-request of one disk in the system is R/N. When every disk finishes sub-request, the system returns the data requested to client

Table 1. Modeling parameters

Variable	Definition
$T_{\text{resp},i}$	The mean response time of disk i
W	The mean throughput of disk system
$W_{i,j}$	The mean throughput of disk i

We presume that among all the disks in the heterogeneous disk system exists disk j, the throughput of which is

$$W_j = \min_{i=1}^n (W_i) . \quad (1)$$

The mean throughput of the heterogeneous disk system is

$$W = < W_j . \quad (2)$$

According to formula 1 and 2, we have

$$W = < W_j = < W_i, i \in [1, n] . \quad (3)$$

We can draw a conclusion from formula 3, that the throughput of heterogeneous disk system is limited by the performance of the slowest disk.

When data requested is not striping over all disks in the system, the system performance is also limited by the performance of the slowest one among the disks which data requested stripe on.

In condition of small data request, model of striping in heterogeneous disk system:

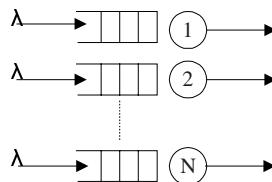


Fig. 2. This shows model of small data request. In the situation of small data request, the data requested is located on one disk i ($1 \leq i \leq N$)

The mean throughput of the system W is

$$W = \sum_{i=1}^N W_i = \sum_{i=1}^N \frac{1}{T_{resp,i}}. \quad (4)$$

Formula 4 represents that the throughput of system is relative to summation of throughput of disks among the system.

In this paper, we propose a method that divides the physical disks in heterogeneous system into several logical disks with identical bandwidth, which are organized and managed through striping.

3 A Data Assignment Policy for Heterogeneous Disk Systems

We use Iometer to simulate disk load [6]. The parameter of size of data requested can be set as striping unit. We can look a request for a single disk as multiple sequential sub-requests for the disk. The size of each sub-request is one striping unit. The percent of access specification can be set as 100%. The percents of read/write and random request are up to the workload of real conditions.

3.1 Striping on the Logical Disks

1. Measure bandwidth of disks under typical workload of heterogeneous system. The bandwidth of disks is sorted in order, represented by (W_1, W_2, \dots, W_n) .
2. Assume that the slowest disk 1 is composed of N_1 logical disks. That is to say that the bandwidth of each logical disk is W_1/N_1 . So other disks can be divided into $\left[\frac{W_i}{W_1} \cdot N_1\right], (1 \leq i \leq N)$ logical disks. If we want each logical to have the

same bandwidth, we should select a N_1 to make $\frac{W_i}{W_1} \cdot N_1$ an integer. The range of N_1 is $(1 \leq N_1 \leq W_1)$. If N_1 is too large, the heterogeneous system is partitioned

into many groups, which increases the complexity of storage system management. If N_i is too small, much bandwidth of faster disks is wasted.

3. Select one logical disk from each physical disk to group an array of logical disks, which we can strip data on.
4. Repeat step 3 until no logical disk lefts.

4 Experiment Validation

We compare the performance of three systems: System A: Composed of four SEAGATE ST39236LW disks. Striping is used to assign across the disks. The striping unit is 16KB. System B: Composed of two SEAGATE ST39236LW disks and two SEAGATE ST318406LW disks. Striping is used to assign across disks. The striping unit is 16KB. System C: Composed of two SEAGATE ST39236LW disks and two SEAGATE ST318406LW disks. Striping is used to assign across logical disks, which have equal throughput. The striping unit is 16KB.

4.1 Data Assignment on Experiment System

We use Iometer to get the throughput of disk. The workload is: 67% Read /33% Write, 100% Random accessing. Throughput of ST318406LW is 1.3~1.5 times faster than that of ST39236LW. We set the size of request as 16KB according to above systems.

We take ST39236LW as one logical disk S1 and ST318406LW can also be looked as one logical disk F1.



Fig. 3. That shows the organization of logical disk. We group (S1, F1, F1, S1) in an array of logical disks. This structure has no difference with system C

If disk ST39236LW is portioned into two logical disks S1 and S2, ST318406LW can be portioned into three logical disks F1, F2 and F3.

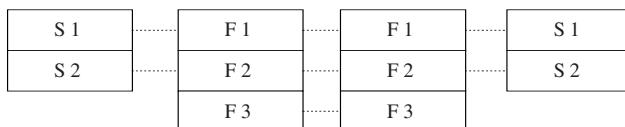


Fig. 4. That shows the organization of logical disk. We divide logical disks into three groups, (S1, F1, F1, S1), (S2, F2, F2, S2) and (F3, F3)

We use Iometer measure the throughput of the three systems. The experiment parameters are: 67% Read/33% Write, 100% Random accessing.

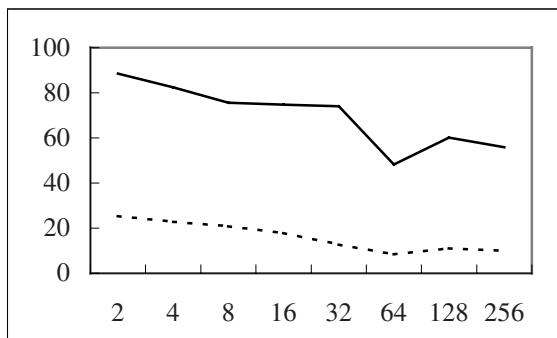


Fig. 5. This shows the improving ratio of throughput of system B and C comparing with that of system A. *Y-axis* represents improving ratio of throughput of system B (*dotted line*) and C (*solid line*) comparing with that of system A, and *x-axis* represents size of requested data. We can see that the new method using in system C improves the throughput of heterogeneous system

5 Conclusion

Traditional policies used for homogeneous disk system do not adapt to heterogeneous disk system, they cannot make full use of the bandwidth of disks in systems.

In this paper, we propose a new method of data assignment on heterogeneous disk system. Firstly, it constructs a logical collection of disk drives from an array of heterogeneous disk drives. Each logical disk has identical bandwidth. Then, these logical disks are reorganized to strip the data. Experimental results show that this method can increase the bandwidth of the heterogeneous disk system effectively.

References

1. Marc Farley, Building Storage Network, Second Edition, McGraw-Hill Education, 2002
2. Zhang Jianglin, Feng Dan, Massive Information Storage, Science Publishing, 2003
3. <http://sourceforge.net/projects/iometer/>
4. Ling Tony Chen, Doron Rotem, and Sridhar Seshadri. "Declustering Databases on Heterogeneous Disk Systems," In Proceedings of the 21st International Conference on Very Large Data Bases, pages 110–121, Switzerland, September 1995
5. Asit Dan and Dinkar Sitaram. "An Online Video Placement Policy based on Bandwidth to Space Ratio (BSR)," in Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 376–385, San Jose, May 1995
6. <http://www.digit-life.com/articles/hddide2k1feb/iometer.html>

Apply Aggregate I/O to Improve Performance of Network Storage Based on IP*

Qiang Cao and Changsheng Xie

Chinese National Storage System Lab
Computer School of Huazhong University of Science and Technology
Wuhan, Hubei, P.R.China 430074
caoqiang_2003@mails1.hust.edu.cn

Abstract. Network storage provide high scalability, availability and flexible for storage system, and are widely applied to many fields. Small I/O heavily effects performance of storage system include of network storage. Current several methods to solve this problem need theory model and but only improve write performance. In this paper, the aggregate I/O model is presented first. Moreover, we also illustrate how to apply this methodology in NAS system based on IP network. The aggregate I/O module called CFS (Chunk File System) is designed in both NAS device and customer end. In our experiment, the analysis of results shows correctness of model and an improvement made by aggregate I/O for network storage performance. Our experimental results show that CFS using aggregate I/O can improve throughput by factor of 2-4 in same response time without it.

1 Introduction

The growing network storage technologies shake off the traditional storage architecture (DAS) limit, and provide high scalability, availability and flexible for storage system, moreover are widely applied to many fields. Storage area network (SAN), network attached storage (NAS) and iSCSI are proven approaches to network storage [1].

In many factors issuing storage performance, small I/O is one of most important problems. Studies have shown that 75 percent or more web documents in web proxy traffic are less than 8KB [2]. It is well-known that FFS and other conventional file systems cannot efficiently handle small files, leading to very poor performance. In the network storage, the problem of small I/O is more severe since the network storage has longer data path and more overhead than DAS, extremely depends on the ability of network processing. At the same time, small I/O leads many little data packages in network, which makes system performance further worse than in DAS.

In this paper, the method combining several I/O to large I/O is called aggregate I/O. So we first consider a single server and batch processing queuing system, which

* This paper is supported by both National Natural Science Foundation of P R China under the Grant NO 60173043, NO 60273073 and National 973 Great Research Project of P R China under the Grant NO G1999033006.

named by batch processing model or aggregate I/O for storage system. The key parameters and their relation with performance of aggregate I/O can be defined by the model. Second, we also illustrate how to apply this methodology in NAS, which is typical network storage system based on IP. A novel aggregate I/O module, called Chunk file system (CFS), in NAS, has a good write performance, similar to LFS, as both combining large I/O transfers. Unlike LFS, CFS also significantly improves read performance by chunk caching and I/O aggregating. Third, our experimental results show that CFS using aggregate I/O can improve throughput by factor of 2-4 in same response time without it.

2 Aggregate I/O Model

In real world, the service time of many tasks break into two parts, one is relative to the task characters, such as work quantity and difficult level, the other is fixed process overhead of server. So the overall time of service can be denoted as $T_s = T_{req} + \tau$, where T_{req} is real processing task time and τ is fixed overhead independent of task itself. For instance, in computer system, the process time of task is consisted of the time where CPU prepares for initialize register and loads program into memory etc and the time where code of program is executing. Ahead is nothing with code number, however, later is almost linear to code number.

2.1 Assumption

Apparently, if the delay of fixed overhead is not neglected comparably to overall service time, the method of batch processing can be used to apportion the fixed overhead into each tasks of batch and therefore reduce the real service time of each task. Simply, the mean service time of a task in batch is $E[T_B]$,

$$E[T_B] = (\sum_1^K T_{req} + \tau) / K = \frac{1}{K} \cdot \sum_1^K T_{req} + \frac{\tau}{K}, \text{ where } K \text{ is length of batch.}$$

Since the throughput of system is $1/E[T]$, the batch processing method can improve throughput, which is a foundational argument for our aggregate I/O. However, until all K tasks have arrived and batch is filled, the server can't begin to process task. The mean response time of each task increases intuitively, because earlier arrive tasks have to wait until K tasks arrived. In following subsection, we will construct batch processing model in order to get the relation between throughput and mean response time.

For analytically tractable system, the batch processing model behaves as follows: The customers arriving to system has rate λ Poisson arrive and are served by single server with a infinite queue length. Server only processes a batch that already has K customers. Because in network storage, the service time is linear to the size of I/O request [3], the service time can be given by $T_s = C \cdot s + \tau$, where C is ratio of the time per data unit and s is size of the requesting data. For further simplicity, s is fixed as unit data. When s is minimal I/O unit, a large I/O request can be regard as several small I/O arriving in the same time. With the assumption the service time for all requests is same and server process is exponential distribution. So the whole service

time of K tasks is $T = K \cdot C \cdot s + \tau$. $\mu = K/T$ is service rate of system. Only until the number of arrival tasks is beyond K, the process can't start.

The goal of our model is study the relation between key parameters such as mean response time, throughput and the value of K.

2.2 The Model of Batch Process

In this section, we exploit queuing theory[8] to construct batch processing model. It is proven that process method in previous section is meet the condition of Markov chain. Let $Y(i)$ denote as probability of the state i that there i requests in queue in Equilibrium conditions. Figure 1 show the state transition graph in our system.

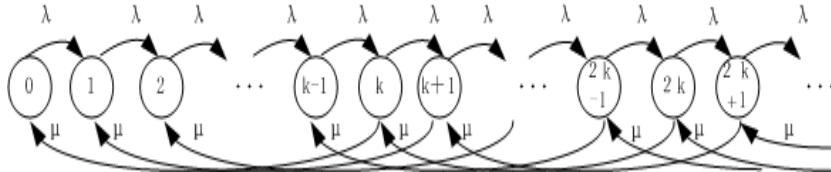


Fig. 1. State transition diagram for the aggregate I/O

Chapman-Kolmogorov equation is defined as,

$$\begin{cases} \lambda \cdot Y(0) = \mu \cdot Y(K) \\ \lambda \cdot Y(i) - \lambda \cdot Y(i-1) = \mu \cdot Y(i+K) & 1 \leq i < K \\ (\lambda + \mu) \cdot Y(i) = \lambda \cdot Y(i-1) + \mu \cdot Y(i+K) & K \leq i \\ \sum_0^{\infty} Y(i) = 1 \end{cases} \quad (1)$$

we can give following results,

$$\begin{aligned} Y(i) &= Y(0) \cdot \frac{1 - x_0^{i+1}}{1 - x_0} & 1 \leq i < K \\ Y(i) &= Y(0) \cdot \frac{\lambda}{\mu} \cdot x_0^{i-K} & K \leq i \\ Y(0) &= \frac{1 - x_0}{K} \end{aligned} \quad (2)$$

x_0 is minimal value of equation $\mu \cdot x^{K+1} - (\lambda + \mu) \cdot x + \lambda = 0$, and $0 < x_0 < 1$, so the mean tasks number in system queue is given by $E[N] = \sum_0^{\infty} i \cdot Y(i)$.

With computing, it is:

$$E[N] = \frac{K-1}{2} + \frac{\lambda}{\mu} + \frac{\lambda}{K \cdot \mu} \cdot \frac{x_0}{(1-x_0)} - \frac{x_0^2 - x_0^{K+1}}{K \cdot (1-x_0)^2} + \frac{(K-1)x_0^2}{K \cdot (1-x_0)} \quad (3)$$

According to Little 公式, the mean response time of system can be given by:

$$E[T] = E[N]/\lambda = \frac{K-1}{2 \cdot \lambda} + \frac{1}{\mu} + \frac{1}{K \cdot \mu} \cdot \frac{x_0}{(1-x_0)} - \frac{x_0^2 - x_0^{K+1}}{K \cdot \lambda \cdot (1-x_0)^2} + \frac{(K-1)x_0^2}{K \cdot \lambda \cdot (1-x_0)} . \quad (4)$$

Based on above formulas observation, since system can't process until the length of arrival queue is beyond K, the mean length of waiting queue is nearly $(K-1)/2$. It is same as intuition.

In other hand, we can see:

When λ is small compared to μ or K is large, $f(0.5) < 0$. So according to Feimi law, $x_0 < 0.5$. Thus,

$$E[T] < \frac{K}{2 \cdot \lambda} + \frac{1}{\mu} \cdot \left(1 + \frac{1}{K}\right) - \frac{3}{K \cdot \lambda} = \frac{K}{2 \cdot \lambda} + \left(C \cdot s + \frac{\tau}{K}\right) \cdot \left(1 + \frac{1}{K}\right) - \frac{3}{K \cdot \lambda} . \quad (5)$$

3 Implement and Experiment of Aggregate I/O

In this section, the aggregate I/O module in NAS is designed and called CFS (Chunk file system). The iSCSI system is similar to NAS in many aspects and will be further discussed in other paper.

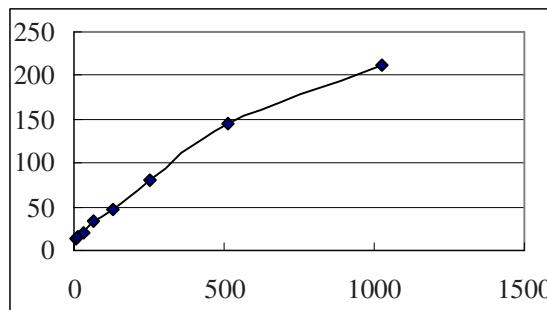


Fig. 2. Shows the relation between chunk size and response time. X axis represents chunk size (KB), y axis represents the mean response time of request (ms). The result meets the equation 4

In network storage application, many operations need copy whole directory even entire file system, such as backup and data immigrating. These operations need copy many files including small file and target may be NAS device. So in our experiment, the methodology is copy a typical web site for CFS using aggregate I/O, which can test practical effect of CFS. Because the size of file isn't same, computing K by defined minimal unit is trivial and insignificant. So in this experiment, adjusting the size of chunk replaces the adjusting K.

In experiment, there are Procom1000 as NAS device and Pentium 4 PC as server, connecting with Calayst3524 switch. Network full width is 100Mbps.

Figure 2 shows the relation of throughput and chunk size, figure 3 shows the relation of mean response time and chunk size. The results can be satisfied with batch processing model.

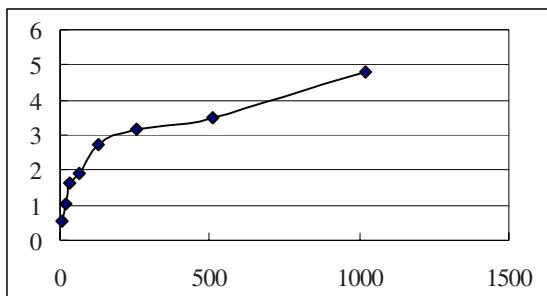


Fig. 3. Shows the relation between throughput and chunk size. *X axis* represents chunk size (KB), *y axis* represents throughput (MB/s) of transferring data

4 Conclusion

In this paper, we use queuing theory to build batch processing model which is base of aggregate I/O. We also illustrate how to apply aggregate I/O in NAS, which is typical network storage based on IP. In the end of paper, we test CFS and results show that CFS using aggregate I/O can improve throughput by factor of 2-4 in same response time without it.

Reference

- [1] Garth A. Gibson, Rodney Van Meter: Network Attached Storage Architecture, IEEE Internet Computing. Vol.43, No.11, 2001:Page(s): 50–58
- [2] Yingwu Zhu, Yiming Hu, UCFS – A Novel User-Space, High Performance. Customized File System for Web Proxy Servers, IEEE Trans. on computers, 51(9), SEP 2002,pp1056–1071
- [3] Cao Qiang, Xie Chang-sheng: The Study of the I/O Request Response Time in Network Storage System. Vol(40), NO.6, in Journal of Computer Research & Development.

Orthogonal Design Method for Optimal Cache Configuration

Hongsong Chen, Zhenzhou Ji, and Mingzeng Hu

Department of Computer Science and Technology
Harbin Institute of Technology
Harbin, 150001
chs68@pact518.hit.edu.cn

Abstract. A cache configuration is one of the most important factors to improve speed of transferring information between CPU(s) and memory. In order to find the optimum cache configurations that give high hit rate with small cache size, designers have to deal with many cache parameters such as number of sets for each cache, block size for each cache line, and so on. Beside the parameters, the designers have to find what the maximum hits of the cache under the consumption that the size is unlimited. Orthogonal method is apply modern statistical methods treating simultaneously several statistical variables. A purpose of this paper is to use orthogonal design method to search for the optimal cache configuration which produces high hit rate with small size. Using the orthogonal method for the cache configuration is successful because high hit rate with small size of cache is considered in the tests.

1 Introduction

Computer performance relies on many factors. One of them is cache efficiency to provide the necessary information to processor as soon as possible. In order to provide requested information to the processor, the cache has to search through all the cache space. If the size of the cache in fully set associative style is tremendous huge, searching time will be much longer than that in direct map. On the other hand, if lot of information needs to be kept in the cache, the set associative style mostly can store the data more than direct map with the same size. Processor companies have made central processing units with different cache configuration (Table 1).

There seem to be a little controversial when designers tried to design I-cache, D-cache. Some designers prefer to have more I-cache space than D-cache space, while others prefer to have more D-cache space than I-cache space. Another factor that designers need to consider is cache hierarchy. The cache can be designed to have one, two, or more levels. In general (might not be the best), the primary cache is split in to I-cache and D-cache, while the secondary cache is unified and able to dynamically allocate cache blocks to contain data or instruction depending on the program's requirements[1].

In the paper I proposes a technique using orthogonal method to find out which combinations give the near-optimal results based on hit ratio. The orthogonal method usually works well on large amount of data with a combination method to search near optimal results.

Table 1. Example of on-chip memory in microprocessors

Processor	Size	I-cache		Size	D-cache	
		assoc	line		assoc	line
Intel Pentium	8-KB	2-way	8-word	8KB	2-way	8-word
DEC 21064 (Alpha)	8-KB	1-way	8-word	8-KB	1-way	8-word
Hitachi HARP-1	8-KB	1-way	8-word	16-KB	1-way	8-word
MIPS R4200	16-KB	1-way	8-word	8-KB	1-way	4-word
MIPS R4400	16-KB	1-way	8-word	16-KB	1-way	8-word
SuperSPARC	20-KB	5-way	16-word	16-KB	4-way	8-word
TeraSPARC	4-KB	1-way	8-word	4-KB	1-way	8-word

2 Methodology

2.1 Orthogonal Method

Experimental designs based on using orthogonal arrays change more than one variable to effectively determine the importance of several variables with a single test run. Orthogonal arrays were invented by Jacques Hadamar and first used for statistical experimental design by Plackett and Burman[2].

The motivation is to minimise the size of an experiment, to control the amount of information required to be evaluated and to minimize the time and cost to execute an experiment. In this paper, I use it to select the optimum cache configuration.

2.2 Cache Architecture

Cache provides high speed data access for CPU to the memory by copying the portion of the program or data which will be used often into the cache line (cache block). How the caches are mapped to memories is important. And it also affects the performance of the computer[3]. Cache architecture are included with fully mapping, direct mapping and n-way set associative mapping.

2.3 Use Orthogonal Method to Design Cache

The main parameters used to construct the cache are number of set, block size, number of lines in one set and replacement policy. There are many kinds of combination, so we should analyze each factor and its level.

2.3.1 Assign Orthogonal Array

Because level one cache consists of instruction cache and data cache, I design that the parameters of level one instruction cache and data cache are equal. The level two cache is the uniform one. So we need design six factors. The number of columns

represents the levels of each kind of factors, every factor is parted into five levels. Factors and levels are showed in table 2.

Main cache's parameters are number of set (nset), block size (bsize), and number of lines in one set (assoc)[4]. In general, cache hierarchy has two levels and level one cache composes of instruction cache (I-cache) and data cache (D-cache).

Table 2. Fators and levels

Factors	Level1	Level2	Level3	Level4	Level5
L1-nset	8	16	32	64	128
L1-bsize,byte	16	32	64	128	256
L1-assoc	1	2	4	8	16
L2-nset	16	32	64	128	256
L2-bsize,byte	128	256	512	1024	2048
L2-assoc	1	2	4	8	16

In my design, the parameters of level one data cache and level one instruction cache are equal. Therefore, there are six parameters of cache configuration.

The default replacement policy is LRU (least recently used), when we find the optimum cache configuration, I will change the replacement policy with FIFO(first in first out) and Random replacement policy, then compare them to find the best one.

2.3.2 Evaluation Function

Evaluation Function used in this paper is related to hit rate and cache size which are directly related to the performance of computers. If the number of hits is high, it means that the CPU seldom fetches the information from the slower storages. If the cache size is small, the seek time will be short.

This function will give the good, reasonable, and acceptable performance for the computer in the idea of speed and size of the caches because when the cache has the high hit rate and small cache size it will give the high evaluation value.

$$\begin{aligned} \text{Evaluation function} = & 100 * \text{il1_hit_rate} + \text{il1_weight} / \text{il1_size} \\ & + 100 * \text{dl1_hit_rate} + \text{il1_weight} / \text{dl1_size} \\ & + 100 * \text{ul2_hit_rate} + \text{l2_weight} / \text{ul2_size} \end{aligned}$$

il1_hit_rate: the rate of cache hits in level 1 instruction cache

dl1_hit_rate: the number of cache hits in level 1 data cache

ul2_hit_rate: the number of cache hits in level 2 uniform cache

il1_weight: the value of evaluating level 1 cache, designed with 10^6

l2_weight: the value of evaluating level 2 cache, designed with 10^6

il1_size: the size of level 1 instruction cache

dl1_size: the size of level 1 data cache ul2_size: the size of uniform 2 cache

The evaluation value of each cache consists of two parts. The first part is the rate of cache hits. The second part is the size of each cache. When the hit rate is high and the size of cache is small, the total evalution value is great.

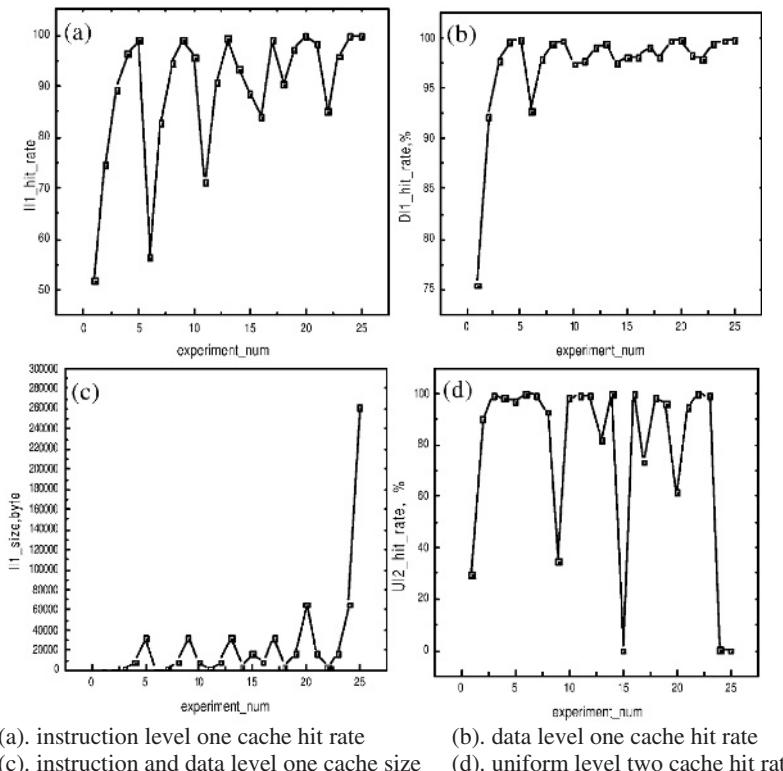
3 Testing Environment

Environment that I used to test includes hardware and simulation. The simulation will describe the parameters and benchmarks that I used for testing the idea.

On this paper, the test is based on Solaris 2.6 operating system. The simulator that I have used is simplescalar tool set[5]. The benchmark that I have used in this paper is test-math which is included in the Tool Set.

4 Result and Analysis

The result of each configuration is shown in Fig 1. Graphs show the change of evaluation function. As the rule of thumb, the bigger the cache size, the higher the hit rate.



(a). instruction level one cache hit rate (b). data level one cache hit rate
 (c). instruction and data level one cache size (d). uniform level two cache hit rate

Fig 1. Results of Orthogonal experiment

As the Fig . 1 show, with the change of level one cache size, the instruction cache hit rate changes much more than that of level one data cache. When the parameters of level two cache changes, the level two cache hit rate changes much.

However, with the cache size starts to be bigger and bigger, the high hit rate is stable. Analyse the result data to determine the optimum levels of the factors. From Fig 1, we can selected the number of experiment 13, 14, 16 and 17 to find the fine cache parameters. Near optimum cache parameter and evaluation value List in table 3.

Table 3. Near optimum cache parameter and evaluation value

Experiment number	IL1_configure	DL1_configure	UL2_configure	Evaluation value
13	32:64:16	32:64:16	32:1024:1	372.03
14	32:128:1	32:128:1	64:2048:2	744.09
16	64:16:8	64:16:8	32:2048:4	342.89
17	64:32:16	64:32:16	64:128:8	347.63

Since on this paper, I give the importance of the evaluation function to the cache configuratton, we can select the cache parameter with quantitative analysis. The higher the cache hit rate, the smaller the cache size, the more the evaluation value is. See from table 3, we can select the best evaluation value set, the number 14, this cache configure- tion can get high hit rate with small cache size, it is fit to our design.

From the result above, I have selected the best cache parameters by much tests and theorise analyse. It is exact and accepted.

5 Conclusions

I bring forward a orthogonal design method to optimize the parameters of cache. The most important factors that affect the performance of the cache are hit rate and cache size. On this paper, the evaluation value is the function of these parameters. The evaluation value will increase when the number of hits increase and the cache size is small. From the results, I strongly believe that using the orthogonal design method to search for the best cache configuration is very effective.

References

1. Norman P. Jouppi and Steven J.E. Wilton.Tradeoffs in Two-Level On-Chip Caching. Compaq Computer Corporation, October, 1993
2. Owen, A.B. Orthogonal arrays for computer experiments, integration and visualization. Statist Sinica.1992,(2):439-452
3. David A. Patterson and John L. Hennessy. Computer Architecture, A Quantitative Approach. Morgan Kaufmann Publishers Inc, 2002
4. Jeffrey B. Rothman and Alan Jay Smith.The Pool of Subsectors Cache Design.. Computer Science Division, University of California, Berkeley.1999
5. Doug Burger, and Todd M. Austin.The SimpleScalar Tool Set, Version 2.0. Computer Sciences Department, University of Wisconsin-Madison.1997

A Probabilistically Correct Election Protocol in Asynchronous Distributed Systems

Sung-Hoon Park

Dept. of Computer Science,
NamSeoul University,
Chung-Nam 330-800, Korea
spark@nsu.ac.kr

Abstract. A Leader is a Coordinator that supports a set of processes to cooperate a given task. This concept is used in several domains such as distributed systems, parallelism and cooperative support for cooperative work. In completely asynchronous systems, there is no solution for the election problem satisfying both of safety and liveness properties in asynchronous distributed systems. Therefore, to solve the election problem in those systems, one property should be weaker than the other property. If an election algorithm strengthens the safety property in sacrifice of liveness property, it would never progress. But on the contrary, an election algorithm strengthening the liveness property in sacrifice of the safety property would have the high probability of violating the safety property. In this paper, we presents a safety strengthened Leader Election protocol with an unreliable failure detector and analyses it in terms of safety and liveness properties in asynchronous distributed systems.

1 Introduction

Distributed systems consist of groups of processes that cooperate in order to complete specific tasks. A *Leader* is a Coordinator that supports a set of processes to cooperate a given task. This concept is used in several domains such as distributed systems, parallelism and cooperative support for cooperative work.

To elect a *Leader* (or Coordinator) in a distributed system, an *agreement* problem must be solved among a set of participating processes. This problem, called the *Election* problem, requires the participants to agree on only one leader in the system [1]. The problem has been widely studied in the research community [2,3,4,5,6]. One reason for this wide interest is that many distributed protocols need an election protocol. The Election problem is described as follows. At any time, there is at most one process that considers itself a *leader* and all other processes consider it as to be their only leader. If there is no leader, a leader is eventually elected.

The so-called FLP impossibility result, which states that it is impossible to solve any non-trivial agreement in an asynchronous system even with a single crash failure, also applies to the election problem [7]. That means that there is no solution for the election problem satisfying both of safety and liveness properties in completely asynchronous distributed systems. It must be pointed out, however, that the impossibility result really means “not always possible,” as opposed to “never

possible.” As a matter of fact, any algorithm that tries to solve the Election Problem cannot always make progress; there exist cases in which the algorithm blocks forever, although it is very unlikely.

Therefore, to solve the election problem in those systems, one property should be weaker than the other property. If an election algorithm strengthens the safety property in sacrifice of liveness property, it would be difficult to progress. But on the contrary, an election algorithm strengthening the liveness property in sacrifice of the safety property would have the high probability of violating the safety property. There exists a trade-off between safety property and liveness property.

A probabilistically correct election protocol, which implies the safety strengthened election protocol, is needed in a practical distributed computing environment. Consider a mission critical distributed system such as an electronic commerce system that runs multiple servers in which one of them roles a master (leader) and others are slaves. To have data consistency among the servers in the system, this system should not violate safety property, which means that all processes connected the system never disagree on a *leader*. In those system the safety property is more important property than the liveness property.

As a classic paper, there is Garcia-Molina’s Invitation algorithm to solve election problem in asynchronous distributed systems. The algorithm strengthens the progress property rather than safety and it allows more than two leaders in the systems.

Our idea is based upon the Garcia-Molina’s Invitation algorithm for solving the election problem in asynchronous distributed systems [2]. He redesigns the Bully algorithm for synchronous distributed systems into the Invitation algorithm for asynchronous distributed systems by using a specification that is weak enough to be solvable, allowing the algorithm to progress even in completely asynchronous distributed systems. His specification uses a strong progress requirement, allowing executions in which even a single process crash and its attempted leader election from the members allow to elect two leaders.

We propose an election algorithm that requires processes to elect a new leader only when they agree with the current leader’s crash. This requirement is strong because, if no set of processes agrees on the current leader’s crash, no progress is made. The requirement is, however, much stronger than the one proposed by Garcia-Molina’s Invitation algorithm in that it implicitly states that the leader election of any process be allowed only on the basis of only its own knowledge.

In this paper, we presents a safety strengthened Leader Election protocol with an unreliable failure detector and analyses it in terms of safety and liveness properties in asynchronous distributed systems.

Our algorithm, based on a standard three phases commit protocol, is fully distributed. It does not extend the asynchronous model of concurrent computation to include global failure detectors. Progress of the algorithm can be guaranteed only in case of minimal violating a safety property.

The rest of the paper is organized as follows. In Section 2, we describe our system model and definitions. In Section 3, this paper relates the election specification to other ways to solve the election problem. In Section 4, this paper provides a robust algorithm that solves the Leader Election problem. In Section 5, we ensure the correctness of the algorithm by proving that it satisfies the two properties of the specification given in Section 4. Finally, Section 6 summarizes the main contributions of this paper and discusses related and future works.

2 Model and Definitions

Our model of asynchronous computation with failure detection is the one described in [9,10]. In the following, we only recall some informal definitions and results that are needed in this paper.

2.1 Processes

We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \dots, p_n\}$ where processes are identified by unique id's. Communication is by message passing, *asynchronous* and *reliable*. Processes fail by crashing; Byzantine failures are not considered.

Every pair of processes is connected by a communication channel. That is, every process can send messages to and can receive messages from any other. We assume processes are able to probe a communication channel for incoming messages. Communication channels are considered to be reliable, FIFO, and to have an infinite buffer capacity. A reliable channel ensures that a message, sent by a process p_i to a process p_j , is eventually received by p_j if p_i and p_j are correct (i.e. do not crash).

Asynchrony means that there is no bound on communication delays or process relative speeds. A process that has been infinitely slow for some time and has been unresponsive to other processes may become responsive again at any time. Therefore, processes can only suspect other processes to have crashed, using local failure detectors.

A failure detector is a distributed oracle which gives hints on failed processes. We consider algorithms that use failure detectors. Local failure detectors are assumed to be inaccurate and incomplete. That is, local failure detectors may erroneously suspect that other, operational processes have crashed or that crashed processes are operational. Since local failure detectors run independently at each process, one local failure detector may perceive a failure, but other detectors may perceive it at a different time or not at all. The failure model allows processes to crash, silently halting their execution. Because of the unpredictable delays experienced by the system, it is impossible to use time-outs to accurately detect a process crash.

We assume that a process communicates with its local failure detector through a special receive-only channel on which the local failure detector may place a new list of id's of processes not suspected to have crashed. We call this list the local system view of the process. Each process considers the last local system view received from its local failure detector as the current one.

2.2 Election Specifications

The Election problem is described as follows: At any time, at most one process considers itself the leader, and at any time, if there is no leader, a leader is eventually elected. More formally, the Election Problem is specified by the following two properties:

- *Safety*: All processes in the local system view of the process never disagree on a *leader*.

- *Liveness*: All processes should eventually progress to be in a state in which all processes connected to the system agree to the *only one* leader.

3 Circumventing the Impossibility Result

In this section, we relate the election specification to other ways to solve the election problem.

- In an asynchronous model augmented by global failure detectors, processes have access to modules that (by definition) eventually reflect the state of the system. Therefore, progress can be guaranteed unconditionally.
- In a timed asynchronous model, processes must react to an input, producing the corresponding output or changing state, within a known time bound. Under this model, progress can be guaranteed if no failures and recoveries occur for a known time needed to communicate in a timely manner.
- In a completely asynchronous model, progress cannot always be guaranteed and failure detectors in practice eventually reflect the system state, but they must be considered arbitrary. Correct processes react in practice within finite time, but this time cannot be quantified. Therefore, in order to guarantee a solution, we need a weaker specification of the problem.

Our approach falls into the last category that originated with Garcia-Molina's work.

Our election algorithm, however, differs from Garcia-Molina's in several ways.

- Processes in Garcia-Molina's model do not need to wait to get consensus about the current leader's crash. If one process suspects that the leader failed, it may attempt to elect the new leader. Garcia-Molina's specification says that, if one process attempts to be a new leader, it eventually should be elected as a leader. Our specification requires all processes in a set to agree on the current leader crash before changing their new leader.
- Garcia-Molina's specification allows a solution in which the attempted change of a leader divides all processes into several sub-groups. Our specification does not allow such a sub-group because it states that if all processes in a system agree on a new leader, they must eventually accept such a leader.

In our model stability is also required for progress, but, at variance of the above case, it is not necessarily related to the state of the system. In other words, eventual progress is required when there is agreement among a set of the local failure detectors, even if failures and recoveries continue to occur in the system.

4 Election Algorithm

We provide a robust algorithm that solves the Leader Election problem given in Section 2. The algorithm is based on the three asynchronous phases.

- A prepare phase, in which a process propose a new leader that the other processes agree with.
- A ready phase, in which all process that agree on the new leader acknowledge the reservation of the potential leader.
- A commit phase, in which the new leader is finally elected, and all process accept it their only leader.

4.1 Solution Sketch

The main idea for the algorithm is as follows. A process p that is informed by its local failure detector of a leader's crash and that has the smallest id among processes in its new local connectivity view sends a message to all processes in its view proposing to update the current leader with the new leader.

Each process received the message records this proposal until the newly elected leader is the same as the proposed new leader in its local view. At which point, it responds by sending back an Accept or Retry message to the process that proposed the leader update. The Accept message is sent if the process agrees on the proposed leader in its local current view.

Upon sending the Accept message, the process reserves the prospective leader, so that no other proposal is accepted for that system. Upon receiving a Retry message, the proposing process restarts the first phase of the algorithm, sending a new Propose message to all processes in its view.

When the proposing process has collected Accept messages from all processes in its view, it starts the commit phase by sending commit messages, ordering other processes in its view to commit the leader update. Upon receiving a commit message, the processes accept the reserved prospective leader as a their new leader.

4.2 Code Description

The code is shown in Fig. 1. The first guarded command in Fig. 1 shows how a process p , when informed of a change in its local connectivity view, set its view to be current and checks if the current leader has crashed. If the leader has crashed, it set the variable *LeaderStatus* to be false. When *leaderStatus* is false in the third guarded command in Fig. 1, the process p checks it is the minimum id among the processes in v_p . If p is the minimum id, it increases the round and proposes itself as a new prospective leader and initializes its *ack* array to zero.

The second guarded command in Fig. 1 checks for incoming messages from other processes. These may be proposals for a new leader (Propose), rejections to propose a new leader (Rej), acceptances of a proposed new leader (Acc), orders to commit a new leader (Commit) or orders to abort a proposed new leader (Abort).

Upon receiving a proposal message from process q , process p stores the new leader's id proposed by q at position q of the array *NewLeader* and stores the proposed round at position q of the array *RoundIn*, then sets position q of the array *Prop* to true to record the receipt of the proposal from q and sets the *CurView* to false to refresh the current view of the system.

```

*[[ local → local ?(vp) ;
    CurView := true;
    [ CurLeader ∉ vp → LeaderStatus := 0;
    [] else → skip
    ]
[]<[]q:: Rq → Rq?m ;
    [ m.type = Propose → (NewLeader[q], RoundIn[q]) := m;
        Prop[q] := true;
        CurView := false;
    [] m.type = Rej → PropRound := m;
        <||q ∈ vp SNq ! Abort (PropRound)>
        <||q:: ack[q] := 0>
    [] m.type = Acc → i := m;
        [Round = i → ack[q] := 1
        [] else → skip
    ]
    [ ∀q:q ∈ vp ack[q] = 1 → <||q:q ∈ vp SNq !
        Commit (PropLeader, Round)>
        <||q:: ack[q] := 0>
    [] else → skip
]
[] m.type = Commit → ( myLeader, i ) := m; Round := i;
    CurLeader = myLeader; LeaderStatus := 1
]
[] LeaderStatus = 0 → [ p = min(vp) → Round := Round + 1; PropLeader := p;
    <||q ∈ vp SNq ! Propose (PropLeader, Round)>;
    <||q:: ack[q] := 0>
]
[] else → skip
]
[] ( CurView ∧ Prop[q] ) → Prop[q]:=false;
[ (Newleader[q] ≤ min(vp) ∧ RoundIn[q] > Round) → SNq ! Acc ( PropIn[q] );
    Next = RoundIn[q] + 1
]
[] else → SNq ! Rej (PropIn[q])
]
>]]

```

Fig. 1. The Algorithm.

Upon receiving a proposal message from process q, process p stores the new leader's id proposed by q at position q of the array NewLeader and stores the proposed round at position q of the array RoundIn, then sets position q of the array Prop to true to record the receipt of the proposal from q and sets the CurView to false to refresh the current view of the system.

If process p later agrees on the proposed new leader, it sends a response to process q (see last guarded command in Fig. 1). The response is either an acceptance of the

new leader at position $\text{NewLeader}[q]$ if the minimum id among the process in v_p is greater or equal than the id of proposed $\text{NewLeader}[q]$ and the proposed round greater than the current round; or it is an rejection to the proposed new leader if the minimum id among the process in v_p is less than the id of proposed $\text{NewLeader}[q]$ or the proposed round less or equal than the current round.

A rejection to the proposed new leader consists of sending back to q the proposed round. An acceptance consists of acknowledging the proposed new leader at position $\text{NewLeader}[q]$.

We now examine the guarded commands of the remaining message types. A process p that receives a rejection to its proposal sends all processes in v_p a message to abort the proposed round and reinitializes the ack array to zero.

A process p that receives an acceptance regarding its proposed new leader receives the proposed round. If the received round is equal to the round of the most recent proposal sent, process p sets the element at position q in the array ack to 1 to record the acceptance.

Then, it inspects the ack array to check if all entries are 1. If so, p starts the commit phase by broadcasting its previously proposed new leader and the corresponding proposed round PropRound to all processes in v_p and reinitializes the ack array to zero.

A process p that receives an order to commit a new leader at position q from process q , simply sets the current leader to the proposed new leader and sets the current round to the proposed round.

5 Correctness

We can ensure the correctness of the algorithm by proving that it satisfies the two properties of the specification given in Section 4.

5.1 Safety

Theorem 1. The algorithm described in Section 4 satisfies the safety condition of the specification (Property 1, Section 2): At any point in time, all processes connected the system never disagree on a leader.

Proof. Either all processes remain in the start state or some process p receives the proposed leader as its leader. In the start state, the safety property holds since all processes are in the state in which a leader has not been elected. If some process p receives its leader by committing a proposed leader at a given position q , it must have received a Commit message from some process q ; therefore, q must have received Accept messages regarding its proposal of a new leader from all processes in v_p including p . It follows from the last guarded command in Fig. 1 that, if process p has accepted the proposal of process q , it will not accept any other proposal for new leader, making it possible to commit at most single proposed leader. Therefore, process p either commits the process at position q as a new leader or ends up with position q by aborting the proposed new leader. Therefore safety property holds.

5.2 Liveness

Theorem 2. The algorithm described in Section 4 satisfies the liveness condition of the specification (Property 2, Section 4): All processes should eventually progress to be in a state in which all processes connected to the system agree to the only one leader.

Proof. By contradiction, a non-progress means that the new leader is not elected forever even though there is no leader; therefore, no Commit messages must be sent. Since the number of processes is finite, there must be at least one process whose id is the minimum value in v_p and that process eventually sends a Propose message. Call this process p . By the code in Fig. 1, we see that, to have no Commit message, each time p sends a Propose message, it should be rejected by other process. It follows that, in order to abort infinitely many Propose messages, other process q must reject the proposed messages infinitely often.

Propose messages are rejected either when the minimum id of v_p is greater than the id of the proposed leader or because of a Propose message already has been received (see Fig. 1).

The first case is ruled out because it implies that some process always considers that there is a process that is alive and whose id is less than the id of proposed new leader. But by strong completeness of a failure detector it is contradiction.

The second case is also ruled out, because it implies that other process q sends infinitely many proposals of the other leader. But by eventual strong accuracy of a failure detector, the process q knows that there is a process whose id is less than its id. Therefore it is contradiction.

6 Concluding Remarks

We have presented a robust election protocol with a reliable failure detector in completely asynchronous systems. We have assumed our local failure detectors to be inaccurate and incomplete. With this approach, the leader election specification states explicitly that progress cannot always be guaranteed. In practice, our requirement for progress is weaker than that stated in the original specification of having a set of processes sharing the same leader.

In fact, if the rate of perceived a leader failures in the system is lower than the time it takes the protocol to make progress and accept a new leader, then it is possible for the algorithm to make progress every time there is a leader failure in the system. This depends on the actual rate of a leader failures and on the capacity of the failure detectors to track such failures.

In [10], Chandra and Toueg note that failure detectors defined in terms of global system properties cannot be implemented. This result gives strength to the approach of relaxing the specification and of having a robust election protocol. In real world systems, where process crashes actually lead a connected cluster of processes to share the same connectivity view of the network, convergence on a new leader can be easily reached in practice.

References

1. G. LeLann, "Distributed Systems—towards a Formal Approach," in *Information Processing 77*, B. Gilchrist, Ed. North-Holland, 1977.
2. H. Garcia-Molian, "Elections in a Distributed Computing System," *IEEE Transactions on Computers*, vol. C-31, no. 1, pp. 49–59, Jan. 1982.
3. H. Abu-Amara and J. Lokre, "Election in Asynchronous Complete Networks with Intermittent Link Failures." *IEEE Transactions on Computers*, vol. 43, no. 7, pp.778–788, 1994.
4. H.M. Sayeed, M. Abu-Amara, and H. Abu-Avara, "Optimal Asynchronous Agreement and Leader Election Algorithm for Complete Networks with Byzantine Faulty Links," *Distributed Computing*, vol. 9, no. 3, pp.147–156, 1995.
5. J. Brunekreef, J.-P. Katoen, R. Koymans, and S. Mauw, "Design and Analysis of Dynamic Leader Election Protocols in Broadcast Networks," *Distributed Computing*, vol. 9, no. 4, pp.157–171, 1996.
6. G. Singh, "Leader Election in the Presence of Link Failures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 3, pp.231–236, March 1996.
7. M. Fischer, N. Lynch, and M. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," *Journal of the ACM*, pp. 374–382. (32) 1985.
8. T. Chandra and S.Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of ACM*, Vol.43 No.2, pp. 225–267, 1996.
9. D. Dolev and R Strong, "A Simple Model For Agreement in Distributed Systems," In *Fault-Tolerant Distributed Computing*, pp. 42–50. B. Simons and A. Spector ed, Springer Verlag (LNCS 448), 1987.
10. T. Chandra, V. Hadzilacos and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *Journal of ACM*, Vol.43 No.4, pp. 685–722, 1996.

Part II

Software and Theory

A Formal Specification and Method for MAS as a Distributed System*

Yan Qi¹, Xin Wang¹, Wei Yan², Xinjun Mao¹, and Zhichang Qi¹

¹ Department of Computer Science and Technology,
National University of Defense Technology,
Changsha, 410073 China

yanqi_nudt@yahoo.com.cn

² Department of Computer Science,
Huazhong University of Science and Technology,
Wuhan, 430074 China
thman@163.com

Abstract. With the recent rapid growth of interest in Multi-Agent System (MAS), both in distributed artificial intelligence and software engineering, has come an associated difficulty concerning basic terms, concepts and corresponding formal methods. In this paper, a new definition of agent companying with soft gene and role is proposed. As a distributed system, a MAS formal method is presented.

1 Introduction

Multi-Agent System (MAS) [1] attracts much attentions in software development field. They are even regarded as a revolution by some researchers. We have defined agents in [2] through the underlying homogeneousness between human being and software agent.

The remainder of this paper is organized as follows. Section 2 presents the main idea of soft gene, role and agent. Section 3 gives a formal method to be applied to distributed MAS. Section 4 discusses related works and gives some conclusions.

2 Soft Gene, Role, and Agent

In [2], by considering Fig 1, we define soft gene, role and then give a definition for agent:

Definition 1: A soft gene is an entity that comprises a set of behaviors and a set of attributes.

* This work is supported by the National Natural Science Foundation of China under Grant No.600003002; the National High Technology Development 863 Program of China under Grant No.2002AA116070.

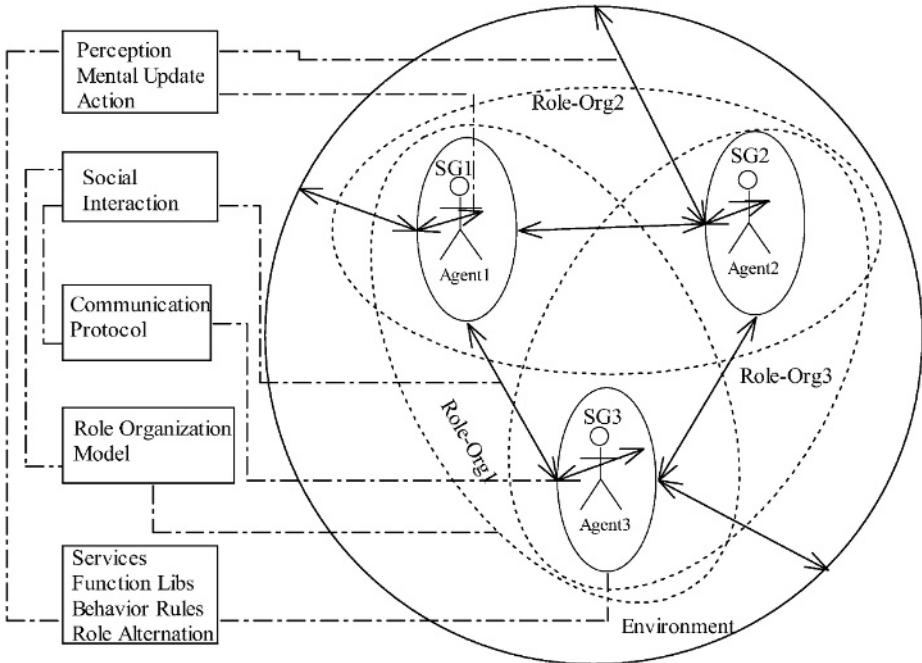


Fig. 1. Agent's activities, models and layers [2], SG_i means a soft gene, Role-Org_i is composed by roles that may be bound to SG_i. Rectangles represent actions and tool system that used by agents.

Definition 2: A role is a software component that satisfies a goal or set of goals, provides some services.

Definition 3: An agent is a software component that composed by certain soft genes and bound with certain roles.

For detailed illustration about Fig 1, readers referred to [2]. We also explain in [2] that such definitions may help to model dynamic and open MAS.

3 The Formal Method for MAS as a Distributed System

MAS is regarded as a series of processing units

$$AU = \{AU_1, AU_2, \dots, AU_n\}$$

These units do not share memory but only communicate through messages. Such a distributed system is a series of cooperative processes(agents)

$$A = \{A_1, A_2, \dots, A_n\}$$

For convenience, we assume that every process is related to only one processing unit, although the real condition is not so.

3.1 Finite State Machine

Looking from a process, a distributed system comprises only two parts: clock and local state. To a MAS, local state is the set of mental attitudes of an agent. Local state is changed by events: external or internal. An external event includes a send event and a receive event.

We use Finite State Machine (FSM) as the formal method for MAS as a distributed system. Normally a FSM comprises a set of possible commands C and a set of possible global states S . Each global state is the set of local states and its accurate definition includes local states and channel states. A function e (event) is defined as:

$$e: C \times S \rightarrow S$$

$s \rightarrow s'$ under c show that the execution of command c in C under state s may cause system state shift from s to s' . A system is determinate if and only if for all $s \in S$ and $c \in C$, there are no more than one s' : $s \rightarrow s'$ under c , otherwise the system is not determinate.

3.2 Event Order

In a centralized system, the order of every two events of two processes since that there is a shared memory and common clock. We define a relation to describe the order:

Definition 4: \rightarrow is a not reflexive but partial order:

1. if a and b are events in the same process and a is executed before b , then $a \rightarrow b$.
2. if a is the sending message event in one process and b is the corresponding receiving message event in another process, then $a \rightarrow b$.
3. if $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$.
4. Generally $\neg(a \rightarrow a)$ is right to every event a .

The \rightarrow relation can be illustrated by time-space view. In the view, horizontal way represents space and vertical way represents time; the vertical lines with notations are processes; the points with notations are events; the arrow lines are messages. Figure 2 shows a MAS comprises three agents A_1, A_2, A_3 , each of which has some events with notations. We can conclude $b_2 \rightarrow a_3$ from $b_2 \rightarrow a_2, a_2 \rightarrow a_3$. If for two different events a and b , $\neg(a \rightarrow b)$ and $\neg(b \rightarrow a)$, then we say that a and b are concurrent. For example, in figure 1, events c_2 and a_0 are concurrent.

3.3 Global State in Time-Space View

Global state GS is composed of the set of local states LS_i and the set of channel states. We first define GS as the set of local states, and then consider channel states to keep a consistent global state. $GS = (LS_1, LS_2, \dots, LS_n)$. Two more concepts are needed, which denote sets. $s(m), r(m)$ respectively represents message m 's sending event and receiving event.

In transition: $transition(LS_i, LS_j) = \{m | s(m) \in LS_i \wedge r(m) \in LS_j\}$

Inconsistent: $inconsistent(LS_i, LS_j) = \{m | \neg(s(m) \in LS_i) \wedge r(m) \in LS_j\}$

The set transition contains all the messages on the channel between agent A_i and A_j . The set inconsistent contains all the messages whose receiving events have been recorded by A_j while sending event have not been recorded by A_i .

GS is consistent if and only if $\forall i, \forall j, inconsistent(LS_i, LS_j) = \emptyset$

GS is not in transition if and only if $\forall i, \forall j, transition(LS_i, LS_j) = \emptyset$

We call a GS is strong consistent if it is consistent and not in transition. In other words, all the local states are consistent and no messages are in transition.

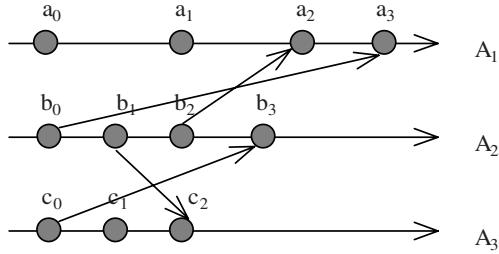


Fig. 2. A time-space view of a distributed system, which is composed by agent A_1, A_2, A_3

3.4 Logical Clock

In our method, each agent A_i has a logical clock LC_i , which keeps increasing and is initiated as $init_i (\geq 0)$. Every message m sent by agent A_i is denoted with the current LC_i value and agent id i , thus compose a triple $\langle m, LC_i, i \rangle$. To update LC_i , we use these rules:

1. before an event happening, update LC_i : $LC_i = LC_i + d (d > 0)$, where d is different in different applications.
2. when receiving a time-stamped message $\langle m, LC_j, j \rangle$, A_i updates: $LC_i = max(LC_i, LC_j) + d (d > 0)$.

Thus we can determine the logical time of the events in figure 1 as table 1 represents. We can easily prove that: for events a and b , if $a \rightarrow b$ then $LC(a) < LC(b)$, where $LC(x)$ is x 's logical time. Besides, other types of logical clock exist [3].

Table 1. Events and their logical time in figure 1, assuming that $d=1$, for A_i , $init_i=0$.

event	a_0	a_1	a_2	a_3	b_0	b_1	b_2	b_3	c_0	c_1	c_2
Logical Time	1	2	4	5	1	2	3	4	1	2	3

4 Related Works and Conclusions

Among role concept researches [4,5], role-based (including Gaia and MaSE, etc.) methodologies have not given the formal specification of corresponding concepts. Few formal methods for MAS as distributed systems are proposed before.

In this paper, we propose a new definition of agent based on soft gene and role. The new definition leads us to a diagrammatic role-based modelling method supporting MAS analysis and design [6,2]. A formal method for MAS is proposed to describe MAS agents actions, messages and (global and local) states. All these works may be helpful to agent's autonomy and MAS formalization as a distributed system.

References

1. Wooldridge, M., Ciancarini, P.: Agent-Oriented Software Engineering: The State of The Art. in Handbook of Software Engineering and Knowledge Engineering (2001,World Scientific Publishing)
2. Yan, Q., Mao, X.J., Zhu, H., Qi, Z.C. (P. Giorgini, and J. Odell (Eds.), Agent-Oriented Software Engineering IV, Proceedings of the Fourth International Workshop AOSE 2003, Melbourne, Australia, July 2003, Springer, 2004)
3. Mattern, F.: Virtual Time and Global States of Distributed Systems. Proc. Of Parallel and Distributed Algorithms Conf. (1988) 215–226
4. Reenskaug, T., Wold, P., Lehne, O.A.: Working with Objects, The OOram Software Engineering Method. Manning Publications Co, Greenwich (1996)
5. Wooldridge, M., Jennings, N., Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems 3 (2000) 285–312
6. Yan, Q., Shan, L.J., Mao, X.J.: RoMAS: A Role-Based Modeling Method for Multi-Agent System. Proceedings of International Conference on Active Media Technology, World Scientific Publishing (2003)

Optimal Fixed Priority Assignment with Limited Priority Levels*

Xuelian Bin¹, Yuhai Yang², and Shiyao Jin³

¹ School of Computer Science,
National University of Defense Technology,
ChangSha 410073,P.R.China
binxuelian@yeah.net

² Staff Room of Computer,
Airforce Army Radar Academy,
Wuhan 430010, P.R.China
yhyang@wtwh.com.cn

³ School of Computer Science,
National University of Defense Technology,
ChangSha 410073,P.R.China
syjin@nudt.edu.cn

Abstract. The schedulability utilization of fixed priority scheduling will be reduced greatly if priority levels of the system are insufficient. In this case, more than one task must be grouped into a same priority. This paper extends necessary and sufficient conditions for analyzing the schedulability of fixed priority algorithms on resources with limited priority levels. We develop an optimal fixed priority assignment algorithm (FPA) with limited priority levels. As it turns out, FPA is optimal in the sense that no other fixed priority rule can schedule a task set that can't be scheduled by FPA and the priority levels required by FPA algorithm is minimum. Furthermore, in order to reduce its time complexity, an improved algorithm (OFPA) is presented. The simulation results show that the schedulability ratio of FPA and OFPA is much higher than that of Constant Ratio Grid algorithm.

1 Introduction

The scheduling of periodic tasks with hard deadlines is an important problem for real-time systems. The scheduling algorithms are mainly divided into two classes: fixed-priority scheduling and dynamic-priority scheduling algorithms. Because of its easier implementation, lower overhead and better prediction when transient overload occurs, the fixed priority scheduling is of greater practical importance than dynamic priority scheduling. Though fixed priority scheduling has been widely used in real-time systems, it has the essential disadvantages: lower utilization, poorer adaptability, constrained by the number of priority levels supported by a system etc. This paper focuses on the fixed priority scheduling with limited priority levels.

* This work is supported by the fund of National Nature Science under Grant No.60073003.

The optimal fixed scheduling algorithms [1-4] are based on the assumption that the system supports as many priority levels as necessary. In the processor scheduling case, it is reasonable to assume that there is sufficient number of priority levels to assign distinct priorities to each task. However, the assumption is not realistic if considering bus or network. For example, MSI STD BUS [9] has 2 priority levels. IEEE 802.6 Metropolitan Area Network [10] defines 4 priority levels. As the number of tasks is less than or equal to the number of the priority levels provided by a system, the optimal priority assignment algorithm will not change. Nonetheless, a task set may require more priority levels than a system can support. In this case, more than one task must be grouped into the same priority level. This will result in priority inversion and significantly reduced schedulability utilization.

The problem of scheduling with a limited number of priority levels was first treated in [1]. In [1], a CRG (Constant Ratio Grid) algorithm has been proposed. The system was considered to have a logarithmic priority grid $\{l_0, l_1, \dots, l_k\}$, where each l_i was assigned a task period. l_k was the largest period in the task set and $l_0 \rightarrow 0$ was smallest possible period. If a task had a period T such that $l_{i-1} < T \leq l_i$, it was assigned priority l_i . It is easy to implement, nevertheless the schedulability utilization is very low. When the number of priority levels is 4, the schedulability utilization is less than 0.1. [6] presented necessary and sufficient conditions for the tasks with deadlines less than or equal to their periods, scheduled on a system with limited priority levels. However, no priority assignment algorithm has been presented in the paper. The optimal priority assignment can be obtained through a search of exponential complexity. [7] studied the problem of the minimum number of priority levels required for RMS. However no priority assignment algorithm was shown in it.

The main purpose of this paper is to provide an optimal fixed priority assignment algorithm with limited priority levels. This paper builds on the work [6], and extends both necessary and sufficient conditions for tasks with arbitrary deadlines to determine whether the timing constraints of a given task set can be guaranteed if it is scheduled on a system with limited priority levels. Then an optimal priority assignment algorithm for the case of limited priority levels is presented. Furthermore, in order to reduce its time complexity, an improved algorithm is developed in the paper. Finally, the simulation results show that although running time of the algorithms in the paper is greater than that of CRG algorithm, their schedulability ratio is much greater than that of CRG algorithm.

2 Necessary and Sufficient Schedulability Conditions with Limited Priority Levels

In this paper, we consider a fixed priority scheduling tasks set $\Gamma = \{r_i | 1 \leq i \leq n\}$ on a single processor. Each task is characterized by a period (T_i), a deadline (D_i) and a computation time (C_i). The deadline of a task r_i is arbitrary, where it is only required that $C_i \leq D_i$ for $\forall i$. (In this case many requests of a single task may be simultaneously active even if the system is feasible, i.e., all deadlines are met.) We assume that (1) all tasks are independent of each other (e.g. they do not interact); (2) All tasks are periodic and their offsets are zero. The j^{th} request of a task r_i is called a task instant denoted by r_{ij} . Suppose the number of priority levels that a system provides is N . The

relation between tasks and priority levels can be regarded as a surjective mapping between tasks and priority levels. A priority level can be regarded as a priority group denoted by g_z , where z is its system priority. Each priority group is a set of tasks that execute at the same priority level on the system. We can think that there are two priorities of a task r_i . One is natural priority assigned using the optimal priority assignment with unlimited priority levels [1-4]. Let $p(r_i)$ denote the natural priority of r_i . The other is system priority denoted by $l(r_i)$. Suppose tasks r_1, r_2, \dots, r_n and priority levels are arranged in decreasing priority order.

Let $r_i \in g_z$, if $D_i \leq T_i$, then r_i will meet all its deadlines if and only if the equation (1) holds [6].

$$\min_{0 \leq t \leq D_i} W_i(t)/t \leq 1 \quad (1)$$

$$\text{Where } W_i(t) = \sum_{r_j \in g_p, p < z} C_j \left\lceil \frac{t}{T_j} \right\rceil + \sum_{\forall r_q \in g_z} C_q \quad (2)$$

Let $r_i \in g_z$, if $D_i > T_i$, then r_i will meet all its deadlines if and only if the equation (3) holds.

$$\max_{k=0,1,2,\dots,d} W_i(k) - kT_i \leq D_i \quad (3)$$

$$\text{Where } W_i(d) - (d+1)T_i < 0 \quad (4)$$

$$W_i(k) = (k+1)C_i + \sum_{r_j \in g_p, p < z} \left\lceil \frac{W_i(k)}{T_j} \right\rceil C_j + \sum_{r_q \in g_z, q \neq i} m_q * C_q \quad (5)$$

$$m_q = \left\lfloor \frac{S_i(k)}{T_q} \right\rfloor + 1 \quad (6)$$

$$S_i(k) = k * T_i \quad (7)$$

Proof. $S_i(k)$ is the arrival time of a task instance r_{ik} (formula 7). A priority queue corresponds to a priority group. In a priority queue, there are two locations in the priority queue for instances of a task mapped to g_z ($r_i \in g_z$). One location is ahead of the task instance r_{ik} if they arrive before $S_i(k)$. Another is at the back of r_{ik} as their arrival time after $S_i(k)$. All task instances are assumed to be queued ahead of r_{ik} if their arrival time coincides with $S_i(k)$ and they are members of group g_z . According to fixed priority scheduling, in a priority queue the task instances preceding r_{ik} will delay r_{ik} , while the task instances at the back of r_{ik} has no effect on r_{ik} . Only when the system completes the cumulative work consisting of both the task instances with higher system priorities than that of r_i , which arrive before time t , and those queued ahead of r_{ik} in the same priority queue, can r_{ik} start running. Once r_{ik} has started running, only the tasks with higher system priorities are capable of preempt r_{ik} . The cumulative required computation time to finish r_{ik} is given by $W_i(k)$. The number of instances of a

task r_q arriving before $S_i(k)$ is $\left\lfloor \frac{S_i(k)}{T_q} \right\rfloor + 1$, which is mapped to g_z . If there is an instance of r_q sharing the arrival time of r_{ik} , then the number of instances of task r_q queued before r_{ik} is $\frac{S_i(k)}{T_q} + 1 = \left\lfloor \frac{S_i(k)}{T_q} \right\rfloor + 1$ (formula 6). The number of instances of a task r_j with higher system priority arriving before $W_i(k)$ is $\left\lceil \frac{W_i(k)}{T_j} \right\rceil$. At $W_i(k)$, the number of instances of r_i is $(k+1)$. As it can be seen from lemma 8 in [8], in a synchronous system, we simply need to study the schedule of the most demanding arrival pattern in the first busy period, i.e. in the first interval from time $t=0$ up to the first process idle time (formula 4). According to fixed scheduling policy, the system will not execute any task instances whose system priorities are lower than that of r_i before $W_i(k)$. Therefore, formula 5 holds.

According to fixed priority scheduling, since the worst-case response time of a task is less than or equal to its deadline, the task is feasibly schedulable. Thus formula 3 holds.

3 Priority Assignment Algorithm with Limited Priority Levels

For convenience, priority assignment algorithm with limited priority levels will be called system priority assignment for short.

Consider the ways that n tasks may be arranged into N priority groups, where $N < n$. The total number of ways that n tasks can be arranged into N priority groups is $(n-1)!/((N-1)!(n-N)!)$. It is therefore the optimal priority assignment requires a search of exponential complexity. Hence, an effective method to assign system priority should be developed to reduce time complexity.

Lemma1. The task with system priority lower than that of r_i will not affect the worst-case response time of r_i .

According to fixed priority scheduling, we can know r_j with lower system priority will not affect the worst-case response time of r_i , no matter that its natural priority is higher or lower than that of r_i .

Theorem 1. Remove a task r_i from group g_k and add it to group g_j may only affect the worst-case response time of its own and those which are mapped between g_k and g_j .

Proof. It can be seen by examining the equation (1) and (3), the system priority of a task r_i determines which tasks may be blocked. These tasks are those with the same system priority as task r_i . The response time of these tasks may be thereby affected when the system priority of r_i is modified. It also affects its own response time since it changes the set of tasks that can preempt it once it has started running. Note that, a lower system priority task r_j is not impacted by r_i . Therefore, changing the system priority of r_i will not affect any lower system priority task.

Since the number of tasks is greater than the number of priority levels supported by a system, there is a need that more than one task will be grouped into one priority level. Formally, we use the following rules to assign system priority of tasks.

Rule (1) [6]. Assume task $r_i \in g_k$ and task $r_j \in g_l$ ($k \neq l$). If $p(r_i) > p(r_j)$, then it must be the case that $k < l$; if $p(r_i) < p(r_j)$, then it must be the case $k > l$; if $p(r_i) = p(r_j)$, then it must be the case that either $k = l + 1$ or $l = k + 1$.

Rule (2). If r_i is mapped to g_k , all tasks in g_k will satisfy the necessary and sufficient (N&S) conditions (formula 1 or 3). Namely if $D_i \leq T_i$, formula (1) should hold; if $D_i > T_i$, formula (3) should hold.

According to rule (1), since the natural priority of r_i is higher than that of r_j , either the system priority of r_i is equal to that of task r_j or the system priority of r_i is higher than that of r_j . In addition, according to rule (2), whenever a task is added into a priority group, it is of necessity to check whether there is a task in the group that doesn't satisfy N&S conditions.

Thus the assignment rules form the basis of the optimal system priority assignment algorithm FPA.

Basic idea. The natural priorities of tasks can be set using the optimal priority assignment algorithm [1-4]. According to rule (1), a task arranged higher natural priority should also be mapped to higher priority group. Consequently, the highest system priority coincides with the highest natural priority. Hence, it offers a heuristic way to assign system priorities of tasks in decreasing natural priority order. Firstly, the highest natural priority task r_1 is mapped to the highest priority group g_1 . Secondly, according to formula (1) or (3), the next highest natural priority task r_2 is checked whether it can be mapped to the highest priority group. If it can, then it is arranged the highest system priority. Otherwise it is mapped to the next highest priority group g_2 . Repeat the above processing until all tasks are arranged system priorities or the number of system priority levels required is greater than that of the system provides.

Before a new task is mapped to a group g_z , all tasks in g_z should be checked whether they satisfy N&S conditions. When the first task is mapped to group g_z , it needs to check whether its own satisfies N&S conditions. When the second task is mapped to group g_z , it needs to check whether the first task and its own satisfy N&S conditions. When the m^{th} task is mapped to g_z , it needs to check whether the previous $m-1$ tasks and its own in g_z satisfy N&S conditions. Consequently, if there are m tasks in the group g_z , the total checking number is $1+2+\dots+m=(m+1)m/2$. When all tasks are uniformly distributed in all groups, the number needs to check is minimal. The least checking number is $(n/N+1)(n/N)*N/2=(n+N)n/(2N)$. When all tasks are partitioned in one group, the checking number is maximal. The maximal checking number is $(n+1)n/2$. Hence, time complexity of FPA is $O(n^2)$.

The FPA is optimum in the sense that for a given a task set and a given number of priority levels, if the task set can be feasibly scheduled using FPA, the number of priority levels required is minimal; if there is a algorithm making the task set schedulable, then FPA can also make it schedulable.

Theorem 2. Assume FPA can feasibly schedule a given task set with a given number of priority levels. The number of priority levels required by FPA is minimal.

Proof. Suppose task set Γ can be schedulable using FPA and the number of priority levels required is p . Let $A(i)$ denote the priority group for FPA. If an algorithm B exists, making Γ schedulable and the number of priority levels required is q ($q < p$),

then there is at least one priority group, where tasks must be re-partitioned over other groups. If the first task r_j of $A(l)$ is mapped to $A(l-1)$, then a certain task in $A(l-1)$ will not satisfy rule (2). Otherwise, it will be mapped to $A(l-1)$ using FPA rather than $A(l)$. Similar to the above analysis, it will consequently result in a certain task in $A(l+1)$ unschedulable if r_j is mapped to group $A(l+1)$. It is in contradiction with the assumption that algorithm B can make task set Γ schedulable. Hence, theorem 2 holds.

Theorem 3. For a given task set and a given number of priority levels, if there is a fixed priority assignment algorithm making the task set schedulable, then FPA can also make it schedulable.

Theorem 3 implies that if it isn't possible for FPA to schedule a task set, it is also of no possibility for other fixed priority assignment algorithm to schedule it.

Proof. There is no fixed priority assignment algorithm can make a task set schedulable, if it is not schedulable with unlimited priority levels.

Hence in the following, we will only discuss the task set that is schedulable with unlimited priority levels. Assume a task set $\Gamma = \{r_i | 1 \leq i \leq n\}$ and the number of priority levels supported by a system is m ($m < n$). Tasks in the sets are arranged in decreasing natural priority order, whose natural priorities are given using the optimal priority assignment algorithm [1-4].

Suppose the task set Γ is not schedulable using FPA and r_i is the first unschedulable task. This means that tasks r_1, \dots, r_{i-1} are schedulable and they are mapped to m groups. Moreover, if task r_i is mapped to the m^{th} group, it will result in the worst-case response time of a certain task in the group, missing its deadline.

Suppose the task set Γ is schedulable by an algorithm B and the tasks r_1, \dots, r_{i-1} are mapped to m' ($m' \leq m$) groups. According to theorem 2, m' is no less than m , hence $m' = m$. Since task r_i is not schedulable using FPA, it will result in a certain task r_j in the m^{th} group unschedulable if adds it to m^{th} group. Remove r_j from m^{th} group and add it to $(m-1)^{\text{th}}$ group. According to theorem 2, it will also lead to a certain task in the $(m-1)^{\text{th}}$ unschedulable. Repeatedly adds the unschedulable task in a group to its preceding group until the first group. Due to adding a new task, a certain task in the first group will not schedulable. Therefore, algorithm B is not able to make task set Γ schedulable. It is in contradiction with the assumption that algorithm B makes task set Γ schedulable.

Hence, theorem 3 holds.

4 Improved System Priority Assignment Algorithm: Opt-FPA

It need check all the tasks in a group whether they are schedulable using FPA algorithm. In the worst case, the checking numbers are $(n+1) n/2$. In order to reduce the running time of FPA, it is necessary to reduce the checking numbers.

As more than one tasks are mapped to a group, lower natural priority tasks may block a higher natural priority task within its group. Thus, to check whether a task satisfies N&S conditions is equivalent to check whether the blocking time of lower natural tasks is greater than the maximal blocked time that allowed for a higher natural priority task.

Definition. It is called saturation of a level if a task is mapped to a group resulting in at least one task in the group unschedulable.

As $D_i \leq T_i$, formula (1) can be rewritten as formula (8).

$$\min_{0 < i \leq D_i} (t - \sum_{r_j \in g_p, l(r_p) > l(r_i)} C_j \left\lceil \frac{t}{T_j} \right\rceil) \geq \sum_{l(r_q) = l(r_i)} C_q \quad (8)$$

It can be seen from [11], checking whether a task r_i satisfies formula (8), it only needs to examine the time instances $t = \{k * T_j | r_j \in g_p, l(r_p) > l(r_i); k=1, \dots, \left\lfloor \frac{D_i}{T_j} \right\rfloor\}$.

According to formula (8), we know that for a given time instance, the maximal blocked time that allowed for a task r_i is dependent on tasks with higher system priority, not related with tasks of the same system priority $l(r_i)$. Let H_i denote the maximal blocked time that allowed for a task r_i .

$$H_i = \min_{0 < i \leq D_i} (t - \sum_{r_j \in g_p, l(r_p) > l(r_i)} C_j \left\lceil \frac{t}{T_j} \right\rceil) \quad (9)$$

Only if $H_i \geq \sum_{l(r_q) = l(r_i)} C_q$, does r_i satisfy the N&S conditions.

As $D_i > T_i$, formula (3) can be rewritten as formula (10).

$$\min_{0 < i \leq D_i} (t - \sum_{r_j \in g_p, l(r_j) > l(r_i)} \left\lceil \frac{t}{T_j} \right\rceil) \geq \sum_{l(r_q) = l(r_i), l \neq i} m_l * C_l + \left\lceil \frac{t}{T_i} \right\rceil * C_i \quad (10)$$

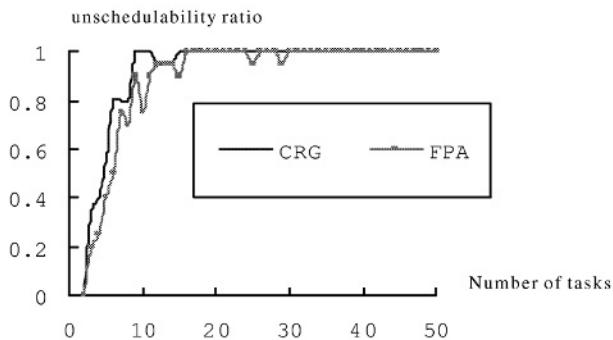
It can be seen from formula (10) for a given time instance the maximal blocked time that allowed for a task r_i is related with tasks of higher and the same system priority $l(r_j)$. It is therefore that formula (9) is not actual maximal blocked time that allowed for a task r_i . However, it is obvious that H_i should be at least as large as $\sum_{l(r_q) = l(r_i)} C_q$, if task r_i is schedulable.

Therefore, we are able to apply the following method to reduce the checking numbers. It only needs to check whether the task with the minimal H_i satisfy N&S conditions, not all tasks in a group need to be checked. The tasks are classified into two categories according to their deadlines. One is the case that tasks with deadlines less than or equal to their periods. Let $H = \min_{l(r_j) = l(r_i) \wedge D_j \leq T_j, r_j \in \Gamma} H_j$. If $H < \sum_{l(r_q) = l(r_i)} C_q$, the level

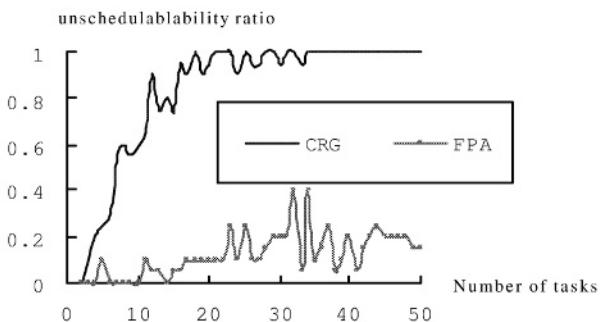
is saturated. Hence, the system priority of the next task should be $l(r_i) + 1$. Otherwise its system priority is $l(r_i)$. Another is the case that tasks with deadlines larger than their periods. Let $H = \min_{l(r_j) = l(r_i) \wedge D_j > T_j, r_j \in \Gamma} H_j$. If $H < \sum_{l(r_q) = l(r_i)} C_q$, then the level is saturated and the

system priority of the next task should be $l(r_i) + 1$. Otherwise, it needs further checking whether a task with deadline larger than its period is schedulable.

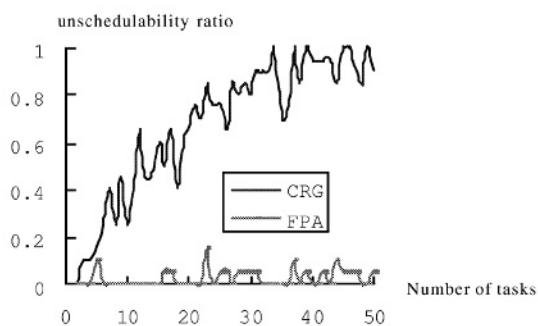
The only difference between OPT-FPA and FPA is that the way of determining whether a task can be mapped to a priority group. OPT-FPA uses the above method to reduce checking numbers.



(a). The number 2 of priority levels



(b). The number 4 of priority levels



(C). The number 8 of priority levels

Fig. 1. Unschedulability distribution of CRG and FPA

5 Performance Comparison

In order to analyze the performance of our algorithms, we compare them with CRG algorithm. The quantitative measurements are the unschedulability ratio and the running time of the algorithms.

Assume the number of priority levels supported is 2, 4, and 8 respectively.

We divide the task sets into 49 classes according to the number of tasks in a set. The number of tasks in the set increases from 2 to 50 and each class has 20 sets. Therefore, the number of the total task sets to be experimented is 980.

Each set in the classes is generated by the following conditions:

- The period of a task is uniform distributed over the range [10,1000].
- The deadline of a task is uniformly distributed over the range [1/4* period, period].
- To keep the computation time randomly generated and make the set schedulable with unlimited priority levels, the computation time is uniformly probability distributed in the range [1, 4*deadline/the number of tasks in the set].

The tasks are arranged in decreasing natural priority order.

Figure 1 and figure 2 are the results of the experiments. Unschedulability ratio is defined as the number of unschedulable sets divides 20 (all the sets in one class are schedulable with unlimited priority levels). The ratio in figure 2 is the result of the running time of FPA divides by that of Opt-FPA.

Figure 1 is the unschedulability distribution of FPA and CRG. Due to the same results of priority assignment using FPA and Opt-FPA, results of the unschedulability ratio of CRG and FPA are depicted in figure 1. Figure (1)a, (1)b, (1)c are the unschedulability ratio distribution with 2,4, 8 priority levels, respectively.

For little overhead of CRG, its running time is near zero. Thus, we only consider the running time of FPA and Opt-FPA. When the number of tasks is very small (less than or equal to 10), the running time of Opt-FPA is also near zero. Therefore, we just plotted the running time of FPA and Opt-FPA for sets where the number of the tasks is larger than 10 and the number of system priority level is 8 in Figure 2.

It is known from figure 1 schedulability ratio increases with increasing number of priority levels for a given task set. While schedulability ratio of CRG algorithm is the least, the schedulability of FPA and Opt-FPAs are much greater than that of CRG. We can see from figure 2 that the running time of Opt-FPA is much less than that of FPA. Therefore, the running time of CRG is minimal and FPA is largest.

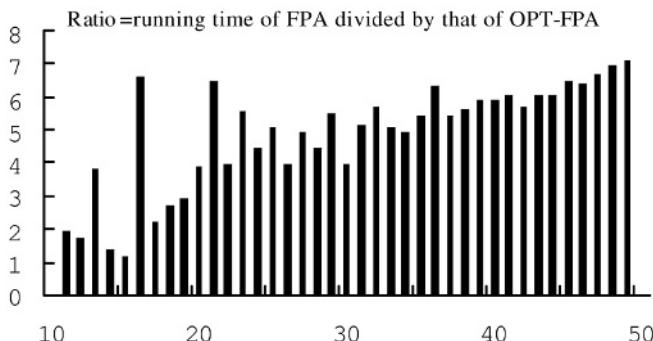


Fig. 2. Running time of FPA and OPT-FPA

6 Conclusions

Fixed priority scheduling has been widely used in real-time systems. Nonetheless, it is constrained by the priority levels supported by a system. The focus of this paper is the fixed priority assignment algorithm with limited priority levels. A necessary and sufficient condition for fixed priority scheduling with limited priority levels is developed for tasks with deadlines greater than their periods. An optimal priority assignment algorithm FPA is then presented. In order to reduce the time complexity of FPA, an improved algorithm Opt-FPA is further derived. The simulation results show that although the running time of CRG less than that of FPA and Opt-FPA, the schedulability ratio of FPA and Opt-FPA is much greater than that of CRG.

References

1. C.L.Liu. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, Vol. 20, No. 1, (1973) 46–61
2. Jan. N. Audsley, A. Burns, M. Richardson, K. Tindell, A. Wellings. Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling. *Software Engineering Journal*, Vol. 8, No. 5, (1993) 284–292
3. N. C. Audsley. Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times. *YCS 164*, Dept. Computer Science, University of York (December 1991)
4. Wei Kuan Shih, J.W.S. Liu, C.L. Liu. Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines. *IEEE transactions on Software Engineering*, Vol.19, No.12, (1993) 1171–1179
5. John P. Lehoczky, Lui Sha. Performance of Real-time Bus Scheduling Algorithms. *ACM SIGMETRICS Performance Evaluation Review*, Vol. 14, No.1, (1986) 44–53
6. Danel I. Katcher, Shirish S. Sathaye, Jay K. Strosnider. Fixed Priority Scheduling with Limited Priority Levels. *IEEE Transactions on Computers*, Vol. 44, No.9, (1995) 1140–1144
7. Javier Orozco, Ricardo Cayssials, Jorge Santos, Rodrigo Santos. On The Minimum Number of Priority Levels Required for Rate Monotonic Scheduling of Real-Time Systems. *10th EUROMICRO Workshop on Real Time Systems*, June 17–19th, 1998, Berlin, Germany
8. Laurent George, Nicolas Rivierre, Marco Spuri. Preemptive and Non-Preemptive Real-Time Uniprocessor Scheduling. *Rapport de Recherche n°2966*, Inria, 1996
9. MSI-C851 STD BUS 80C51 Microcontroller Card. *Microcomputer Systems*. www.microcomputersystems.com
10. IEEE, 802.6 Distributed Queue Dual Bus- Metropolitan Area Network, 1990
11. J.Lehoczky, L Sha, Y. Ding. The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior. In *Proceedings of 8th IEEE Real-Time Systems Symposium*, IEEE Computer Society Press, (1989) 166–171

A Proof Assistant for Mobile Processes*

Xutao Du and Zhoujun Li

School of Computer Science

National University of Defence Technology

Changsha 410073, P.R.China

Abstract. The π -calculus is invented for modelling *mobility* (the ability to dynamically change the communication structures) within systems. It can conveniently represents hardware or software concurrent systems with evolving communication structures. After modelling those systems with the π -calculus, we can formally verify the correctness of them. The verification process is error-prone since there are many details. Hence some mechanical support is necessary. We have developed an interactive proof assistant for the monadic π -calculus, namely PiM(for the Pi-calculus Manipulator). It is based on *symbolic* style proof systems with respect to different bisimulation congruences. And we have generalized a version of the unique fixpoint induction to deal with recursions. Using PiM users can check different bisimulation congruences between π -calculus agents by directly manipulating syntactic terms of them.

1 Introduction

Large hardware or software systems are created with the hope that they will work as intended. To verify this is difficult due to the complexity of the problem. And it is especially a hard work when concurrent systems, in which certain parts may present some *mobile* characteristics, are under consideration. In order to carry out the verification the system and its intended behavior, or in other words, the implementation and the specification, should be described in a *formal* way before we can go on to *prove* their *equality*.

Process algebras are widely used in describing and analyzing concurrent systems. Among them, the π -calculus is the one capable of modelling concurrent systems with evolving communication structure. That is why it is called “a calculus of mobile processes” [16].

In process algebras, *bisimilarity* is used as the definition of *equality*. As for the π -calculus, the late and early bisimulation congruences, and the open bisimulation congruences are typical ones. Based on these bisimulation congruences we can check whether two *processes*, for example, the implementation and the specification of a concurrent system, are *equal*.

* This work is supported by National Natural Science Foundation of China (No.90104026 and No.60073001), 863 Project(2002AA144040) and Visiting Scholar Foundation of Key Lab. In University.

There are two ways to go in such a verification. Through operational semantics, we can prove the equality “semantically”. Or we can work directly on syntactic terms to do the same task, which requires a sound and complete axiomatization for a specific bisimulation congruence. No matter which way we choose, the proof is not so easy when a large system is under consideration that some kind of mechanical support is needed to avoid errors.

We have developed a proof assistant PiM. Users can interactively check whether two π -calculus agents are bisimilar by directly manipulating syntactic terms of them. Based on complete axiomatizations for the late and early bisimulation congruences [19,23] and those for the open bisimulation congruences [17], PiM allows users to do verifications with respect to the specific bisimulation they choose. A generalized version of the unique fixpoint induction is implemented to deal with recursions.

As far as we know, PiM is the first interactive theorem prover for the π -calculus. We have proved a real-world problem(the correctness of the AB protocol) using PiM. The whole proof is about 300 lines. And our proof is at variance with the content in Robin Milner’s book [9]. In later section, we will say more about this. The aim of this paper is to introduce our tool PiM.

The rest of this paper is organized as follows: we relate our work to that of others in the next section. In Sect.3 the language of the monadic π -calculus and a simple theory of conditions are introduced. Section 4 describes the proof systems underlining PiM. How to carry out proofs in PiM is shown through examples in Sect.5. We conclude this paper with Sect.6 where some future work are also discussed.

2 Related Work

There are many proof tools based on the theoretical results in process algebras. The CWB[1], AUTO[13], PSF[4], PAM[7]/VPAM[8], MWB[10] and HAL[11] are a few of them. Some of them are automated and some are interactive.

As for the π -calculus, there are only two verification tools in existence: MWB and HAL. Both of them are automated. The main functionality of MWB is the verification of open bisimulation equivalences between π -calculus agents. It can also be used to find deadlocks and for interactively simulating agents through various commands provided. MWB employs the “on-the-fly” method which generates the state spaces of systems at the same time as the bisimulation relation is being built.

HAL is an automated tool which can check whether two π -calculus agents are (strong or weak) bisimilar. The language HAL uses is the monadic π -calculus with the positive match operator. HAL is built upon the JACK tool. It translates π -calculus agents to ordinary automata, and then passes them to JACK’s equivalence checker to verify their equivalences. Automata minimization, according to the weak bisimilarity is also possible, by using the functionalities provided in JACK by the HOGGAR tool.

Value-Passing Algebra Manipulator(VPAM) [8] is the first proof tool that can deal with value-passing process calculi. As an interactive proof tool for value-passing process algebras, VPAM has influenced us much in several aspects. It implements *symbolic* inference systems [6]. Users are allowed to define their own calculi by specifying their own operators and axioms(inference rules are fixed). Hence, it actually provides a general proof platform for value-passing calculi. Because of the data domain involved in value-passing calculi, VPAM relies on an independent theorem prover for reasoning about data. In fact, we have got much help from Prof. Lin, the author of VPAM. Thanks to Lin.

3 The Language and Conditions

There are two basic types of π -calculus: the *monadic* π -calculus, where exactly *one* name is communicated at each synchronization, and the *polyadic* π -calculus, where *zero or more* names are communicated. PiM deals with the monadic one.

In the π -calculus the data domain is a plain set of *names*. Lin has developed a simple theory of *conditions* which plays an important role in his work of axiomatizations of the π -calculus for late and early bisimulation congruences [19,23] as well as in Li's work of axiomatizations for open bisimulation congruences [17]. They have both done their work in a *symbolic* way [5]. And the proof systems they have obtained make the foundation of PiM.

3.1 The Monadic π -Calculus

We let a, b, x, y, \dots range over names and t ranges over agents. Then the language of the monadic π -calculus with the extension of *mismatch* can be described by the following BNF grammar

$$\begin{aligned} t ::= & 0 \mid \alpha.t \mid t + t \mid [x = y]t \mid [x \neq y]t \mid (x)t \mid t \mid t \mid A(y_1, \dots, y_n) \\ \alpha ::= & \tau \mid a(x) \mid \bar{a}x \end{aligned}$$

Most of these operators are from CCS [9] : 0 is a process that will do nothing, $\alpha.t$ is action prefixing, $+$ is nondeterministic choice, $|$ is parallel composition, $(x)t$ is scope restriction. The *match* construction $[x = y]t$ compares two *names* : if x and y are the same name then the process will behave like t ; otherwise it will have no action. The *mismatch* construction $[x \neq y]t$ compares x and y , and if they are equal it will have no action; otherwise it will behave like t . Each identifier A has a defining equation $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} t$ associated with it.

As in CCS τ represents communication. An input-prefixed process $a(x).t$ can receive a name along the port a then behave like t with the received name in place of x . An output-prefixed process $\bar{a}x.t$ can emit the name x along a and continue like t .

In $a(x).t$ and $(x)t$, x is a *bound* name with the scope t . The *free names* $fn(t)$ of t are the set of names which occur in t but not bound either by an input prefix or by a restriction. And in a defining equation $A(x_1, \dots, x_n) \stackrel{\text{def}}{=} t$, it is required that $fn(t) \subseteq \{x_1, \dots, x_n\}$.

3.2 Conditions

Conditions, ranged over by ϕ and ψ , are finite conjunctions of equality or inequality tests on *names*. The empty condition will be denoted by *true*.

Substitutions, ranged over by σ and δ , are mappings from N to N . $[\bar{y}/\bar{x}]$ is the substitution sending \bar{x} to \bar{y} and identity elsewhere.

A substitution σ satisfies a condition ϕ , written $\sigma \models \phi$, if $x\sigma = y\sigma$ for any $x = y$ in ϕ and $x\sigma \neq y\sigma$ for any $x \neq y$ in ϕ . We write $\phi \Rightarrow \psi$ to mean that $\sigma \models \phi$ implies $\sigma \models \psi$ for any substitution σ . The relation $\phi \Rightarrow \psi$ is decidable [18] [17].

4 The Proof Systems

The proof systems underlining PiM are each composed of a set of inference rules and some equational axioms. They are both sound and complete for finite π -calculus. And we have generalized a version of unique fixpoint induction to deal with recursions.

4.1 The Inference Rules

The inference rules for the late bisimulation congruence is presented in Fig.1.

With the standard equations for choice(S1-S4) and restriction(R1-R5) in Fig.2, we obtain a complete proof system for the late strong bisimulation congruence. If the three τ -laws in Fig.3 are added, the proof system is complete with respect to the observation congruence.

The judgments in our proof systems are of the form

$$\phi \triangleright t = u$$

where t, u are terms in the π -calculus language and ϕ is a condition.

As we have said, the semantic implication \Rightarrow between conditions is *decidable*. And since the data domain is a plain set of *names*, all questions about conditions can be answered inside PiM. Therefore, in the proof systems, questions about names are treated as side-conditions to the inference rules. If users want to apply a rule which is guarded by a side-condition, PiM will first check whether the side-condition is true. For example, in order to prove $[a = b][x = y] \triangleright \bar{a}x.t = \bar{b}y.u$, we want to use the rule `output`(in Fig.1). PiM will first check the side-conditions $[a = b][x = y] \Rightarrow a = b$ and $[a = b][x = y] \Rightarrow x = y$. Since both of them are easily checked by PiM itself following the method provided in [18] and [17], `output` will be applied and the goal will be reduced to $[a = b][x = y] \triangleright t = u$.

4.2 The Axioms

The standard axioms for choice(S1-S4) and restriction(R1-R5) in Fig.2 and τ -laws(T1-T3) in Fig.3 are rather straightforward. Each of them consists of a name, an equation, and possibly, a side-condition. Yet there is another rule called the expansion law, which is used to reduce parallelism to nondeterminism.

EQUIV	$\frac{}{true \triangleright t = t}$	$\frac{\phi \triangleright t = u}{\phi \triangleright u = t}$	$\frac{\phi \triangleright t = u \quad \phi \triangleright u = v}{\phi \triangleright t = v}$		
AXIOM	$\frac{}{true \triangleright t = u}$	$t = u$ is an axiom instance			
CHOICE	$\frac{\phi \triangleright t_i = u_i \quad i = 1, 2}{\phi \triangleright t_1 + t_2 = u_1 + u_2}$				
INPUT	$\frac{\phi \triangleright t = u}{\phi \triangleright a(x).t = b(x).u}$	$\phi \Rightarrow a = b, x \notin n(\phi)$			
OUTPUT	$\frac{\phi \triangleright t = u}{\phi \triangleright \bar{a}x.t = \bar{b}y.u}$	$\phi \Rightarrow a = b, \phi \Rightarrow x = y$			
TAU	$\frac{\phi \triangleright t = u}{\phi \triangleright \tau.t = \tau.u}$				
MATCH	$\frac{\phi \wedge [x = y] \triangleright t = u \quad \phi \wedge [x \neq y] \triangleright 0 = u}{\phi \triangleright [x = y]t = u}$				
RES	$\frac{\phi \wedge \bigwedge \{x \neq y \mid y \in f_n((x)t, (x)u)\} \triangleright t = u}{\phi \triangleright (x)t = (x)u} \quad x \notin n(\phi)$				
PARTITION	$\frac{\phi \wedge [x = y] \triangleright t = u \quad \phi \wedge [x \neq y] \triangleright t = u}{\phi \triangleright t = u}$				
CONSEQ	$\frac{\phi \triangleright t = u}{\phi' \triangleright t = u}$	$\phi' \Rightarrow \phi$			
ABSURD	$\frac{}{false \triangleright t = u}$				

Fig. 1. The Inference Rules for Late Symbolic Bisimulation

S1	$X + 0 = X$	R1	$(x)0 = 0$
S2	$X + X = X$	R2	$(x)\alpha.X = \alpha.(x)X \quad \text{if } x \notin n(\alpha)$
S3	$X + Y = Y + X$	R3	$(x)\alpha.x = 0 \quad \text{if } x \text{ is the port of } \alpha$
S4	$(X + Y) + Z = X + (Y + Z)$	R4	$(x)(y)X = (y)(x)X$
		R5	$(x)(X + Y) = (x)X + (x)Y$

Fig. 2. The Axioms for Choice and Restriction

4.3 The Unique Fixpoint Induction

In any non-trivial problems, processes(agents) are recursively defined and hence some kind of induction is needed. The unique fixpoint induction is widely used

T1	$\alpha.\tau.X = \alpha.X$
T2	$X + \tau.X = \tau.X$
T3	$\alpha.(X + \tau.Y) + \alpha.Y = \alpha.(X + \tau.Y)$

Fig. 3. τ -Laws

in the process algebra community. Lin presented a version of unique fixpoint induction for the π -calculus in [19] which is of the form:

$$\frac{\text{true} \triangleright F = G[F/X]}{\text{true} \triangleright F = \mathbf{fix} XG} \quad X \text{ guarded in } G \quad (1)$$

where F and G are *abstractions*. We refer to [19] for details about this rule. Now we give a brief description of abstraction, definition and declaration. An abstraction is either a process abstraction $(\tilde{x})T$ or a recursive abstraction $\mathbf{fix} XF$. A *definition* for X has the form

$$X \Leftarrow G$$

where G is an abstraction of the form $(\tilde{x})T$. A set of definitions

$$D = \{X_i \Leftarrow G_i \mid 1 \leq i \leq n\}$$

is called a *declaration* if each X_i is distinct and all identifiers appearing in the G_i are in the set $\{X_i \mid 1 \leq i \leq n\}$. X_i is *guarded* if every occurrence of X_i in any G_j is within some subexpression $\alpha.T(\alpha \neq \tau)$ of G_j . The declaration D is guarded if every X_i is guarded.

Rule (1) is simple and sufficient from the theoretical point of view, however, it cannot be directly applied to recursions that have more than one definition clauses and it can only be applied to abstractions. Inspired by Hennessy, Lin [21] and Rathke [22], we have obtained a generalized version of the unique fixpoint induction:

If $D = \{X_i \Leftarrow (\tilde{x}_i)T_i \mid 1 \leq i \leq n\}$ is a guarded declaration and $F_i \equiv (\tilde{x}_i)U_i$ is a collection of abstractions then

$$\text{UFI-F} \quad \frac{\phi_i \triangleright U_i = T_i[\tilde{F}/\tilde{X}]}{\phi_1 \triangleright U_1 = X_1(\tilde{x}_1)}$$

provided that $\phi_i \wedge \phi \Rightarrow \phi_j[\tilde{e}/\tilde{x}_j]$ whenever $(\phi, j, \tilde{e}) \in \text{Calls}(X_i)$.

The function $\text{Calls}(X_i)$, which is transplanted from its value-passing version [22], returns the set (ϕ, j, \tilde{e}) such that $X_j(\tilde{e})$ appears in a subexpression of T_i guarded by the condition ϕ . Details about this rule are described in [15].

The implementation of UFI-F is inspired by Lin [20]. It involves three commands `ufi`, `bind` and `known`. To apply `ufi`, we require the right hand side of the current equation to be a recursively defined identifier, possibly with parameters. When the `ufi` command is invoked PiM generates a subgoal for each of

```

Conjecture
  B2(a,c)==Buff2(a,c)
where
  B2(a,c)==(b)(Buff1(a,b)|Buff1(b,c))
  Buff1(a,b)==a(x).~b x.Buff1(a,b)

  Buff2(a,c)==a(x).Buff2'(a,c,x)
  Buff2'(a,c,x)==a(y).Buff2''(a,c,x,y)+~c x.Buff2(a,c)
  Buff2''(a,c,x,y)==~c x.Buff2'(a,c,y)
end

```

Fig. 4. Example: Two-space Buffer

the relevant definition clauses. In case that the relevant definition clauses are more than one, a pair of “unknowns” are generated for each of the identifiers in the definition clauses. One is “process unknown” of the form X'_n and the other is “condition unknown” of the form C'_n , where n is a number. Every “process unknown” should be bound to some terms using `bind` in order to complete the proof. When users binding a “process unknown”, the “condition unknown” associated with it is also being bound to the context condition of the current equation. After been bound, a “process unknown” can be “made” known, which will result in replacing the unknown with the terms bound to it previously.

5 Examples

In Fig.4 we give an example of the input file. It illustrates the syntax for problem definition in PiM. The conjecture to prove follows the keyword **Conjecture**. Agents are defined by mutual recursions after the keyword **where**. The definition clauses are self-evident except for $\bar{a} x$, which stands for $\bar{a} x$ in the π -calculus.

`B2` is the implementation of a two-space buffer by running two one-space buffer(`Buff1`) in parallel. `Buff2` is the specification of a two-space buffer. The conjecture says that the implementation of a two-space buffer and the specification of it are *equal* (observation congruent).

The whole proof process is presented in Fig.5. In the beginning only the conjecture `B2(a,c)==Buff2(a,c)` is displayed. Users choose appropriate rules and click corresponding buttons to apply certain rules.

Proofs in PiM are displayed in a mixed way: invoking inference rules is goal-directed, while applying equational axioms is implemented as rewriting. The goal-directed style can be described as follows: the equation to prove is viewed as a goal; applying an inference rule to it means matching it with the conclusion of the inference rule; a subgoal will be created for each of the premises of this rule; and if all the subgoals are proved the goal is proved.

Equational axioms should be applied to terms instead of the whole equation. One side of the equation will get rewritten and thus a new equation will be generated. If this new equation is proved the original equation is proved.

The left half of the proof window is a command panel where there is a command button for each inference rules and axioms. There are two switches:

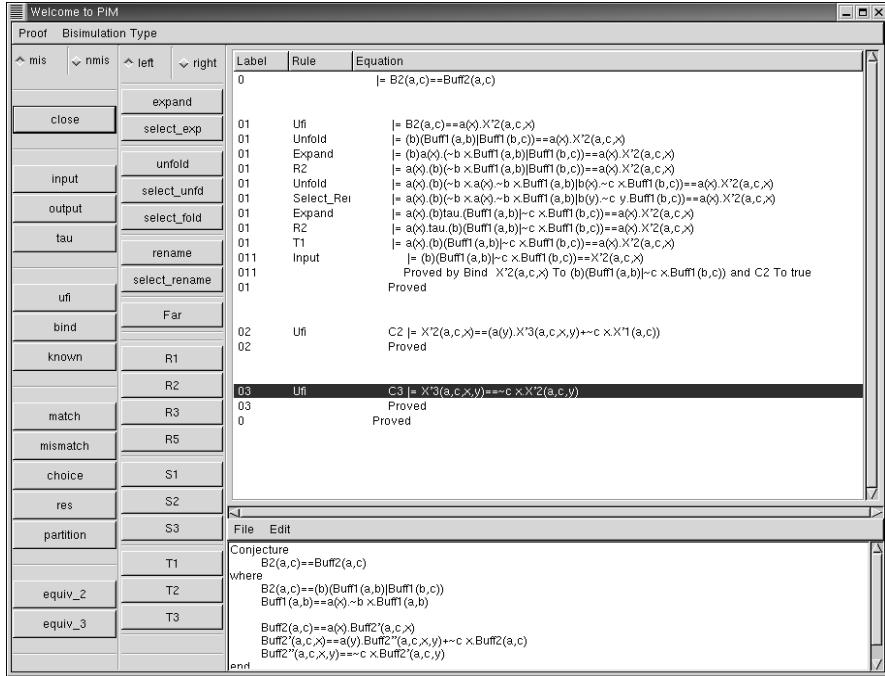


Fig. 5. The Proof Window

mis-nmis and *left-right*. *Mis-nmis* determines whether *mismatch* is allowed in current proof and *left-right* is used to determine which side of the equation is going to be rewritten when applying a rewrite rule.

There are two subwindows on the right. The upper one is for displaying proofs. Through the lower subwindow, which is a normal editor, users can see the input file corresponding to the current proof. There are three columns in the upper subwindow, namely *label*, *rule* and *equation*. *Label* is used to notify users the position of this goal equation in the whole proof goal-tree. *Rule* tells users how this goal equation is obtained from its "father". For example,

011 Input $\models ([x = t]P'(x) + [x = f]P) == ([x = t]X'2(x) + [x = f]X'1)$

means that this goal equation is the first subgoal of the goal labeled by "01" who is indeed its "father" and it is obtained by applying the rule *input* to its "father". However, if the name of a rewrite rule is in the position of *rule* then it means that this goal equation is obtained by rewriting the goal with the same *label*, which is just above it.

Every inference rules and axioms can only be applied to the *current goal*, which is highlighted when users select it. To invoke a rewrite rule, one need to choose the correct side by *left-right* switch and press the corresponding button. PiM will then rewrite the correct side of the current goal referring to the *left-right* switch and generate a new goal equation with the same *label*. Invoking an inference rule will cause PiM to match the current goal with the conclusion of

the rule and generate new subgoals according to the premises. However, some of the inference rules and axioms are guarded by some side-conditions. If such a rule is applied PiM will first check the side-conditions and if they are true PiM will go on as stated in the previous paragraph otherwise it will give an error message.

Moreover, we have proved the AB protocol in PiM. And in doing so, our proof is at variance with the content in Robin Milner's book [9]. In Chap.6 of [9], using CCS the author presented a model as the implementation of the AB protocol and a definition of a one-place buffer as the specification of the AB protocol. He gave a weak bisimulation and said that since both the implementation and the specification of the AB protocol are *stable* the weak bisimulation is actually an observation congruence (the readers are referred to [9] for details about the concepts of stable, weak bisimulation and observation congruence). In page 145, the author said "Note that both agents are stable, so this implies equality". But we found that the implementation of the AB protocol has a τ -derivative, which means it is not stable. Therefore we modified his model and transplanted it to the setting of π -calculus with the aim of getting an observation congruence in mind. We got it and proved it in PiM. Interested readers are referred to [14] for our work about the AB protocol. The whole proof of the AB protocol in PiM is about 300 lines.

We allow the not-finished proof to be *saved* using the `save proof` button because the proof of a not-so-short problem will be long and time-consuming. And a previously saved proof can be *loaded* using the `load proof` button. Since the space for displaying proofs is limited to one screen, we provide a `close` command through which one can make the sub-proof tree of the selected goal equation not to be displayed. To apply `close`, the current goal equation must be a proved one because a "closed" goal is always assumed to be proved.

In Fig.5 we can see the whole proof process of the two-space buffer problem. The second and third subgoals of the top level goal equation have been "closed" to make the whole proof fit into one page.

O'Caml is the programming language we have chosen to implement PiM. The graphical interface is implemented using *lablgtk*.

6 Conclusions

In this paper we have presented PiM, an interactive proof tool for the monadic π -calculus. It is interactive, and works as follows: users choose an appropriate rule and tell PiM to apply it to the current goal; this will result in new goals to be generated; if the generated goals are all proved, the original goal is proved; this process ends when the ultimate goal is proved. The proof systems underlying PiM are the axiomatizations for late and early bisimulation congruences [18,23] and those for open bisimulation congruences [17]. Users can choose a specific proof system and go on to decide the corresponding bisimulation congruence between π -calculus agents.

In order to deal with recursions, we have generalized a version of the unique fixpoint induction in the π -calculus setting. This is inspired by Hennessy, Lin [21] and Rathke [22]. The soundness and completeness of proof systems with this new rule are described in [15].

As far as we know, our proof assistant PiM is the first interactive theorem prover for the π -calculus. However, the PiM implementation is preliminary.

For the future work, we want to integrate PiM with MWB or HAL. That means users can call MWB or HAL in PiM in order to finish their proofs automatically. Some times users may want to use some *theorems* instead of invoking a set of "basic" rules one by one. Therefore, to allow users to input their own theorems and use them later as standard axioms is another challenging task.

References

1. R. Cleaveland, J. Parrow and B. Steffen: A semantic based verification tool for finite state systems. In *Proceedings of the 9th International symposium on Protocol Specification, Testing and Verification*, North Holland (1989)
2. H. Lin: Complete proof systems for observation congruences in finite-control π -calculus. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 443–454. Springer-Verlag (1998)
3. M. Dam: Model checking mobile processes. In *Proceedings of the CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, Springer-Verlag (1993)
4. S. Mavw and G. Veltink: A proof assistant PSF. In *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, Springer-Verlag (1991) 158–168
5. M. Hennessy and H. Lin: Symbolic bisimulations. Technical Report 1/92, Computer Science, University of Sussex (1992)
6. M. Hennessy and H. Lin: Proof systems for message-passing process algebras. Technical Report 5/93, Computer Science, University of Sussex (1993)
7. H. Lin: PAM: A process algebra manipulator. In *Computer Aided Verification*, volume 575 of *Lecture Notes in Computer Science*, Springer-Verlag (1991) 136–146
8. H. Lin: A verification tool for value-passing processes. In *Proceedings of 13th International Symposium on Protocol Specification, Testing and Verification*, IFIP Transactions, North Holland (1993)
9. R. Milner: *Communication and Concurrency*. Prentice Hall (1989)
10. B. Victor and F. Moller: The Mobility Workbench – a tool for the π -calculus. In *Proceedings of CAV'94*, volume 818 of *Lecture Notes in Computer Science*, Springer-Verlag (1994) 428–440
11. S. Gnesi, U. Montanari, M. Pistore, G. Ferrari, G. Ferro and G. Risori: An automata-based verification environment for mobile processes. In *Proceedings of TACAS'97*, volume 1217 of *Lecture Notes in Computer Science*, Springer-Verlag (1997)
12. U. Montanari and M. Pistore: Checking bisimilarity for finitary π -calculus. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, Springer-Verlag (1995)
13. R. De Simone and D. Vergamini: Aboard auto. Technical Report RT111, INRIA (1989)

14. Du. Xutao and Li. Zhoujun: Proving the ab-protocol in PiM. Technical Report, National University of Defence Technology (2001)
15. Du. Xutao and Li. Zhoujun: On formulating the unique fixpoint induction for the π -calculus. Technical Report, National University of Defence Technology (2001)
16. J. Parrow, R. Milner and D. Walker: A calculus of mobile processes, part I and II. *Information and Computation* (1992) 100:1–77
17. Li. Zhoujun: The Verification Theory and Algorithms for the Value-Passing CCS and π -Calculus. PhD Thesis, National University of Defence Technology (1999)
18. H. Lin: Complete inference systems for weak bisimulation equivalences in the π -calculus. In *Proceedings of TAPSOFT'95*, volume 915 of *Lecture Notes in Computer Science*, Springer-Verlag (1995) 187–201
19. H. Lin: Unique fixpoint induction for mobile processes. In *Proceedings of CONCUR'95*, volume 962 of *Lecture Notes in Computer Science*, Springer-Verlag (1995) 88–192
20. H. Lin: On implementing unique fixpoint induction for value-passing processes. In *Proceedings of TACAS95 workshop on Tools and Algorithms for the Construction and Analysis of Systems*, Aarhus (1995)
21. M. Hennessy and H. Lin: Unique fixpoint induction for messing-passing process calculi. In *Proceedings of CATS97 Computing: Australian Theory Symposium* (1997)
22. J. Rathke: Unique fixpoint induction for value-passing processes (Extended Abstract). In *Proceedings of the 12th Symposium on Logic in Computer Science* (1997)
23. H. Lin: Complete proof systems for observation congruences in finite-control π -calculus. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, Springer-Verlag (1998) 443–454
24. Z. Li and H. Chen: Checking strong/weak bisimulation equivalences and observation congruence for the π -calculus. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, Springer-Verlag (1998) 707–718
25. Z. Li and H. Chen: Computing strong/weak bisimulation equivalences and observation congruence for value-passing processes. In *Proceedings of TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, Springer-Verlag (1998) 300–314

Data Space Fusion Based Approach for Effective Alignment of Computation and Data*

Jun Xia, Xuejun Yang, and Huadong Dai

School of Computer Science,
National University of Defense Technology,
Changsha 410073, Hunan, China
ddk@nudt.edu.cn

Abstract. The alignment of computation and data has an important affection on the performance of parallel programs running on distributed memory multicomputers. This paper presents a new approach of the effective alignment of computation and data, namely the data space fusion based approach for partitioning computation and data, which can be used to solve the computation and data decomposition problems on distributed memory multicomputers. This approach can maximize parallelism and minimize communication over processors by exploiting the parallelism of computation space as high as possible and using the technique of data space fusion to optimize data distribution. The approach can be integrated with data replication and offset alignment naturally and therefore can make the communication overhead as low as possible. The experimental results on eight programs show that the approach presented in this paper is effective for aligning computation and data.

1 Introduction

Distributed memory multicomputers and shared memory multicomputers are the two most popular parallel computers in high speed computing. Distributed memory multicomputers are superior to shared memory multicomputers in scalability and increasingly have been used to solve various scientific problems. The major disadvantage of distributed memory multicomputers is the difficulty in programming due to the lack of a single global address space. Hence, programmers and compilers are responsible for distributing code and data over processors. As the difference of overhead between local memory references and remote memory references is large, the code and data have to be carefully distributed to get good performance. Over the last decade, many researchers have paid attention to the problem of effective alignment of computation and data for a given program executed on a parallel machine [1-5]. But in their research, there exist some disadvantages, which will be discussed in related work later.

In this paper, we discuss the problem of effective alignment of computation and data and present a data space fusion based approach for partitioning computation and

* This research is supported by NNSF (National Natural Science Foundation grant No. 69825104)

data. In this approach, the vector space containing all distance vectors [6] of a loop nest is first determined, which is used to partition the iteration space of the loop nest, then the data spaces referenced by iteration partitions are determined for different references of each array and the data spaces referenced by the same iteration partition for different references of the same array are fused to a single data space, which is used to partition the data space of the array. To solve the problem of data decompositions, we present a theoretical framework of data space fusion, which can be used to organize data effectively according to the decompositions of iteration spaces and optimize data distribution. The approach can deal with the computation and data decompositions of a group of loop nests with arrays having any dimensions. It also can make the parallelisms of computation and data decompositions as high as possible and can be naturally integrated with data replication and offset alignment to make the communication overhead as low as possible. At last, we show the effectiveness of the approach with experimental results.

2 Technical Preliminaries

We view the iteration space of a loop nest of depth n as an n -dimensional polyhedron where each point is denoted by an $n \times 1$ column vector $\bar{I} = (i_1, i_2, \dots, i_n)^T$. We call \bar{I} as *iteration vector* and use L^n to denote the iteration space of the loop nest. Similarly, every m -dimensional array X declared in the program defines an m -dimensional polyhedron (namely data space), each point of which represents an array element and can be denoted by an $m \times 1$ column vector. We assume that all loop bounds and subscript expressions are affine functions of enclosing loop indices and symbolic constants. The array reference can be denoted by

$A\bar{I} + \bar{o}$, $m \times n$ matrix A is called as *reference matrix*, and $m \times 1$ column vector \bar{o} is called as *offset vector* [7].

We use $\text{span}\{\bar{b}_1, \dots, \bar{b}_l\}$ to denote the space spanned by vector $\bar{b}_1, \dots, \bar{b}_l$, $\text{span}\{\Omega\}$ to denote the space spanned by vectors in set Ω , Q to denote rational number field, Q^n to denote the space composed of all the $n \times 1$ rational number vectors, $\dim(\Psi)$ to denote the dimension of vector space Ψ and $\dim(\bar{b}_1, \dots, \bar{b}_l)$ to denote the rank of vector $\bar{b}_1, \dots, \bar{b}_l$. We define $\bar{b} + \Psi = \left\{ \bar{q} \mid \bar{q} = \bar{b} + \bar{p}, \bar{p} \in \Psi \right\}$, where

\bar{b} is $n \times 1$ column vector, and Ψ is a vector space composed of $n \times 1$ column vectors. The vector spaces considered in this paper are all vector spaces over Q .

3 Computation Space Decompositions

Given a n -nested loop L with arrays X_1, \dots, X_p , assume Ψ^L is the vector space containing all distance vectors of L . To get Ψ^L , we can do data dependence analysis on each array respectively and get the vector space containing all distance vectors related to it first. Assume vector space containing all distance vectors related to X_j as $\Psi_{X_j}^L$, then $\Psi^L = \text{span}\{\Psi_{X_1}^L \cup \dots \cup \Psi_{X_p}^L\}$. By doing precise data dependence analysis [6, 8] on the loop nest and eliminating redundant computations [5], we can make the dimension of Ψ^L as low as possible and therefore can exploit the parallelism of the loop nest as high as possible.

After Ψ^L is got, it can be used to partition L^n . We assume $\dim(\Psi^L) < n$ in default, for we only consider the circumstance that the iteration space can be partitioned into more than one independent iteration partition that can be parallel executed.

Definition 1. Given a loop nest L with depth n , assume Ψ^L is the vector space containing all distance vectors of L , $\bar{\beta}_1, \dots, \bar{\beta}_u$ are a basis of Ψ^L , and $\bar{\alpha}_1, \dots, \bar{\alpha}_v$, $\bar{\beta}_1, \dots, \bar{\beta}_u$ are a basis of Q^n . Let $B^L(k_1, \dots, k_v) = k_1 \bar{\alpha}_1 + \dots + k_v \bar{\alpha}_v + \Psi^L (v \geq 1)$, we call $\{B^L(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$ as a *linear computation decomposition* of L^n , $B^L(k_1, \dots, k_v)$ as *iteration partition*, and the *parallelism* of this linear computation decomposition $\{B^L(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$ is v (or $n - \dim(\Psi^L)$).

4 Data Space Decompositions and Fusion

4.1 Linear Data Decompositions

Definition 2. Given an m -dimensional array X , a set of $m \times 1$ column vectors $\bar{\delta}, \bar{\gamma}_1, \dots, \bar{\gamma}_v$, and a vector space Ω_X composed of $m \times 1$ column vectors. Let

$$D_X(k_1, \dots, k_v) = \bar{\delta} + k_1 \bar{\gamma}_1 + \dots + k_v \bar{\gamma}_v + \Omega_X (v \geq 1),$$

we call $\{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$ as a *linear data decomposition* of the data space of X , $D_X(k_1, \dots, k_v)$ as *data partition*. Assume $\dim(\Omega_X) = u$, $\bar{\eta}_1, \dots, \bar{\eta}_u$ are a basis of Ω_X , and $r = \dim(\bar{\gamma}_1, \dots, \bar{\gamma}_v, \bar{\eta}_1, \dots, \bar{\eta}_u)$, then we call the *parallelism* of the linear data decomposition $\{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$ is $r - u$.

Given an n -nested loop L with array X , assume $A\bar{I}+\bar{o}$ is a certain reference of X , $\left\{B^L(k_1, \dots, k_v) = k_1\bar{\alpha}_1 + \dots + k_v\bar{\alpha}_v + \Psi^L \middle| k_1, \dots, k_v \in Q\right\}$ is the linear computation decomposition of L^n and $\bar{\beta}_1, \dots, \bar{\beta}_u$ are the basis of Ψ^L . For $A\bar{I}+\bar{o}$, the data space referenced by iteration partition $B^L(k_1, \dots, k_v)$ is $AB^L(k_1, \dots, k_v) + \bar{o} = \bar{o} + k_1 A\bar{\alpha}_1 + \dots + k_v A\bar{\alpha}_v + \text{span}\left\{A\bar{\beta}_1, \dots, A\bar{\beta}_u\right\}$. If we let $D_X(k_1, \dots, k_v) = AB^L(k_1, \dots, k_v) + \bar{o}$, then according to definition 2 we can know $\{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$ is a linear data decomposition of the data space of X . From above, we can conclude that given a linear computation decomposition and a reference of an array, we can get the corresponding linear data decomposition of the data space of the array.

4.2 Standard Forms of Linear Data Decompositions

Definition 3. Given a linear data decomposition

$$\left\{D(k_1, \dots, k_v) = \bar{\delta} + k_1\bar{\gamma}_1 + \dots + k_v\bar{\gamma}_v + \Omega \middle| k_1, \dots, k_v \in Q\right\}$$

and set $\left\{D'((F\bar{K})^T) = \bar{\delta}' + HF\bar{K} + \Omega' \middle| \bar{K} \in Q^v\right\}$,

where Ω and Ω' are vector space composed of $m \times 1$ column vectors, $H = (\bar{\varphi}_1, \dots, \bar{\varphi}_l)$, F is $l \times v$ matrix, $\bar{K} = (k_1, \dots, k_v)^T$, $\bar{\delta}, \bar{\delta}', \bar{\gamma}_1, \dots, \bar{\gamma}_v, \bar{\varphi}_1, \dots, \bar{\varphi}_l$ are $m \times 1$ column vectors.

1. If $\forall 1 \leq j \leq v, \bar{\gamma}_j \in \Omega$, we call $\left\{D'((F\bar{K})^T) \middle| \bar{K} \in Q^v\right\}$ as the *standard form* of linear data decomposition $\{D(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$, where $\bar{\delta}' = \bar{\delta}$, $\Omega' = \Omega$, H is $m \times 1$ zero vector, F is $1 \times v$ zero vector.
2. If $\exists 1 \leq j \leq v, \bar{\gamma}_j \notin \Omega$, assume $\bar{\eta}_1, \dots, \bar{\eta}_u$ are the basis of Ω' , if the row rank of F is full, $\bar{\varphi}_1, \dots, \bar{\varphi}_l, \bar{\eta}_1, \dots, \bar{\eta}_u$ are linearly independent and $\forall k_1, \dots, k_v \in Q$, there has $D(k_1, \dots, k_v) = D'((F\bar{K})^T)$, then we call $\left\{D'((F\bar{K})^T) \middle| \bar{K} \in Q^v\right\}$ as the *standard form* of $\{D(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$.

From definition 3 we can know if $\forall 1 \leq j \leq v, \bar{\gamma}_j \in \Omega$, we can get the unique standard form of the linear data decomposition directly; otherwise we can use the algorithm given in Fig. 1 to transform it to its standard form. It is easy to prove the row rank of matrix F in any standard form of the linear data decomposition equals to this linear data decomposition's parallelism.

Input: linear data decomposition

$$\left\{ D(k_1, \dots, k_v) = \bar{\delta} + k_1 \bar{\gamma}_1 + \dots + k_v \bar{\gamma}_v + \Omega \middle| k_1, \dots, k_v \in Q \right\}, \text{ where } \exists 1 \leq j \leq v, \bar{\gamma}_j \notin \Omega.$$

Output: its standard form $\left\{ D'((F \bar{K})^T) = \bar{\delta}' + HF \bar{K} + \Omega' \middle| \bar{K} \in Q^v \right\}$

1. Assume $\bar{\eta}_1, \dots, \bar{\eta}_u$ are a basis of Ω , $r = \dim(\bar{\gamma}_1, \dots, \bar{\gamma}_v, \bar{\eta}_1, \dots, \bar{\eta}_u)$. Find a set of linearly independent column vectors with the least number that contains $\bar{\eta}_1, \dots, \bar{\eta}_u$ and can express $\bar{\gamma}_1, \dots, \bar{\gamma}_v$, obviously the number of this linearly independent set is r , assume it is $\bar{\varphi}_1, \dots, \bar{\varphi}_{r-u}, \bar{\eta}_1, \dots, \bar{\eta}_u$;

2. Assume $\bar{\gamma}_j = a_{j1} \bar{\varphi}_1 + \dots + a_{j(r-u)} \bar{\varphi}_{r-u} + c_{j1} \bar{\eta}_1 + \dots + c_{ju} \bar{\eta}_u, 1 \leq j \leq v$. Put them into $D(k_1, \dots, k_v)$ and we can get:

$$D(k_1, \dots, k_v) = \bar{\delta} + (a_{11}k_1 + \dots + a_{v1}k_v)\bar{\varphi}_1 + \dots + (a_{1(r-u)}k_1 + \dots + a_{v(r-u)}k_v)\bar{\varphi}_{r-u} + \text{span}\left\{\bar{\eta}_1, \dots, \bar{\eta}_u, \bar{0}\right\}$$

3. $\bar{\delta}' = \bar{\delta}$, $F = \begin{pmatrix} a_{11} & \cdots & a_{v1} \\ \vdots & \ddots & \vdots \\ a_{1(r-u)} & \cdots & a_{v(r-u)} \end{pmatrix}$, $\bar{K} = (k_1, \dots, k_v)^T$, $H = (\bar{\varphi}_1, \dots, \bar{\varphi}_{r-u})$, $\Omega' = \Omega$;

RETURN $\left\{ D'((F \bar{K})^T) = \bar{\delta}' + HF \bar{K} + \Omega' \middle| \bar{K} \in Q^v \right\}$

Fig. 1. Algorithm of finding the standard form of linear data decomposition

4.3 Data Replication

Given a loop nest L with array X , for a certain reference of X we can get a linear data decomposition of the data space of X with a linear computation decomposition of the iteration space of L . Assume the parallelism of this linear computation decomposition is v , the standard form of this linear data decomposition is $\left\{ D'_X((F \bar{K})^T) \middle| \bar{K} \in Q^v \right\}$, and F 's row rank is f . If $f < v$, then equation $F \bar{d} = \bar{0}$

has non-zero solutions and the dimension of the solution space is $v - f$, so there exists the circumstance that several iteration partitions reference the same data partition, therefore the data partitions have to be replicated to make the reference local. Assume the dimension of processor space is v . As the parallelism of the linear data decomposition is f , we need to replicate data partitions over $v - f$ dimensions which is the same as the dimension of the solution space of $F \bar{d} = \bar{0}$.

Definition 4. Given a linear computation decomposition of the iteration space of a loop nest and a linear data decomposition of an array referenced in the loop nest that is got by the linear computation decomposition, assume the parallelism of the linear computation decomposition is v and the parallelism of the linear data decomposition is f , then we call the *replication degree* of the linear data decomposition relative to the linear computation decomposition is $v - f$.

We always hope the replication degree is as low as possible, because it will make the amount of data needed to be replicated as small as possible. Moreover if written arrays are replicated, generally compilers will responsible for the consistency of the replicated data [6], therefore reducing replication degree will reduce the runtime overhead of maintaining the consistency of the replicated data too.

4.4 Data Space Fusion

Given a n -nested loop L with array X , assume the linear computation decomposition of L^n is $\left\{ B^L(k_1, \dots, k_v) = k_1 \bar{\alpha}_1 + \dots + k_v \bar{\alpha}_v + \Psi^L \middle| k_1, \dots, k_v \in Q \right\}$,

and X has q different references $A_1 \bar{I} + \bar{o}_1, \dots, A_q \bar{I} + \bar{o}_q$. For $A_j \bar{I} + \bar{o}_j$ ($1 \leq j \leq q$), let $\left\{ D_{X^j}(k_1, \dots, k_v) = \bar{\delta}_j + k_1 \bar{\gamma}_{j1} + \dots + k_v \bar{\gamma}_{jv} + \text{span} \left\{ \bar{\eta}_{j1}, \dots, \bar{\eta}_{ju}, \bar{0} \right\} \middle| k_1, \dots, k_v \in Q \right\}$

be the corresponding linear data decomposition of X got by the linear computation decomposition of L^n . If $q > 1$, for each different reference of X we can get its corresponding linear data decomposition of X , but the linear data decomposition of X should be unique (for we only consider static data decompositions in the paper and don't consider data redistribution), therefore we have to fuse all those linear data decompositions into a unique one. Assume the fused unique linear data decomposition is $\{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$, then it should satisfy the condition:

$\forall k_1, \dots, k_v \in Q, 1 \leq j \leq q, D_{X^j}(k_1, \dots, k_v) \subseteq D_X(k_1, \dots, k_v)$, which means the data spaces referenced by the same iteration partition should be fused into a single data space. It is possible that many linear data decompositions satisfy the condition, and we want to acquire the one with the highest parallelism among them, which will make the replication degree the lowest. We give the algorithm of acquiring the fused unique linear data decomposition that satisfies the condition we impose on it and has the highest parallelism in Fig. 2.

Input: array X 's q linear data decompositions:

$$\{D_{X_j}(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}, 1 \leq j \leq q$$

Output: array X 's fused unique linear data decomposition:

$$\{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$$

Initialization:

$$\theta = \{\bar{0}\};$$

DO $j = 1, q$

DO $f = 1, u$

$$\theta = \theta \cup \{\bar{\eta}_{jf}\}$$

ENDDO

ENDDO

DO $j = 2, q$

DO $p = 1, v$

$$\theta = \theta \cup \{\bar{\gamma}_{jp} - \bar{\gamma}_{1p}\}$$

ENDDO

ENDDO

DO $j = 2, q$

$$\theta = \theta \cup \{\bar{\delta}_j - \bar{\delta}_1\}$$

ENDDO

$$\Omega_X = \text{span}\{\theta\};$$

$$D_X(k_1, \dots, k_v) = \bar{\delta}_1 + k_1 \bar{\gamma}_{11} + \dots + k_v \bar{\gamma}_{1v} + \Omega_X;$$

$$\text{Return } \{D_X(k_1, \dots, k_v) | k_1, \dots, k_v \in Q\}$$

Fig. 2. Algorithm of acquiring the fused unique linear data decomposition

5 Mapping of Virtual Processor Space

Given the linear computation decomposition of L^n and the standard forms of the fused unique linear data decompositions of arrays X_1, \dots, X_l accessed in L . Assume virtual processor space is Q^v , where v is the dimension of virtual processor space. In the following we define two mappings: $IP : \Delta \rightarrow Q^v$ is the mapping from the iteration partitions to coordinates of virtual processor space, where Δ is the set composed of iteration partitions. $DP_{X_j} : \Theta \rightarrow Q^v$ is the mapping from the data partitions of X_j to coordinates of virtual processor space, where Θ is the set composed of data partitions. We can determine mappings $IP, DP_{X_1}, \dots, DP_{X_l}$ and the expressions to realize them to make all references of X_1, \dots, X_l local, and then loop and data transformations can be used to express these mappings in the program languages like HPF and MPI etc., and after that we can further optimize the local references' spatial locality [9], which may further improve performance. Due to the limitation of the space, the steps of finding mappings and mapping expressions, code transformations and optimizing local references' spatial locality are omitted here.

The loop nest's parallelism and the array's replication degree associate with each other. We can reduce replication degree or make it zero by reducing the loop nest's parallelism. For example, if a written array is replicated and the overhead of maintaining its data consistency is high, then we can reduce the loop nest's parallelism to make this array's replication degree lower or zero.

In the above, we discuss the approach of aligning computation and data for a single loop nest. Base on it, we can extend this approach to deal with a group of loop nests with their corresponding arrays. As the limitation of the space, the approach of global computation and data decompositions is omitted here.

6 Experimental Results

We will present performance results for the following eight programs: *matmult* is a matrix-multiplication routine; *stencil* is a five-point stencil computing code. *syr2k* is a banded matrix update routine from BLAS; *adi* and *hydrodynamics* are two routines from Livermore kernel. *btrix* and *mxm* are two test programs from Spec92/NASA benchmark suite; *tomcatv* is the mesh generation program from Spec95. We conduct the experiments with the xHPF versions of these programs. For each program, we experiment with five different versions: the version with parallelization analysis and column distribution for all arrays (denoted by *col*); the version with parallelization analysis and row distribution for all arrays (denoted by *row*); the version with parallelization analysis and complete replication for all arrays (pgHPF compiler's default data distribution, denoted by *acopy*); the version optimized by the optimizing policies of native xHPF compiler (optimizing option *-auto=1*, denoted by *xhpf*); the version optimized by the approach studied in this paper (denoted by *opt*).

We report speedups for up to 64 processors on some distributed memory machine. This machine has 32 nodes and each node has two CPUs. Shared memory architecture is adopted inside each node while distributed memory architecture is adopted among nodes. Figure 3 gives speedups of the test programs in different versions.

From the experimental results and the analysis of the original codes, we find: different arrays may have different reference modes. If we adopt the same data distribution for all arrays, it may make references of some arrays local while references of the others need communication and therefore can't get good performance. For the complete replication for all arrays, as they will be replicated to each node, the high runtime overhead of maintaining the written arrays' consistency will make performance bad. The optimizing polices of xHPF compiler can only do simple parallelization and data distribution analyses on loop nests and arrays, and therefore its optimizing ability is limited. Our approach can deal with the complex parallelization and data distribution analyses based on the global information and make the replication degrees of arrays as low as possible, therefore compared with the approaches described above, our approach can get better effect, which is showed by the experimental results: for the eight test programs our approach gets the best effect, and with the increasing of processor number, the gap of optimizing effect between our approach and the other approaches become larger for most of the test programs.

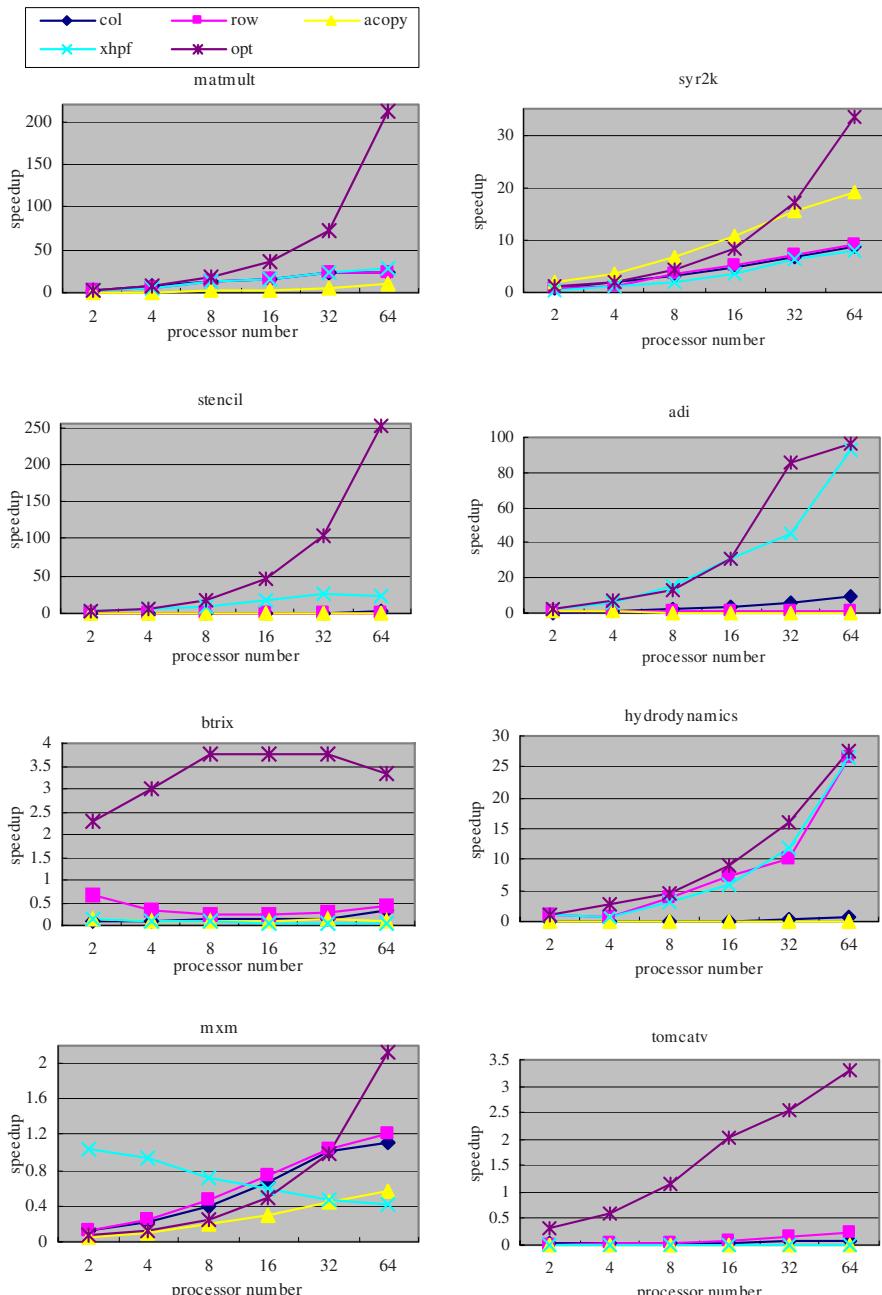


Fig. 3. Speedups of the test programs in different versions [matmult-1024×1024 matrices; syr2k-1024×1024 matrices; stencil-1024×1024 arrays; adi-1024×1024×3 arrays; btrix-the size parameters are set to 200; hydrodynamics-2048×2048 arrays; mxm-1024×1024 matrices; tomcatv-513×513 arrays]

7 Related Work

Chen and Chang [1] present a skewed alignment instead of traditional dimension-order alignment techniques to align arrays. Compared with dimension-order alignment, their approach can deal with more complex programs with minimized data communication than that of the dimension-order scheme. But their approach can be applied only after computation decompositions are determined and they don't consider data replication and offset alignment, therefore their approach can't make communication overhead the lowest. Compared with their approach, we consider computation decompositions and data decompositions simultaneously. Moreover, our approach can integrate with data replication and offset alignment naturally, and therefore can make communication overhead as low as possible.

Chang and Chu [2] present two new alignment functions for loop iteration space and arrays with linear subscripts in three loop index variables or quadratic subscripts, but they don't consider data replication and their approach can only be applied to align arrays with linear subscripts in three loop index variables or quadratic subscripts. Compared with their approach, although our approach can't deal with arrays with quadratic subscripts, it can solve the alignment problem of arrays with general linear subscripts and considers data replication.

Chen and Sheu [5] partition iteration space and organize data elements by data dependence distance vectors. Although they give the approach of communication-free computation and data decompositions with and without replication, their approach can only be applied to a single loop nest and requires the reference matrices of all references of the same array must be equal. Moreover, they don't give the systemic approach of distributing data. Compared with their approach, we consider computation and data decompositions for a group of loop nests with arrays having any dimensions and the reference matrices of the same array's references can be different. Moreover, we give the systemic approach of organizing and distributing data.

Shih and Sheu [3] present statement-level communication free hyperplane partitioning techniques. They assume each statement has an independent iteration space and partition them respectively. They also give the necessary and sufficient conditions for the feasibility of communication-free hyperplane partitions. As their approach is based on hyperplane, the parallelisms of computation and data decompositions found by it are only 1 (namely only one loop of each loop nest can be parallelized and one dimension of each array can be distributed in the final transformed code). Moreover, although viewing each statement as an independent scheduled unit can extend the application of their approach, it increases the difficulty of code transformations too.

Anderson [4] presents a linear algebra based computation and data decomposition approach. He first formulates a set of equations that computation and data distribution matrices should satisfy, and then find computation and data distribution matrices by solving their null spaces' bases. However, when solving the equations, he assumes iteration spaces are unlimited and therefore can't exploit the full parallelism of loop nests. Moreover, his replication approach can't be used to replicate data along dimensions, which restricts its application. Compared with his approach, we can exploit the full parallelism of loop nests by finding the vector spaces containing all distance vectors of loop nests precisely and reducing their dimensions. Furthermore, our replication approach can replicate data along dimensions.

8 Conclusions

The alignment of computation and data is a key factor of affecting the performance of parallel programs executing on distributed memory multiprocessors, and is also an important problem in parallel compiling optimization. To solving this problem, a data space fusion based approach for partitioning computation spaces and data spaces is presented in this paper. The approach can deal with the computation and data decompositions of a group of loop nests with arrays having any dimensions. It also can make the parallelisms of computation and data decompositions as high as possible and can be naturally integrated with data replication and offset alignment to make the communication overhead as low as possible. The experimental results show the effectiveness of our approach.

References

1. T.-S.Chen and C.-Y.Chang. Skewed data partition and alignment techniques for compiling programs on distributed memory multicomputers. *The Journal of Supercomputing*, 21(2): 191–211, 2002.
2. W.-L.Chang, C.-P.Chiu and J.-H.Wu. Communication-free alignment for array references with linear subscripts in three loop index variables or quadratic subscripts. *The Journal of Supuercomputer*, 20(1): 67–83, 2001.
3. K.-P.Shih, J.-P.Sheu and C.-H.Huang. Statement-level communication-free partitioning technique for parallelizing compilers. *The Journal of Supercomputing*, 15(3): 243–269, 2000.
4. J.M.Anderson. Automatic computation and data decomposition for multiprocessors. Ph.D. Thesis, Stanford University, Palo Alto, Cal, 1997.
5. T.-S.Chen and J.-P.Sheu. Communication-free data allocation techniques for parallelizing compilers on multicomputers. *IEEE Transaction on Parallel and Distributed Systems*, 5(9): 924–938, 1994.
6. M.Wolf. *High Performance Compilers for Parallel Computing*. Redwood City: Addison-Wesley Publishing Company, 1996.
7. M.Wolf and M.Lam. A data locality optimizing algorithm. In *Proceedings of the SIGPLAN '91 Conference on Programming Language Design and Implementation*, Toronto, Canada, pp. 30–44, 1991.
8. SHEN Zhi-Yu, HU Zhi-Ang, LIAO Xiang-Ke, WU Hai-Ping, ZHAO Ke-Jia and LU Yu-Tong. *Methods of Parallel Compilation*. National Defense Industry Publishing Company, China, 2000.
9. XIA Jun, YANG Xue-Jun, ZENG Li-Fang and ZHOU Hai-Fang. A projection-delamination based approach for optimizing spatial locality in loop nests. *Chinese Journal of Computers*, 26(5):539–551, 2003.

Optimization Parameter Selection by Means of Limited Execution and Genetic Algorithms

Yonggang Che, Zhenghua Wang, and Xiaomei Li

School of Computer
National University of Defense Technology,
Changsha, PRC, 410073
ygche2003@yahoo.com
zhhwang@cndt.edu.cn
lxmcjh@sohu.com

Abstract. In this article, we present an optimization parameter selection framework based on limited execution and genetic algorithm. In this framework, the parameter selection problem is transformed into a combinatorial minimization problem. We first perform reduction transformation to reduce the program's runtime while maintaining its relative performance as regard to different parameter vectors. Then we search for the near-optimal optimization parameter vector based on the reduced program's real execution time. The search engine is guided by genetic algorithm, which converges to near-optimal solution quickly. The reduction transformation reduces the time to evaluate the quality of each parameter vector. And the genetic algorithm reduces the number of candidate parameter vectors evaluated. This makes execution-driven optimization parameter search practical. Our experiments for 5 scientific applications on 3 different platforms suggest that our approach can find excellent optimization parameters in reasonable time. It obtains highly architecture specific optimizations in an architecture independent manner and can solve nearly all combined optimization parameter selection problems.

1 Introduction

In recent years, high performance architectures have become widely available. However, these architectures also become increasingly hard to analyze and predict, which makes it difficult to obtain peak performance. On the one hand, complex program optimizations are required to exploit the hardware resources. On the other hand, the changes in architectures force programmers to repeat program optimization task for architecture anew. This is especially true for scientific applications.

Parallel scientific application's performance is mainly determined by tow components: inter-processor parallelism and single-processor performance. So the high performance computing community pay a lot attentions to program performance optimization for single-processor. On a single processor, an application's performance is mainly determined by how it exploits the machine's memory hierarchies and pipelines. Enormous performance optimization methods are introduced, such as loop permutation, loop tiling, loop fusion, loop distribution, loop unrolling, data layout transformations, and array padding. Many of these optimizations depend on right optimization parameters to get

good performance. For example, the program's performance is highly sensitive to the tile sizes (i.e., the width and height of each tile) for loop tiling. Another example is loop unrolling, in which the unroll factors (i.e., the number the loop body is replicated) must be judiciously selected so as to expose enough ILP (Instruction Level Parallelism) while suppress negative effects, such as extra register spills when the working set of the unrolled loop body exceeds the number of available registers, or the increased instruction-cache misses when code size of the unrolled loop exceeds instruction cache [1]. Padding sizes are also critical parameters in array padding optimization. Determining these parameters is often the key task for these optimizations.

Many optimization parameter selection algorithms exist in literature, such as [2], [3], [4], [5] for tile shape selection and [6] for unroll factor selection. Most of them are based on static analysis of the program codes and simplified machine models. These algorithms are useful when the program behaviors on the target platform can be precisely modeled. But scientific codes are becoming increasingly complicated [7] due to complex program structure (e.g., pointer-based data structures), complex program behaviors (e.g., irregular computations) and their complicated interactions with the machines. And the computer architectures are increasingly complex (long pipelines, cache hierarchies, branch prediction, trace cache, hardware prefetching, etc) [8]. It is rarely possible for these analytical methods to obtain the optimal (or excellent) optimization parameters. This is especially true when several optimizations are applied in combination. Significant gaps appear to exist between theory and practice in the field. T. Kisuki et al [9] performed experiments to some benchmarks on several platforms. They found that tile sizes selected based on tow analytical methods are far from optimal and are outperformed by simple search methods remarkably.

In this article, the task of optimization parameters selection is transformed into a combinatorial minimization problem. We search for the near optimal optimization parameters in a manner that is adaptable for different architectures. First, some transformations are applied to the application, leaving the optimization parameter vector be read from configure file. Second, we compile the application into executable with the native compiler. Then we repeatedly generate configure file with different parameter vector selected by a search engine, and measure the executable's runtime. The search engine is based on genetic algorithm to find the parameter vector that has the shortest execution time. We call this approach *Execution-Driven Parametric Search based on Genetic Algorithm (EDPS-GA)*. Because real execution of the target application is time-consuming, we propose to transform the optimized application code so as to reduce its runtime while maintaining its relative performance as regard to different optimization parameter vectors. Experiments show that our approach can find excellent optimization parameters in reasonable time.

The rest of this article is organized as follows: Section 2 presents the formulation of parameter selection problem and related concepts. Section 3 presents the ideal and implementation of EDPS-GA in detail. Experimental results are presented and discussed in Section 4. In Section 5, we present a discussion of related work in the areas of optimization parameter selection. Section 6 presents our conclusions along with a discussion of future work.

2 Problem Definition

2.1 Formulation of the Parameter Selection Problem

Assume P_0 is the original program to be optimized. Suppose the user selects m performance critical units $u_i (1 \leq i \leq m)$ from P_0 and apply some transformations to them. Here each unit can be a basic block, a loop nest or a procedure. After the transformations, P_0 becomes P_1 . Assume P_1 depends on n parameters $v_1, v_2, \dots, v_n (v_j \in N, 1 \leq j \leq n)$. We call each parameter an *optimization parameter*. Each optimization parameter v_j is limited by a lower bound low_j and an upper bound up_j , that is $low_j \leq v_j \leq up_j$. $V = (v_1, v_2, \dots, v_n)$ is called a *parameter vector*.

The aim of performance optimization is to select right optimization vector V so as to minimize the execution time of P_1 . Let $T(P, V)$ denote the execution time of program P when it takes optimization parameter vector V . Then the problem can be transformed into a constrained optimization problem that finds a vector $V = (v_1, v_2, \dots, v_n)$ minimizing an object function $T(P_1, V)$:

$$\left. \begin{array}{l} \text{Minimize} \left(T(P_1, V = (v_1, v_2, \dots, v_n)) \right) \\ \text{Subject to} \\ low_j \leq v_j \leq up_j, j = 1, 2, \dots, n, v_j \in N \end{array} \right\} \quad (1)$$

In problem (1), a parameter vector is also called a *solution*. The space enclosed by $low_j \leq v_j \leq up_j, j = 1, 2, \dots, n, v_j \in N$ is called the *solution space*. Denote it as Ω .

2.2 Reduce the Execution Cost

We use real execution time to evaluate the quality of each solution, which enjoys the superiority than other metrics such as predicted cache misses used in [4] for it is precise in nature. But real execution is time-consuming. Obviously, the user can provide tighter bounds of the parameters based on their own knowledge, so as to reduce the number of solutions evaluated (hence, the time spent on executing them). But this is not enough.

We notice that the user only cares about the *relative quality* of each solution, not *absolute execution time* that is related to it. If we apply certain transformation to the optimized program P_1 , which shorten the execution time of each solution while maintaining the solution's relative quality to other solutions, then the reduced program can represent program P_1 when evaluating the quality of each solution. Let R be a transformation performed to the optimized program P_1 . Let P' be the transformed program. That is, $P' = R(P_1)$. We call R a *reduction transformation*. R is qualified for our purpose, if $\forall V^1, V^2 \in \Omega$, the following conditions are satisfied:

$$\begin{aligned} T(P', V^1) < T(P', V^2) &\Leftrightarrow T(P_1, V^1) < T(P_1, V^2) \\ T(P', V^1) \geq T(P', V^2) &\Leftrightarrow T(P_1, V^1) \geq T(P_1, V^2) \end{aligned} \quad (2)$$

When R satisfies condition (2), we call it a *legal reduction transformation*. Note here R is neither a performance optimization transformation, nor a semantically legal program transformation. If R is so performed that it changes little of the program's memory and computation behaviors, it can closely satisfy condition (2). We also notice

that for optimization problem (1), near-optimal solution is also satisfactory. So R doesn't need to strictly satisfy condition (2). Fortunately, we find that for a large fraction of real scientific applications, such reduction transformations do exist.

Iterative schemes are widely used in PDE (Partial Differential Equation) solvers. The kernel module of such solvers is a time-step loop, whose loop body performs almost the same operations in different iterations. Note that in this article, we do not consider those applications with changing computation patterns for different time-step, such as adaptive mesh refinement programs. Scaling down the number of the iterations will not influence the memory access pattern and computation order of the enclosed computations. Mgrid from spec95/2000, swim from spec95/2000, wave5 from spec95 are all benchmark codes that hold such property. The number of time-step iteration in real application is usually a large number, leaving much room for runtime reduction.

There are other cases when the number of computations of a program is determined by certain program parameters. If these parameters can be altered while not change the computation and memory characteristics much, they can also be scaled down to reduce the program's runtime. A typical example is loop tiling optimization for matrix multiplication, where the computation pattern and memory reference pattern are mainly determined by the inner loops (element loops). The bounds of the outer loops (tile loops) can be scaled down to reduce the size of iteration space (hence, the execution time). We have proved the legality of these reduction transformations in another article.

3 Optimization Parameter Selection with EDPS-GA

3.1 Genetic Algorithm

The solution space Ω consists of $\prod_{1 \leq j \leq n} (up_j - low_j + 1)$ possible solutions, which is a huge number in many cases. Clearly exhaustive search is too expensive. If different optimization parameters of problem (1) can be independently optimized, one may devise heuristic optimization algorithms for different parameters separately. However, as many factors influence a program's performance, different optimization parameters are nonlinearly interdependent. Problem (1) is a NLP (Nonlinear Integer Optimization) one [4]. One of the challenges in NLP is that some problems exhibit local minima and search algorithms can be stuck at them. We also noticed that program optimization is a task where near optimal solutions would be satisfactory. This makes genetic algorithm well suited for this problem [10].

Genetic algorithm is a search algorithm that identifies high quality solution for large complex optimization problems. It is based on the Darwinian principle of reproduction and survival of the fittest and analogs of naturally occurring genetic operations such as *crossover (recombination)* and mutation. In the genetic algorithm, each *individual* in the population represents a candidate solution to the given problem. The genetic algorithm transforms a *population (set)* of individuals, each with an associated *fitness value*, into a new generation of the population using reproduction, crossover, and mutation. Genetic algorithms have shown good results when applied and tuned to many problems [11], [12].

3.2 Our Genetic Algorithm Formulation

The individual. In our method, an individual S in the genetic algorithm is a vector of integer values (s_1, s_2, \dots, s_n) , with each integer s_j corresponds to an optimization parameter v_j of solution V . In the following of the article we do not distinguish between V and S , v_j and s_j .

The encoding. We encode each individual S in a sequence of integers. In our implementation, no illegal solution is allowed. Each genetic operator will restrict all parameters of an individual in legal range.

The population. The population of the i -th generation is denoted as Pop_i . Each population has a fixed size. We denote it as N_{pop} .

The fitness function. The fitness value reflects the quality of the individual S in relation to other individuals. We use linear rank-based fitness assignment scheme [13] to calculate each individual's fitness value. First, the individuals are sorted in descending order based on their corresponding execution time. Let $Pos(S)$ denote the order of S in the population Pop , $SP (SP \in [1.0, 2.0])$ be the selection pressure. Then the fitness function is defined as:

$$Fit(S) = 2 - SP + \frac{2(SP - 1)(Pos(S) - 1)}{N_{pop} - 1} \quad (3)$$

The selection operator. First we select a small fraction of elitists (i.e., the individuals with the shortest execution time) of population Pop_{i-1} and directly put them into population Pop_i . Then we select other individuals based on a roulette wheel selection scheme.

The recombination operator. We use an integer number recombination scheme. Let $S^1 = (s_1^1, s_2^1, \dots, s_n^1)$ and $S^2 = (s_1^2, s_2^2, \dots, s_n^2)$ be the tow parent individuals, $S^a = (s_1^a, s_2^a, \dots, s_n^a)$ and $S^b = (s_1^b, s_2^b, \dots, s_n^b)$ be the tow child individuals, then $\forall j \in [1, n]$:

$$\begin{aligned} s_j^a &= Mod(s_j^1 + RandR(-0.25, 1.25) * (s_j^1 - s_j^2), up_j) \\ &if(s_j^a < low_j) s_j^a = RandI(low_j, up_j) \\ s_j^b &= Mod(s_j^2 + RandR(-0.25, 1.25) * (s_j^2 - s_j^1), up_j) \\ &if(s_j^b < low_j) s_j^b = RandI(low_j, up_j) \end{aligned} \quad (4)$$

Where $RandR(low_j, up_j)$ is the function that randomly selects one real number in the range from low_j to up_j , $RandI(low_j, up_j)$ is the function that randomly selects one integer number in the range from low_j to up_j .

The mutation operator. The mutation operator is based on integer value mutation. For $\forall j \in [1, n]$ and $k = a, b$:

$$\begin{aligned} s_j^k &= Mod(s_j^k + RandI(-muStep_j, muStep_j), up_j) \\ &if(s_j^k < low_j) S_j^k = RandI(low_j, up_j) \end{aligned} \quad (5)$$

Where $muStep_j (1 \leq j \leq n)$ are positive integers set by the user.

3.3 Algorithm EDPS-GA

The algorithm of EDPS-GA is shown in figure 1. EvLst is a list structure that records the individuals that have been evaluated (i.e., whose corresponding execution time have been measured) along with their corresponding execution time. NumElitist is the number of enlists that will be put directly into next generation. MaxGen is the maximal number of generations.

Algorithm: EDPS-GA

1. Read input parameters and initialize data structures
2. Randomly generate population Pop_0
3. For each individual S in Pop_0 , measure $T(P', S)$
5. Put all unique individuals of Pop_0 into EvLst
6. For i =1 to MaxGen do
 - (1) Reproduce NumElitist individuals with the highest fitness value from Pop_{i-1} into Pop_i
 - (2) For j = (NumElitist+1) to N_{pop} step 2 do
 - a. Select S_1 and S_2 from Pop_{i-1} using above selection operator
 - b. If recombination, apply recombination operator (5) to S_1 and S_2 , which generates tow child individuals S_a and S_b ;
 - Else $S_a = S_1$, $S_b = S_2$
 - c. For k=a,b do
 - If mutation, apply mutation operator (6) to S_k
 - If $S_k \in$ EvLst, retrieve $T(P', S_k)$ from EvLst;
 - Else, measure $T(P', S_k)$, and put S_k into EvLst
 - d. Put S_a and S_b into Pop_i
 - (3) For each individual S in Pop_i , calculate $Pos(S)$ and $Fit(S)$
7. Select the individual with the highest fitness value in Pop_{MaxGen} as the solution

Fig. 1. Algorithm EDPS-GA

As can be seen from the above algorithm, each parameter vector's corresponding execution time will be measured for only once, no matter how many times it appears in the search process.

4 Experimental Results

4.1 Experiment Environments

We report the results of experiments conducted on 3 different platforms, the basic configurations of which are shown in Table 1.

4.2 Applications

We use 5 typical scientific programs in our test. The data type is double precision floating-point. We first apply some transformations to these programs and leave optimization parameters to be determined. Table 2 describes these programs. In the fourth column of

Table 1. Hardware and Software Setup of the Three Platforms

Platform	C3	P3	P4
CPU	VIA C3 733MHz	Intel PIII 550MHz	Intel P4 1.6GHz
Cache(KB)	L1: 64/64, L2: 64	L1: 16/16, L2: 256	L1: 12/8, L2: 512
Main Mem	PC133 256MB	PC100 256MB	PC133 256MB
OS	Windows 2000 Pro	Windows 2000 Pro	Windows XP Pro
Compiler	Compaq Visual Fortran 6.5, -O4 (Full Optimizations)		

Table 2. Programs in Our Experiment Setting

Program Description		Problem size	Num. Parameters
bcmul	Complex matrix multiplication	1024*1024	2(tile)
rbsor	3D Red Black SOR	256*256*256	2(tile)
zgemm	Level 3 BLAS routine	1024*1024	1(tile) + 1(unroll)
MxM	Matrix multiplication	1024*1024	2(tile) + 1(unroll)
mgrid	multigrid solver from Spec2000 Ref input set	2(tile) + 2(tile)	

Table 3. Execution Time in Seconds and the Corresponding Speedup

	C3			P3			P4		
	T_{ori}	T_{opt}	SP	T_{ori}	T_{opt}	SP	T_{ori}	T_{opt}	SP
bcmul	838.95	235.40	3.56	180.295	72.195	2.50	238.79	40.03	5.97
rbsor	9.854	4.776	2.06	3.844	2.236	1.72	1.151	0.891	1.29
zgemm	366.05	175.90	2.08	95.83	29.44	3.26	119.65	17.77	6.73
MxM	85.312	43.792	1.95	25.516	9.556	2.67	6.078	3.725	1.63
mgrid	4876.5	3898.9	1.25	2253.7	1768.7	1.27	526.6		

table 2, “2(tile)” means there are 2 optimization parameters and the parameters are tile sizes for loop tiling.

The execution time of these programs is reduced either by altering the number of time-step iterations, or by altering the number of iterations that the outer most loop(s) executed in the optimized programs. In EDPS-GA, the time-reduced programs are used when searching for the near-optimal optimization parameters.

4.3 Selection Qualities

Table 3 shows the selection quality in terms of execution time. T_{ori} is the execution time of the original program P_0 . T_{opt} is the execution time of the optimized program P_1 with the optimization parameter vector selected by EDPS-GA. SP is the speedup ($T_{ori} : T_{opt}$). For mgrid runs on P4, tiling of the tow loop nests fails to optimize its performance (even when exhaustive parameter search is used), so its optimized time and corresponding cost of optimization is not given in table 3, table 4 and table 5.

From table 3 we see that our EDPS-GA can adapt to different execution environments automatically. For each platform, it always selects excellent optimization parameters for

Table 4. Cost of Optimization in Terms of Solution Time

	C3		P3		P4	
	T_{op}	T_{all}	T_{op}	T_{all}	T_{op}	T_{all}
bcmul	804	2344	267	331	134	269
rbsor	788	1667	121	1439	40	361
zgemm	1525	1687	133	258	518	569
MxM	2554	2904	431	1177	474	903
mgrid	2704	5414	2065	2323		

Table 5. Cost of Optimization in Terms of the Number of Individuals Evaluated

	C3		P3		P4	
	N_{op}	N_{all}	N_{op}	N_{all}	N_{op}	N_{all}
bcmul	248	748	490	603	295	597
rbsor	337	714	72	859	73	856
zgemm	240	265	98	183	384	433
MxM	1785	2036	828	2349	588	1159
mgrid	420	853	734	826		

bcmul, rbsor, zgemm and MxM. The speedups for mgrid are not so notable only because loop tiling improves the performance of the stencil computations in mgrid not much.

4.4 Cost of Optimization

Table 4 shows the EDPS-GA cost in terms of solution time in seconds, including the time spent on execute the reduced programs. T_{op} is the time spent when the near-optimal solution is found. T_{all} is the time spent when the search stops.

Table 5 shows the EDPS-GA cost in terms of the number of individuals evaluated. N_{op} is the number of individuals being evaluated when the near-optimal solution is found. N_{all} is the number of individuals being evaluated when the search stops.

From table 4 and table 5 we can see that the number of individuals evaluated is far smaller than the size of solution space for each program on each platform. The EDPS-GA is an efficient search algorithm for the problems we have considered. We also see that the optimization times is small and is acceptable by the user.

5 Related Work

The Automatically Tuned Linear Algebra Software (ATLAS) [14] project optimize BLAS and some LAPACK routines by first probes to determine the characteristics of the target platform, then determine the optimal unrolling factor base on the probed information. It also experimentally searches possible unroll factors and tiling factors for the optimal. They use exhaustive search method, which is time-consuming.

Jaume Abella et al [4] use a genetic algorithm based technique to search the solution space for near-optimal tile size that with minimal capacity misses. They evaluate the

quality of each candidate tile size based on the number of capacity misses predicted by an analytical model: CME (Cache Miss Equations). As their method doesn't actually execute the target application, the solution time is comparatively shorter. But it also suffers from the limitations of CME. For example, CMEs can be applied only to loop nest with affine array reference pattern. They are often not general and accurate enough for realistic problems.

Our approach differs from the above methods in several ways. First, it is more general. It requires neither machine specific knowledge, nor program specific knowledge. So it can be used for any optimization parameter selection problem on any platform. Second, it is more accurate as it evaluates the quality of each parameter vector based on its corresponding execution time. Third, the reduction transformation (which can be used for a large fraction of scientific applications) reduce the solution time greatly with little loss in selection quality. This makes execution-driven parameter search practical. Fourth, our method requires no repeated compilation. Our method has produced excellent solution for the applications we have considered.

6 Conclusion and Future Work

In this article, we present a performance optimization framework EDPS-GA that uses genetic algorithms to select near-optimal optimization parameters for scientific applications. The optimization framework is platform neutral because by real execution, it accurately accounts for all performance components of the target platform. Based on the observation of the characteristics that many scientific codes possess, the solution time can be greatly reduced by reduction transformation. Our experiment results for 5 scientific codes on 3 different platforms show that EDPS-GA is able to identify high quality optimization parameters in acceptable time.

Our method is very attractive in performance critical situations such as embedded systems and vendor supplied library codes. It is also useful for application whose array size is fixed but consumes different data. In contexts where the underlying architecture changes, it is also suitable. Energy consumption can also be optimized within the same framework, if the energy consumption can be by some means measured and integrated into the fitness function of EDPS-GA.

Currently, EDPS-GA uses fixed control parameters (e.g., selection pressure, recombination probability, mutation probability). The effectiveness of GA depends on the problem it tries to solve and the control parameters it used. So we think more work should be done on improving EDPS-GA to make its control parameters dynamically adaptable during the solution process. We are also revising EDPS-GA to allow user to put more "domain specific" knowledge into the optimization to guide its search process.

Acknowledgements. This work is supported by the National Natural Science Foundation of China, under grant number 69933030.

References

1. K. Heydemann et al.: Global Trade-off between Code Size and Performance for Loop Unrolling on VLIW Architectures (2001)
2. Marta Jiménez, et al.: Register tiling in nonrectangular iteration spaces, ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 24, Issue 4 (2002) 409–453
3. Monica S. Lam, et al.: The Cache Performance and Optimizations of Blocked Algorithms. In Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV), Santa Clara (1991) 63–74
4. Jaume Abella, et al.: Near-Optimal Loop Tiling by means of Cache Miss Equations and Genetic Algorithms, In Workshop on Compile/Runtime Techniques for Parallel Computing, Vancouver, BC, Canada, (2002).
5. Karin Hogstedt, Larry Carter, and Jeanne Ferrante.: Selecting tile shape for minimal execution time. In Symposium on Parallel Algorithms and Architectures (1999) 201–211
6. Vivek Sarkar.: Optimized Unrolling of Nested Loops, ICS 2000, Santa Fe, USA (2000) 153–166
7. Chau-Wen Tseng.: Software Support For Improving Locality in Advanced Scientific Codes. Technical Report CS-TR-4168, Dept. of Computer Science, University of Maryland (2000)
8. David Parello.: On Increasing Architecture Awareness in Program Optimizations to Bridge the Gap between Peak and Sustained Processor Performance – C Matrix-Multiply Revisited, SC2002.
9. T. Kisuki, et al.: Incorporating cache models in iterative compilation for combined tiling and unrolling, Technical Report 2000-10, LIACS, Leiden University (2000)
10. Niclos G. Fournier.: Enhancement of an Evolutionary Optimizing Compiler, Ph.D Thesis, Department of Computer Science, University of Manchester (1999)
11. Neal K. Bambha and Shuvra S. Bhattacharyya.: A Joint Power/Performance Optimization Algorithm for Multiprocessor Systems using a Period Graph Construct, ISSS 2000, Madrid, Spain, (2000) 91-99
12. Y. Chen, et al.: Automatic parallel I/O performance optimization using Genetic Algorithms, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, Chicago, Illinois (1998) 155–162
13. Xiaoping Wang, Liming Cao.: Genetic Algorithm: Theories, Applications and Software Implementation. Xi'an Jiaotong University Press, Xi'an (2002)
14. R.Clint Whaley, Antoine Petitet, Jack. J. Dongarra.: Automated Empirical Optimization of Software and the ATLAS Project, Parallel Computing, 27(1–2), (2001) 3–35

Study on CORBA-Based Load Balance Algorithm*

Gongxuan Zhang¹, Jianfang Ge², and Changan Jiang¹

¹ Department of Computer Science and Technology,
Nanjing University of Science and Technology,
210094 Nanjing, China

Gongxuan@mail.njust.edu.cn

² Department of applied mathematics,
Nantong Institute of Technology,
226007 Nantong, China

Abstract. With the increment of the number of Internet applications, in order to solve some problems such as heterogeneous access, dynamic services and their migration, a number of middleware and load balancing methods have been developed. CORBA is a standard of application middleware made by OMG. After studying some load balancing algorithms, we present an improved, CORBA-based load balancing model and its migration algorithm.

1 Introduction

In recent years, the Internet has been applied over most of our real world. Companies use it for their businesses to provide customers with their products or services. With the explosion of the number and type of services, some new mechanisms and frameworks are required to meet the needs of customers with ease. The mechanisms and frameworks are developed to discover, invoke and migrate web resources. Many new technologies are developed to improve business processes and quality of services over Internet.

CORBA, Common Object Request Broker Architecture, is a standard of middleware gave by Object Management Group[2]. There are many specifications of object services, for instance, naming service and trading service, described in the standard. With naming service, a client computer can locate an object reference by symbol name. And with trading services, a client is able to take advantage of the service's dynamic discovery mechanism for the object location. The mechanism is called dynamic binding. Trading services are hosted on a computer, with providing query service descriptions that clients can dynamically invoke the services just as the yellow pages of telephones. The proposed CORBA based load balancing model (in brief, CBLB model) is based on CORBA trading services. The rests of this paper are organized as follows: Section 2 describes trading service and layered load balancing abstract. Section 3 discusses the CBLB model, Section 4 shows migration algorithm. And the last is the conclusion.

* This is partly sponsored by the National Science Fund Committee, and the project number is 40120140817.

2 Trading Service

A CORBA Trading Service provides the function of detecting a dynamic object, and the client can ask it to locate the object. Similar to the naming service, the trading service stores object references. Note that, the trading service does not save the reference's name but saves the service description that the reference provides. The follow are some concepts about the object trading service:

1. A bulletin, used for trading storage, is called a service offer that contains the service's description and an object reference that provides the service. The service offer is still of some concrete service types.
2. The action of registering a bulletin is called 'export', and the actor of exporting is called to an exporter. That is a server's action.
3. The object reference inside the service offer is an object of which provides bulletin service, that is a service provider which can't be changed after the service offer is imported. And before the object reference does not remove and the service offer is imported again, the object reference can't be changed either
4. Inside a service offer, its service description, of which means the bulletin's "original text", is constructed by many name-value pairs. Compared to service supporters, the values may update. The same service promoters may be repeatedly advertised, but the values are not the same. Many bulletins have same name-values but different service promoters. This compare to an advertisement can list many stores in different areas
5. Bulletins can be withdrawn from the trading service. The bulletins are withdrawn or deleted only by servers.
6. The action for standard services to search for bulletins is called 'import', and the actor is an importer. That is a client's action.

In practice, we can take the Trading Service as the trading platform between CORBA object developers and CORBA clients. The developers register, explain and promote their objects through the application interfaces or with tools provided by object trading services, and the clients query, search, get and invoke the objects.

3 CBLB Model

In the CORBA environment, there are several layered abstracts for load balancing classification in order to construct a practical system. They are object level load balancing policy that consists of several CORBA objects identified by object reference, object implementing level load balancing policy that load balancing is implemented through active entities, for instance, processes of operating system, and system level load balancing policy that

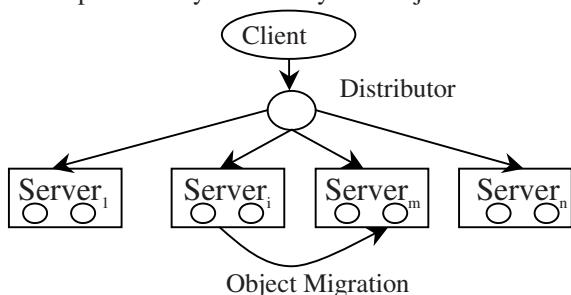


Fig. 1. Basic load balancing model

processes are distributed to different computing nodes just like traditional load balancing.

In this paper, the first two policies are introduced. With object level load balancing, an object reference will be taken if another object send a request to it. The client's requests are distributed to available servers. A client can contact the trading server and obtain appropriate object references if he needs specific services. The load balancing actions are also done within a trading server, and the server selects suitable object references and then sends the references back to the client. Finally, the client choices the object references and the object services.

With object implementing level load balancing, objects are special inner object instances of processes. The load balancing is implemented by distributing the instances' services uniformly to servers' processes, or by migrating the instances' services. As depicted in figure 1 is the basic way for load balancing.

As all known, the load balancing strategies are more complicated in a client/server environment. The client carries out a series of operations, each operation may send a request to a specific service. And many servers provide services together to meet all the requests. So the client must find out the servers that can handle its request.

The main responsibility of a trading server is to delivers services between the client and the server. By a trading server, a client can dynamically and accurately find out the servers that provide the services. The basic steps are shown in figure 2. As service-exporters, the servers export the services. This means they register their services in a trading server. As service-importers, the clients first invoke some operations of a trading server and acquire the information of a server's services. And then the clients can directly send its requests to the servers and invoke the services. The server can remove its registered services from the trading server by invoking the function withdraw() to the trading server.

The important problems are that how services are registered up to the trading server (called trader below) and specified by the clients. The solution of CBLB model is that a server must give its service description by specifying some special properties or service functions, and exports the description for clients to acquire the service. This means that a client must specify the name of a service type when importing a service, and then the trader maps the type to a corresponding object reference and sends the reference back to the client. It is very convenient for the client to communicate with the related servers. In the CORBA environment, the trader can be transparently embedded into an ORB (Object Request Broker) as one of basic CORBA services. In this paper, the trader is also a component used for the clients and servers.

In our proposed model, the mission of a server is to determine what resources are available for a client. A server is a process running in a specific host with the CORBA environment. For the sake of the convenience of description, we suppose each host circulate a server only. The available server can handle with a series of requests from the client. And the trader is to deliver the services between clients and servers. In addition, a supervisor is designed to deal with the service objects' migration among

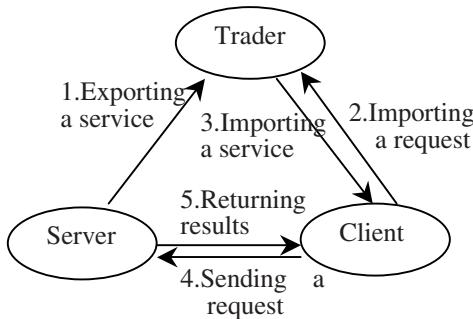


Fig. 2. Steps of services delivery

servers. So there are two aspects to deal with the client's requests to some servers with balance: one is by the trader's allocation policy and the other is by the monitor's migration algorithm.

There are three kinds of nodes in the CBLB model. The server node exports services to the trader node for registration. The monitor or the supervisor, as a special server, gathers load balance information from the server and then migrates some services to different servers if necessary. On the other hand, the client node imports available services from the trader and then invokes the services. When receiving a client's 'importing' request, the trader must quickly search the specific services by executing the function: $t:T \rightarrow p(S)$. If the service type T_i is provided by the current servers $\{S_1, S_2, \dots, S_k\}$, the function's result is $t(T_i) = \{S_1, S_2, \dots, S_k\}$. Of course, the function t is dynamically affected when the server withdraws the registered services or the monitor migrates services among the servers.

4 Migration Algorithm

As stated above, the service migration is the monitor's job. So the monitor must gather load balance information from the servers and then make a decision whether the service objects are migrated. Once making the decision, the monitor starts a migrating process. During the migration, he announces the destination server to immigrate the specific service objects from the source server. A basic implementation for migration is that the destination server creates a new service object as the same type one as in the source server and then receives the objects' status from the source server. Afterwards, the destination server announces the source server receives the new object reference in reverse. At the end, the destination server announces the trader that the object has been migrated correctly and asks the source server to remove the object. The entire procedure is depicted in Figure 3 and the following two cases should be considered if a client sends a request to a server while the server is doing migration:

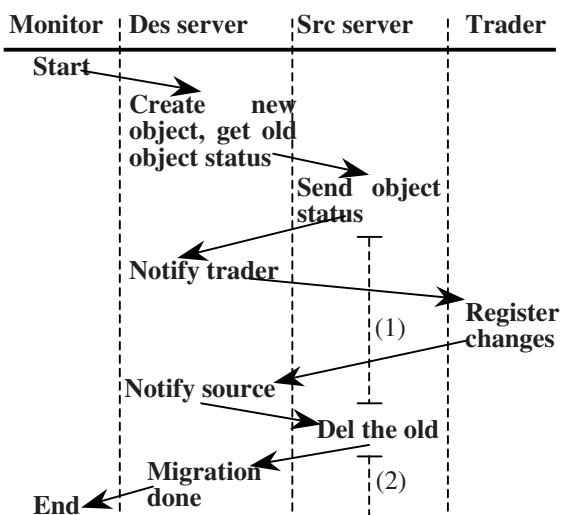


Fig. 3. The whole process of migration

- (1) One case is that the source object is not removed just after its states have been migrated when the request comes to the source server. The solution is the source server sends the new reference to the client and then the client sends the request to the destination server.

- (2) The other is that the migration has finished and the old object has been removed when the client sends the same request. The client will get error information and must import service objects again from the trader. In this case, the client invokes the new object reference from the trader.

Just as the discussion, a client always acquires correct service object references.

For the load balance algorithm, Peter Scheuermann's idea [4], with which the file is moved from a 'hottest disk' to a 'coolest disk' (called 'disk cooling process'), is introduced to our service object migration that the server stands for the disk and the service object for the file. The task of the migration algorithm is to move the service object from a 'heavier load server' to a 'lighter load server'. This is called 'the cooling server migration algorithm'.

5 Conclusion

With the CBLB model and the migration algorithm, some tests have been taken in a communication network with 7 computers. The performance is better but the trader will become a neck-bottle and block the migration when the service objects migration occur much frequently.

But there are two benefits in this proposed approach. First the CBLB model can easily join to existing applications as an ORB. The trader is embedded and taken as the ORB. Second the migration algorithm may easily be implemented in programming C++ or Java.

References

1. Andrew S. Tanenbaum. *Distributed Systems: Principles and Paradigms*. Prentice-Hall, Inc (2002).
2. K. Wallnau. Common Object Request Broker Architecture. Software Technology Review, Software Engineering Institute, Carnegie Mellon University (2000)
3. B. Schiemann, L. Borrmann. A new approach for load balancing in high-performance decision support systems. Future Generations Computer Systems, No. 12, Elsevier / North-Holland (1997) 345–355
4. Peter Scheuermann Gerhard Weikum and Peter Zabback. Disk Cooling Parallel Disk Systems. Vol. 17. No.3. IEEE Data Engineering Bulletin (1994) 29–40

Parallel Algorithm for Mining Maximal Frequent Patterns*

Hui Wang^{1,2**}, Zhitong Xiao², Hongjun Zhang¹, and Shengyi Jiang¹

¹ Computer School,
Huazhong University of Science & Technology,
430074, WuHan, China

² Wuhan Communication College,
430010,Wuhan, China
hustwanghui@mail.china.com

Abstract. We present a novel and powerful parallel algorithm for mining maximal frequent patterns, called Par-MinMax. It decomposes the search space by prefix-based equivalence classes, distributes work among the processors and selectively duplicates databases in such a way that each processor can compute the maximal frequent patterns independently. It utilizes multiple level backtrack pruning strategy and other novel pruning strategies, along with vertical database format, counting frequency by simple tid-list intersection operation. These techniques eliminate the need for synchronization, drastically cutting down the I/O overhead. The analysis and experimental results demonstrate the superb efficiency of our approach in comparison with the existing work.

1 Introduction

Mining frequent patterns is to discover all frequent patterns in a given database. It comprises the core of several data mining algorithms such as association rule mining and sequence mining, and dominates the running time of those algorithms. It has been shown to have an exponential worst case running time in the number of items, therefore much research [1,2,3,4] has been devoted to increasing the efficiency of the task.

Since both the data size and the computational costs are large, parallel algorithms have been studied extensively. Frequent pattern discovery has become a challenge for parallel programming since it is a highly complex operation on huge datasets demanding efficient and scalable algorithms. Most previous parallel algorithms [5,6,7,8,9,10] use complicated hash structures, make repeated passes over the database partition, have to exchange the partial results among all the processors during each iteration, resulting in additional maintaining overhead, high I/O overhead, expensive communication and synchronization cost.

* This paper is supported by the National Natural Science Foundation of China under Grant No.60273075.

** Hui Wang is a PhD candidate in computer school, Huazhong university of science and technology. Her research interests include data mining and parallel computing.

We present a novel and powerful parallel algorithm for mining maximal frequent patterns, called Par-MinMax, which is based on its serial version MinMax[11]. The new algorithm decomposes the original search space into smaller pieces by prefix-based equivalence classes, schedules classes among processors by the weights, distributes work among the processors and selectively duplicates databases in such a way that each processor can compute the frequent patterns independently. It uses depth-first search and a novel multiple level backtrack pruning strategy[11] and other powerful pruning strategies, along with vertical tid-list database format, counting frequency by simple tid-list intersection operation. These techniques eliminate the need for synchronization, drastically cutting down the I/O overhead. The analysis and experimental results demonstrate the superb efficiency of our approach in comparison with the previous work.

The rest of this paper is organized as follows: In section 2 we describe the maximal frequent pattern problem. The serial version MinMax is briefly described in Section 3. Section 4 describes our new algorithm Par-MinMax. We show the experimental results in section 5. The conclusions are in section 6.

2 Problem Statement

The problem of mining maximal frequent patterns is formally stated by definitions 1-4 and theorems 1-2. To describe our algorithm clearly, definition 5-9, propositions 1-2 and theorem 3 are given in this paper.

Let $\mathbf{I}=\{i_1, i_2, \dots, i_m\}$ be a set of m distinct items. Let \mathbf{D} denote a database of transactions where each transaction has a unique identifier (tid) and contains a set of items.

Definition 1: (pattern)

A set $X \subseteq \mathbf{I}$ is called a pattern (an itemset). A pattern with k items is called a k -pattern.

Definition 2: (pattern's frequency σ)

The frequency of a pattern X , denoted by $\sigma(X)$, is defined as the number of transactions in which the pattern occurs as a subset, called the support of the pattern.

With the vertical database layout, the database is comprised by items with corresponding tid-lists.

Let $x \in \mathbf{I}$, $\text{tid-list}(x)=\{t_i \mid x \text{ appeared in } t_i\}$. $\sigma(x)=|\text{tid-list}(x)|$. Let $X \subseteq \mathbf{I}$, $X=\{x_1, x_2, \dots, x_k\}$, $\sigma(X)=|\bigcap_{x_i \in X} \text{tid-list}(x_i)|$.

Definition 3: (frequent pattern)

Let ϵ be the threshold minimum frequency value specified by user. If $\sigma(X) \geq \epsilon$, X is called a frequent pattern. The frequent 1-pattern is called a frequent item. The set of all the frequent k -patterns in \mathbf{D} is denoted by F_k .

Definition 4: (maximal frequent pattern)

If $\sigma(X) \geq \epsilon \wedge \neg \exists Y (Y \supset X) \wedge \sigma(Y) \geq \epsilon$, we say X is a maximal frequent pattern.

Let $\mathbf{F} = \{X \mid X \subseteq \mathbf{I} \wedge \sigma(X) \geq \mathcal{E}\}$, $\mathbf{M} = \{X \mid X \in \mathbf{F} \wedge \neg \exists Y (Y \supset X) \wedge Y \in \mathbf{F}\}$, so $\mathbf{M} \subseteq \mathbf{F}$. Since a maximal frequent m -pattern includes 2^m frequent patterns, $|\mathbf{M}| << |\mathbf{F}|$. Given a threshold minimum frequency value \mathcal{E} and a database \mathbf{D} , the mining goal is to find \mathbf{F} from \mathbf{D} . Since the maximal frequent patterns contain all the frequent ones, it is wise to compute \mathbf{M} instead of \mathbf{F} .

It is profitable to view the frequency mining as a search problem. Each node in search space is composed of two parts: *head* and *tail*. In the initial status, the only node is root where *head*= \emptyset and *tail*= F_1 . The number of items in *node's tail* is the number of nodes in the next level that the node *node* can be frequently extended. Let *node's tail*= $\{a_1, a_2, \dots, a_n\}$, where *head* $\cup \{a_i\}$ is frequent, $i=1, 2, \dots, n$, then node (*head*, *tail*) can be extended n nodes: *head* $\cup \{a_i\}$, $\{a_{i+1}, a_{i+2}, \dots, a_n\}$, $i=1, 2, \dots, n$. The goal is to find all the *heads*. We use *head* \cup *tail* to represent the node itself.

According to definition 3, the following theorems hold.

Theorem 1: Any sub_patterns of a frequent pattern are frequent.

Theorem 2: Any super_patterns of an infrequent pattern are infrequent.

Definition 5: (item's infrequency λ)

A frequent item x 's infrequency λ is defined as the number of infrequent 2-patterns it makes.

$$\lambda(x) = |\{y \mid y \in F_1 \wedge x \in F_1 \wedge \sigma(\{x,y\}) < \mathcal{E}\}|$$

Proposition 1: If $\lambda(x_1) > \lambda(x_2)$, then x_1 makes more infrequent patterns than x_2 .

Proof: Due to the definition 5, the proposition holds.

Proposition 2: If $\sigma(x_1) < \sigma(x_2)$, then x_1 makes more infrequent patterns than x_2 .

Proof: Let T be a transaction in \mathbf{D} , $x_1, x_2, y \in \mathbf{I}$. Let $P(x)$ be the probability that x occurs in T . Let $P(xy)$ denote the probability that both x and y occur in T . Since $\sigma(x_1) < \sigma(x_2)$, so $P(x_1) < P(x_2)$. Because $P(xy) = P(x)*P(y)$, so $P(x_1y) < P(x_2y)$, then $\sigma(x_1y) < \sigma(x_2y)$. This implies that x_1 makes more infrequent patterns than x_2 .

Theorem 3: Let P be maximal frequent pattern in the sub-tree rooted with node *node(head,tail)*. If $P = \text{head} \cup \text{tail}$, then all nodes in this sub-tree are frequent.

Proof: In terms of the extending process, for any node *son-node* in the sub-tree rooted with *node*, *son-node* $\subseteq \text{head} \cup \text{tail}$ holds. Since $P = \text{head} \cup \text{tail}$, we have *son-node* $\subseteq P$. As P is frequent, so *son-node* is frequent.

Definition 6: (equivalence relation)

Let P be a set. An equivalence relation on P is a binary relation \equiv such that for all $X, Y, Z \in P$, the relation is:

- 1) Reflexive: $X \equiv X$.
- 2) Symmetric: $X \equiv Y$ implies $Y \equiv X$.
- 3) Transitive: $X \equiv Y$ and $Y \equiv Z$, implies $X \equiv Z$.

Definition 7: (equivalence class)

The equivalence relation partitions the set P into disjoint subsets called equivalence classes. The equivalence class of an element $X \in P$ is given as $[X] = \{Y | Y \in P \wedge X \equiv Y\}$.

Define a function $p: P(\mathbf{I}) \times N \mapsto P(\mathbf{I})$ where $p(X, k) = X[1:k]$, the k length prefix of X .

Definition 8: (prefix-based equivalence relation)

Define an equivalence relation θ_k on the lattice $P(\mathbf{I})$ as follows: $\forall X, Y \in P(\mathbf{I}), X \theta_k Y \Leftrightarrow p(X, k) = p(Y, k)$. That is, two patterns are in the same class if they share a common k length prefix. We therefore call θ_k a prefix-based equivalence relation.

Definition 9: (equivalence class weight)

Let $[x]$ denote an equivalence class on F_1 , based on equivalence relation θ_1 , m be the number of class $[x]$, $\omega(x)$ denote the weight of class $[x]$, $\omega(x) = \sum_{y: \{x\} \cup \{y\} \in [x]} \lambda(y) / m$.

3 MinMax: An Efficient Serial Algorithm

MinMax is an iterative algorithm based on a depth-first traversal over the search tree rooted with F_1 and returns the exact set of M . The basic idea of MinMax is to find out maximal frequent patterns as soon as possible and to use them to prune away the non-maximal frequent patterns which have superset in M . It has a stack keeping search trace and performs multiple level backtrack pruning (see line 7). Initially, F_1 was sorted by $\lambda \downarrow \sigma \uparrow$ according to propositions 1 and 2. It makes head with smaller tail and gets to a maximal frequent pattern much faster. MinMax also uses pruning strategies based on theorems 1(see line 6) and 2(see line 5) to make the process more efficiently.

MinMax(F_1, M)/* MinMax outputs the set of all the maximal frequent patterns M , F_1 was the input */

1. sort F_1 by $\lambda \downarrow \sigma \uparrow$;
2. stack P was initialized as $(\emptyset, F_1, 0)$;
3. select the most left item a_i in tail which flagbits(a_i) is 0;
4. current_head $\leftarrow P.\text{head} \cup \{a_i\}$;
5. current_tail := $\{y | y \in P.\text{tail} \wedge y > a_i \wedge \text{current_head} \cup \{y\} \text{ is frequent}\}$;
6. if $\text{current_head} \cup \text{current_tail}$ has a superset in M then flagbits(a_i) $\leftarrow 1$; goto 3 else goto 7
7. if $\text{current_tail} == \emptyset$ and current_head has no superset in M then $M = M \cup \text{current_head}$; backtrack to the oldest ancestor which $\text{head} \cup \text{tail} == \text{current_head}$; flagbits(a_i) $\leftarrow 1$; goto 3
8. if $\text{current_tail} \neq \emptyset$ then push $(\text{current_head}, \text{current_tail}, 0)$; goto 3

9. if all the flagbits==1 then $x_0 \leftarrow$ the last item in P.head; pop; flagbits(x_0) $\leftarrow 1$; goto 3
 10. if the stack P's end status is arrived then return M.
- End

4 Par-MinMax: Algorithm Design and Implementation

The new algorithm Par-MinMax overcomes the shortcomings of the Count and Candidate Distribution algorithms. It utilizes the aggregate memory of the system by partitioning the patterns into disjoint sets, which are assigned to different processors. The dependence among the processors is decoupled right in the beginning. Since each processor can proceed independently, there is no costly synchronization. Furthermore the new algorithm uses the vertical database layout which clusters all relevant information in a pattern's tid-list. Each processor computes all the frequent patterns from one class before proceeding to the next. The local database partition is scanned only once. As the pattern size increases, the size of the tid-list decreases, resulting in very fast intersections.

There are three distinct phases in the algorithms. The initialization phase, responsible for scheduling equivalence classes and distributing related tid-lists among the processors; the asynchronous phase, which generates local maximal frequent patterns, and the final reduction phase, which kicks out all the local maximal but not maximal frequent patterns in global. The more detail is as following:

- (1) Equivalence Class Generating and Scheduling: We first partition F_2 into equivalence classes by θ_1 , then computer each class's weight by definition 9, and sort the classes on the weights. We use a greedy algorithm to schedule the classes among the processors by assigning each class in turn to the least loaded processor at that point.
- (2) Database Repartitioning: Database was partitioned roughly equal among processors. To minimize communication and make each processor work independently, each processor scans the item tid-lists in its local database partition and writes it to a transmit region which is mapped for receive on other processors. The other processors extract the tid-list from the receive region if it belongs to any class assigned to them.
- (3) Asynchronous Mining: At the end of the initialization step, the relevant tid-lists are available locally on each host, thus each processor can independently generate the maximal frequent patterns from its assigned classes eliminating the need for synchronization with other processors. Each class is processed in its entirety before moving on to the next class in the schedule.
- (4) Final Post-processing: Since the results by each processor might be local maximal but not global maximal frequent patterns, so the non-maximal frequent patterns will be killed in the final post-processing.

The new algorithm Par-MinMax is described as follows:

```

Par-MinMax(F,M)/* Par-MinMax outputs all the maximal frequent patterns, Fi is
the input */
/* Initialization phase */
Generate independent classes from F2 by  $\theta_1$ ;
Schedule the classes among the processors on the weight of each class, by
definition 9.
Scan local database partition ; Transmit relevant tid-lists to other processors; Receive
tid-lists from other processors;
/* Asynchronous Phase */
for each processor Pi:
for each assigned class [x]:
Y={y|{x} ∪ {y} ∈ [x]};
MinMax(Y,Mi);
/* Final post-process Phase */
Aggregate Results and kick out all the non-maximal patterns and Output M
End

```

Table 1. The synthetic databases

Database	T	I	D	D1	D2	D4	D6
T10.I4.D2084K	10	4	2,084,000	91MB	182MB	364MB	546MB
T15.I4.D1471K	15	4	1,471,000	93MB	186MB	372MB	558MB
T20.I6.D1137K	20	6	1,137,000	92MB	184MB	368MB	552MB

Table 2. Speedup experiments

Number of processors	T10.I4.D2084 K	T15.I4.D1471K	T20.I6.D1137K
1	144	315	810
2	81	225	495
4	72	180	360
8	54	135	262

Table 3. Sizeup experiments

r	T10.I4.D2084K	T15.I4.D1471K	T20.I6.D1137K
1	20	40	80
2	35	60	150
4	72	180	360
6	158	320	700

5 Experimental Results

We implemented the algorithms on Dawn 3000 with 3 hosts, each host has 4 processors shared 2GB RAM, 9GB hard disk, each processor has the CPU 375MHz, by several synthetic datasets. It shows that our parallel algorithm Par-MinMax is well scalable in speedup and in sizeup.

We use the synthetic databases shown in table 1 to test our algorithm. Where T denotes the average transaction size , I the average maximal potentially frequent pattern size, D , the number of transactions , where r is the replication factor. For $r = 1$, all the databases are roughly 90MB in size. In speedup experiments, $r=4$. In sizeup experiments, the number of processors is 4. The speedup and sizeup experiment results are shown in table 2 and table 3 respectively. It shows that the speedup and sizeup are nearly linear.

6 Conclusions

In this paper we proposed a new parallel algorithm Par-MinMax for mining maximal frequent patterns. The algorithm uses the prefix-based decomposition technique, and the multiple level backtrack pruning strategy. The set of independent classes is scheduled among the processors, and the database is also selectively replicated so that the portion of the database needed for the computation of frequency is local to each processor. After the initial setup phase the algorithm does not need any further communication or synchronization. We implemented the algorithms on Dawn 3000 by several synthetic datasets. It shows that our parallel algorithm Par-MinMax is well scalable in speedup and in sizeup. Naturally, the load balance schema used by most parallel mining algorithms are static, our current research is directed to applying dynamic load balance schema to our algorithm, it would be hopeful to achieve better performance.

References

1. R.Agrawal, H.Mannila, R.Srikant, H.Toivonen and A.I. Verkamo: Fast Discovery of Association Rules. Advances in Knowledge Discovery and Data Mining, Chapter 12, AAAI/MIT Press, 1995.
2. R. J. Bayardo Jr.. Efficiently Mining Long Patterns from Databases. *Proc. of the ACM SIGMOD Conference on Management of Data*, Seattle, pages 85-93, June 1998.
3. D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: a maximal frequent itemset algorithm for transactional databases. In Intl. Conf. on Data Engineering, Apr. 2001.
4. K. Gouda, M. J. Zaki. Efficiently Mining Maximal Frequent Itemsets. In 1st IEEE Intl. Conf. On data mining, Nov. 2001.
5. R. Agrawal and J. Shafer, Parallel Mining of Association Rules, IEEE Trans. on Knowledge and Data Engg., 8(6):962-969, December 1996.
6. M.J.Zaki, S. Parthasarathy, M. Ogihara, and W. Li, New parallel algorithms for fast discovery of association rules, Data Mining and Knowledge Discovery: An International Journal, 1(4):343-373, December 1997.

7. M.J.Zaki: Parallel and Distributed Association Mining: A Survey, IEEE Concurrency, Vol.7, No.4, pp.14–25, 1999.
8. Ruoming Jin and Gagan Agrawal. Shared Memory Parallelization of Data Mining Algorithms: Techniques, Programming Interface, and Performance. In Proceedings of the second SIAM conference on Data Mining, April 2002.
9. Srinivasan Parthasarathy, Mohammed Zaki, Mitsunori Ogihara, Wei Li, Parallel Data Mining for Association Rules on Shared-memory Systems, in Knowledge and Information Systems, Volume 3, Number 1, pp 1–29, Feb 2001.
10. Karlton Sequeira, Mohammed J. Zaki, ADMIT: Anomaly-base Data Mining for Intrusions, 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 2002.
11. H.Wang, Q.Li, A Maximal Frequent Itemset Algorithm, Lecture Notes in Computer Science 2639, May 2003.

Design of Cluster Safe File System

Gang Zheng, Kai Ding, and Zhenxing He

North China Institute of Computing Technology,
Beijing, 100083, China
{zhengg,dingk,hezx}@nci.ac.cn

Abstract. The Cluster Safe File System (CSFS) is a distributed file system that can provide features such as single name space and remote data mirror. Files in the file system are scattered on all or some of the nodes in the cluster, and all nodes in the cluster can access the CSFS file system by mounting it on a local directory, clients can access the file system via NFS, FTP, Samba, etc. Every file in the file system has its synchronized mirror on another node, and that is why we call it SAFE. And in fact we can easily expand it to support more than one synchronized mirrors.

1 Introduction

High Available clusters are more and more widely used to provide non-stop services, such as web services, commercial and banking system, etc, in which cases, temporary failure might cause great loss and catastrophic consequences. And in many case, data loss is more intolerable than service failure, so it is also very important for a HA cluster to provide HA for its data. The CSFS is such a cluster file system which provides a mirror for each file in the file system on another node in the cluster to prevent from data loss. The CSFS is a filtering file system which is based on other file systems such as ext3, reiserfs, etc. The CSFS is both safe and single file system..

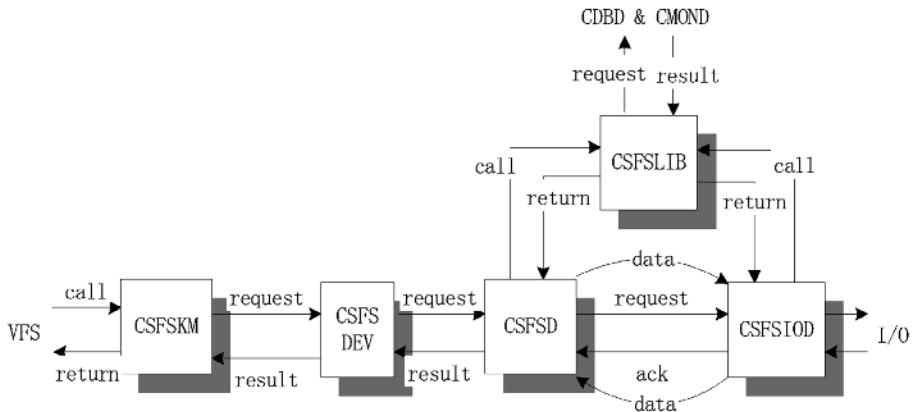
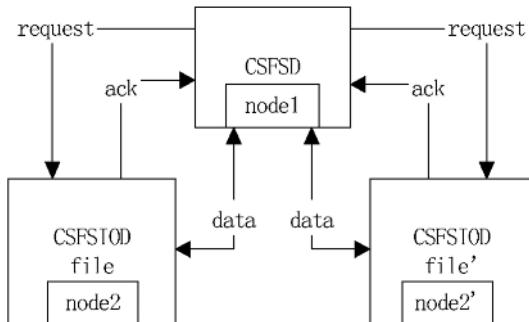
The CSFS is not designed from scratch. It is based on some existing open source projects such as FailSafe, PVFS and Intermezzo. It is designed for our cluster system based on the SGI's Linux FailSafe.

2 Architecture

The CSFS system is a distributed file system. Its physical arrangement and data mirroring mechanism are transparent to users.

The architecture of CSFS is made up of five parts (see figure 1, the CMOND is the cluster monitor daemon which can monitor processes), Linux kernel module (CSFSKM), Linux virtual device (CSFSDEV), CSFS library (CSFSLIB), file system service daemon (CSFSD) and file system input/output daemon (CSFSIOD).

Every file in the file system has its synchronized mirror on another node. In figure 2, the file' on node2' is a mirror of the file on node2.

**Fig. 1.** The CSFS's Architecture**Fig. 2.** File Mirror

The Linux kernel module (CSFSKM) generates file system operation request and Modification Log (Modification Log is generated only if the operation will modify the file system) for an operation, which are sent to and handled by the CSFSD. The CSFSD looks up the file or directory of the operation in the FailSafe's CDB (Cluster Database), and gets its two mirrored physical paths, then send the request to those nodes. And finally the CSFSIOD on those two nodes do the real I/O work.

2.1 CSFS Kernel Module (CSFSKM)

The CSFSKM is a bridge between the VFS and CSFSD. The CSFSKM gets an operation from VFS, translates it into request, and send it to the CSFSD via the CSFSDEV. After the result has arrived from the CSFSD, the CSFSKM gets it from the CSFSDEV and returns it to the VFS. If the request from the VFS belongs to writing requests, the CSFSKM will generate Modification Logs (ML), and fill it into the request that will be sent to the CSFSD.

2.2 CSFS Device (CSFSDEV)

The CSFSKM runs in the kernel space, while the CSFSD runs in the user space. So the CSFSKM need to communicate with the CSFSD via the CSFSDEV.

2.3 CSFS Library (CSFSLIB)

The CSFS's directory entries that are information about mapping logical paths into physical paths are stored in the FailSafe's CDB (Cluster Data Base), which will ensure that all nodes in the cluster will have exact copies of that information.

The CSFSLIB is a function library that provides interfaces to the CDB and the CMOND, etc. The CSFSD can use those functions to access the CDB

The mapping information between logical path and physical path of files and directories in the CSFS is stored in the CDB. Its format is

“logical_path → hostname1:physical_path1:hostname2:physical_path2”.

The logical_path is the file or directory's path in the single name space. The hostname is the node the file or directory is on. The physical_path is the local path of the file or directory on the node which it is stored. The hostname2:physical_path2 is the mirror of the hostname1:physical_path1. We also did some optimization to reduce the entries in the CDB. If only a directory or file are not on the same node with its parent directory will we create an entry in the CDB to record the mapping information, otherwise, we don't need it.

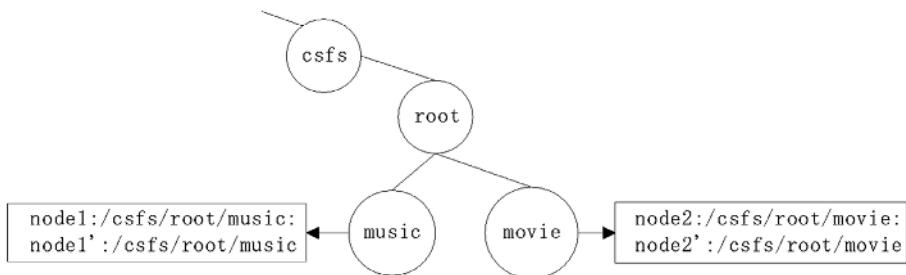


Fig. 3. Logical Directory Structure 1

We create an entry in CDB named “csfs”, and an entry named “root” under the node “csfs” which is the root of the CSFS's single name space. Every child entry under the “root” corresponds to a file or directory in the single name space. We store a file or directory's physical paths as the value of the CDB's entry. If the value is empty, it means that the file or directory corresponding to this entry and its parent directory are on the same node.

On the assumption that there are four machines which are referred to as node1, node1', node2 and node2' respectively, and two directories in the CSFS, one is music, which is stored on node1 and node1' with the local path “/csfs/root/music” and the other is movie, which is stored on node2 and node2' with the local path “/csfs/root/movie”. (see figure 3, the paths in the rectangle are the value of the corresponding CDB entry that stores the two mirrored physical paths).

2.4 CSFS Daemon (CSFSD)

The CSFSD's responsibility is receiving and handling the CSFSKM's requests. The CSFSD parses a request from CSFSKM, calls functions provided by the CSFSLIB to look up the file or directory of the request in the CDB and get its two mirrored physical paths, then send the request to those nodes' CSFSIOD.

Operation types of the request are divided into two classes, writing requests and reading requests. Reading requests are requests which don't modify the file system. Contrasting to reading requests, writing requests will modify the file system.

2.5 CSFS I/O Daemon (CSFSIOD)

The CSFSIOD receives requests from the CSFSD, does the real I/O work, and finally returns the results to the CSFSD by which the requests are sent. The content of a result message is similar to that of a result message from the CSFSD to the CSFSKM.

3 Reading Requests

Reading requests include null, getmeta, lookup, rlookup, readlink, getdents, statfs, export, hint, fsync. When the CSFSKM sends the reading request to the CSFSD, which doesn't include a ML, the CSFSD only sends the request to one of the two CSFSIODs on those two mirror nodes respectively. If the CSFSIOD returns success, the CSFSD returns success. Otherwise, the CSFSD sends the request to the other CSFSIOD. If the other CSFSIOD returns success, the CSFSD returns success. Otherwise, the CSFSD returns failure and send a notification to report the situation.

As for the read operation, because the data being transferred could be too large to be filled in the result message's buffer, so we decide to create another TCP connection between the CSFSD and the CSFSIOD, which is used to transfer data especially. The getdents operation is similar to read operation.

4 Writing Requests

Writing requests include setmeta, creat, remove, rename, symlink, mkdir, rmdir, mkdir, rmdir, read, write, which are attached with a ML. The CSFSD must send the request to both of CSFSIODs simultaneously on those two mirror nodes. The CSFSD waits for the CSFSIODs on both nodes to finish their jobs or time out. If both nodes finish their jobs successfully, CSFSD returns success. If one of them fails or time out, the CSFSD will send a notification to the administrator and then returns success. If both nodes fail or time out, the CSFSD will send a notification to report the situation and return failure.

In terms of the write operations, we also create other TCP connections between the CSFSD and those two CSFSIODs, which are used to transfer data especially.

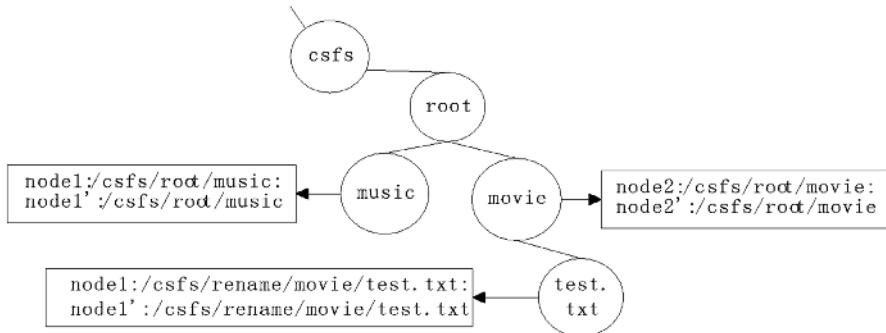


Fig. 4. Logical Directory Structure 2

5 Moving Files

In common condition, the implementation of rename is easy. But some times, the rename operation could lead to move files from one node to another node, because the old physical path and the new physical path are not on the same node. For example, we suppose that there is a file named test.txt under the directory /csfs/root/music on node1 and node1' (figure 3). After the user executes “rename /music/test.txt /movie/test.txt”, the CSFSD receives and parses the request, then maps the logical paths into physical paths. It finds out that the physical path of /movie is on node2 and node2'. It means that we need to move the file test.txt from node1 and node1' to node2 and node2'. Because of the expensive cost, We can solve it by the following method. The CSFSD sends the request “rename /csfs/root/music/test.txt /csfs/ rename/movie/test.txt” to the CSFSIODs on node1 and node1' simultaneously. The new physical path is made up of /csfs/ rename and the file or directory's logical path, so that the new physical path is unique under the directory /csfs/ rename. After the CSFSIODs on node1 and node1' return success, the CSFSD creates a logical path “/movie/test.txt” and sets its physical path as “node1:/csfs/ rename/movie/test.txt:node1':/csfs/rena-me/movie/test.txt”, then delete the logical path “/music/test.txt” in the CDB. (We create the directory /csfs/ rename on every node, under which all the files or directories that should have been moved out of this node). After that, the logical directory structure looks like the figure 4.

References

1. Mallik Mahalingam, Christos Karamanolis, Lisa Liu, Dan Muntz, Zheng Zhang. *Data Migration in a Distributed File Service*. Computer Systems and Technology Laboratory, HP Laboratories Palo Alto ,HPL-2001-128, May 23rd, 2001.
2. Philip H.Cams, Walter B.Ligon III, Robert B. Ross Rajeev Thakur. *PVFS: A Parallel File System for Linux Cluters*. October 2000.
3. Peter J.Braam. *InterMezzo: File Synchronization with Intersync*.
4. Joshua Rodman, Steven Levine. *Linux Failsafe Administrator's Guide*.2000.
5. Joshua Rodman, Lori Jhonson. *Linux Failsafe Programmer's Guide*. 2000.

Graph Scaling: A Technique for Automating Program Construction and Deployment in ClusterGOP

Fan Chan, Jiannong Cao, and Yudong Sun

Software Management & Development Lab, Department of Computing
The Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong

Abstract. Program development and resource management are critical issues in large-scaled parallel applications and they raise difficulties for the programmers. Automation tools can benefit the programmer by reducing the time and work required for programming, deploying, and managing parallel applications. In our previous work, we have developed a visual tool, VisualGOP, to help visual construction and automatic mapping of parallel programs to execute on the ClusterGOP platform, which provides a graph-oriented model and the environment for running the parallel applications on clusters. In VisualGOP, the programmer needs to manually build the task interaction graph. This may lead to scalability problem for large applications. In this paper, we propose a graph scaling approach that helps the programmer to develop and deploy a large-scale parallel application minimizing the effort of graph construction, task binding and program deployment. The graph scaling algorithms expand or reduce a task graph to match the specified scale of the program and the hardware architecture, e.g., the problem size, the number of processors and interconnection topology, so as to produce an automatic mapping. An example is used to illustrate the proposed approach and how programmer benefits in the automation tools.

1 Introduction

Programming with parallel applications is a difficult task involving programming details, processors information and network configurations. A method to simplify parallel program development is to abstract and represent the parallel programming structure by logical graphs.

Task graph is a graphical representation of a parallel program. It describes the logical structure of the program in which the nodes represent the computational tasks and the edges denote the communication links and precedence relationships among the nodes. Varieties of task graph have been proposed. Directed acyclic graph (DAG) [1, 2] and task interaction graph (TIG) [3, 4] are two ordinary types of task graphs. TIG is a concise representation of parallel program. The edges can represent any relationships between the nodes, for example, communication, synchronization, and execution precedence. The edges

can form loop to represent the iterative operations. Thus, TIG is more flexible to describe different program structures. Designing programs in TIG can simplify many programming details. However, there is a restriction for programming. TIG may have difficulty in handling complex relationships between tasks in a large-scaled graph. For the implementation, network configurations such as nodes-to-processors and LPs(local programs)-to-nodes mapping are time consuming tasks and difficult to handle manually. The programmer needs some tools for designing the TIG efficiently and managing the task mapping automatically.

In our previous work, we have developed tools for supporting the development of parallel applications, based on the graph-orient programming (GOP) model [5, 6]. We have developed a visual programming tool, VisualGOP [7], for designing the GOP program graphically. VisualGOP has a highly visual and interactive user interface, and provides a framework in which the design and coding of GOP programs, and the associated information can be viewed and modified. It also facilitates the compilation, mapping, and execution of the programs. Programs constructed in VisualGOP are deployed to the ClusterGOP, a high-level parallel computing platform for the cluster [8]. However, in VisualGOP, the programmer needs to manually build the task interaction graph. This may lead to scalability problem for large applications. In this paper, we describe the improvement to VisualGOP with an automation tool for constructing the graph and deploying the application in an efficiently way. In the tool, the graph scaling algorithms adapts a basic graph to the parameters of the application and determines the mapping between the programs and the processors automatically.

Section 2 introduces the ClusterGOP framework. Section 3 discusses the graph scaling approach. Section 4 presents the implementation of the task scaling and mapping tool in VisualGOP and the experiment using an example. Section 5 concludes the paper with the discussion of our future work.

2 The ClusterGOP Framework for Programming on Clusters

2.1 The ClusterGOP Model and Architecture

ClusterGOP is based on the GOP model, in which parallel/distributed program is defined as a collection of *local programs* (LPs) that may execute on several processors. Parallelism is expressed through explicit creation of LPs and communication between LPs is solely via message passing. The distinct feature of GOP is that it allows programmers to write distributed programs based on user-specified graphs, which serve the purpose of naming, grouping and configuring LPs. The graph construct is also used as the underlying structure for implementing uniform message passing and LP co-ordination mechanisms.

The key elements of GOP are a logical graph construct to be associated with the LPs of a parallel/distributed program and their relationships, and a collection of functions defined in terms of the graph and invoked by messages traversing the graph. As shown in Figure 1, the GOP model consists of the following:

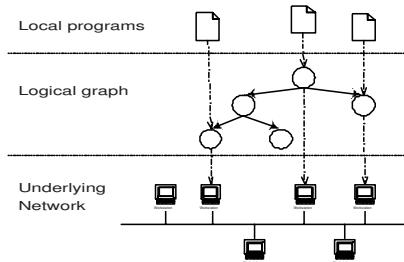


Fig. 1. The GOP conceptual model

- A *logical graph* (directed or undirected) whose nodes are associated with local programs (LPs), and whose edges define the relationships between the LPs.
- An *LPs-to-nodes mapping*, which allows the programmer to bind LPs to specific nodes.
- An optional *nodes-to-processors mapping*, which allows the programmer to explicitly specify the mapping of the logical graph to the underlying network of processors. When the mapping specification is omitted, a default mapping will be performed.
- A library of language-level graph-oriented programming primitives.

The GOP model provides high-level abstractions for programming distributed programs, easing the expression of parallelism, configuration, communication and coordination by directly supporting logical graph operations. It is important to note that GOP is independent of any particular language and platform. It can be implemented as library routines incorporated in familiar sequential languages and integrated with programming platforms such as PVM and MPI [9, 10].

ClusterGOP is an implementation of the GOP framework on MPI. The ClusterGOP software environment is illustrated in Figure 2. The top layer is a visual programming environment, VisualGOP, which supports the design and construction of parallel/distributed programs. A set of GOP API is provided for the programmer to use in parallel programming, so that the programmer can build application based on the GOP model, ignoring the details of low-level operations and concentrating on the logic of the parallel program. The GOP library provides a collection of routines implementing the GOP API. The goal in the GOP library implementation is to introduce a minimum number of services with a very simple functionality to minimize the package overhead.

The runtime system is responsible of compiling the application, maintaining structure, and executing the application. In the target machine, there exists two runtimes. The first one is the GOP runtime, a background process that provides graph deployment, update, query and synchronization. When deploying and updating the graph, it will block other machines to further update the graph and synchronize the graph update on all machines. Another runtime is the MPI

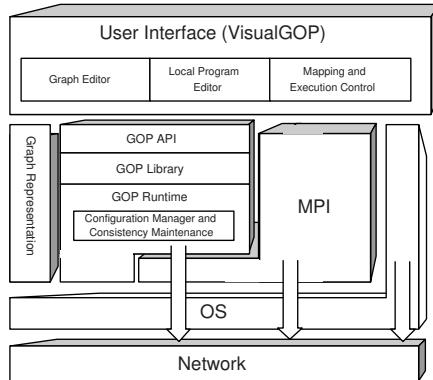


Fig. 2. The ClusterGOP Framework

runtime, which provides a complete set of parallel programming library for the GOP implementation.

3 Graph Scaling and Mapping

In this section, we will first introduce the basic patterns of the task graph, and then discuss the graph scaling method and the graph mapping strategy.

3.1 Regular Graphs for Parallel Application

The programmer needs to create a realistic graphical representation for parallel applications. The task graph should be defined as an abstraction of program structure. In addition, it should be high scalability to adapt to the parameters such as the problem size and number of processors. Our graph scaling approach is based on TIG by which the graph scaling algorithms and the graph mapping strategy will be implemented. In graph scaling, the nodes in the graph can be decomposed or merged, and the edges are reconstructed based on the original graph structure to produce a new task graph to match the parameters. TIG provides a concise topology to describe process-level computation and communication. It has a flexible structure for graph scaling.

Task graphs may have an arbitrary structure. More often, however, a task graph can take a regular topology such a tree, mesh, hypercube, etc., as parallel algorithms are often developed based on a regular topological model [11]. The following are typical topologies of task graph:

Tree. A *tree* has one root node and multiple leaf nodes (leaves). Each node except the root node has one parent. The edges are acyclic. If a graph satisfies these conditions, it can be identified as a tree.

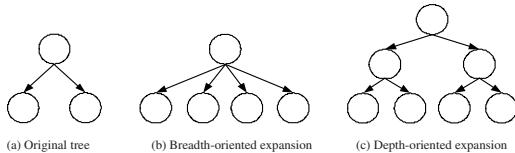


Fig. 3. Graph expansion for tree structure

Mesh. In an $m \times n$ mesh, there are four corner nodes with two neighbors each, $2(m-2)+2(n-2)$ boundary nodes with three neighbors each, and $(m-2)(n-2)$ inner nodes with four neighbors each.

Hypercube. In a hypercube with 2^n nodes, each node has n neighbors.

Arbitrary topologies. A task graph can present any other arbitrary topology.

The topology of a task graph is identified by the programmer. If the size of the graph, measured in the number of tasks that can be executed in parallel, does not conform to the parameters specified at runtime such as the problem size and the number of processors, graph scaling is required to derive a new graph from the original one to match the parameters.

3.2 Graph Expansion and Compression

Graph scaling can be made in two modes. If the number of parallel tasks is less than the required problem size or the number of processors, graph expansion will be performed to generate more tasks. On the other hand, if the number of parallel tasks is greater than the problem size, the graph should be compressed to include fewer nodes, although this is a rare situation in task graph. In addition, if the number of parallel tasks is greater than the available processors, the graph may be compressed.

In *graph expansion*, some nodes are decomposed and the edges are re-linked between the nodes based on the graph topology. The tasks should be redistributed among the expanded nodes. Figure 3 shows the expansion of a tree structure. The tree in Figure 3(a) can be expanded in two directions. One is breadth-oriented expansion as shown in Figure 3(b), in which all expanded nodes are attached to the root. The other is depth-oriented expansion in which the nodes spawn children beneath as shown in Figure 3(c).

Figure 4 shows the graph expansion for a mesh. The original 2×2 mesh is expanded to 2×4 and then 4×4 meshes by redeploying the decomposed nodes. If there are n nodes in a mesh, they are deployed as an $\sqrt{n} \times \frac{n}{\sqrt{n}}$ array. The nodes are linked by edges according to the mesh topology.

A hypercube is expanded in a similar way. The decomposed nodes are linked based on the hypercube topology. That is, each node is linked to k neighbors if there are totally 2^k nodes. Figure 5 shows the expansion of a 4-node hypercube to 8-node and 16-node hypercube.

Graph compression can use the same approach as the clustering in task scheduling [1, 2, 12]. It merges the nodes of a graph to clusters when the number

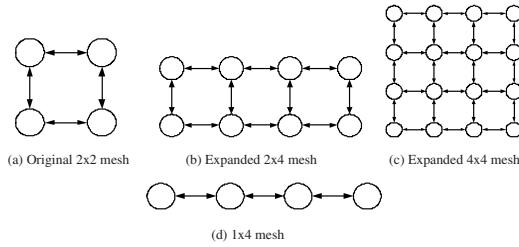


Fig. 4. Graph expansion for mesh structure

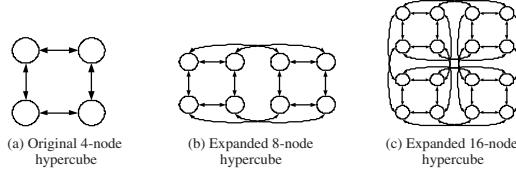


Fig. 5. Graph expansion for hypercube structure

of nodes is greater than the available processors. Graph compression is particularly useful to the cyclic graphs such as mesh and hypercube. The criterion of graph compression is the reduction of communication between the nodes. It analyzes the edges between the nodes and determines the neighboring nodes to be merged; meanwhile the topology of a compressed graph can be maintained. To compress the 16-node mesh in Figure 4(c), for example, the analysis can determine that the merge of two adjacent rows or columns has the same effect in reducing the inter-node communication. Thus, the compression can be made on either direction. If we choose row-oriented compression, the graph can be compressed into an 8-node in Figure 4(b). To compress the 8-node mesh to a 4-node one, the row-oriented compression will produce a one-dimensional mesh as shown in Figure 4(d). The column-oriented compression will result in a 2×2 mesh as shown in Figure 4(a). Using the criterion of communication reduction, it can be decided that the row-oriented compression can result in a graph with minimal the communication (denoted by three edges), in contrast to the result of column-oriented compression (i.e., the 2×2 mesh with four edges).

3.3 Graph Mapping

After the graph analysis and the graph scaling, LPs will be mapped to the task graph which has expanded or compressed. In SPMD (Single Program Multiple Data) model, all the nodes share the same copy of the program, so the mapping is simple. In the MPMD (Multiple Program Multiple Data) model, each node may work on different tasks. The programmer can choose a set of rules to do the LPs-to-nodes mapping automatically. There are rules for classifying the LP into different groups, e.g., a range of node ID, similar node names, and node

types. Finally, a task graph will be mapped to processors. Each processor is responsible for executing a node of in the task graph; i.e., there is a one-to-one correspondence between a processor and a node.

4 Implementation

We have developed algorithms for graph scaling as described in the previous section and implemented in VisualGOP. In this section, we first describe the scaling algorithms for graph expansion and then use our example to show how the proposed algorithms help the program design.

4.1 Scaling Algorithms

In the following parts, we will introduce three basic scaling algorithms for graph expansion. We assume that G_n is a 2-D graph and not weighted. We also assume that $|N_n|$ (the number of nodes) = $|N_p|$ (the number of processors), and there is a one-to-one mapping between N_n and N_p .

Tree. This structure can have two expansion types, the breadth-oriented and the depth-oriented expansion. We choose the binary tree as the example, which belongs to the depth-oriented expansion. The graph of binary tree has $2^n - 1$ nodes, where n starts from 2. During the graph expansion, the function `ExpDepBTree()` updates the graph according to the input node number. In each loop of the graph expansion, `Search_Leaf_Nodes()` will be invoked for adding the leaf nodes into a list. Each tree leaf node in the list will be expanded to produce a new level of leaf nodes. The psuedo-code segment for this algorithm is shown below:

```
public void ExpDepBTree( graph Gi, int nodenum ) {
    Until Gi's nodenum >= input nodenum
        initialize the graph and nodelist for saving the result
        Search_Leaf_Nodes(Gi, root, nodelist_t)
        For each node k in nodelist_t
            add new_left_node, new_right_node to Gi
            add new_left_node, new_right_node to Gi.alist[k]
        End Loop
        update Gi
}
```

Mesh. We use a 2×2 mesh as the original graph. This graph has 2^n nodes, where n starts from 2. In order to get a better communication performance, the expanded mesh should be maintained in a regular shape, so that the communication paths among the nodes are short in average. By identifying the node number, the graph expands vertically if n is an odd number; otherwise, the graph expands horizontally. Graph expansion is done by joining two identical graphs to form a new one. After that, all nodes are re-ranked

for getting new node IDs. The psedueo-code segment for this algorithm is shown below:

```

public void ExpMesh( graph Gi, int nodenum ) {
    initialize the graph for saving the result
    Until Gi's nodenum >= input nodenum
        Gj := Graph_Copy(Gi)
        Find_Coloumn_Row_Num(Gi's nodenum, col_num, row_num);
        calculate n_multiple for graph expands direction
        If n_multiple is odd Then
            For n=0 to col_num-1
                u:= node nodeID(col_num*(row_num-1)+n) Gi's last row
                v:= node nodeID(n) Gj's first row
                add v to Gi.alist[u]
                add u to Gj.alist[v]
            End Loop
        Else
            For n=0 to row_num-1
                u:= node nodeID(col_num*(n+1)-1) Gi's last column
                v:= node nodeID(col_num*n) Gj's first column
                add v to Gi.alist[u]
                add u to Gj.alist[v]
            End Loop
        End If
        rerank Gi and Gj
        join Gi and Gj
}

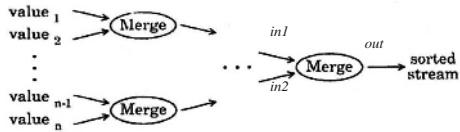
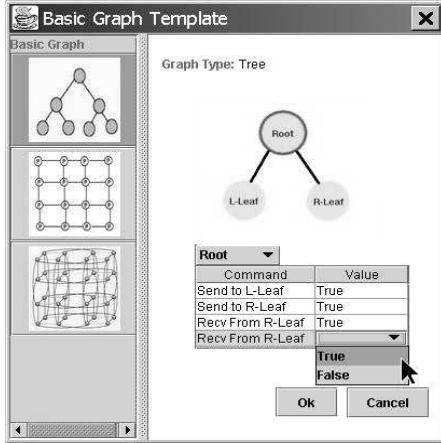
```

Hypercube. We choose a 2x2 hypercube as the original graph. This graph has 2^n nodes, where n starts from 2. Hypercube expands by connecting corresponding nodes from two identical graphs. All nodes are re-ranked after the creation of the new graph. The psedueo-code segment for this algorithm is shown below:

```

public void ExpHypercube( graph Gi, int nodenum ) {
    initialize the graph for saving the result
    Until Gi's nodenum >= input nodenum
        Gj := Graph_Copy(Gi)
        For n=0 to Gi's nodenum-1
            add v to Gi.alist[u]
            add u to Gj.alist[v]
        End Loop
        rerank Gi and Gj
        join Gi and Gj
}

```

**Fig. 6.** The Merge Sorting Algorithm**Fig. 7.** The Graph Template for the Basic Graph Diagram

4.2 An Example: The Parallel Merge Sorting

In our example, we use VisualGOP for the programming demonstration, which involves the program design, the graph scaling and the graph mapping. The programmer first uses VisualGOP to design the logical graph and LPs, then a processor list is created and the graph will be expanded or decompressed. Finally, VisualGOP applies the LPs-to-nodes and the nodes-to-processors mappings automatically.

We use a typical parallel algorithm, the parallel merge sorting, to show how programmer benefits by using the VisualGOP as an automation tool for designing parallel applications. The idea behind a merge sorting is to merge two sorted lists into a longer sorted list repeatedly in parallel (see in Figure 6). The graph is constructed out of instances of merge program. Each merge process receives values from two ordered input streams, *in1* and *in2*. It merges these values to produce one ordered output stream, *out*.

Step 1: Define the Basic Graph

The first step for the programmer is to define the graph for the parallel application (see in Figure 7). VisualGOP will ask for the graph pattern, which is required for the basic graph of the application. In this example, we choose the binary tree template for the merge sorting application.

Step 2: Program LPs and Define the LPs-to-nodes Mapping

In the next step, the programmer needs to define programs for LPs-to-nodes mapping. The example uses MPMD programming model, which contains three source codes (root.c, transfer.c and leaf.c) for different tasks in the merge sorting application. The first LP (root.c) is mapped to the root node, for collecting the final result. The second LP (transfer.c) is mapped to the transfer nodes (non-root and non-leaf nodes). Their major tasks are accepting the input from child nodes, merge and sort the data, and then transfer the results to their parent nodes. The last LP (leaf.c) is mapped to the leaf nodes, they are the nodes for retrieving the input list and pass the data to their parent nodes (transfer nodes or root nodes). After defining the LPs, VisualGOP provides mapping rules for the LPs-to-nodes mapping automatically.

Step 3: Prepare the Processors

The next step is the creation of the processor list for nodes-to-processors mapping. Programmer edits the processor list through adding, removing, or modifying the existing processor information. VisualGOP restricts the processor number input according to the graph pattern to prevent the programmer input unaccepted processor number to the algorithm.

Step 4: Graph Expansion

After receiving the processor number, VisualGOP will choose graph expansion if the available processor number is larger than the graph node number, otherwise the graph compression is used. The programmer can also make a preview on the expanded or compressed graph, for accepting or rejecting the changes.

Step 5: Graph Mapping

Finally, the expanded graph is created and the graph mapping starts automatically. Different LPs will be mapped to the specified nodes. The nodes-to-processors mapping helps programmer mapping the nodes to the processors automatically. Programmer can make the final changes to the mapping if needed. For example, the programmer can change the nodes-to-processors mapping to use more powerful processor to increase the performance of a specified node. After the graph mapping, the graph scheduling for the merge sort application has finished. Programmer can compile and execute the application directly.

5 Conclusions and Future Work

We have introduced the ClusterGOP system, which provides high-level abstractions for programming parallel applications, easing the expression of parallelism, configuration, communication and coordination by directly supporting logical graph operations. We also provide a visual programming environment, VisualGOP, to provide a visual and interactive way for the programmer to develop and deploy parallel applications. We propose the scaling algorithms for the task graph, which supports graph expansion and compression to match the specified parameters. The graph scaling realizes the draw-once run-variedly feature of task graph that contributes to the practicability of task graph based scheduling in parallel computing.

In our future work, we will take into account the computation load on the nodes and the communication costs on the edges when map the nodes to processors. One target of the mapping is to reduce inter-processor communication, and another one is to balance the workload among processors so as to minimize the execution time of entire program.

Acknowledgement. This work is partially supported by the Hong Kong Polytechnic University under the research grant H-ZJ80.

References

1. Kwok, Y.K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys* **31** (1999) 406–471
2. El-Rewini, H., Lewis, T.G., Ali, H.H.: Task Scheduling in Parallel and Distributed Systems. Prentice-Hall (1994)
3. Hui, C., Chanson, S.: Allocating task interaction graphs to processors in heterogeneous networks. *IEEE Transactions on Parallel and Distributed Systems* **8** (1997) 908–925
4. Senar, M.A., Ripoll, A., Cortes, A., Luque, E.: Clustering and reassignment-based mapping strategy for message-passing. In: 12th International Parallel Processing Symposium and 9th Symposium on Parallel and Distributed Processing, Florida, United States (1998) 415–421
5. Cao, J., Fernando, L., Zhang, K.: Programming distributed systems based on graphs. *Intensional Programming I*, World Scientific (1994)
6. Cao, J., Fernando, L., Zhang, K.: Dig: A graph-based construct for programming distributed systems. In: Proceedings of 2nd Int'l Conference on High Performance Computing, New Delhi, India (1995)
7. Chan, F., Cao, J., Chan, A.T., Zhang, K.: Visual programming support for graph-oriented parallel/distributed processing. Submitted for publication (2002)
8. Chan, F., Cao, J., Sun, Y.: High-level abstractions for message-passing parallel programming. To appear in *Parallel Computing* (Elsevier Science) (2003)
9. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderam, V.S.: PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA (1994)
10. Snir, M., et al: MPI: the complete reference. MIT Press, Cambridge, MA, USA (1996)
11. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing: Design and Analysis of Algorithms, 2nd Ed. Addison Wesley (2002)
12. Darbha, S., Agrawal, D.P.: A fast and scalable scheduling algorithm for distributed memory systems. In: Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing, San Antonio, TX (1995) 60–63

Pattern Classification with Parallel Processing of the Cellular Neural Networks-Based Dynamic Programming

Hyongsuk Kim, Taewan Oh, Sangik Na, and Changbae Yoon

School of Electronics and Information Eng.

Chonbuk National Univ.,

Republic of Korea

hskim@moak.chonbuk.ac.kr,

{control,hunter209}@mail.chonbuk.ac.kr,

gubooban@hanmail.net

Abstract. A Cellular Neural Networks (CNN)-based fast pattern classification algorithm utilizing the most likely path finding feature of the dynamic programming is proposed. Previous study shows that the dynamic programming for the most likely path finding algorithm can be implemented with CNN. If exemplars and test patterns are assigned as the goals and the start positions, respectively, on the CNN-based dynamic programming, the paths from test patterns to their closest exemplars are found with the optimality feature of the CNN-based dynamic programming. Such paths are utilized as aggregating keys for the classification. The algorithm is similar to the conventional neural network-based method in the use of the exemplar patterns but quite different in the use of the most likely path finding. Simulation results are included.

1 Introduction

One of the most important superiority human cognition over the conventional digital processing is the parallel processing in the image recognition. The pattern classification is an essential technique for the image recognition.

The pattern classification techniques ever developed can be classified roughly into the statistical probability-based, Fuzzy-based, and the neural network-based ones. The statistical probability-based technique [1][2][3] is the classical method where probabilities to belong to specific classes are computed utilizing the features of each class. However, choosing the adequate features to minimize the classification error is difficult. Also, the order of the function to compute the probability grows high due to the requirement of many features for better classification performance. In the Fuzzy-based technique [4][5][6], the patterns are classified according to fusion of Fuzzy rules. Analysis of exemplar patterns and creation of the Fuzzy rules for the classification are burdensome. The neural network-based technique [7][8] is the one which does not require either analysis of patterns or the effort of creating the Fuzzy rules. If some typical patterns called exemplars are presented for learning, the neural networks create classification rules automatically on the networks, which is the similar way to human learning. The classification is performed with learning of exemplar patterns without prior knowledge about them. The classification principle

under this approach is the use of nearest pattern classification. The drawbacks are long time taking in learning and unsuccessful learning due to the local minima.

The proposed pattern classification algorithm is with the utilization of fast parallel processing of the Cellular Neural Networks (CNN) [9][10]. The algorithm is similar to the neural networks-based classification in the use of exemplars but quite different in the use of the most likely path finding feature. Also, successful classification is always guaranteed and learning is not required. The basic principle under the proposed algorithm is the closest exemplar finding with the use of optimal computation property of dynamic programming. Since the proposed algorithm utilizes the competition among the classes, the optimality for the classification is kept. Also, since the classification is performed simply with presenting the exemplar patterns, prior knowledge or analysis of classes is not required.

2 Principle of CNN

The CNN is a massively parallel computational structure invented by Chua et al [9][10] in 1988. The CNN is composed of 2-D analog arithmetic cells which are disposed at nodes of 2-D grid as shown in Fig. 1(a). At each cell, there are 3 different sets of information sources: one from outputs of its and its neighboring cells through weighted connections called A template, the other from its and its neighbor's inputs through another set of weighted connections called B template, and the last one from its bias value z . In the CNN, each cell performs same dynamic operation as denoted in (1).

$$\frac{dx(i,j)}{dt} = -x(i,j) + \sum_{kl \in N_r} A(ij,kl)y_{kl} + \sum_{kl \in N_r} B(ij,kl)U(k,l) + z_{ij} \quad (1)$$

where $x(i,j)$ is the state of the cell (i,j). $A(ij,kl)$ and $B(ij,kl)$ are the templates(weights) between cell (i,j) and cell (k,l). Also, Z_j is the bias value of cell (i,j).

The output $y(i,j)$ of the cell (i,j) is

$$y(i,j) = f(x(i,j)) \quad (2)$$

where f is a nonlinear output function defined as in Fig. 1(b).

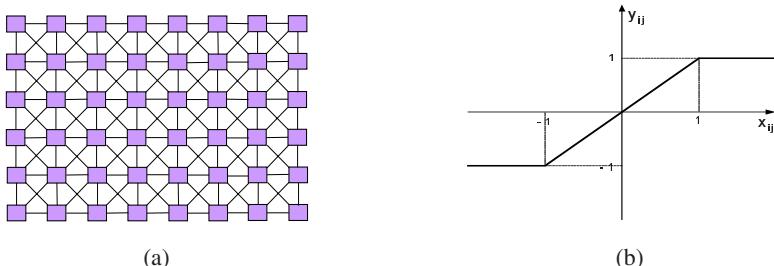


Fig. 1. Cellular Neural Networks(CNN) (a) cell disposition of the CNN (b) nonlinear output function of a CNN cell.

Since every CNN cell has same weighted connections and dynamic mechanism, every cell performs the same actions as long as input and state value are the same. However, if different template sets are employed, different processing results are expected even though same input or initial state values are given. Many sets of templates for different type of image processing have already been developed [11].

3 Implementation of the DP with CNN Nonlinear Templates

The Dynamic Programming (DP) is an efficient optimal path finding algorithm. Let $D(i,j)$ and $D(k,l)$ be the shortest distance to the goal from a node (i,j) and a node (k,l) , respectively. Then, $D(i,j)$ is computed utilizing $D(k,l)$ as

$$D(i,j) = \min \left\{ d_{ij,kl} + D(k,l) , (k,l) \in R(i,j) \right\} \quad (3)$$

where $d_{ij,kl}$ is the shortest distance between the node (i,j) and (k,l) , and $R(i,j)$ is the set of neighboring nodes of (i,j) . Note that $d_{ij,kl}$ equals to zero if $ij=kl$. By setting the initial value of D at a goal (k,l) with zero and all other D s with a big value (bigger value than shortest distance between the most far nodes), $D(i,j)$ in (3) computes the minimum distance to the goal (k,l) at a node (i,j) . If an arithmetic cell is arranged to compute (3) at each node, the processing of (3) is confined to the local operation of *min* and *summation*. Since the *min* circuit is known to be more complicated than *max* circuits for the practical implementation [12], some arrangement of (3) enables *min* to be replaced by *max* operation.

Let's $y(i,j)$ be the complement value of $D(i,j)$ from a dummy constant I_{\max} for the node at (i,j) as

$$y(i,j) = I_{\max} - D(i,j) \quad (4)$$

Plugging $D(i,j)$ of (4) into (3) and rearranging its result, we have

$$y(i,j) = \max \left\{ y(k,l) - d_{ij,kl} ; (k,l) \in R(i,j) \right\} \quad (5)$$

In the proposed algorithm, (5) is computed by a cell at each node. On the networks with these cells, I_{\max} is set for the state of the goal node and zero is set for all other states. At the goal cell, $y(k,l)$ and $d_{ij,kl}$ in the parenthesis of (5) are I_{\max} and zero, respectively since $(k,l) = (i,j)$. Therefore, the operation of (5) can be summarized as

$$y(i,j) = \begin{cases} I_{\max} & ; \text{if } (i,j) \text{ is designated as the goal.} \\ \max \{ y(k,l) - d_{ij,kl} ; (k,l) \in R(i,j) \} & ; \text{otherwise.} \end{cases} \quad (6)$$

With $y(i,j)$, the shortest distance to the goal, $D(i,j)$, can easily be computed with (4).

The physical meaning of the modified dynamic programming of (5) is associated with the propagation of reference information from its goal while its magnitude is reduced by the amount of the distance weight value between cells.

The nonlinear function of the *max* and the linear function of subtraction are involved in operation (5). It is not easy to be directly implemented with the CNN linear template due to the nonlinearity of these operations. The proposed strategy to implement the functions (5) with CNN is associated with the decomposition into several simple functions. Arranging them to be processed by cells at different layer, the CNN to compute (5) is feasible. Adding y_{ij} to both side of (5), (5) becomes

$$2y_{ij} = \max \{y_{ij} + y_{kl} - d_{ij,kl}, (k,l) \in R(i,j)\} \quad (7)$$

or

$$y_{ij} = \max \left\{ \frac{y_{ij} + y_{kl} - d_{ij,kl}}{2}, (k,l) \in R(i,j) \right\} \quad (8)$$

Let the *max* in (8) be processed by a cell on a layer called Distance Computation layer and the terms inside of the parenthesis be processed by a cell on the other layer called Intermediate layer. Then,

$$y_{DC}(i,j) = \max \{y_I(i,j)\} \quad (9)$$

and

$$y_I(i,j) = \frac{1}{2}(y_{DC}(i,j) + y_{DC}(k,l)) - \frac{d_{ij,kl}}{2} \quad (10)$$

where $y_I(i,j)$ and $y_{DC}(i,j)$ are the output of the cell (i,j) in the I layer and DC layer, respectively. The structure to find the most likely path through such processing is as in Figure 2. The cells of I layer are placed at the locations corresponding to the links between the nodes and the metric information is provided externally to its input. The cells in DC and PF layer have connections only with their adjacent cells in I layer.

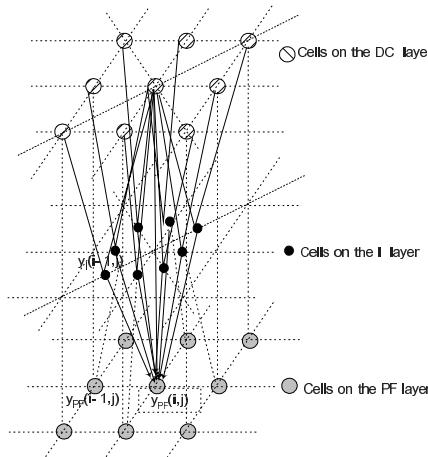


Fig. 2. Connections of a cell in the proposed multi-layer CNN structure.

Thus, the *max* computation in (9) is performed with the interaction between the cells in DC layer and I layer. Path decision criterion can also be more simplified utilizing the cells in the I layer as

$$y_I(i, j) = y_{DC}(i, j) \quad (11)$$

The average operation and the subtraction in (10) can easily be performed with the linear template [11]. The nonlinear operation of *max* in (9) and comparison in (11) can be implemented with nonlinear templates as in [13].

4 Patten Classification Utilizing the Optimization Property of the Dynamic Programming

The proposed pattern classification algorithm is the utilization of both benefits of parallel processing property of the CNN and the optimization property of the dynamic programming. The CNN determines appropriate group for each test pattern based on the most likely path finding mechanism where the exemplars and the test patterns are treated as the goals and the start points, respectively.

Let j th pattern in the i th class be denoted as $P_i(j)$ and P_s be the whole set of the patterns. That is

$$P_i(j) \in P_s, \quad 1 \leq i \leq M, 1 \leq j \leq N_m \quad (12)$$

where M is the number of classes and N_m is the number of exemplar patterns in each class. Also, let the distance between the test pattern P_t and an exemplar $P_i(j)$ (j th pattern in the i th class) be the distance d_{t-ij} . If the test pattern has the closest distance to the l th pattern in the k th class such as

$$d_{t-kl} = \min\{d_{t-ij}, \quad 1 \leq i \leq M, 1 \leq j \leq N_m\} \quad (13)$$

then, k is the class to which the test pattern has the shortest path among M classes. In this case, the pattern P_t is classified into the class k .

The algorithm to implement above concept with CNN is shown in Figure 3. For this algorithm, the location of each pattern is expressed with the cell position in the multilayer CNN. The first step of the algorithm is the distance weight setting on the I layer of the CNN. The information is utilized for computing shortest distance to the nearest exemplar. The next step is the exemplar and test pattern setting on the DC and PF layer, respectively. If the identical reference values $I_{max,s}$ are given at the exemplar positions as goals, the reference values propagate to all directions while they are reduced by the amount of the inter cell distance value provided by I layer. When the reference value reaches each test pattern, the path connection from each test pattern to the closest exemplar begins to be made. The path information obtained after this procedure is bi-levels with zero and infinitive big value. Providing such paths as distance information on I layer again and presenting class identification (ID) values at the exemplar cell positions, the identification values are propagated without alteration through the paths with zero distance weighting. When the class identification values reach the positions of the test patterns, the classification is performed by simply reading the class identification values which appear at the cells corresponding to test patterns.

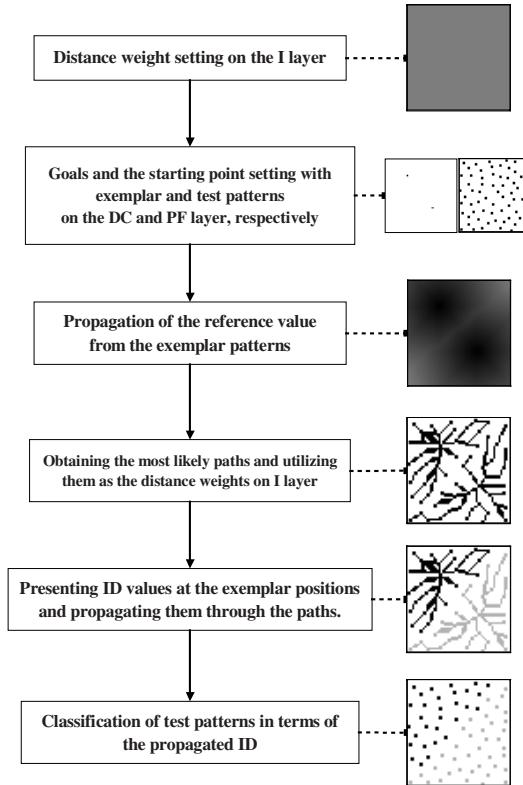


Fig. 3. Flow diagram of the proposed algorithm.

5 Simulation

Pattern classification capability of the proposed algorithm has been shown about patterns with multiple classes. Also, its classification capability for data with highly nonlinear class boundary has been demonstrated.

5.1 Classification of Patterns with Multiple Classes

Figure 4 shows the data with multiple classes, where figures 4(a) shows exemplar patterns of 4 classes and Figure 4(b) is test patterns. It is assumed that the inter cell distances are all identical. Being provided four identical reference inputs at the exemplar positions, the reference values propagate through the CNN networks while they are reduced by the amount of inter-cell distance as in Figure 4(c). If the reduced reference values reach the locations of the test patterns, the paths initiated at the locations of test patterns proceed along the direction which the strongest reference value comes from. The most likely paths obtained with such procedure are shown in Figure 4(d). These paths are provided again on the I layer as the inter-cell distance of

zero and 1 for path and non-path, respectively. With such bi-leveled inter-cell distance, the identification values presented at exemplar positions are propagated through the paths. Figure 4(e) shows the propagated identification values on the paths. Note that the distance weights on weights are zero. The classification is performed simply with reading the propagated class identification value at each pattern position. Figure 4(f) is the classified results with this procedure. It can be seen that the test patterns are all reasonably classified in terms of the Euclidian distance to the closest classes of exemplars.

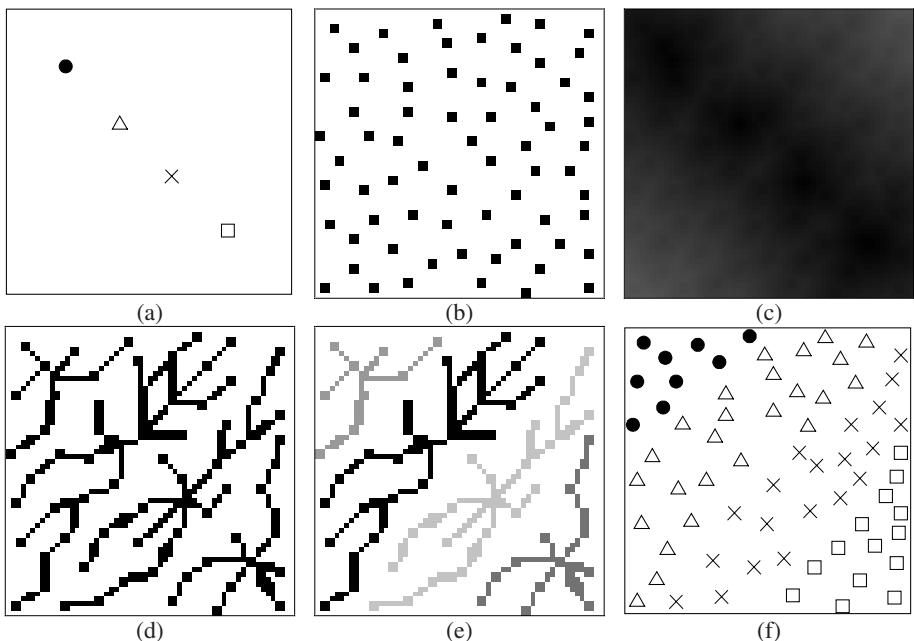


Fig. 4. Classification of patterns of multiple classes (a) exemplar patterns (b) test pattern (c) propagated reference values (d) paths from the test patterns to the closest exemplars (e) class identification values propagated from the exemplar patterns through the paths in (d) (f) classified patterns.

5.2 Classification of Data with Nonlinear Class Boundaries

The pattern classification about data with the nonlinear class boundary is a difficult task in classification. Figure 5 is the data with very highly nonlinear class boundaries where Figure 5(a) is the exemplar patterns and Figure 5(b) is test patterns. Also, the Figures 5(c)-5(e) shows the propagated reference values, the paths to the nearest exemplar, and the propagated identification values, respectively. The classified result is as shown in Figure 5(f). No extra difficulty is experienced in classifying patterns with highly nonlinear class boundaries.

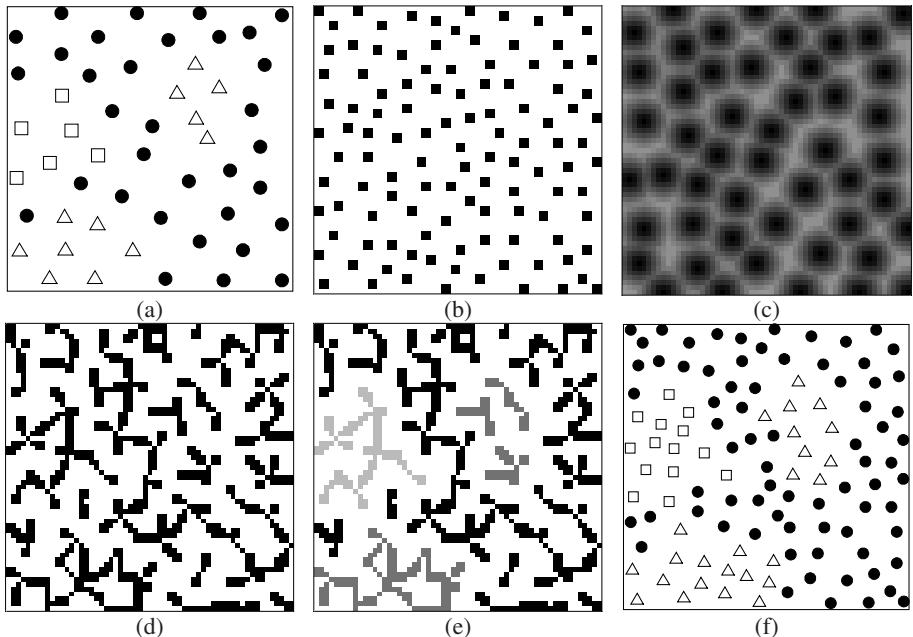


Fig. 5. Classification of the patterns with highly nonlinear class boundaries (a) exemplar patterns (b) test pattern (c) propagated reference values (d) paths from the test patterns to the closest exemplars (e) class identification values propagated from the exemplar patterns through the paths in (d) (f) classified patterns.

6 Conclusion

A pattern classification algorithm using the analog parallel processing of the CNN technology has been proposed. The algorithm utilizes the both benefits of parallel processing of the CNN and the optimization of the dynamic programming. Another feature of the proposed classification algorithm is the use of exemplars as in the conventional neural networks.

In this classification algorithm, the exemplars and test patterns are considered as goals and starting points in the most likely path finding algorithm, respectively. The classification is composed of two steps of information propagation; the distance reference signal propagation and the class identification value propagation. Since principle of the proposed algorithm is the use of competition among the class, the optimality for the classification is kept. Also, prior knowledge or analysis of classes is not necessary in this algorithm.

Simulations have been done to test the properties of the proposed algorithm. With the algorithm, the classification has been performed successfully for all the problems in the simulations. Perhaps the strongest feature of the proposed algorithm is the classification capability about the data with highly nonlinear class boundaries. No extra difficulty is encountered in classifying patterns with highly nonlinear class boundaries.

References

1. Patra, P.K. Nayak, M. Nayak, S.K. Gobbak, N.K.: Probabilistic Neural Network for Pattern Classification. *Neural Networks*, 2002. IJCNN '02. Proceedings of the 2002 International Joint Conference on , vol. 2, pp.1200–1205, 2002.
2. Lee Luan Ling, Cavalcanti, H.M.: Fast and efficient feature extraction based on Bayesian decision boundaries. *Pattern Recognition*, 2000. Proceedings. 15th International Conference on , vol. 2, pp. 390–393, 2000.
3. Horiuchi, T.: Decision rule for pattern classification by integrating interval feature values. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol. 20, pp. 440–448, 1998, Apr.
4. Hahn-Ming Lee, Chih-Ming Chen, Jyh-Ming Chen, Yu-Lu Jou.: An efficient fuzzy classifier with feature selection based on fuzzy entropy, *Systems, Man and Cybernetics, Part B, IEEE Transactions on* , vol. 31, pp. 426–432, 2001, June.
5. Ishibuchi, H. Nakashima, T. Murata, T.: Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *Systems, Man and Cybernetics, Part B, IEEE Transactions on* , vol. 29, pp. 601–618, 1999, Oct.
6. Xi Chen, Dongming Jin, Zhijian Li.: Fuzzy petri nets for rule-based pattern classification, *Communications, Circuits and Systems and West Sino Expositions, IEEE 2002 International Conference on* , vol. 2 , pp. 1218–1222, 2002.
7. Jinwook Go, Gunhee Han, Hagbae Kim, Chulhee Lee.: Multigradient: A New Neural Network Learning Algorithm for Pattern Classification, *Geoscience and Remote Sensing, IEEE Transactions on* , vol. 39, pp. 986–993, 2001, May.
8. Murphy, Y. L. Yun Luo,: Feature Extraction for a Multiple Pattern Classification Neural Network System. *Pattern Recognition*, 2002. Proceedings. 16th International Conference on , vol. 2, pp. 220–223, 2002.
9. Chua, L.O. and Yang, L.: Cellular Neural Networks: theory. *IEEE Tr. on Circuits Systems*, vol.35, pp.1257–1272, 1988.
10. Chua, L.O. and Yang, L.: Cellular Neural Networks: applications. *IEEE Tr. on Circuits Systems*, vol. 35, pp. 1273–1290, 1988.
11. Analogical and Neural Computing Lab, SimCNN-Multi-layer CNN Simulator for Visual Mouse Platform: Reference Manual version 2.2. Computer and Automation Institute(MTA SzTAKI) of the Hungarian Academy of Science, 1998.
12. Espejo, S., Dominguez-Castro, R., Linan, G., Rodriguez-Vazquez, A.: A 64×64 CNN universal chip with analog and digital I/O. *Electronics, Circuits and Systems*, 1998 IEEE International Conference on , vol. 1, pp. 203–206, 1998.
13. Kim, H., Son, H., Roska, T., Chua, L.O.: Optimal path finding with space- and time-variant metric weights with Multi-layer CNN. *International Journal of Circuit Theory and Applications*, vol. 30, pp. 247–270, 2002, 2.
14. Roska, T. and Chua, L.O.: The CNN universal machine: an analogic array computer. *IEEE Tr. on Circuits Systems II, CAS-40*, pp. 163–173, 1993.
15. Bellman, R.: *Dynamic Programming*, Princeton, NJ: Princeton Univ. Press, 1957.

An Effective Molecular Algorithm for Solving the Satisfiability Problem*

Wen Yu and Weimin Zheng

Department of Computer and Technology,
Tsinghua University,
Beijing, 100084, China
yuwenyw@263.sina.com
zwm-dcs@tsinghua.edu.cn

Abstract. A well-defined satisfiability problem (SAT) is mapped into a unique expression of logical array by introducing a transformation. Such an expression forms a unique molecular algorithm for solving SAT, which is prone to computation on an electronic computer. A famous 20-variable instance of the three-satisfiability problem is solved via Matrix laboratory (MATLAB) installed on a notebook computer. The special computation implements huge parallelism that goes beyond 1 million (2^{20}). The unique solution is found and the computational complexity is analyzed.

1 Introduction

After Adleman[1] and subsequently Lipton [2]demonstrated how to solve instances of NP-complete combinatorial problems by using DNA, numerous models of molecular computing as well as efficient methods for solving the intractable computational problems have been presented[3-9], several of have been explored experimentally and found to be feasible[5-9]. Approaches of examining the complexity of molecular computation also have been investigated [10,11].

Among the methods of molecular computing, the NP-complete satisfaction problem (SAT) were greatly investigated [2,5,7-9]. “The problem became the benchmark for testing the performance of DNA computers”[12, p. 499]. In 2002, Ravinderjit S.et al reported a 20-variable NP-complete 3-satisfiability problem, a special instance of SAT, was solved experimentally on DNA computers with exhaustive search, which may be the largest solved using DNA means [12].

Massively parallel features that appear in DNA computing are the reason for the gain of time. However, most of molecular computations are based on some idealized assumptions and deterministic working pattern[2]. Because of the absence of technical breakthroughs, almost each step of molecular operations is probabilistic and prone to error in practical [7]. Thus, it is the task of theoretical computer science to find out some ideal models and techniques of the new computation method.

* Supported by the National Natural Science Foundation of China under Grant No. 60273007 and No.60131160743.

A silicon-based model of molecular computing was investigated uslately, which may solve a class of SAT problem in polynomial time. This new model has an unconventional, parallel logic access memory, where each literal input, a Boolean variable or its negation, is automatically transformed into a binary periodic array or outputs via a hardwired decoding circuitry. Such an array can control exponentially bounded cells of memory for implementing massively parallel access. Here we show that the transform technique can be applied to the conventional computation through MATLAB, and a unique molecular computation for solving the SAT problem is presented. For a brief comparison with the DNA-based approach, we use MATLAB 6.0 installed on a notebook computer [13] to solve the famous 20-variable instance of 3-SAT problem .The unique solution is found in polynomial time after massively parallel search.

In our molecular computing, each Boolean variable (or its negation) is transformed into a unique binary periodic matrix that has exponential length, with respect to the size of input variable. Each of Boolean operators also is transformed into the corresponding operator of logical matrix. As a result, a standard Boolean formula is mapped into a unique expression of logical array. To use the basic matrix operators step by step, a final output matrix is returned or produced in polynomial steps. The final output matrix just is the transformed solution space, which represents exactly the all set of solutions to the original Boolean formula. Naturally, the desired answer and solutions can be tested or read out.

Briefly, our molecular computing for solving arbitrary SAT problem consists of five main steps and may be summarized as follows. 1) Generate all initial input matrices. 2) Obtain all OR-ed condition matrices. 3) Obtain an AND-ed output matrix. 4). Test the output matrix, accept if it is a nonzero matrix; otherwise reject. 5) Readout the existing solution if accepted.

2 The Computational Problem and Method

2.1 The Satisfiability Problem

The satisfiability problem with n variables can be formally written as a conjunction of m clauses (well-defined formula):

$$f = C_1 \cap \dots \cap C_i \cap \dots \cap C_m \quad (1)$$

where symbol “ \cap ” denotes the conjunction (Boolean *and*), and C_i ($i = 1, 2, \dots, m$) stands for a clause that is some literals connected with “ \cup s” (Boolean *or*). The problem is to determine whether a Boolean formula is satisfiable, this is, if there is a truth-value for all variables that satisfy all clauses.

2.2 Solution Representation with Matrix

For any satisfiability problem with n -variable, there are 2^n possible truth assignments, each of which is a unique n -tuple of 0s and 1s and corresponds a either

true or false state. To represent the truth assignments and corresponding states to a given instance, a binary logical matrix (array) with 2^n elements is employed. In our strategy, each position of elements that starts with 0 and ends with $2^n - 1$ represents a fixed truth assignment, and the value of element such as either “1” or “0” represents a either true or false state. Intuitively, such a binary matrix is able to act as a unique solution space for computation. For example, the identity matrix $[1,1,\dots,1,1]_{2^n}$ denoted by \mathbf{I}_{2^n} stands for a full solution space that all truth assignments are true, whereas the zero matrix $[0,0,\dots,0,0]_{2^n}$ denoted by $\mathbf{0}_{2^n}$ corresponds an empty solution space that all truth assignments are false. Identity matrix and zero matrices are seen two trivial ones. Fig. 1 demonstrates that a binary matrix with 16 elements represents a unique solution space that has specified 2 true truth assignments and 14 false ones.

A	$[1, 0, \dots, 0, 1, 0]_{16}$
	$\downarrow \quad \downarrow \quad \quad \downarrow \quad \downarrow \quad \downarrow$
B	<u>0000,0001,.....,1101,1110,1111</u>
C	{0000, 1110}

Fig. 1. A. A binary matrix with 16 elements. B. The corresponding 16 truth assignments. C. Two true truth assignments.

2.3 The Overall Method for Solving SAT

An overall method for solving SAT problem with n variables and m clauses is summarized as follows:

0) Transform: make a transformed formula according to the following rule:

$$T : (x_k, \sim x_k, \cap, \cup, C_i) \rightarrow (Xk, \sim Xk, \&, |, C_i) \quad (2)$$

where literal x_k and $\sim x_k$ ($k=1,\dots,n$) correspond the specified periodic matrices Xk and $\sim Xk$ ($k=1,\dots, n$), respectively, and operators “ \cap ” and “ \cup ” correspond matrix operators AND “ $\&$ ” and OR “ $|$ ”, respectively. Correspondingly, a clause C_i is converted to a “condition” C_i that consists of some matrices connected with “ls”, and usually is enclosed in braces “()”.

1) Initializing: Generate each matrix of Xk and $\sim Xk$ k ($k=1,\dots,n$) as follows through MATLAB.

$$Xk = [\underbrace{0, \dots, 0}_{2^{n-k}}, \underbrace{1, \dots, 1}_{2^{n-k}}, \dots, \underbrace{0, \dots, 0}_{2^{n-k}}, \underbrace{1, \dots, 1}_{2^{n-k}}]_{2^n} \quad (k=1,1,\dots,n) \quad (3)$$

and

$$\sim Xk = [\underbrace{1, \dots, 1}_{2^{n-k}}, \underbrace{0, \dots, 0}_{2^{n-k}}, \dots, \underbrace{1, \dots, 1}_{2^{n-k}}, \underbrace{0, \dots, 0}_{2^{n-k}}]_{2^n} \quad (k=1,1,\dots,n) \quad (4)$$

2) OR operation: Use logical OR ($\&$) obtain matrix \mathbf{C}_i ($i = 1, 2, \dots, m$) according to i-th transformed condition.

3) AND operation: Use logical AND ($\|$) obtain the output matrix F according to the transformed formula.

4) Test operation: Accept if the output matrix is a nonzero matrix; otherwise reject.

5) Readout: Use MATLAB function “`find()`”, to read out the subscripts of all nonzero elements, each subscript of which corresponds to a fixed truth assignment according to the following mapping:

$$i \mapsto B(i - 1) \quad (5)$$

where $i \in \{1, 2, 3, \dots, 2^n\}$ is a natural number and $B(i - 1) \in \{0, 1\}^n$ is a binary string.

3 Example

3.1 The Considered Boolean Formula

The Boolean formula of 20-variable instance of 3-SAT problem is transformed into a unique formula (see Fig 2), which consists of the corresponding logical matrices and operations. Note that this transformed expression can be viewed as an algorithm and computed directly in MATLAB space in parallel and deterministic manner. As a result of computation, it returns an output matrix that stands for the instance's solution space.

- A. $f = (\sim x_3 \text{ or } \sim x_{16} \text{ or } x_{18}) \text{ and } (x_5 \text{ or } x_{12} \text{ or } \sim x_9) \text{ and } (\sim x_{13} \text{ or } \sim x_2 \text{ or } x_{20}) \text{ and } (x_{12} \text{ or } \sim x_9 \text{ or } \sim x_5) \text{ and } (x_{19} \text{ or } \sim x_4 \text{ or } x_6) \text{ and } (x_9 \text{ or } x_{12} \text{ or } \sim x_3) \text{ and } (\sim x_1 \text{ or } x_4 \text{ or } \sim x_{11}) \text{ and } (x_{13} \text{ or } \sim x_2 \text{ or } \sim x_{19}) \text{ and } (x_5 \text{ or } x_{17} \text{ or } x_9) \text{ and } (x_{15} \text{ or } x_9 \text{ or } \sim x_{17}) \text{ and } (\sim x_5 \text{ or } \sim x_9 \text{ or } \sim x_{12}) \text{ and } (x_6 \text{ or } x_{11} \text{ or } x_4) \text{ and } (\sim x_{15} \text{ or } \sim x_{17} \text{ or } x_7) \text{ and } (\sim x_6 \text{ or } x_{19} \text{ or } x_{13}) \text{ and } (\sim x_{12} \text{ or } \sim x_9 \text{ or } x_5) \text{ and } (x_{12} \text{ or } x_1 \text{ or } x_{14}) \text{ and } (x_{20} \text{ or } x_3 \text{ or } x_2) \text{ and } (x_{10} \text{ or } \sim x_7 \text{ or } \sim x_8) \text{ and } (\sim x_5 \text{ or } x_9 \text{ or } \sim x_{12}) \text{ and } (x_{18} \text{ or } \sim x_{20} \text{ or } x_3) \text{ and } (\sim x_{10} \text{ or } \sim x_{18} \text{ or } \sim x_{16}) \text{ and } (x_1 \text{ or } \sim x_{11} \text{ or } \sim x_{14}) \text{ and } (x_8 \text{ or } \sim x_7 \text{ or } \sim x_{15}) \text{ and } (\sim x_8 \text{ or } x_{16} \text{ or } \sim x_{10})$
- B. $F = (\sim X3 | \sim X16 | X18) \& (X5 | X12 | \sim X9) \& (\sim X13 | \sim X2 | X20) \& (X12 | \sim X9 | \sim X5) \& (X19 | \sim X4 | X6) \& (X9 | X12 | \sim X5) \& (\sim X1 | X4 | \sim X11) \& (X13 | \sim X2 | \sim X19) \& (X5 | X17 | X9) \& (X15 | X9 | \sim X17) \& (\sim X5 | \sim X9 | \sim X12) \& (X6 | X11 | X4) \& (\sim X15 | \sim X17 | X7) \& (\sim X6 | X19 | X13) \& (\sim X12 | \sim X9 | X5) \& (X12 | X1 | X14) \& (X20 | X3 | X2) \& (X10 | \sim X7 | \sim X8) \& (\sim X5 | X9 | \sim X12) \& (X18 | \sim X20 | X3) \& (\sim X10 | \sim X18 | \sim X16) \& (X1 | \sim X11 | \sim X14) \& (X8 | \sim X7 | \sim X15) \& (\sim X8 | X16 | \sim X10)$

Fig. 2. A. The Boolean formula. B. The transformed formula.

3.2 Parallel Realization with MATLAB Procedure

We now roughly describe how to realize the parallel computation with MATLAB procedure.

Step 1: Generate all initial periodic matrices X_k according to the following procedure:

```
for k = 1:1:20
    X(k)=[ zeros(1,2^(k-1)), ones(1,2^(k-1)) ] % X(k) represent Xk
    while length(X(k))<(2^20)
        X(k)=[X(k),X(k)] % X(k) is created by replicating
    end
end
```

Since $\sim X_k$ and is the negation of X_k , it can be produced when needed via logical “NOT” and X_k .

Step 2: Use logical OR twice to return each $C(i)$, with respect to condition C_i according to the following procedure:

```
for i= 1:1:24
    C(i) = C_i % Each  $C_i$  is a known condition.
end
```

For example, $C(1)$ can be obtained according to the first transformed condition $C_1, (\sim X_3 \sim X_{16} | X_{18})$, where “ \sim ” denotes the logical “NOT”. Note that such a strategy might cost a great of space.

Step 3: Use logical AND (“ $\&$ ”) to return the final output matrix F according to the following procedure:

```
F=ones(1,2^20)
for i =1:1:24
    F=F& C(i)
    s(i)=sum(F) % s(i) is the number of nonzero elements in F.
end
```

this is, matrix F could be obtained by using logical AND ($\&$) 23 times. As expected, the output matrix F represents the instance’s solution space.

Step 4: Test the output matrix by using the following function:

```
if F ~=zeros(1,2^20) % symbol “~=” represent “not equal to”
```

Accept if it is not equal to the zero matrix; otherwise, reject. Since the instance was designed to a unique satisfying truth assignment beforehand, as expected, the final matrix F has only one nonzero element.

In general, the output matrix usually contains more than one nonzero element whose location tells us the corresponding satisfying truth assignment. If we wonder the satisfying truth assignments, we may continue the following operations called “Readout”.

Step 5 (Readout): Use MATLAB function “*find()*” to read out any nonzero elements.

```
N= find (F) % read out the subscripts of all nonzero elements
```

It is showed that the output N is 267281, this is, the sole satisfying truth assignment

$B(267280) = 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0$

The same result was obtained by [12] via DNA-based mean.

4 The Running Time and Space

The proposed method is suitable for a generalization of SAT problem in theory. Thus we consider an arbitrary well-defined Boolean formula with n variables and m clauses. We first take a rough measurement according to each proposed step and subsequently indicate the computational performance.

In first computational step, each X_k or ($k=1,\dots,n$) can be generated in n time and 2^n space (cells or bits). Hence it costs at most n^2 time and $n 2^n$ space.

In second and third step, there are at most $2nm$ logical OR “|”, AND “&” and NOT “~”, each of which is run only once, thus it needs at most $2nm$ time. In addition, m matrices C_i need $m 2^n$ space at all.

The fourth step is to test if the output is not equal to the zero matrix. It is expected to be quite parallel and only constant time and 2^n cells were needed.

At last, we hope that “*find()*” function needs linear time and 2^n space for computation.

Consequently, the algorithm runs in $O((m+n)n)$ time and $O((m+n)2^n)$ space.

5 Conclusion

An exhaustive molecular algorithm that is based on conventional computer is presented for solving arbitrary SAT problem. It needs $O((m+n)n)$ time and $O((m+n)2^n)$ space. Three basic matrix operators “AND”, “OR” and “NOT” are employed. A famous 20-variable instance of the three-satisfiability (3-SAT) problem is solved in parallel and deterministic manner.

It is noticed that the proposed algorithm requires exponential space for the gain of polynomial time. Hence, it means that the SAT problem is “hard” one on the other hand.

However, it is significant to solve effectively the NP-complete SAT problem by trading space for time, because the “space” is reusable for multiple computations. Moreover, it suggests that any a problem in NP can be effectively solved in the same way. Thus the proposed method has practical and theoretical implications. Theoretically, it can open the way for theories of computation that go beyond the Turing limit. On the practical level, it shows that an alternative computing model with exponentially bounded parallelism might be reasonable and realizable, moreover, such model can ultimately speed up the current computation.

References

- [1] Adleman L M. Molecular computation of solutions to combination problems. science,266(11):1021–1023, 1994.
- [2] Lipton R J. DNA solution of hard computational problems. Science,268(28):542–545, 1995.

- [3] Roweis S et al, A sticker based model of DNA computation. In Proceedings of the Second Annual Meeting on DNA-based computers, 1996.
- [4] Bach E et al. DNA model and algorithms for NP-complete problem. In Proceedings of the 11th Annual Conference on Structure in Complexity Theory, 290–299, 1996
- [5] Smith L M et al. A surface-based approach to DNA computation. Journal of computational biology, 5 (2): 255–267, 1998.
- [6] Ouyang Q et al, DNA solution of the maximal clique problem. Science, 278, 446–449, 1997.
- [7] Qinghua Liu et al. DNA computing on surfaces, Natural, 403(13), 175–178, 2000.
- [8] Sakamoto K et al. Molecular computation by DNA hairpin formation, Science, 288, 1223–1226, 2000
- [9] D. Faulhammer, A. R. Cukras, R. J. Lipton, L. F. Landweber, Proc. Natl. Acad. Sci. U.S.A. **97**, 1385 (2000)
- [10] D.Roos, K. Wagner. On the power of bio-computers. Information and Comoutation, 131:95–109, 1996
- [11] Max H Garzon et al. The bounded complexity of DNAcomputing. BioSystem, 52 63–72, 1999
- [12] Braich R S et al , Solution of a 20-Variable 3-SAT Problem on a DNA Computer Science, Vol. 296, 499–502, 2002.
- [13] The used notebook computer is a Intel Pentium III (TOSHIBA 1300), 1.0 GHz-M, 384 MB (RAM), 30G HD, installed Microsoft® window® XP Home Edition. MBTLAB 6.0 is used.

Scheduling Outages in Distributed Environments

Anthony Butler¹, Hema Sharda¹, and David Taniar²

¹ Department of Electrical Engineering and Computer System, RMIT,
Melbourne, Australia,
`{s9814033,hemas}@rmit.edu.au`

² School of Business Systems,
Monash University,
Melbourne, Australia,
`David.Taniar@infotech.monash.edu.au`

Abstract. This paper focuses on the problem of scheduling outages to computer systems in complex distributed environments. The interconnected nature of these systems makes scheduling global change very difficult and time consuming, with a high degree of error. In addressing this problem, we describe the constraints on successful outage scheduling. Using these constraints, it describes a solution using Constraint Logic Programming, which is able to deliver rapid schedules for complex architectures. The developed approach is then applied and tested to several real world problems to ascertain its effectiveness and performance.

1 Introduction

A significant challenge faced by large enterprises relates to the scheduling of upgrade activities and the outages associated with them on hosts within a large, *distributed environment*. Complex distributed environments consist of many nodes that are interconnected and dependent upon one another. Coulouris et al [1] defined such environments as “one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.” These interconnections can be transactional dependencies, physical connections, or logical dependencies. With the world having moved away from the centralised model of mainframe computing to a distributed model, there are now new challenges to be addressed in the maintenance of these environments. These challenges were not present in the old mainframe architectures of a single centralised host and many dumb terminals running from it.

Amongst the challenges of managing a complex distributed environment is the challenge of managing the upgrade of components in such a way as to minimise downtime and ensure that all the many dependencies between systems are maintained. The challenge of scheduling these activities is compounded by the need to take into account the various dependencies as well as resourcing constraints. Any schedule is constrained by the availability of the human resources with the skills and availability to perform the necessary upgrades.

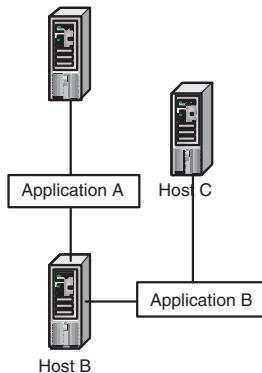


Fig. 1. Example Cluster

To attempt to solve these problems manually has proven to be a very time consuming and error-prone process. Furthermore, it is not scalable. As the number of hosts increase, so does the complexity, level of difficulty, and risk of error. This paper presents a solution to this complex problem by applying *Constraint Logic Programming* (CLP).

2 Migration Scheduling Problem: Background

Software migration is the managed upgrade or deployment of software to an organisation's infrastructure – either in its entirety or a large subset. Some of these applications may have hardware dependencies between them, which makes scheduling difficult. A shared disk between two hosts, or multiple applications installed on a single host means that host-wide changes affect different applications, which in turn affect their own dependencies.

We defined the problem as “scheduling” and not “timetabling” or “planning”. There has been considerable confusion amongst people outside of the scheduling community, as to the distinction between scheduling and planning. In short, planning is the process that decides what to do, whereas scheduling is the process which decides how and when to do the tasks specified within the plan [2]. Scheduling also involves resource allocation, wherein each activity is assigned to a resource(s). This is an important distinction. That said, there is still a planning component to the software migration effort. This planning consists of the determination as to what needs to be done, i.e. which software elements need to be upgraded and which hosts are affected.

3 Problem Definition

A detailed analysis of the problem domain was conducted in order to ascertain what constraints and dependencies existed that made the scheduling of outages in distributed environments difficult.

3.1 Using Clusters to Manage Complexity

In order to simplify the handling of a large number of hosts, it is useful to logically group hosts into clusters. The dependencies that bind hosts together into a cluster are: multiple applications on a common host, hosts sharing a common storage device, and up-stream and downstream transactional dependencies.

In Figure 1, Host A, Host B, and Host C are clustered because Host A and Host B share the same application and Host B and Host C share the same application.

The usefulness of clusters to scheduling is that it enables the scheduler to divide a large number of infrastructure components into human manageable chunks that can be dealt with in isolation. It is an example of the well-known principle of “divide and conquer”.

3.2 Environmental Constraints

Based on discussions with business managers at the telecommunications company, it was found that one of the most important considerations in scheduling software migrations is need to maintain required levels of uptime for their systems.

As a general rule, no organization should perform an upgrade to its production infrastructure or to production instances of its applications, until that same upgrade has been applied and tested against test infrastructure. Likewise, the development environment should be upgraded before test, whilst keeping in mind the need to maintain at least some test and development environments on the old versions of the software in order to be able to continue to support the existing production environment.

3.3 Summary of Rules and Constraints

Examining the dependencies between hosts and applications, we determined the following set of scheduling rules that need to be applied:

- 1 For a given application, hosts must be migrated in the order of development, test, then production. Additional environments, such as QA or Training, are inserted between test and production.
- 2 Hosts that share storage migrate at the same time if multiple hosts share storage, and that storage is used for bootdisk, then all hosts must be migrated together. If multiple hosts share storage, and that storage is not used for bootdisk, then the hosts should be scheduled to migrate together, though it is not necessary.
- 3 Hosts that send data, but do not receive any from the target, are unidirectional. Therefore, the sending system can be migrated without consideration of the recipients.
- 4 Hosts that receive data, but do not send any to the target, are unidirectional. If an outage of this data link would bring the host(s) down, then they should be migrated together.

- 5 Hosts that send and receive data from a host, should be scheduled to be migrated at the same time.
- 6 Hosts that are failover or backup for other Hosts. If it is necessary that a given host is failover or backup for another host, and it is necessary to maintain uptime of the application, then the two hosts must be scheduled to be migrated at different times.
- 7 Hosts that have infrastructure-level dependencies on other hosts, should be migrated at the same time as that other host, if possible.
- 8 Each machine has a system administrator who performs the migration. A system administrator can only work on one machine at any given time.

4 Methodology

There are many approaches to solving complex scheduling problems. Whilst they vary greatly in their detail, efficiency and effectiveness, they basically fall into one of two categories: *Artificial Intelligence* (AI) or *Operations Research* (OR) methods.

In distinguishing between them, OR methods tend to be based on mathematical models that aim to achieve high levels of efficiency in their models. OR methods also tended to focus on problems that could be expressed in purely mathematical terms. Whereas, AI approaches attempted to solve scheduling problems with more general problem solving paradigms. In summarising the benefits of each type of approach, the distinguishing quality between the two was the efficiency of OR methods versus the general applicability of AI methods.

In the OR domain, scheduling problems have typically been approached through dynamic programming, branch and bound algorithms, simulated annealing, and heuristic searches. Research in the Artificial Intelligence domain has centered around the application of genetic algorithms and constraint logic programming (CLP); with the latter being the most significant.

4.1 Constraint Logic Programming

Constraint Logic Programming (CLP) represents one of the most exciting sub-domains within AI research. As Freuder [3] opined, “Constraint Programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.”

A constraint was defined by Bartak[4] as a logical relation between several unknowns (or variables), each taking a value within a given domain. The constraint thus restricts the range of possible values that a variable can take. Humans use constraints constantly. For instance, we may say to someone that we will visit them between 10am and 11am. This is an example of a constraint. The simplest form of constraint in CLP is what is known as a finite-domain variable. A finite-domain is a set of values, and a finite-domain variable is a variable that is constrained to only take on values from the defined set.

```

Start::1..10
Name::[Fred,Albert,John]
alldifferent([X1,X2,X3])

```

The above example requires that Start must be a value between 1 and 10; that Name must be one of either Fred, Albert or John; and that X1, X2 and X3 are unique.

The solution to a constraint problem is therefore when there has been an assignment of values to all of the variables consistent with the problem constraints. A constraint problem can have many correct answers. For instance, the constraint $X < Y$, can be solved by any value of X and Y, provided X is less than Y. Therefore, X=1, X=2 and X=10,Y=100 are both correct answers.

A domain to which CLP has been successfully applied is that of scheduling. The vast majority of literature dealing with scheduling or timetabling utilise CLP techniques or their variants in order to solve the problem. Constraint Logic Programming has been utilised in solving varied scheduling problems, such as the scheduling of oil well activity [5], the scheduling of forest insecticide treatment [6], through to the scheduling of job shop style activities. It is generally accepted, within the literature, that CLP represents one of the most effective methods of solving complex scheduling problems. Whilst it seems there has been no research performed in scheduling software migrations, there has been a significant amount of effort expended in maintenance scheduling. The maintenance scheduling problem bares some similarity to the migration problem, because maintenance typically involves a resource being unavailable for a period of time (outage), and optimising for minimised downtime, cost and order of execution. One area where considerable research is being performed is in the area of Power Station Maintenance Scheduling.

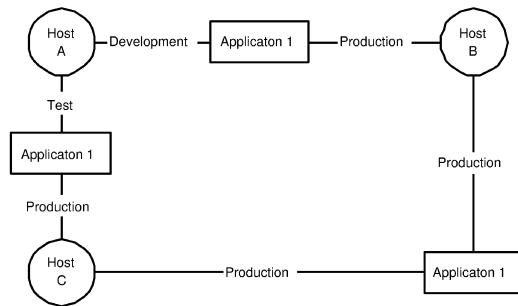
A typical power plant maintenance operation would involve between 5,000 and 45,000 activities that need to be scheduled. Each activity has explicit constraints that dictate the order of work, for instance, the powering down of one generator would be a prerequisite activity for any maintenance work. In that sense, there is a great parallel with the problem of migration scheduling.

In addition, the total outage for a plant must be as small as possible, ensuring ordering constraints are adhered to, and also that the safety constraints are enforced.

Alguire and Gomes [7] have also carried out extensive research in the application of CLP to power outage scheduling. They found that the current method of scheduling power station maintenance was a manual task, based heavily on the Critical Path Methodology (CPM).

They found that whilst this method worked quite adequately in generating schedules, it did not take into account the safety constraints. As such, after each schedule was generated, a Safety System Functional Assessment (SSFA) Model was applied to the schedule, to determine whether it met all of the required safety constraints.

CLP was selected as the approach due to the requirement that the resulting solutions be readily proveable as optimal solutions. A further reason was the

**Fig. 2.** Dependency Model

relative simplicity by which the problem could be defined using CLP constructs, relative to the use of genetic algorithms' fitness functions.

5 Applying CLP: Proposed Method

Figure 2 shows three hosts associated to one another by shared applications (Application 1). In order for a schedule to be built, it is necessary to determine the dependencies between each of the hosts.

The shared application resolves to a shared application dependency – where Host A has the test environment of Application 1 and Host C has the production environment.

5.1 Constraint Generation

For each host, identify the applications installed on it. Then, for each application, identify the other hosts that application is stored on and the environments. Referring to the rule that development must be scheduled prior to production, we can impose the following constraints:

Host B > Host A (Host B is scheduled after Host A); Host C > Host A (Host C is scheduled after Host A);

There is no constraint on the ordering of Host C versus Host B, as both are production hosts.

If there other dependencies that affect ordering, such as the existence of shared storage or transactional dependencies, then these would also be modelled during this phase.

5.2 Resourcing Constraints

Referring to the data that was sourced, each machine has a system administrator. Assuming that, for the example in Figure 3 each machine has the same system administrator.

Based on that, there is a constraint that only one machine can be migrated at any one time.

5.3 Solve

The resulting constraints are then fed into the solver. The solver uses the Eclipse 5.2 CLP environment, developed by Imperial College's Centre for Planning and Research Control (IC-PARC). The tool is freely available to academic users, and incorporates Application Programming Interfaces (APIs) to Visual Basic, Java, C++ and other languages.

CLP Problem Expression. First, the required libraries are loaded into Eclipse. In this case, it is a finite domain problem (FD), so it is only necessary to load the FD library. A finite domain problem is one where the range of possible values is limited to a defined (finite) domain.

```
:lib(fd).
```

Each machine being scheduled is represented by a data structure consisting of the following attributes:

1. When a machine will be migrated (start);
2. Prerequisite Machines (pre);
3. Machines that must be scheduled at the same time (same);
4. Machines that cannot be scheduled at the same time (prohib);
5. The Human Resource that must perform the migration (use);

```
:local struct(host(start,need,same,prohib,use)).
```

The solution space is then modelled, with all the hosts enumerated. As CLP (like prolog) works on the principle of defining goals, it is necessary to define a goal called solve:

```
solve(Solution):-
```

The “atoms” that make up the solution are then defined. In this case, it is the hosts that are being scheduled. Once the solution domain is defined, the structures that contain the dependencies for each host are built:

```
solve(Solution):- Solution=[HostA,HostB,HostC], HostA = host with
[need : [] ,same : [], prohib: [], use : Dev], HostB = host with
[need : [HostA] ,same : [], prohib: [],use : Dev], HostC = host
with [need : [HostA] ,same : [], prohib: [],use : Dev],
starts(Solution,L), L :: 1..3,
```

The following constraints specify that machines that have prerequisites must not be scheduled before those machines, that machines that must be scheduled at the same time as others are, and that machines that are forbidden to be scheduled together are not allowed.

```
% Precedence constraints

( foreach(host with [start:Si,need:Neededhosts], Solution) do (
foreach(host with [start:Sj], Neededhosts), param(Si) do Si #> Sj)
),

% Select machines that have to be migrated at the same time
( foreach(host with [start:Si,same:Samehosts], Solution) do (
foreach(host with [start:Sj], Samehosts), param(Si) do Si #= Sj)
),

% Select machines that cannot be migrated at the same time
( foreach(host with [start:Si,prohib:Prohibhosts], Solution) do (
foreach(host with [start:Sj], Prohibhosts), param(Si) do Si #\= Sj)
),

```

The remaining code imposes several generic constraints that apply to all problems in this domain:

1. The constraint that no two machines can be upgraded simultaneously if both require the same resource.
2. The constraint that the schedule must be the optimum solution (lowest cost);

```
% minimize cost
min_max(( no_overlaps(Solution), labeling(L)),X), nl, write(L),nl.
max([X],X). max([X|L],X):-max(L,Y), Y =< X.
max([X|L],Y):-max(L,Y), Y > X.

starts([],[]). starts([host with [start : X]
|L],[X|L2]):-starts(L,L2).

no_overlaps(hosts) :- ( fromto(hosts, [host0|hosts0], hosts0, [])
do ( foreach(host1,hosts0), param(host0) do host0 = host with
[start:S0, use:R0], host1 = host with [start:S1, use:R1], ( R0 ==
R1 -> no_overlap(S0, S1) ; true ) ) ).

no_overlap(Si,Sj) :- Sj #> Si. no_overlap(Si,Sj) :- Si #>
Sj.
```

5.4 Analysis of Output

The solver outputs the following cryptic message:

[1, 2, 3]

Table 1. Performance Results

<i>Number of Hosts</i>	<i>Time to Solve (secs)</i>
18	0.01
36	0.03
54	0.13
108	2.83
162	19.71
216	91
270	138.75

This represents, [Host A = 1, Host B = 2, Host C = 3], where the number denotes the “slot” where that activity should be performed.

This data was then entered into Microsoft Project in order to produce a Gantt chart. Once in that format, real dates are assigned to the notional slots, and the project plan is used as the basis for the execution phase.

6 Conclusion

In assessing the effectiveness of the approach, it was necessary that the system perform better than the manual method, with a lower margin of error, and with an output that can be readily proven and explaining to people not literate with scheduling algorithms.

6.1 Performance Tests

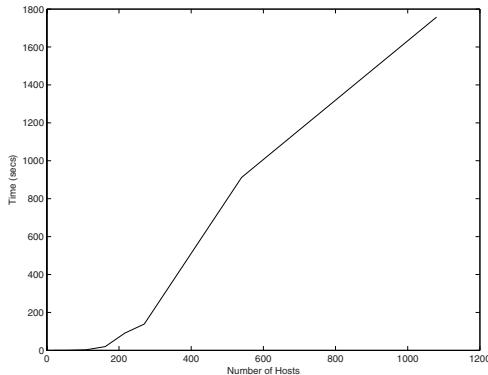
In order to ascertain the scalability of this approach, a series of tests were conducted with varying numbers of hosts and constraints. One of the key success factors to the usability of the developed approach is that it performs faster than manual methods and that it is scaleable.

Tests were conducted on infrastructure models of 18, 36, 54, 108, 162, 216 and 270 hosts, using a 1.4gb Athlon processor, with 512mb Ram. 1 documents the results.

Based on these results, some projections can be made as to performance against larger numbers of hosts. As Figure 3 shows, overall performance is very good, with even 2,000 hosts taking approximately only 53 minutes (3197 seconds) of CPU time.

It is expected that most clusters would still be under 100 hosts, which would place performance well within an acceptable range.

In order to develop some baseline for measuring the effectiveness of this approach versus manual approaches, we surveyed a large IT outsourcing company that had previously attempted a number of large scale migrations. They estimated that for each cluster (of approximately 20 hosts), the time to build a schedule was 4 hours, with an additional 3–4 hours of rework once all clusters

**Fig. 3.** Performance

were completed. This represents a time, per host, of 12 minutes as compared to less than 1/100th of a second per host using the method outlined here.

7 Further Work

Further research can be performed in the development of some concept of “soft constraints”. A “soft constraint” is a constraint that can be broken in certain circumstances. For instance, we may specify that we want all development hosts to migrate before production, however if it is not possible, then as long as at least one development host has been migrated then production can be done.

There is a well-known issue in CLP where a problem can be over-constrained – meaning that it is essentially impossible to solve without breaking constraints. Through the utilisation of soft constraints, it is possible to put priorities on constraints – separating them into mandatory and preferred.

There are significant opportunities for automating and streamlining the process of schedule generation. Of particular interest, would be to automate the interface between the Eclipse solver and the configuration database. As eclipse has a Java-based interface, a GUI-based front end could be built that enables the data to be dynamically sourced from a configuration database and sent to the solver in a single pass.

The scheduling of activities that take place during an outage provides opportunities for further research. For instance, in some cases, there is a need for cross-discipline support, such as from both DBA’s and applications support staff. Being able to handle multiple resources and the activities performed within each outage would improve the ability to plan more complicated migration efforts, some of which may require multiple outages.

Whilst an assessment was made regarding the suitability of several scheduling approaches, there is still some research that can be conducted into the usefulness of genetic algorithms (GAs) to solving these problems.

The challenge in a GA-based approach is constructing a fitness function to determine a “good schedule”. An interesting aside might be to investigate how user feedback can be incorporated into the model to capture some of the constraints that are not necessarily easy to define. For instance, in the authors reading of the literature regarding power plant scheduling, it mentioned consistently the role that “gut feel” plays in building good schedules. Using some of the existing AI technologies, it may be possible to capture these abstract constraints via user feedback on the quality of generated schedules.

References

1. Coulouris, G.e.a.: *Distributed Systems Concepts and Design*. third edn. Addison-Wesley (2001)
2. Bartak, R.: Towards mixing planning and scheduling. In: *Proceedings of CPDC2000 Workshop*. (2000)
3. Freuder, E.C.: Synthesizing constraint expressions. *Communications of the ACM* **21** (1978) 958–966
4. Bartak, R.: Constraint programming: In pursuit of the holy grail. In: *Proceedings of Workshop of Doctoral Students'99*. (1999)
5. Johansen, B.e.a.: Well activity scheduling – an application of constraint reasoning. In: *Proceedings of Practical Applications of Constraint Technology (PACT)*. (1997)
6. J. Adhikary, G.H., Misund, G.: Constraint technology applied to forest treatment scheduling. In: *Proceeding of Practical Application of Constraint Technology*. (1997) 15–34
7. Alguire, K., Gomes, C.: Roman – an application of advanced technology to outage management. In: *Proceedings of 1996 Dual-Use Technologies and Applications Conference*. (1996)
8. Gomes, C.: Automatic scheduling of outages in nuclear power plants with time windows. Rome Lab Technical Report (1996)
9. Langdon, W.: Scheduling maintenance of electrical power transmission networks using genetic programming. *Research Note RN/96/49* (1996)
10. Louis, R., e.a.: Software agents activities. In: *Proceedings of 1st International Conference on Enterprise Information Systems*. (1999)
11. Mason: *Genetic Algorithms and Job Scheduling*. PhD thesis, Department of Engineering (1992)

An Improved Parallel Algorithm for Certain Toeplitz Cyclic Tridiagonal Systems on Distributed-Memory Multicomputer

Xuebo Zhang¹, Zhigang Luo², and Xiaomei Li^{1,2}

¹ College of Equipment Command and Technology,
Beijing 101416, PRC

² National Laboratory for Parallel and Distributed Processing,
Changsha 410073, PRC

Abstract. Based on Luo's parallel algorithm [4] for certain Toeplitz cyclic tridiagonal systems on distributed-memory multicomputer, we present an improved algorithm. Its communication mechanism is simple and redundant computing is small for solving massively systems. The numerical experiments show that the parallel efficiency of the improved algorithm is higher than Luo's algorithm [4].

1 Introduction

In massively science and engineering computing problems, when solving the implicit finite difference equations derived from the linear first order hyperbolic equation, e.g. transport equation, under a variety of boundary conditions, we will solve certain Toeplitz cyclic tridiagonal systems as follows again and again:

$$\begin{bmatrix} a_0 & a_1 & & -a_1 \\ -a_1 & a_0 & \ddots & \\ & \ddots & \ddots & a_1 \\ a_1 & & -a_1 & a_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \quad (1)$$

where a_0 and a_1 are real numbers ($a_0 > 0$), n is the size of the systems.

For solving systems (1), in 1977 Buckley presented an algorithm [1] derived from ordinary Gauss elimination method; in 1980 Evans presented a fast algorithm [2], which is the fastest serial algorithm at present, based on the factorization of the coefficient matrix; in 1989 Zichun Zhao presented a parallel algorithm [3] fit for vector machine; in 2001 Zhigang Luo and Xiaomei Li presented a parallel algorithm [4], which is a more perfect algorithm than others on distributed-memory multicomputer. But Zhigang Luo's algorithm [4] can be improved further when the size of systems (1) is big enough.

Based on Luo's parallel algorithm [4], we present an improved algorithm according to the characters of massively systems. In this paper, we will discuss the deduction and description of this algorithm, the error analysis and the performance analysis; and then we will give the data of the numerical experiments.

2 Algorithms

2.1 Algorithm Deduction

Do some operations as follows according to literature [2]:

$$\alpha = \frac{2a_1}{a_0 + \sqrt{a_0^2 + 4a_1^2}} \quad \beta = \frac{1 - \alpha^2}{a_0} \quad b_k = \beta * f_k \quad k = 1, 2, \dots, n \quad (2)$$

Systems (1) will become equivalent cyclic tridiagonal systems:

$$Ax = b \quad (3)$$

$$\text{where } A = \begin{bmatrix} 1 - \alpha^2 & \alpha & & -\alpha \\ -\alpha & 1 - \alpha^2 & \ddots & \\ & \ddots & \ddots & \alpha \\ \alpha & & -\alpha & 1 - \alpha^2 \end{bmatrix}_{n \times n}, \quad x = (x_1, x_2, \dots, x_n)^T, \quad b = (b_1, b_2, \dots, b_n)^T.$$

The coefficient matrix A can be factorized to matrix Q and matrix R : $A = QR$,

$$\text{where } Q = \begin{bmatrix} 1 & & & -\alpha \\ -\alpha & 1 & & \\ & \ddots & \ddots & \\ & & -\alpha & 1 \end{bmatrix}_{n \times n}, \quad R = \begin{bmatrix} 1 & \alpha & & \\ & 1 & \ddots & \\ & & \ddots & \alpha \\ \alpha & & & 1 \end{bmatrix}_{n \times n}.$$

So solving systems (3) will become solving two systems as follows:

$$Qv = b \quad (4)$$

$$Rx = v \quad (5)$$

where $v = (v_1, v_2, \dots, v_n)^T$.

Since $a_0 > 0 \Rightarrow |\alpha| < 1$, we can prove that systems (4) are diseased systems when α is close to 1 very much, and that systems (5) are diseased systems when α is close to -1 very much. So, when $|\alpha| \approx 1 (|a_0| \approx 0)$ we should not adopt the method presented in this paper for solving systems (3), the α in this area is not discussed here.

Now we discuss the parallel solver for systems (4) on distributed-memory multicomputer. We suppose p is the number of the processors and distribute equation $im+1, im+2, \dots, (i+1)m$ of systems (4) to processor P_i ($i = 0, 1, \dots, p-1$), where $m = \lceil n/p \rceil$, we consider $m = n/p$ so as to say easily. The equations processed by processor P_i ($i = 0, 1, \dots, p-1$) are:

$$\begin{bmatrix} -\alpha & 1 & & & \\ -\alpha & 1 & & & \\ \ddots & \ddots & \ddots & & \\ & & & -\alpha & 1 \end{bmatrix}_{m \times (m+1)} \begin{bmatrix} v_{im} \\ v_{im+1} \\ \vdots \\ v_{(i+1)m} \end{bmatrix}_{(m+1) \times 1} = \begin{bmatrix} b_{im+1} \\ b_{im+2} \\ \vdots \\ b_{(i+1)m} \end{bmatrix}_{m \times 1} \quad (6)$$

where $v_0 = v_n$.

If each processor computes the product of α^{m-k} and equation k of systems (6) and adds the product to equation m, systems (6) will become:

$$\begin{bmatrix} -\alpha & 1 & & & \\ -\alpha & 1 & & & \\ \ddots & \ddots & \ddots & & \\ -\alpha^m & 0 & 1 \end{bmatrix}_{m \times (m+1)} \begin{bmatrix} v_{im} \\ v_{im+1} \\ \vdots \\ v_{(i+1)m} \end{bmatrix}_{(m+1) \times 1} = \begin{bmatrix} b_{im+1} \\ b_{im+2} \\ \vdots \\ s_{i+1} \end{bmatrix}_{m \times 1} \quad (7)$$

where $s_{i+1} = \sum_{k=1}^m \alpha^{m-k} b_{im+k}$ ($i = 0, 1, \dots, p-1$).

Since $|\alpha| < 1$, formula $\lim_{n \rightarrow \infty} \alpha^n = \lim_{n \rightarrow \infty} \alpha^{\frac{n}{p}} = 0$ must be right. In massively science and engineering computing, if the size of the coefficient matrix of systems (1) is big enough to make $-\alpha^m \approx 0$, processor P_i ($i = 0, 1, \dots, p-1$) will get the value of $v_{(i+1)m}$ by formula $v_{(i+1)m} \approx s_{i+1}$ of systems (7). We compute s_{i+1} by the following formula (8) to avoid computing $\alpha^{m-1}, \alpha^{m-2}, \dots, \alpha^2$.

$$s_{i+1} = b_{(i+1)m} + \alpha(b_{(i+1)m-1} + \alpha(\dots + \alpha(b_{im+2} + \alpha b_{im+1})\dots)) \quad (8)$$

After sending the value of $v_{(i+1)m}$ to processor $P_{(i+1) \bmod p}$, processor P_i ($i = 0, 1, \dots, p-1$) will have the values of v_{im} and $v_{(i+1)m}$. We also can get the formula as follows from systems (6):

$$v_{im+k} = b_{im+k} + \alpha v_{im+k-1} \quad (k = 1, 2, \dots, m-1) \quad (9)$$

Each processor will get the partial roots of systems (4) by computing the above formula. So the parallel solver for systems (4) is finished.

Because the parallel solver for systems (5) is similar with systems (4), we need not say it again.

2.2 Algorithm Description

1. We distribute a_0, a_1 and $f_{im+1}, f_{im+2}, \dots, f_{im+m}$ of systems (1) to processor P_i ($i = 0, 1, \dots, p-1$);
2. Processor P_i ($i = 0, 1, \dots, p-1$) computes α, β and b_{im+k} ($k = 1, 2, \dots, m$) according to the formula (2). If $\alpha^m \approx 0$, we continue with the following steps, otherwise we solve systems (1) according to algorithm [4];
3. Processor P_i ($i = 0, 1, \dots, p-1$) computes s_{i+1} according to formula (8);
4. Processor P_i ($i = 0, 1, \dots, p-1$) gets the value of $v_{(i+1)m}$ by formula $v_{(i+1)m} \approx s_{i+1}$ and sends it to processor $P_{(i+1) \bmod p}$;
5. Processor P_i ($i = 0, 1, \dots, p-1$) computes v_{im+k} ($k = 1, 2, 3, \dots, m-1$) according to formula (9). So we will get the total roots of systems (4);
6. Solving systems (5) according to steps 3~5, we will get the roots of systems (5), which are also the roots of the primary Toeplitz cyclic tridiagonal systems (1).

2.3 Error Analysis

We write systems (7) as follows so as to say easily,

$$Q^* \hat{v}_i = b_i \quad i = 0, 1, \dots, p-1 \quad (10)$$

$$\text{Supposing } M = \begin{bmatrix} -\alpha & 1 & & & \\ & -\alpha & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \end{bmatrix}_{m \times (m+1)}, N = \begin{bmatrix} & & & \\ & & & \\ & & & \\ & & & \\ & & & \\ & & & \alpha^m \end{bmatrix}_{m \times (m+1)},$$

$Q^* = M - N$. So systems (10) will become:

$$(M - N) \hat{v}_i = b_i \quad i = 0, 1, \dots, p-1 \quad (11)$$

We suppose that $\hat{v}_i^* = (v_{im}^*, v_{im+1}^*, \dots, v_{im+m}^*)^T$ are the roots of systems (10), that $\hat{v}_i = (v_{im}, v_{im+1}, \dots, v_{im+m})^T$ are the roots of systems $M \hat{v}_i = b_i$, and that the errors

are $\Delta\hat{v}_i = \hat{v}_i^* - \hat{v}_i = (\Delta v_{im}, \Delta v_{im+1}, \dots, \Delta v_{im+m})^T$, where $v_0^* = v_n^*$, $v_0 = v_n$, $\Delta v_0 = \Delta v_n$. We take \hat{v}_i^* into systems (11) and take \hat{v}_i into systems $M\hat{v}_i = b_i$, then

$$(M - N)\hat{v}_i^* = M\hat{v}_i \Rightarrow M(\hat{v}_i^* - \hat{v}_i) = N\hat{v}_i^* \\ \Rightarrow M\Delta\hat{v}_i = N\hat{v}_i^* \quad i = 0, 1, \dots, p-1 \quad (12)$$

Writing formula (12) in the matrix form:

$$\begin{bmatrix} -\alpha & 1 & & & \\ & -\alpha & 1 & & \\ & & \ddots & \ddots & \\ & & & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta v_{im} \\ \Delta v_{im+1} \\ \vdots \\ \Delta v_{(i+1)m} \end{bmatrix} = \begin{bmatrix} v_{im}^* \\ v_{im+1}^* \\ \vdots \\ v_{(i+1)m}^* \end{bmatrix} \\ \Rightarrow \begin{bmatrix} \Delta v_{im} \\ \Delta v_{im+1} \\ \vdots \\ \Delta v_{(i+1)m-1} \\ \Delta v_{(i+1)m} \end{bmatrix} = \begin{bmatrix} \Delta v_{im} \\ \alpha\Delta v_{im} \\ \vdots \\ \alpha\Delta v_{(i+1)m-2} \\ \alpha^m v_{im}^* \end{bmatrix} = \begin{bmatrix} \alpha^m v_{(i-1)m}^* \\ \alpha^{m+1} v_{(i-1)m}^* \\ \vdots \\ \alpha^{2m-1} v_{(i-1)m}^* \\ \alpha^m v_{im}^* \end{bmatrix} \quad i = 0, 1, \dots, p-1 \quad (13)$$

where $v_{-m}^* = v_{(p-1)m}^*$.

If we take the precise roots and the approximate roots of systems (4) into systems (5) respectively, two systems as follows will appear in subsequent computing.

$$R^*\hat{x}_i = \hat{v}_i^* \quad i = 0, 1, \dots, p-1 \quad (14)$$

$$R^*\hat{x}_i = \hat{v}_i \quad i = 0, 1, \dots, p-1 \quad (15)$$

$$\text{where } R^* = \begin{bmatrix} 1 & 0 & & -(-\alpha)^m \\ & 1 & \alpha & \\ & & \ddots & \ddots \\ & & & 1 & \alpha \end{bmatrix}_{m \times (m+1)}, \quad \hat{x}_i = (x_{im+1}, x_{im+2}, \dots, x_{(i+1)m+1})^T,$$

$$\hat{v}_i^* = \left(\sum_{k=1}^m (-\alpha)^{k-1} v_{im+k}^*, v_{im+2}^*, \dots, v_{im+m}^* \right)^T, \quad \hat{v}_i = \left(\sum_{k=1}^m (-\alpha)^{k-1} v_{im+k}, v_{im+2}, \dots, v_{im+m} \right)^T.$$

$$\text{Supposing } P = \begin{bmatrix} 1 & 0 & & & \\ & 1 & \alpha & & \\ & & \ddots & \ddots & \\ & & & 1 & \alpha \end{bmatrix}_{m \times (m+1)}, E = \begin{bmatrix} & & (-\alpha)^m \\ & & \\ & & \vdots \\ & & (-\alpha)^m \end{bmatrix}_{m \times (m+1)},$$

$R^* = P - E$. So systems (14) will become:

$$(P - E)\hat{x}_i = \hat{v}_i^* \quad i = 0, 1, \dots, p-1 \quad (16)$$

We suppose that $\hat{x}_i^* = (x_{im+1}^*, x_{im+2}^*, \dots, x_{im+m+1}^*)^T$ are the roots of systems (14), that $\hat{x}_i = (x_{im+1}, x_{im+2}, \dots, x_{im+m+1})^T$ are the roots of systems $P\hat{x}_i = \hat{v}_i$, and that the errors are

$$\Delta\hat{x}_i = \hat{x}_i^* - \hat{x}_i = (\Delta x_{im+1}, \Delta x_{im+2}, \dots, \Delta x_{im+m+1})^T,$$

where $x_1^* = x_{n+1}^*$, $x_1 = x_{n+1}$, $\Delta x_1 = \Delta x_{n+1}$. We take \hat{x}_i^* into systems (16) and take \hat{x}_i into systems $P\hat{x}_i = \hat{v}_i$, then

$$\begin{aligned} (P - E)\hat{x}_i^* - P\hat{x}_i &= \hat{v}_i^* - \hat{v}_i \\ \Rightarrow P(\hat{x}_i^* - \hat{x}_i) &= \Delta\hat{v}_i + E\hat{x}_i^* \\ \Rightarrow P\Delta\hat{x}_i &= \Delta\hat{v}_i + E\hat{x}_i^* \quad i = 0, 1, \dots, p-1 \end{aligned} \quad (17)$$

where $\Delta\hat{v}_i = \hat{v}_i^* - \hat{v}_i = (\sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k}, \Delta v_{im+2}, \dots, \Delta v_{im+m})^T$.

Writing formula (17) in the matrix form:

$$\begin{bmatrix} 1 & 0 & & & \\ & 1 & \alpha & & \\ & & \ddots & \ddots & \\ & & & 1 & \alpha \end{bmatrix} \begin{bmatrix} \Delta x_{im+1} \\ \Delta x_{im+2} \\ \vdots \\ \Delta x_{im+m+1} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} \\ \Delta v_{im+2} \\ \vdots \\ \Delta v_{im+m} \end{bmatrix} + \begin{bmatrix} (-\alpha)^m \\ x_{im+1}^* \\ x_{im+2}^* \\ \vdots \\ x_{im+m+1}^* \end{bmatrix}$$

$$\begin{aligned}
& \Rightarrow \begin{bmatrix} \Delta x_{im+1} \\ \Delta x_{im+2} \\ \vdots \\ \Delta x_{im+m} \\ \Delta x_{im+m+1} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} + (-\alpha)^m x_{im+m+1}^* \\ -\alpha \Delta x_{im+3} \\ \vdots \\ -\alpha \Delta x_{im+m+1} \\ \Delta x_{im+m+1} \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} + (-\alpha)^m x_{im+m+1}^* \\ (\sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} + (-\alpha)^m x_{im+2m+1}^*)(-\alpha)^{m-1} \\ \vdots \\ (\sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} + (-\alpha)^m x_{im+2m+1}^*)(-\alpha) \\ \sum_{k=1}^m (-\alpha)^{k-1} \Delta v_{im+k} + (-\alpha)^m x_{im+2m+1}^* \end{bmatrix} \\
& = \begin{bmatrix} (-1)^{m-1} \alpha^{2m-1} v_{im}^* + \alpha^{m-1} v_{(i-1)m}^* \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} + (-\alpha)^m x_{im+m+1}^* \\ ((-1)^{m-1} \alpha^{2m-1} v_{im+m}^* + \alpha^{m-1} v_{im}^* \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} + (-\alpha)^m x_{im+2m+1}^*)(-\alpha)^{m-1} \\ \vdots \\ ((-1)^{m-1} \alpha^{2m-1} v_{im+m}^* + \alpha^{m-1} v_{im}^* \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} + (-\alpha)^m x_{im+2m+1}^*)(-\alpha) \\ (-1)^{m-1} \alpha^{2m-1} v_{im+m}^* + \alpha^{m-1} v_{im}^* \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} + (-\alpha)^m x_{im+2m+1}^* \end{bmatrix} \quad (18)
\end{aligned}$$

where $x_{m+1}^* = x_{(p+1)m+1}^*$, $\Delta v_{n+k} = \Delta v_k$, $k = 1, 2, \dots, m$, $i = 0, 1, \dots, p-1$.

$$\lim_{m \rightarrow \infty} \Delta x_{im+j} = \lim_{m \rightarrow \infty} [((-1)^{m-1} \alpha^{2m-1} v_{im+m}^* + \alpha^{m-1} v_{im}^* \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} + (-\alpha)^m x_{im+2m+1}^*)(-\alpha)^{m-j+1}] = 0 \quad (19)$$

$$\text{where } \lim_{m \rightarrow \infty} \sum_{k=1}^{m-1} (-1)^{k-1} \alpha^{2k} = \frac{\alpha^2}{1+\alpha^2}, j = 2, \dots, m, m+1 \quad i = 0, 1, \dots, p-1.$$

We know that $\lim_{n \rightarrow \infty} \|\Delta x\|_\infty = 0$ is right according to formula (19). If given the absolute error bound δ , we can consider formula $\|\Delta x\|_\infty \leq \delta$ will be right for certain as long as ensuring the size of the coefficient matrix of systems (1) is big enough.

3 Performance Analysis

In the performance analysis of the improved algorithm, we suppose that the basic operation unit is $flop$, that each arithmetic operation is $1 flop$, that τ is the time of a machine finishing $1 flop$, that t_0 is the additional overhead when each processor

sends or receives a piece of message, and that t_1 is the time of each data passing on the network.

Theorem 1: We suppose the computing time complexity of the improved algorithm is $T_p(n)$ and the absolute speedup is $S_p(n)$, then

$$T_p(n) = 9m\tau + 2(2t_0 + t_1) + c \quad (20)$$

where c is a constant.

$$\lim_{n \rightarrow \infty} S_p(n) = p.$$

Proof: We accumulate the computing time so as to get the formula (20).

We know that the computing time complexity of the fastest serial algorithm [2] for solving systems (1) is $T_1(n) = (9n + 40)\tau$, so

$$S_p(n) = \frac{T_1(n)}{T_p(n)} = \frac{(9n + 40)\tau}{9m\tau + 2(2t_0 + t_1) + c} \quad (21)$$

when $n \rightarrow \infty$, $S_p(n) \rightarrow p$. Finish

4 Numerical Experiments

We get the following data by doing numerical experiments on the clusters including six personal computers, whose CPU are P4 1.7G. The parallel program is written with the Fortran language and has run on MPI (Message Passing Interface) environment. In the following tables, n is the size of the coefficient matrix of systems (1), p is the number of the processors, Sp is the relative speedup, Ep is the parallel efficiency ($Ep = \frac{Sp}{p} \times 100\%$), and the time unit is second.

Table 1. Comparison between the improved algorithm and algorithm [4] on the running time

n	$p = 1$		$p = 2$		$p = 4$		$p = 6$	
	Im algo	Algo [4]						
4E5	12.820	12.863	6.521	6.630	3.324	4.178	2.270	3.499
6E5	19.264	19.324	9.714	9.819	4.925	6.412	3.327	4.977
8E5	25.712	25.693	13.014	13.070	6.565	7.709	4.402	6.241
1E6	32.073	32.080	16.252	16.276	8.167	8.855	5.474	6.509

Table 2. Relative speedup and parallel efficiency of the improved algorithm

n	$p = 2$		$p = 4$		$p = 6$	
	Sp	Ep	Sp	Ep	Sp	Ep
4E5	1.98	99%	3.88	97%	5.65	94.7%
6E5	1.98	99%	3.91	97.8%	5.79	96.5%
8E5	1.96	98%	3.92	98%	5.84	97.3%
1E6	1.97	98.5%	3.92	98%	5.86	97.7%

The data of the numerical experiments shows that the improved algorithm is superior to algorithm [4] on the running time. The most important reason is that the communication times of the improved algorithm are about $1/(2p - 1)$ of algorithm [4], where the communication times of algorithm [4] are $4p - 2$. At the same time, the improved algorithm has a perfect relative speedup and parallel efficiency.

5 Conclusions

This improved algorithm is based on the factorization of the coefficient matrix of systems (3). Through the right estimation and neglect to α^m , we make the communication mechanism simple and the redundant computing small. The numerical experiments show that the parallel efficiency of the improved algorithm is high.

References

- Buckley A. On the solution of certain skew symmetric linear systems. SIAM Journal of Numerical Analysis, 1977, 14: 566–570
- Evans D J. On the solution of certain Toeplitz tridiagonal linear systems. SIAM Journal of Numerical Analysis, 1980, 17(5): 675–680
- Zichun Zhao, Xiaomei Li. Two parallel algorithms for tridiagonal Toeplitz systems. In: Proc China 1st National Conference on parallel algorithms in Science Computing, Beijing. 1989;125–130
- Zhigang Luo, Xiaomei Li. A parallel solver for certain cyclic tridiagonal Toeplitz systems on distributed-memory multicomputer. Journal of Computer Research and Development, 2001. 38(2): 228–233

Generic Programming for Scientific Computing in C++, JavaTM, and C#

Jens Gerlach and Joachim Kneis

Fraunhofer Institute for Computer Architecture and Software Technology (FIRST),
Kekuléstraße 7, 12489 Berlin, Germany

Abstract. Parameterized types (*generics*) have been announced for the JavaTM and C# programming languages. In this paper, we evaluate these extensions with respect to the realm of scientific computing and compare them with C++ templates. At the heart of our comparison are the *set* and *relation* classes from the *Janus* framework which provides abstraction for the efficient representation of meshes, sparse graphs and associated matrices. As an example, we compare the performance of the Bellman-Ford single source shortest path algorithm when using parameterized Janus data structures implemented in C++, JavaTM, and C#. Our measurements suggest that both JavaTM and C# generics introduce only little run time overhead when compared with non-parameterized implementations. With respect to scientific application, C# generics have the advantage of allowing *value types* (including builtin types) as parameters of generic classes and methods.

1 Introduction

In this paper we demonstrate the use of the recently proposed JavaTM and .NET generic classes and methods for the implementation of data structures for the field of scientific computing. We compare their expressiveness with that of C++ templates and evaluate the performance of the corresponding C++, JavaTM, and C# data structures.

C++ has a long history of supporting generic programming through its template mechanism and in particular through the container and algorithm framework of its standard library. Moreover, numerous C++ template libraries, for example the Boost[1] collection, have shown the flexibility, expressiveness, efficiency of the C++ template mechanism.

On the other hand, the JavaTM and .NET platforms did not include parameterized polymorphism in their original releases despite the drawbacks a later introduction could have. Recently, however, support for generic classes and methods have been announced for the JavaTM[2] and C#[3] programming languages. The next major JavaTM release 1.5 will include generic classes and a patch for release 1.4.1 is already available. Regarding .NET, a release has not yet been announced. However, there is a patch for the Shared Source Common Language Infrastructure that supports generics.

There are only a few syntactic differences between JavaTM and C# generics. More important are how their introduction have affected the respective platforms. The generic types of JavaTM have been designed in such a way that there were no changes to the JavaTM virtual machine (JVM) were necessary. This is an advantage for customers that rely on the stability of the run time environment. A drawback, in particular for scientific computing applications, is that builtin types (`int` or `double`) cannot be parameters of generic classes or methods. In .NET, on the contrary, support for generics has been put into the Common Language Runtime (CLR) and its Intermediate Language (IL), so that parameterized types can be used by other .NET languages. As a result, C# generics accept not only classes and interfaces but also *value types* as type parameters. Value types cover builtin integer and floating point types and *structs* and are for efficiency reasons of great importance in scientific computing. Moreover, .NET (and therefore C#) generics support exact runtime types and dynamic linking.

The *Janus* framework[4] is one example that shows the benefits of parameterized polymorphism for the field of scientific computing. *Janus* comprises a conceptual framework and C++ template library for parallel problems that involve grids, meshes, sparse graphs and associated matrices. Section 2 provides an overview of the main abstractions of *Janus*. Programs written in *Janus* can achieve a level of performance that competes favorably with the PETSc[5] applications and the Boost Graph Library[6].

This work attempts to give an evaluation of parameterized types in JavaTM and C# with respect to the application domain of *Janus*. Since performance is a central issues in scientific computing, we are particularly interested in whether there is a significant run time overhead associated when using generics.

As an example of a simple scientific application, we consider the Bellman-Ford single source shortest path algorithm (see Section 4) that can be implemented very easily using one of *Janus* data transfer operations. Performance measurements using parameterized *Janus* data structures implemented in C++, JavaTM, and C# and non-parameterized data structure are given in Section 5. Finally, we summarize our results and draw conclusions.

2 The Janus Framework

Janus is a framework of abstractions that aims at the efficient representation of meshes grids, sparse graphs and associated sparse matrices. *Janus* has been implemented as a C++ template library that provide (distributed) data structures and parallel data-transfer operations.

Janus distinguishes between *sets* and *relations*. Sets are collections of objects, whereas relations describe data dependencies of set elements. In the following subsections we briefly describe the most important features of *Janus* sets and relations.

2.1 The Set Concept

A hierarchy of interfaces describes the most important methods of Janus sets. A very important feature of Janus sets is that there is a clear separation between insertions and access operations. Although this looks like a major restriction, it fits the usage pattern of many scientific applications[4] where a mesh, grid, or graph is used only when it has been completely created. Moreover, these enable the efficient representation of complex and potentially distributed meshes or graphs. Most interfaces are parameterized over the element type of the set.

The interface *Set Insertion* comprises methods `void insert(ElementType)` for the insertion of elements and for the closure of a set data structure. Elements can be inserted into a set only before `void freeze()` has been called.

The interface *Set Access* contains methods for querying the elements of a set. These methods may only be invoked after the initialization has been completed. A Janus set provides random access to its elements through the `ElementType access(int)` method. The index of an element in the set can be queried with the `int position(ElementType)` method. It is this clear separation into insertion and access phases that allows Janus sets to offer features both of *random access* containers and *associative* containers of the C++ standard library.

Simple, for example, regular structures of scientific application can be completely described at initialization time. An example is a rectangular grid where there is no need to insert individual elements into it. The interface *One-Phase Set* expresses that that there is only an access phase but no insertion phase.

Complex (irregular or dynamic) spatial structures, such as a finite element triangulation, usually cannot be completely described without insertion of individual elements. They must be represented by *two-phase* data structures that provide both insertion and access operations. Note, however, that access operations may only be performed after `freeze` has been called.

Both for the *One-Phase Set* and *Two-Phase Set* interface types there are generalizations for distributed computing environments. These interfaces provide descriptors for the distribution of the set elements over a group of processes. Distributed two-phase domains have an additional `insert_at` method whose second argument specifies the process on which the inserted element shall be mapped.

Data Associated with Sets. In scientific computations (numerical) data are often associated with the elements of a set S . An example would be a temperature field $t : S \rightarrow \mathbb{R}$ on the set S .

Since every element $s \in S$ has an index with respect to S we can represent both S and t as sequences $\{S_0, \dots, S_{n-1}\}$, respectively $\{t_0, \dots, t_{n-1}\}$ through assigning $t(S_i)$ the index i . Thus, the real-valued function t can be represented by a one-dimensional array of a floating point type. For most programming languages this is important in order to achieve high performance.

2.2 The Relation Concept

The main purpose of a relation $R \subset X \times Y$ between two Janus sets is the description of data dependencies. Janus relations exploit the fact that there is a one-to-one correspondence between set elements and their positions. A relation is described by pairs of integers instead of pairs of elements. Thus, a relation does not need to know the exact type of its sets. For relations there is the same clear semantic separation into initialization and access phases as in the case of Janus sets.

A finite difference stencil on a rectangular grid is a good example that only needs the *One-Phase Relation* interface. Irregular relations as they occur in finite element methods are best described by two-phase relations.

Note also that the interface *Two-Phase Relation* does not offer a `insert_at` method as in the case for sets. There is no need for it because the mapping of relation elements is determined by the distribution of the elements of its first factor.

Data Associated with Relations. A relation R represents a sparse graph. An example for data that are associated with the elements of a relation are the coefficients $c : R \rightarrow \mathbb{R}$ of a matrix. As in the case of data that are associated with a set, these coefficients can be represented by a one-dimensional array. The `position` method assigns indices to the values of c .

Data Parallel Operations. Janus relations provide a rich collection of generic data parallel operations that generalize (sparse) matrix-vector multiplication. Most of these methods have already been described[4] carefully, thus we concentrate on new features.

As an example, we consider the semantics of the methods `pull_matrix` and `pull_matrix2`. Let $R \subset X \times Y$ be a relation between two sets. Let $c : R \rightarrow \mathbb{F}$ coefficients on R and s and t two \mathbb{F} -valued function on X , respectively Y . Let furthermore \otimes and \oplus two binary operations on \mathbb{F} and init an element of \mathbb{F} . The semantics of `pull_matrix`($R, c, s, t, \text{init}, \otimes, \oplus$) is defined as

$$t_i \leftarrow \text{init} \oplus \left(\bigoplus_{\{j|(i,j) \in R\}} c_{ij} \otimes s_j \right) \quad \forall i \in X.$$

If $u : X \rightarrow \mathbb{F}$ denotes additional data associated with the first factor and f a *binary* function on \mathbb{F} then the semantics of `pull_matrix2`($R, c, s, u, t, f, \text{init}, \otimes, \oplus$) is

$$t_i \leftarrow f \left(\text{init} \oplus \left(\bigoplus_{\{j|(i,j) \in R\}} c_{ij} \otimes s_j \right), u_i \right), \quad (1)$$

for all $i \in X$. In Section 4 we will use this general `pull_matrix2` method to implement the Bellman-Ford shortest path algorithm.

Janus abstraction are well suited for data-parallel programs on both shared-memory and distributed-memory architectures.

- The `pull` and `push` methods have parallel semantics and are highly customizable.
- Using `pull` and `push` methods, users do not have to program *loops* to express data-parallel operations. They can hide both message-passing primitives and shared-memory directives[7].
- Two-phase data structures support the efficient mapping of data onto different processors because the mapping of elements can be efficiently performed before they are accessed.

3 Generic Janus Implementations in JavaTM and C#

The Janus framework[4] has been originally designed for and implemented in C++. In the following we discuss differences between generic programming in C++ on the hand and JavaTM and C# on the other side.

A crucial factor for JavaTM generics is that (due to the backward compatibility) builtin types are not permitted as type parameters. As we will show in Section 5 this poses performance problems in scientific applications.

Further, differences between the JavaTM and C++ implementation of Janus are caused by different handling of generic type parameters. For example, a template parameter representing a function must either be a pointer to a function or a function object that has overloaded the ()-operator. However, typically this is only checked when the actual type parameter is supplied that is a client uses the template in its code.

Both JavaTM and C# check such constraints when the generic class is compiled. Constraints on types can be expressed by indicating that `Type` implements a certain interface or extends a class. The constraining interfaces and classes are referred to as *bounds*. If no bounds are given, the parameter type is treated like `java.lang.Object`. Therefore, a direct transfer of the code to JavaTM fails.

Regarding a binary function parameter, one solution is to require that the parameter `BinaryFunction` is bound to the generic interface `Function2` that defines a binary method `call`, as shown here:

```
interface Function2<Type> {
    Type call(Type, Type);
}
```

The Janus implementation in C# follows straightforward from the JavaTM implementation. This shows the syntactic similarities between these two languages. A major difference is that in C# not only *reference types* (e.g. classes and interfaces) but also *value types* which include *structs* and builtin types can be supplied as type parameters.

As a downside, it has to be pointed out that the currently available implementation of .NET generics as does not generic generic collection classes. In this respect, the implementation of JavaTM generics is more mature.

4 An Application Case Study

The Bellman-Ford algorithm[8] solves the single-source shortest-paths problem. For a given $G = (N, E)$ and a weight function $w : E \rightarrow \mathbb{F}$, this problem consists in finding the shortest path from a fixed source node $s \in N$ to every node of N .

The Bellman-Ford algorithm associates two attributes with each node $v \in N$, namely, the shortest path estimate $d : N \rightarrow R$ and the predecessor $\pi : N \rightarrow N$ in a path from s to v . For the sake of brevity, we consider here only the computation of d . Initially, $d(v)$ is set to ∞ for all nodes except for s where it is set to 0. The essential component of the Bellman-Ford algorithm is the *relaxation* of the shortest path estimate d over all edges and which reads

```
foreach  $u \in N$ 
  foreach  $v : (u, v) \in E$ 
     $d(u) \leftarrow \min(d(u), w(u, v) + d(v)).$ 
```

As with other relaxation algorithms, such as Gauß-Seidel[9], the result of a relaxation sweep depends on the order in which the relaxation is performed. Nevertheless, at most $|N| \cdot |E|$ relaxation steps must be performed to compute the shortest paths and their weights or to indicate that there are negative cycles that are reachable from s . However, depending on the graph and the weights the algorithm can be terminated when no further changes occur.

In order to prepare a Janus implementation relaxation is performed for all nodes and all edges that belong to this node.

It can be written as

$$d_i \leftarrow \min \left(\min \left(\infty, \min_{\{j|(i,j) \in E\}} w_{ij} + d_j \right), d_i \right)$$

for all indices of N . This shows that the relaxation can be implemented using Formula 1 when the following parameters are supplied

`pull_matrix2($E, w, d, d, d, \min, \infty, +, \min$)`

The generic C++ implementation of Bellman-Ford is shown in Listing 1. It uses the `pull_matrix2` method. The data for d and w can be represented by objects of type `std::vector<double>`.

We wrote corresponding generic Bellman-Ford methods also in JavaTM and C#. As in the case of C++, they call the Janus `pull_matrix2` method to perform the relaxation.

5 Performance Measurements

In this section, we analyze the performance of the different implementations of the Bellman-Ford shortest path algorithm. First, we compare the Janus C++ implementation with the Bellman-Ford algorithm of the Boost Graph Library[6].

```

template<typename Edges,typename Nodes,typename Values,typename Equal>
void bellman_ford(const Edges& e, const Nodes& n, const Values& w,
                  Values& d,typename Values::value_type inf, Equal eql) {
    typedef typename Values::value_type V;
    Values y(d);
    for(size_t it = 0; it < n.size(); ++it) {
        e.pull_matrix2(w, d, d, d,jns::min<V>(),
                      inf,std::plus<V>(),jns::min<V>());
        if(std::equal(d.begin(), d.end(), y.begin(), eql)) break;
        y = d;
    }
}

```

Listing 1: Generic Janus implementation in Bellman-Ford algorithm in C++

Table 1. Size of graphs and specification of computers used for performance measurements

	Nodes	Edges		Machine 1	Machine 2
Graph 1	78,877	476,354	CPU	P4 2.0 Ghz	P4 2.4 GHz
Graph 2	567,150	3,395,108	RAM	512 MB	512 MB
			OS	Windows XP SP1	Linux Suse 8.1

We will see that although Janus is not a dedicated graph library such as the BGL it can handle graph problems in a very efficient way.

The second part is a survey on the performance of the JavaTM and C# versions of Janus. Here, we compare the runtime cost when using parameterized classes and methods. A remarkable result that a non-generic JavaTM implementation can achieve almost the same performance as C++ template code.

We use two large sparse test graphs that originate from finite element triangulations.

The size of the graphs are shown in left of Table 1. We measure only the execution time of the Bellman-Ford algorithm. The time to build up the set and relation objects is not considered. Table 1 shows also details of the hardware and used operating systems of our test machines.

5.1 C++ Performance

For our test runs we used a slightly modified version of our generic Bellman-Ford implementation shown in Listing 1. The modifications consists of a an additional call to determine the predecessor (or parent) nodes when the relaxation has come to an end.

In order to represent weights and distance arrays we used the `std::vector` container parameterized with `double`.

The `g++` (version 3.2) was used as C++ compiler with optimization options `-O3 -funroll-loops`. On the windows machine, the `g++` is part of the `cygwin` run time environment.

Table 2. C++ performance on Machine 1 (left) and Machine 2

Graphs	Janus	BGL
Graph 1	6.0s	11.9s
Graph 2	132.9s	262.7s

Graphs	Janus	BGL
Graph 1	4.9 s	10.8s
Graph 2	111.0s	230.4s

Table 3. JavaTM performance on Machine 1 (left) and Machine 2

Graphs	Generic Double double		
Graph 1	24.5s	22.9s	6.9s
Graph 2	632.8s	481.8s	156.2s

Graphs	Generic Double double		
Graph 1	33.1s	30.1s	10.3s
Graph 2	699.4s	636.6s	240.2s

Table 2 shows that the Janus implementation of Bellman-Ford compares favorably with that of BGL. The Janus implementation used the `relation` class which is well-suited for general sparse relations. In the case of BGL, we resorted to the `edge_list` data structure because for our graphs it provides better times than the `adjacency_list` data structure.

The problem of the BGL implementation is that in each relaxation step the parent of a node is determined. Janus does this only when there are no further changes. Note that Listing 1 does *not* show the corresponding code. However, our measurements contain the related times. The determination of the parent node is roughly as intensive as the minimization over an edge. This relativizes the factor of more than 2 when comparing Janus and BGL run-times.

5.2 JavaTM Performance

Two questions arise when investigating the performance of the JavaTM implementation of the generic Janus data structures. Firstly, how fast is Janus in JavaTM compared to Janus in C++. Secondly, how fast are JavaTM generics compared to non-generic JavaTM data structures. In context of scientific computing, the answer to the second question has to take into account that JavaTM generics cannot parameterized with builtin types.

We therefore measured *three* JavaTM implementations of the Bellman-Ford algorithm. A generic implementation that was parameterized with `Double` wrapper class to represent the weights and distances on the graph. A non-generic implementation using `Double` and an implementation that utilized the builtin `double` type. The three implementations differ only in the treatment of the graph weights. There are no differences in the internal representation of the sparse graph. As in the case of the C++ `relation` class the CRS format was used.

Both on machine 1 and machine 2 Sun's JavaTM release 1.4.1_01 has been used.

There are several remarkable observations when analyzing these tables and comparing them with the C++ performance measurements.

Table 4. C# performance on Machine 1

Graphs	Generic double (SSCLI)	double (SSCLI)	double (VS .NET)
Graph 1	136.9s	134.8s	13.0s
Graph 2	2,569.1s	2,529.8s	258.1s

1. The performance overhead between a generic and non-generic implementation (both using `Double`) is with less than 1.4 acceptably small.
2. However, using `double` instead of `Double` in the non-generic implementation accelerates execution time by a factor of 3.
3. On machine 1 the JavaTM implementation (using `double`) is only 40% slower than the Janus C++ implementation.

This shows that JavaTM can achieve a high level of performance when builtin types are used. However, it cannot leverage this performance because its generics accept only class types as parameters.

5.3 C# Performance

Since C# generics accept builtin types as parameters we only compare a generic implementation (working `double` arrays) with a non-generic version that has the `double` hard coded into its implementation. We used the Microsoft's shared source implementation of CLI (SSCLI) because it contains an implementation of the announced C# (.NET) generics. The measurements (see Table 4) have only been performed on Machine 1 because the SSCLI was not directly available for Linux systems.

One problem of SSCLI is that is *very* slow. The last column in Table 4 shows the time of the non-generic implementation when executed under the standard .NET runtime. It is faster by a factor of 10.

Table 4 shows that there are no noticeable differences between the generic and non-generic implementations on the SSCLI. The non-generic version is by a factor of 2 slower than the JavaTM `double` version (see Table 3). We think that this is related to the greater maturity of the JavaTM just-in-time compiler.

6 Conclusions

This paper has described the use of JavaTM and C# generics to implement parameterized data structures and algorithms of the Janus scientific computing framework. We investigated the expressiveness and performance of both type systems at hand of a generic implementation of the Bellman-Ford single source shortest path algorithm. A C++ implementation of Janus that heavily uses templates has served as a reference implementation.

The parameterized type systems of both languages are sufficiently expressive to fulfill the requirements of Janus. A major drawback of the JavaTM approach

with respect to efficiency is that builtin types are not permitted as type parameters. This is regrettable because a JavaTM implementation of the Bellman-Ford algorithm that utilizes builtin types performs almost as well as a corresponding generic Janus implementation in C++.

As in the case of JavaTM, generics in C# do not seem to introduce significant run time overhead. However, a final evaluation can only be delivered if a more mature implementation of C# generics is available.

References

1. Boost.org. *Home Page of the Boost Libraries*. <http://www.boost.org>.
2. G. Bracha, N. Cohen, C. Kemper, S. Marx, M. Odersky, S. Panitz, D. Stoutamire, K. Thorup, and P. Wadler. Adding generics to the java programming language: Participant draft specification. April 2001.
<http://jcp.org/aboutJava/communityprocess/review/jsr014/index.html>.
3. D. Syme and A. Kennedy. The design and implementation of generics for the .net common language runtime. *ACM SIGPLAN Notices*, 36(5), May 2001.
4. Jens Gerlach. *Domain Engineering and Generic Programming for Parallel Scientific Computing*. Ph.D. dissertation, Technical University of Berlin, Germany, July 2002.
http://edocs.tu-berlin.de/diss/2002/gerlach_jens.htm.
5. S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith. Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries Programming. In A. M. Bruaset E. Arge and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhauser Press, 1997.
6. Jeremy Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library*. Addison Wesley, 2002.
7. J. Gerlach, Z.Y. Jiang, and H.P. Pohl. Integrating OpenMP into a Generic Framework for Parallel Scientific Computing. In *Workshop on OpenMP Applications and Tools*, Lecture Notes in Computer Science, Purdue University, West Lafayette, Indiana, USA, July 2001. Springer-Verlag.
8. T.H. Cormen, Leiserson C.E., and R.L. Rivest. *Introduction to Algorithms*. The MIT Press, 1998.
9. W. Hackbusch. *Iterative Solution of Large Sparse Systems of Equations*. Springer, 1994.

A General Metric of Load Balancing in δ -Range*

Xuejun Yang, Huadong Dai, Yuhua Tang, and Xiaodong Yi

School of Computer Science,
National University of Defence Technology,
China
Xjyang@nudt.edu.cn
Hddai@163.net

Abstract. In this paper we present a new general metric of load balancing in δ -range. This metric is different from traditional metrics since it introduces an argument δ to embody the practical circumstances when multiple jobs are running in a parallel computer system. Different demands on resources by various types of jobs and the cost to balance the loads are taken into consideration when we design this metric. As a result, this metric is more practicable and applicable in a real large scale multi-task parallel system. Some properties and two algorithms aiming at the cluster environment are also described in this paper.

1 Introduction

Load balancing is one of the most critical elements in improving both resource utilization and system performance in parallel computers. A great deal of research work has been done in this field and lots of algorithms have been presented. In summary, almost all of the algorithms follow two steps. Firstly, to adopt a metric to measure the loads of the system, and then to choose a policy to balance the loads of the system.

As the basis, a metric must be used to justify whether the loads of a system are balanced or not. Actually, traditional metrics can be seemed as equi-loads metrics. That is, if and only if each node of the system has absolutely equal loads, could the system be thought as load-balanced. But in practical systems, it is nearly impossible for a parallel computer to reach such an absolutely balanced state. Things will be even worse if various classes of jobs are submitted in a heterogeneous system. As a result, the algorithms based on these metrics seldom work as efficiently as expected.

With our researches, we find that the following three factors should be taken into consideration when choosing a practicable and applicable load balancing metric:

- The nodes in a real parallel computer are seldom homogeneous, so we should not draw a conclusion of whether the loads of each node are balanced or not just by measuring the absolute loads of each node.

* This research was supported by the National Natural Science Foundation (No.69825104) and 863 Project (No. 2002AA1Z2101) of China.

- In reality, the jobs can be divided into different classes. And the jobs running in different nodes often fall into different classes and have different attributes. We should not use the same method to measure the loads on each node.
- Load balancing policies, such as task scheduling and process migration, will cause additional cost. If the cost exceeds the benefits brought by load balancing, the policies should not be applied.

Based on the above three considerations, this paper presents a new metric of load balancing in δ -range, which is of general properties. A dynamically calculated δ is introduced to describe the concept of relative load balancing corresponding to absolute balancing in traditional metrics. As a result, this metric is able to overcome the limitations of traditional equi-loads metrics. Instructed by this metric, we also design two load balancing algorithms for cluster systems.

The rest of the paper is organized as follows. We present a brief overview of related work in Section 2. We then describe the metric in δ -range in Section 3. Section 4 gives two load balancing algorithms for cluster systems. Finally, we conclude in Section 5.

2 Related Work

Several commercial load balancing systems, along with their own metrics, developed by academic organizations or corporations, have won their fame by successful applications in practical parallel computing systems. Examples include CONDOR[1] in the University of Wisconsin, Load Leveler[2] by IBM, LSF[3] from Platform, and CODINE/GRD, etc. But the absence of a general metric in theory limits the further populization of these systems. As we can see, to some job classes with particular properties, these systems may improve performance distinctly, while to the job classes showing different properties, they could hardly achieve gratifying effects.

Other load balancing systems aim at particular applications such as web service and distributed filesystems[4][5]. In these systems loads measure goals such as network bandwidth, CPU utilization, memory utilization, and so on, are used as metrics to measure the loads. To other kinds of applications, however, these metrics would be meaningless.

As to load balancing algorithms, traditional static algorithms assign tasks statically to balance the loads[6]. Although heuristic[7] and genetic algorithms[8] have been adopted, they could hardly get ideal speedup with the neglect of complexity and dynamic diversity of worksets and environments.

On the other hand, policies such as self scheduling[9], guided self scheduling[10] and factoring based policies[11] are used to dynamically balance system loads. A major shortcoming of these algorithms is that they are too complex and have huge overheads on system scheduling, as well as lacking of generality.

Some researches have been done to combine the two. Plata and Winckler have both presented a hybrid (static plus dynamic) algorithm[12][13], but the application field is still limited. In comparison, the metric presented in this paper is more general and the algorithms based on the metric with the illumination of Control Theory are relatively new attempts in this field.

3 The General Metric of Load Balancing in δ -Range

In this section, we describe the general metric of load balancing in δ -range through a series of definitions.

The static loads of a node in a parallel system can be considered as the summary of the loads of all the tasks running on it. Suppose the system is constructed by n nodes and let SL_i be the static loads of node i , where $i = 1, 2, \dots, n$. And we define

$\overline{SL} = \frac{\sum_{i=1}^n SL_i}{n}$ to describe average loads of all the nodes in the system. For simplicity, we use the concept of *Static Load Balancing Degree* of the system to describe $\frac{\max_{i=1}^n \{SL_i\} - \overline{SL}}{\max_{i=1}^n \{SL_i\}}$, which is denoted by symbol $\bar{\delta}$. And we call δ the *Static Load Balancing Control Degree* of the system. Then we come to the following definition.

[Definition 1] Static Load Balancing in δ -Range

If a system meets the following two conditions, we call it in a static load balanced state in δ -range.

Cond. 1 For a certain $\delta \in [0,1)$, we have $\bar{\delta} \leq \delta$; and

Cond. 2 For any $\delta' \in [0,1)$ and $\bar{\delta} \leq \delta'$, there must be $\delta \leq \delta'$.

Obviously, when $\bar{\delta} = 0$, the loads of all nodes in the system are absolutely equal. While when $\bar{\delta}$ approaches 1, the traditional thought is that the system's loads are severely unbalanced. But according to *Definition 1*, whether a system is balanced or not is decided by δ , where δ is chosen in accordance with the actual environment. For example, when a user submits to node j an interactive job with heavy loads, while at this moment other nodes in the system are all free, it is apparent that $\max_{i=1}^n \{SL_i\} = SL_j$ and the *Static Load Balancing Degree* of the system, $\bar{\delta}$, approaches 1. As a result, we choose a δ to approach 1, and we think the system is in a relatively balanced state, that is, a state in δ -range. Simply asserting that the system is unbalanced based on the absolute value of δ is a mistake.

For the same reason, when processes are migrated and the system reaches a more balanced state, if the cost caused by migration exceeds the benefits of load balancing, we should also choose a bigger δ so as to avoid the useless migration.

The implication of *Definition 1* is that, if we choose a proper δ according to the practical environment of the system and the properties of user jobs, it is more sensible and meaningful to justify whether the system is balanced in δ -range.

Definition 1 denotes the average loads of the nodes in the system as \overline{SL} . It also uses the subtraction by the average loads to the maximum loads as the load balancing metric. It's suitable to homogeneous parallel systems, i.e. each node with the same processing capability. For heterogeneous systems, we must take different processing capability of each node into consideration and extensions should be made to *Definition 1*.

For a heterogeneous system, let A_j be the processing capability of node j . For any two nodes, i and j , if node i has more processing capability than node j , then $A_i > A_j$.

Let $\mu_j = \frac{A_j}{\max_{i=1}^n \{A_i\}}$, where $\mu_j \leq 1$, be the relative processing capability factor of node j . Then average loads can be defined as $\overline{SL} = \frac{\sum_{i=1}^n SL_i}{\sum_{i=1}^n \mu_i}$. As *Definition 1*, we use

the concept of *Static Load Balancing Degree* of the heterogeneous system, $\bar{\delta}$, to denote $\frac{\max_{i=1}^n \left\{ \frac{SL_i}{\mu_i} \right\} - \overline{SL}}{\max_{i=1}^n \left\{ \frac{SL_i}{\mu_i} \right\}}$. And we call δ the *Static Load Balancing Control Degree* of the heterogeneous system. Based on this, we can give the definition of static load balancing in δ -range for heterogeneous systems.

[Definition 2] Static Load Balancing in δ -Range for Heterogeneous Systems

If a heterogeneous system meets the following two conditions, we call it in a static load balanced state in δ -range.

Cond. 1 For a certain $\delta \in [0,1]$, we have $\bar{\delta} \leq \delta$; and

Cond. 2 For any $\delta' \in [0,1)$ and $\bar{\delta} \leq \delta'$, there must be $\delta \leq \delta'$.

The difference between *Definition 1* and *2* is that the latter introduces a coefficient factor μ to measure a node's processing capability, making it more compatible to heterogeneous systems. If the system is homogeneous, we have $\mu_1 = \mu_2 = \dots = \mu_n = 1$, then *Definition 2* will withdraw to *Definition 1*. In the rest of this paper, unless pointed out explicitly, the concepts refer to *Definition 2*.

In a multi-user and multi-task parallel computer system, the loads of every node is dynamically varied, e.g., the dynamic submission of the jobs and the occurrence of system events. So if we just consider the static information of application programs and the topology of parallel systems, it's difficult to accurately predicate the system's loads during the jobs' running. The metric of static load balancing in δ -range provides only the pre-feed static information. It is deficient in measuring the dynamic loads of the running system. In order to work out a complete evaluation of the load information and control the system, we further define the metric of dynamic load balancing in δ -range.

Let node j 's current loads at time t or node j 's average loads between a segment of time Δt be denoted as DL_j , which is able to reflect the relative computing and processing capability of node j . And as previous definitions, we use the concept of

Dynamic Load Balancing Degree of the system, $\bar{\delta}$, to denote $1 - \frac{\sum_{i=1}^n DL_i}{n \cdot \max_{i=1}^n \{DL_i\}}$.

And we call δ the *Dynamic Load Balancing Control Degree* of the corresponding system. The following definition is given out.

[Definition 3] Dynamic Load Balancing in δ -Range

If a system meets the following two conditions, we call it in a dynamic load balanced state in δ -range.

Cond. 1 For a certain $\delta \in [0,1]$, we have $\bar{\delta} \leq \delta$; and

Cond. 2 For any $\delta' \in [0,1]$ and $\bar{\delta} \leq \delta'$, there must be $\delta \leq \delta'$.

In *Definition 3*, we use the ratio of the system's average dynamic loads and maximum dynamic loads to evaluate the balanced state of the system. The meaning of δ 's value is consistent with that in the static load balancing instance, which reflects the practical system's authorization degree of load balancing.

Combining *Definition 2* with *Definition 3*, we can compose the concept of metric of load balancing in the sense of the whole system.

For a certain $\delta \in [0,1]$, if the system is both static load balanced in δ -range and dynamic load balanced in δ -range, we think the system is load balanced in δ -range in the sense of the whole system. In fact, the *Static Load Balancing Control Degree* and the *Dynamic Load Balancing Control Degree* can hardly be equal. So we define the metric of load balancing in the sense of the whole system as follows.

[Definition 4] Load Balancing in δ -Range in the Sense of the System

Let δ be a 2-tuple, that is, $\delta = \langle \delta_1, \delta_2 \rangle$. Here both δ_1 and δ_2 are certain values predefined in the region $[0,1]$. If the system is both in a static load balanced state in δ_1 -range and in a dynamic load balanced state in δ_2 -range, we say it is in a load balanced state in δ -range in the sense of the system.

Definition 4 considers the metric of both static and dynamic load balancing, i.e., it combines the static system information and predicated information of the jobs before they are running in the system, with the dynamically acquired or feed-back information during the running of the jobs. Therefore this metric can accurately measure the balanced state of a practical multi-task parallel computer system. At the same time, since the corresponding load measuring goals and the value of δ could be ascertained in combination according to the actual environment, this metric has preferable applicability and practicability.

In addition, in a practical system, the submission of the jobs is dynamic. The attributes of different classes of jobs may demand different load measuring goals and metrics. As we all know, common jobs can be divided into interactive jobs, batch jobs, real-time jobs, parallel jobs, and so on. Different kinds of jobs may require different schedule policies and demand different resources. In order to embody these different requirements, we give a further definition in the sense of job classification.

First, we classify the jobs that users may submit. Let J be a kind of job classification, $J = \{J_1, J_2, \dots, J_k\}$, where k means the number of job classes, and J_1, J_2, \dots, J_k each means a set of unique class of jobs. Then we have *Definition 5*.

[Definition 5] Load Balancing in δ -Range in the Sense of Job Classification

Let δ be a vector containing k elements, that is, $\delta = (\delta_1, \delta_2, \dots, \delta_k)^T$. Where $\delta_1, \delta_2, \dots, \delta_k$ are 2-tuples defined in *Definition 4*, e.g., $\delta_i = \langle \delta_{i1}, \delta_{i2} \rangle$, $i = 1, 2, \dots, k$, and $\delta_{i1}, \delta_{i2}, \dots, \delta_{k1}, \delta_{k2}$ each means a certain value predefined in range $[0,1]$. If for any i , when we just consider the running of the i th class of jobs J_i , the system is in a load balanced state in δ_i -range in the sense of the system, then we say that the system is in a load balanced state in δ -range in the sense of job classification.

With *Definition 5*, we extend our metric to a general environment based on the classification of jobs. Different requirements to various resources by distinct classes of jobs lead to different annotations of load balancing metrics. When a batch job containing lots of computing tasks are submitted to a node, the loads of the node may seemed severely overloaded from the point of view of other kinds of jobs, and the system may be thought serious unbalanced. But from the point of view of this batch job, the system is balanced. Formally, there may be a big gap between the values of δ prescribed by different classes of jobs. In other words, let $\delta = (\delta_1, \delta_2, \dots, \delta_k)^T$ and J_i be a special set of jobs, $\forall j$, where $j = 1, 2, \dots, k$, and $j \neq i$, then even if we have $\delta_{j1} \ll \delta_{ii}$, $\delta_{j2} \ll \delta_{i2}$, $\lim \delta_{ji} = 1$ and $\lim \delta_{ii} = 1$, according to *Definition 5*, the system is in a load balanced state in δ -range in the sense of job classification. Hence, with the general metric of load balancing in δ -range, we avoid some misguidances by the absolutely-balanced traditional metrics.

4 Load Balancing Algorithms Based on the General Metric

4.1 Some Properties

Before we describe our algorithms, we firstly probe into two properties of the general metric of load balancing in δ -range. For simplicity, we just consider the instance of static load balancing in a homogeneous system, and the number of processors in each node is supposed to be just 1.

[Property 1] When $\delta = 0$, the system gains the linear Speedup.

Proof: When $\delta = 0$, we have $\max_{i=1}^n \{SL_i\} = \overline{SL}$, and it is obvious that

$$SL_1 = SL_2 = \dots = SL_n.$$

That is, the static loads on each node are absolutely equal. Since the nodes in the system are homogeneous, when the jobs are running on each node, the CPU utilization ratio and CPU free time of each node are the same. In formality,

$$T_{CPU}^1 = T_{CPU}^2 = \dots = T_{CPU}^n, \text{ and } T_{free}^1 = T_{free}^2 = \dots = T_{free}^n,$$

where T_{CPU}^i and T_{free}^i , $i = 1, 2, \dots, n$, denote CPU busy time and CPU free time of node i separately.

Since the total loads in the system should be $n \bullet \overline{SL}$, when the jobs are running on a single node, the CPU busy time and CPU free time should separately be

$$T_{CPU} = \sum_{i=1}^n T_{CPU}^i = n \bullet T_{CPU}^1, \text{ and } T_{free} = \sum_{i=1}^n T_{free}^i = n \bullet T_{free}^1.$$

According to the *Speedup Formula*, the Speedup of the system should be

$$S = \frac{T_{CPU} + T_{free}}{T_{CPU}^1 + T_{free}^1} = \frac{n \bullet (T_{CPU}^1 + T_{free}^1)}{T_{CPU}^1 + T_{free}^1} = n$$

So, when $\delta = 0$, the system gains the linear Speedup. \square

[Property 2] In the case that a system's *Load Balancing Degree* is transformed from $\bar{\delta}_1$ to $\bar{\delta}_2$, $\bar{\delta}_1 > \bar{\delta}_2$, suppose the cost caused by the state transformation is h , and

$\bar{\delta}_2 < \bar{\delta}_1 < \bar{\delta}_2 + h$. Then after the state transformation, the Speedup of the system falls. In other words, $S_1 > S_2$.

Proof: The proof is divided into two steps.

Step 1. Without considering the transformation cost h . Since $\bar{\delta}_1 > \bar{\delta}_2$, we have

$$\left(\frac{\max_{i=1}^n \{SL_i\} - \overline{SL}}{\max_{i=1}^n \{SL_i\}} \right)_1 > \left(\frac{\max_{i=1}^n \{SL_i\} - \overline{SL}}{\max_{i=1}^n \{SL_i\}} \right)_2 \text{ and can deduce } \left(\max_{i=1}^n \{SL_i\} \right)_1 > \left(\max_{i=1}^n \{SL_i\} \right)_2,$$

which means the maximum node loads in state 1 is higher than that in state 2. In the ideal homogeneous system where each CPU has the same processing capability, we can easily find that the completion time of the jobs is longer in state 1 than in state 2. That is, $t_1 > t_2$. As to the Speedup of the two states, it is obvious that $S_1 < S_2$.

Step 2. Take the cost h into consideration and we have $\bar{\delta}_1 < \bar{\delta}_2 + h$. Since the completion time of the jobs in each state without considering the cost is t_1 and t_2 separately, when considering the cost h , the completion time of state 2 should be $t_2 + t_h$, where t_h means additional cost on time caused by the state transformation, e.g., the preemption and migration of tasks. Because $\bar{\delta}_1 < \bar{\delta}_2 + h$, with the consistent relationship among $\bar{\delta}$, maximum node loads and t , we can deduce that $t_1 < t_2 + t_h$. As we can see, the Speedup of a system is at an inversed ratio to the completion time of jobs, that is, $S \propto \frac{1}{t}$. So from the formula $t_1 < t_2 + t_h$, apparently, $S_1 > S_2$ must hold. \square

The implication of Property 2 is that, although the *Load Balancing Degree* of the system decreases and the system seems more balanced with the help of task migration or preemption, due to the costs caused by such actions, the Speedup of the system may decrease. In this case we need not preempt or migrate the tasks. In other words, the system is already in a balanced state.

4.2 Abstraction of a Cluster

Aiming at the characteristics of the architecture and applications of a cluster system, based on the general metric of load balancing in δ -range, we further research the algorithms to balance the loads in a cluster.

A typical cluster is composed of a set of nodes (suppose the number is n) and the interconnection network between the nodes. When constructing a practical system, the cluster is usually configured to m partitions. Each partition contains 1 or more nodes which cooperate with each other to provide a group of services with the same style and a set of jobs with the same characteristics will be scheduled to run in each partition. To enhance the system utilization, partitions can be dynamically adjusted. Jobs are submitted by users and allocated to corresponding partitions by the system. Relevant scheduling policy will schedule the jobs to run on each node in the partition.

A parallel cluster system can be described as $S_{cluster} = \langle P, N, Net, J, C \rangle$, where, P means the set of partitions in the cluster and N means the set of nodes which compose the cluster. Net denotes the interconnection network in the system, which is relevant to the computation of process migration and preemption cost h we have mentioned

before. J is the job sequence submitted to $S_{cluster}$. Besides, to give an abstraction approaching a real system, we suppose J to be an infinite sequence of jobs. C is a set of policies and here we just pay attention to the policies have something to do with load balancing.

4.3 Two Algorithms

In theory, a load balancing system could be viewed as an instance of Modern Control System[14].The controller could be users, system administrators and system software, while the controlled object is the real system. The target of this control system is to balance the computing and processing on each node so as to improve the resource utilization and system performance.

Fig.1 pictures a logical architecture of the load balancing control system. This architecture becomes a framework of our load balancing algorithms. Besides the controller and the controlled system, the static load balancing control information is treated as pre-feed information, and dynamic load information which is hardly to forecast is treated as feedback information. Both the pre-feed and feedback information are combined to balance the loads in the system.

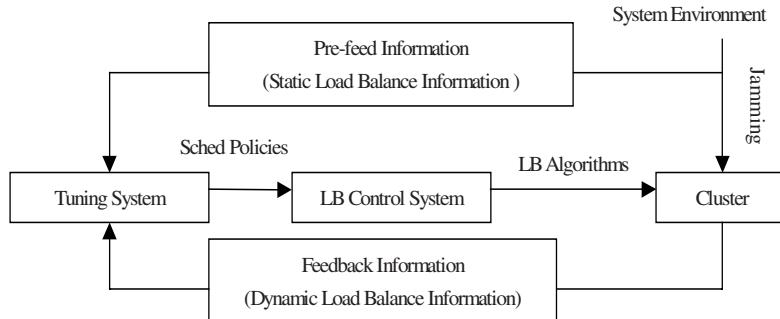


Fig. 1. Logical Architecture of the Load Balancing Control System (LB means Load Balancing)

From the viewpoint of the cluster system, we think a load balancing control system for a parallel cluster mainly involves two levels. One is the control inside a partition and the other is the control among partitions.

Load balancing inside a partition is done mainly through the policies of job scheduling and task migration. Suppose the system be composed of m partitions, each partition has a corresponding *Load Balancing Degree*, $\bar{\delta}$, and a *Load Banlancing Control Degree*, δ . The former is used to evaluate current loads while the latter is a pre-designated value from the angle of load balancing control on the partition.

Then we present the first algorithm, which is the algorithm inside a partition.

```

Algorithm C1 // Inside a Partition
{
  For i = 1 to m, set  $\bar{\delta}_i$  to 0, and set an appropriate
  value to each  $\delta_i$  according to experiences.
  For j = 1 to n, set  $E_j$ , which denotes the utilization
  of each node, to 0. Set  $T_j$ , which denotes total job
  running time on node j, to 0.
  While the queue of submitted jobs, J, is not empty
  {
    Choose a job  $j_{current}$  from J according to the
    priority, remove  $j_{current}$  from J.
    Choose an appropriate partition k for  $j_{current}$  to run
    according to its classification.
    Based on  $E_j$ ,  $T_j$  and the resource needs of  $j_{current}$ ,
    choose a proper set of nodes in partition k for
     $j_{current}$ .
    //Combine pre-feed and feedback information
    Calculate  $\bar{\delta}_k$  for partition k.
    If  $\bar{\delta}_k > \delta_k$ , find a low priority job running in
    partition k, anticipate the process of preempt and
    migrate the job to another node in partition k, and
    calculate the new value of  $\bar{\delta}_k$ , denoted as  $\bar{\delta}'_k$ .
    //Here, we do not really preempt or migrate the job.
    And the cost, h, is considered according to Net of
     $S_{cluster}$ 
    If  $\bar{\delta}'_k < \delta_k$ , preempt and migrate the choosed job.
    Accumulate the new value of  $E_i$  and  $T_i$  for each node
    in partition k.
  } // End of While
  If J is changed, such as the submission of new jobs,
  completion or termination of current jobs, modify the
  queue and re-calculate the priority of the jobs.
} // End of Algorithm C1

```

As to the load balancing among different partitions in the cluster system, we must evaluate the loads, utilization and the job running circumstances of each partition to justify the rationality of the division of partitions. If there exists apparent uniformity among the partitions, for example, some partitions may be too busy, while others may be too idle, we must dynamically adjust the partitions to improve the resource utilization of the cluster. So we further present Algorithm C2.

```

Algorithm C2 // Adjustment of Partitions
{
  For i = 1 to m, set  $\bar{\delta}_i$  to 0, and set an appropriate
  value to each  $\delta_i$  according to experiences.
  For j = 1 to n, set  $E_j$  and  $T_j$  to 0.

```

```

While Queue J is not empty
{
    Choose a job  $j_{current}$  from J according to the priority.
    Choose an appropriate partition k for  $j_{current}$  to run according to its classification.
    According to  $E_j$ ,  $T_j$  and the resource needs of  $j_{current}$ , choose a proper set of nodes in partition k for  $j_{current}$  to run on.

    Calculate  $\lambda = \frac{\sum_{i=1}^m (1 - \bar{\delta}_i) \bullet \bar{L}}{|J|}$ , where  $|J|$  is the size of J
    and  $\bar{L}$  is the average loads in the system.

    //  $\lambda$  is used to evaluate the balanced state of the partitions in the system. If  $\lambda$  is less than a threshold designated by experiences,  $\lambda_0$ , we think the system is unbalanced and partitions should be adjusted.

    If  $\lambda < \lambda_0$ , choose a partition i with the minimum  $\bar{\delta}_i$ , and choose a node  $n_{mig}$  from partition i, migrate the jobs that are currently running on  $n_{mig}$  to other nodes in partition i, then remove  $n_{mig}$  from partition i and add  $n_{mig}$  to the partition with the maximum  $\bar{\delta}_i$ .
    Re-calculate  $\lambda$ , if  $\lambda < \lambda_0$ , repeat the above step.

    Accumulate the new value of  $E_i$  and  $T_i$  for each node in the system.

} // End of While
If Queue J is changed, such as the submission of new jobs, completion or termination of some jobs, modify the queue and re-calculate the priority of the jobs.
} // End of Algorithm C2

```

Remember we omit the caculation process of cost h here, just for simplicity.

In the implementation of both the algorithms, attention should be paid to the choose of *Load Banlancing Control Degree*, δ . We should also pay attention to the cost of migrating a job and adjustment of partitions. Combining the two algorithms could balance the loads both inside and among the partitions in the cluster system.

By the way, to implement both the algorithms, we need a large scale parallel system and we are now on the way to constructing a distributed-shared memory system with a new memory consistency model and programming model[15][16]. Performace evaluation and further optimizations should be presented as soon as possible. In this paper we just provide the algorithms here to verify the rationality of the metric in δ -range.

5 Conclusion

In this paper, we present a new generic metric of load balancing in δ -range. A dynamically calculated δ is introduced to describe the concept of relative load balancing corresponding to absolute balancing in traditional metrics. This metric is able to overcome the confined limitations of traditional load equalization metrics. Several properties of this metric are also proved in this paper. Finally, based on this metric, we describe two load balancing algorithms for cluster systems. Future work includes the implementation and performance evaluation of the algorithms in a real large scale multi-task parallel cluster system, and the results will be inversely used to consummate the metric.

References

1. Jim Basney, Miron Livny, and Todd Tannenbaum, High Throughput Computing with Condor, *HPCU news*, Volume 1(2), June 1997
2. International Business Machines Corporation, IBM LoadLeveler for AIX5L: Using and Administering, Version 3 Release 1, Dec 2001
3. LSF Homepage, <http://www.platform.com>
4. Richard B. Bunt, Derek L. Eager, Gregory M. Oster, and Carey L. Williamson, Achieving Load Balance and Effective Caching in Clustered Web Servers, Proceedings of the 4th International Web Caching Workshop, 1997
5. He jin and Meng Dan, Load Balancing Policies on DCFS (Dawning Cluster File System), Computer Engineering and Applications (China), Vol.38, No.14, pages 109–112, 2002
6. M. Cierniak, M.J.Zaki, and W. Li, Compile-time Scheduling Algorithms for a Heterogeneous NOW, The Computer Journal, Vol. 40(6), pages 356–372, Dec, 1997
7. Shen Shen Wu and David Sweeting, Heuristic Algorithms for Task Assignment and Scheduling in a Processor Network, Parallel Computing, Vol 20, pages 1–14, 1994
8. E. Hou, N. Ansari, and H. Ren, A Genetic Algorithm for Multiprocessor Scheduling, IEEE Transactions on parallel and distributed system, Vol. 5, No. 2, pages 113–120, Feb 1994
9. J.Liu and V.A.Salelore, Self-Scheduling on Distributed Memory Machines, in Proc. of Supercomputing'93, pages 814–823, 1993
10. C. Polychronopoulos and D. Kuck, Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers, IEEE Transactions on Computers, pages 1425–1439, Dec 1987
11. S. F. Hummel, E. Schonberg, and L. E. Flynn, Factoring: A Method for Scheduling Parallel Loops, Comm. ACM 35(8), pages 90–101, 1992
12. O.Plata and F.F. Rivera, Combining Static and Dynamic Scheduling on Distributed-Memory Multiprocessors, in Proc. of the 1994 ACM ICS, pages 186–195, 1994
13. A. Winckler, A Distributed Look-ahead Algorithm for Scheduling Interdependent Tasks, in Proc. ISADS 93, pages 190–197, 1993
14. W. L. Brogan, Modern Control Theory, Prentice-Hall, Englewood Cliffs, New Jersey, 1982
15. H.D.Dai and X.J.Yang, Operating System-Centric Memory Consistency Model-Thread Consistency Model, Journal of Computer Research and Development, Vol.40, No.2, pages 351–359, Feb 2003
16. H.D.Dai and X.J.Yang, A New Framework of Memory Consistency Models in Distributed Shared Memory System-S³C Framework, Chinese Journal of Computers, Vol.25, No.12, pages 1387–1396, Dec 2002

Lattice Boltzmann Simulations of Fluid Flows

Baochang Shi¹, Nangzhong He¹, Nengchao Wang¹, Zhaoli Guo²,
and Weibin Guo¹

¹ Parallel Computing Institute,
Huazhong University of Science and Technology,
Wuhan 430074, People's Republic of China
sbchust@wuhan.cngb.com

² National laboratory of Coal Combustion,
Huazhong University of Science and Technology,
Wuhan 430074, People's Republic of China
pcihust@wuhan.cngb.com

Abstract. Our recent efforts focusing on improving the lattice Boltzmann method (LBM) are introduced, including an incompressible LB model without compressible effect, a flexible thermal LBM with simple structure for Boussinesq fluids, and a robust boundary scheme. We use them to simulate the lid-driven cavity flow at Reynolds numbers 5000–50000, the natural convection due to internal heat generation in a square cavity at Rayleigh number up to 10^{12} , respectively. The numerical results agree well with those of previous studies.

1 Introduction

Today, despite enormous progress in computational fluid dynamics (CFD), limitations still exist because of computer resources. It is apparent that several orders of magnitude improvement in both speed and memory are necessary to solve problems of contemporary interest. These requirements are obtained assuming today's solution algorithms and computer architecture. Since the technologies of scalar and vector computing have had substantial development, further work is unlikely to yield significant increases in computer performance. Massive parallel processing, on the other hand, appears to possess the capability to partially fill the gap between computational needs and present supercomputer performance. Although the supercomputer performance keeps rapid increase, it still could not satisfy the computational need in CFD due to the complex behavior of fluids, especially that of turbulence. Therefore, the efficient use of massively-parallel computers requires new algorithms with high performance. The lattice Boltzmann method (LBM) is a candidate of such algorithms [1-4].

The LBM is a relatively new approach that uses simple microscopic models to simulate complicated macroscopic behavior of transport phenomena. In recent years, the LBM method has achieved great success in simulations of fluid flows and modeling physics in fluids, involving complicated boundaries or/and complex fluids, such as turbulent flow, multi-phase/component fluids, free boundaries in flow systems, particle suspensions in fluid, reactive-diffusive systems and combustions,

magnetohydrodynamics, crystal growing, granular flow and others [3,4]. Compared with the conventional CFD approach, LBM is simple to code, intrinsically parallel, ready for extending to three-dimensional problems and has clear physical pictures. It is also easy to incorporate complicated boundary conditions such as those in porous media.

The LB models commonly used in the solution of the incompressible Navier-Stokes (NS) equations can be viewed as compressible schemes to simulate incompressible fluid flows, and there is the compressible effect which might lead to some undesirable errors in numerical simulations. Some LB models have been proposed to reduce or eliminate such errors [5-7]. However, most of the existing LB models either can be used only to simulate steady flows or are still of artificial compressible form. So, when used to simulate unsteady incompressible flows, these methods require some additional conditions to neglect the artificial compressible effect. In Ref. [8], we have proposed a 9-bit incompressible LB model in two-dimensional space. To our knowledge, this model is the first one without compressible effect for simulating incompressible flows. The approach can be used in the solution of steady or unsteady problems and can also be used to develop other incompressible LB models in either two- or three-dimensional space.

In LBM simulations, boundary condition is a very important issue. Proper boundary conditions can reduce the computational cost and enhance the numerical stability of algorithms. At solid walls, the original schemes are realized by particle density bounce-back. These bounce-back conditions are simple and can be used to some flow problems with complex geometries. But it is known that bounce-back wall boundary conditions are of first-order accuracy, and can not process the complex boundary conditions. To avoid of these problems, several new type boundary-processing schemes have been proposed and improved the overall accuracy of LB methods [9]. But, these schemes are imposed certain restrictions. Although the extrapolation scheme proposed by Chen *et al.* used second-order extrapolation, which is consistent with LBM, we found that the second-order extrapolation scheme has poor stability for high Reynolds numbers. It is necessary to establish a processing scheme for boundary conditions, which is of higher order accuracy, has robust stability and is efficient for arbitrary complex geometric boundaries.

In the present paper, our recent efforts focusing on above problems are introduced, including an LBM without compressible effect, a flexible thermal LBM with simple structure for Bousinesq fluids, and a robust boundary scheme. We use them to simulate the lid-driven cavity flow at Reynolds numbers 5000-50000, and the natural convection due to internal generation in a square cavity at Rayleigh number up to 10^{12} , and Prandtl number 0.25 and 0.6. The numerical results agree well with those of previous studies.

2 Background of Lattice Boltzmann Method

2.1 Lattice Boltzmann Method

A popular LB model is the so-called lattice BGK model (LBGK) with the single relaxation time approximation [3] :

$$f_i(\mathbf{x} + c\mathbf{e}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = -\tau^{-1}[f_i(\mathbf{x}, t) - f_i^{(eq)}(\mathbf{x}, t)], \quad (1)$$

where \mathbf{e}_i is the discrete velocity direction, and $c = \Delta x / \Delta t$ is particle speed, $\Delta x, \Delta t$, and τ are the lattice grid spacing, the time step and the dimensionless relaxation time, respectively. $f_i(\mathbf{x}, t)$ is the distribution function at node \mathbf{x} and time t with velocity \mathbf{e}_i , and $f_i^{(eq)}(\mathbf{x}, t)$ is the corresponding equilibrium distribution depending the lattice model used. The nine-bit square lattice model referred to as D2Q9 model has been successfully used for simulating 2-D flows. For the D2Q9 model, \mathbf{e}_i is defined as

$$\mathbf{e}_0 = \mathbf{0}, \quad \mathbf{e}_i = (\cos[(i-1)\pi/2], \sin[(i-1)\pi/2]) \text{ for } i=1:4,$$

$$\mathbf{e}_i = \sqrt{2}(\cos[(i-5)\pi/2 + \pi/4], \sin[(i-5)\pi/2 + \pi/4]) \text{ for } i=5:8.$$

The equilibrium distribution for the D2Q9 model is in the form of

$$f_i^{(eq)} = \omega_i \rho [1 + 3 \frac{(\mathbf{e}_i \cdot \mathbf{u})}{c} + 4.5 \frac{(\mathbf{e}_i \cdot \mathbf{u})^2}{c^2} - 1.5 \frac{|\mathbf{u}|^2}{c^2}] \stackrel{\Delta}{=} \omega_i \rho [1 + s_i(\mathbf{u})], \quad (2)$$

where ω_i is the weighting factor given by $\omega_0 = 4/9, \omega_i = 1/9 (i = 1 : 4), \omega_i = 1/36 (i = 5 : 8)$.

The flow density, momentum fluxes and kinetic viscosity can be evaluated as

$$\rho = \sum_{i=0}^8 f_i, \quad \rho \mathbf{u} = \sum_{i=1}^8 c \mathbf{e}_i f_i, \quad v = (\tau - 1/2) c_s^2 \Delta t, \quad (3)$$

where $c_s = c/\sqrt{3}$ is the speed of sound in this model and the equation of state is that of an ideal gas, $p = c_s^2 \rho$.

Through multi-scaling expansion, the incompressible NS equation can be derived to the second order under the low Mach number limitation [3]:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0, \quad (4)$$

$$\partial_t (\rho \mathbf{u}) + \nabla \cdot (\rho \mathbf{u} \mathbf{u}) = -\nabla p + v \nabla \cdot (\nabla \rho \mathbf{u} + \nabla (\rho \mathbf{u})^T). \quad (5)$$

From Eqs.(4)-(5), we can see that the LBM is in fact an artificial compressible scheme for the incompressible NS equation. This may lead to compressible effect. We recently proposed a true incompressible LBGK model (ILBGK) for the incompressible NS equation without compressible effect [8]. The equilibrium distribution in ILBGK is defined by

$$f_i^{(eq)} = \lambda_i p + s_i(\mathbf{u}), \quad (6)$$

where $\lambda_0 = -4\sigma/c^2, \lambda_i = \lambda/c^2 (i=1:4)$ and $\lambda_i = \gamma/c^2 (i=5:8)$ with parameters σ, λ , and γ satisfying $\lambda + \gamma = \sigma, \lambda + 2\gamma = 1/2$, and $s_i(\mathbf{u})$ as in Eq.(2).

The flow velocity, pressure and kinetic viscosity are given by

$$\mathbf{u} = \sum_{i=1}^8 c \mathbf{e}_i f_i, \quad p = \frac{c^2}{4\sigma} [\sum_{i=1}^8 f_i + s_0(\mathbf{u})], \quad v = (\tau - 1/2) c_s^2 \Delta t.$$

The ILBGK is a second order scheme for true incompressible NS equation,

$$\nabla \cdot \mathbf{u} = 0, \quad (7)$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u}. \quad (8)$$

2.2 Boundary Conditions

We have found that most of the existing boundary schemes commonly used in LBGK usually encounter numerical instability for flows at large Ra or Re . In a previous study we developed a non-equilibrium extrapolation rule for velocity and pressure boundary condition [10], which is of second order, simple form, and exhibits much better numerical stability. This rule is based on the decomposition of distribution function:

$$f_i(\mathbf{x}_b, t) = f_i^{(eq)}(\mathbf{x}_b, t) + f_i^{(neq)}(\mathbf{x}_b, t), \quad (9)$$

where $f_i^{(neq)}$ is the non-equilibrium part of f_i , \mathbf{x}_b is a node on boundary.

For velocity boundary, we have

$$f_i(\mathbf{x}_b, t) = \bar{f}_i^{(eq)}(\mathbf{x}_b, t) + (f_i(\mathbf{x}_f, t) - f_i^{(eq)}(\mathbf{x}_f, t)), \quad (10)$$

while for pressure boundary, we use

$$f_i(\mathbf{x}_b, t) = \bar{\bar{f}}_i^{(eq)}(\mathbf{x}_b, t) + (f_i(\mathbf{x}_f, t) - f_i^{(eq)}(\mathbf{x}_f, t)), \quad (11)$$

where

$$\bar{f}_i^{(eq)}(\mathbf{x}_b, t) = \lambda_i p(\mathbf{x}_f, t) + s_i(\mathbf{u}(\mathbf{x}_b, t)), \quad \bar{\bar{f}}_i^{(eq)}(\mathbf{x}_b, t) = \lambda_i p(\mathbf{x}_b, t) + s_i(\mathbf{u}(\mathbf{x}_f, t)),$$

\mathbf{x}_f is its nearest neighbor fluid node of \mathbf{x}_b .

3 Numerical Results

3.1 Lid-Driven Cavity Flow

The configuration of the lid-driven cavity flow considered in this paper consists of a two-dimensional square cavity whose top plate moves from left to right with a uniform velocity ($U=1$, here), while the other three walls are fixed. The flow is described by the dimensionless incompressible NS equations, that is Eqs.(7)-(8), where $\mathbf{u} = (u, v)$ is the velocity vector, p is the pressure, and $\nu = 1/Re$ is the kinetic viscosity, Re the Reynolds number.

Numerical simulations were carried out using the methods presented above for the driven cavity flow with $Re=5000, 7500, 10000, 15000, 20000$, and 50000 , respectively, on 256×256 lattice. The relaxation parameter $\omega = \tau^{-1}$ is set to be $1.85, 1.92, 1.95, 1.95, 1.96$ and 1.985 , respectively. $(\sigma, \lambda, \gamma)$ in Eq.(6) is set to be $(5/12, 1/3, 1/12)$ such that $(\lambda, \gamma) = 3 \times (\omega_l, \omega_s)$ which has the symmetry to agree with the

method, and we find that the simulations with this set are more robust. For the walls, no-slip boundary conditions were prescribed by the non-equilibrium extrapolation method given above. The flow with $Re=5000$ is first simulated, where the initial condition is set as $p = 0$, and the velocities at all nodes, except the top nodes, are set as $u = v = 0$. The simulations for higher Re start from the solution obtained for lower Ra as initial condition. In the simulations, steady solutions are obtained for $Re \leq 10^4$. For $Re=1.5 \times 10^4$, the flow becomes periodic, and a period is about 2000 time step. For $Re=2 \times 10^4$ and 5×10^4 , the flow becomes unsteady, and the solutions at 300000 and 350000 time steps are given, respectively.

Fig. 1. shows the contours of the stream function of the flows for the Reynolds numbers considered. These plots show clearly the effect of the Reynolds number on the flow pattern. At $Re=5000$, in addition to the primary, center vortex, and three first-class vortices, a pair of secondary ones of much smaller strength develop in the lower corners of the cavity. When Reynolds number reaches to 7500, a tertiary vortex appears in the lower right corner. Stationary solutions were found for Reynolds numbers up to 10000. We can also see that the center of the primary vortex moves toward the geometric center of the cavity as the Reynolds number increases and becomes fixed in x -direction. As the Reynolds number increases, no more steady solution was found and the flow becomes periodic in time at $Re=15000$. Here streamlines in one period for $Re=15000$ are plotted in Fig. 2. As the Reynolds number increases to 20000, the period begins to be broken, but the primary vortex is still stable. When Re reaches 50000, the flow becomes chaos and the primary vortex is unsteady and broken. To quantify the results, the locations of the vortex are listed in Table 1. From the table, we can see that these values predicted by the LBGK method agree well with those of previous work [2,11,12].

3.2 Natural Convection Flow with Internal Heat Generation

Natural convection (NC) in enclosures is a classical problem in the heat transfer literature and serves as one of the most popular test-problems for numerical methods for incompressible viscous flows. Much more work has addressed two main classes of NC flows, those heated from below and those from side [13,14]. Although NC due to internal heat generation is not less important, it drew much less attention in the past than the two classes above. However, in recent years, it becomes a subject of intense interest mainly due to nuclear safety issues [15,16]. Horvat *et al.* recently simulated the NC flows with internal heat generation in a square cavity for a wide range of Ra and Prandtl numbers (Pr): Ra 10^6 - 10^{13} and Pr 0.25-8 by using the LES method [15,16]. Since the NC flows at large Ra have complex behavior, the solution of them is very difficult. Efficient methods are still needed for further study, especially for 3D problems. The LBGK method is perhaps a suitable one.

The flow considered is in an enclosure of height H and width W (aspect ratio $A = H/W$), and governed by two-dimensional unsteady Boussinesq equations in primitive variables[15]. Since the force or source terms in momentum and temperature are required to be of order $O(\Delta t)$ in LBGK, where Δt is the time step, we use the following dimensionless equations in the paper by setting $U = Ra^{0.5}\mathbf{u}$, $t' = Ra^{-0.5}t$, and $P = pRa$ in the dimensionless equations for the flow in Ref.[15]:

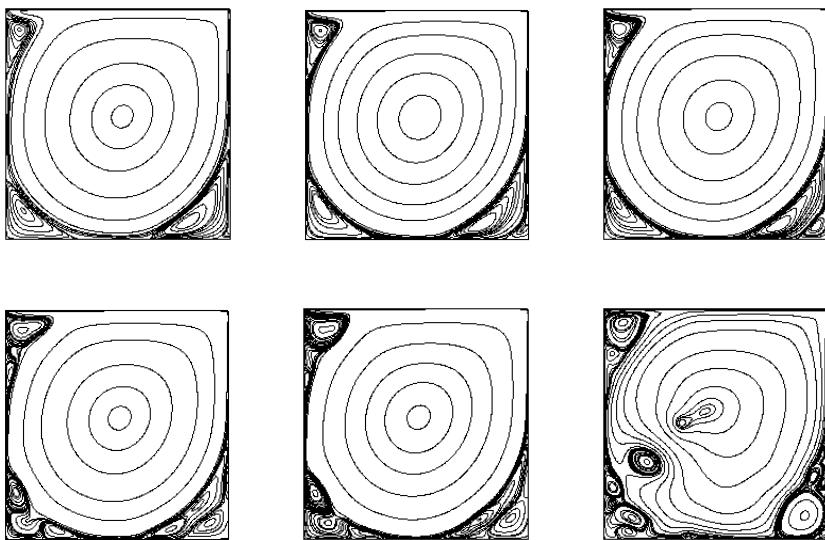


Fig. 1. Streamlines of the flow at: $Re=5000$, 7500 , 10000 (Top: left to right), and $Re=15000$, 20000 , 50000 (Bottom: left to right)

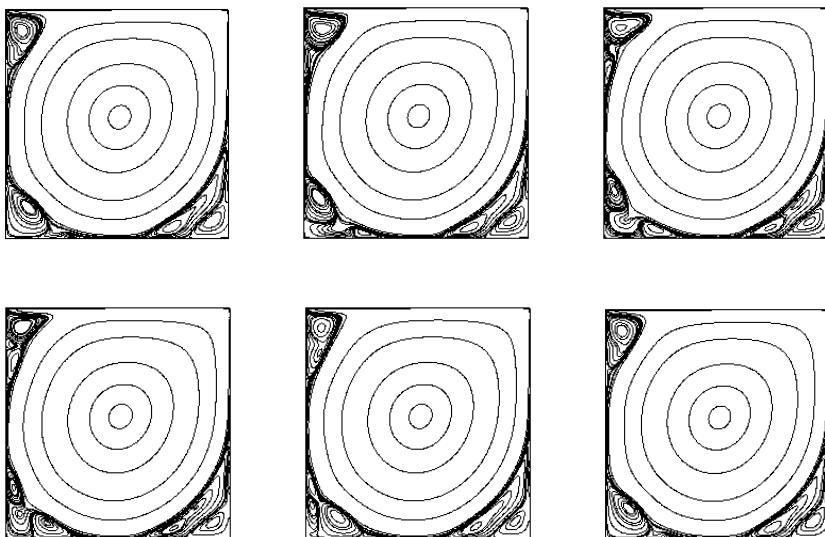


Fig. 2. Streamlines of the flow at $Re=15000$ in one period from 155000 to 157000 time steps; $T=1.9560$

Table 1. Locations of Vortex of the Driven Cavity Flow. The letters T, B, L, and R denote top, bottom, left, and right, respectively; a. Ghia *et al.*[11]; b. Hou and Zhou [2]; c. Present work

<i>Re</i>	Primary	First (T)	First (BL)	First (BR)	Second (BL)	Second (BR)
5000	a	0.5117,	0.0625,	0.0703,	0.8086,	0.0117,
	b	0.5352	0.9102	0.1367	0.0742	0.0078
	c	0.5176,	0.0667,	0.0784,	0.8078,	-
		0.5373	0.9059	0.1373	0.0745	-
		0.5156,	0.0625,	0.0742,	0.8086,	0.0039,
		0.5352	0.9063	0.1328	0.0742	0.0039
		0.5117,	0.0664,	0.0645,	0.7813,	0.0117,
		0.5322	0.9141	0.1504	0.0625	0.0117
	a	0.5176,	0.0706,	0.0706,	0.7922,	-
7500	b	0.5333	0.9098	0.1529	0.0667	-
	c	0.5156,	0.0664,	0.0664,	0.7930,	0.0078,
		0.5352	0.9102	0.1484	0.0664	0.0039
		0.5117,	0.0703,	0.0586,	0.7656,	0.0156,
10000	a	0.5333	0.9141	0.1641	0.0586	0.0195
	c	0.5117,	0.0703,	0.0625,	0.7813,	0.0117,
		0.5313	0.9102	0.1563	0.0625	0.0117
15000	c	0.5117,	0.0781,	0.0547,	0.7227,	-
		0.5313	0.9141	0.1992	0.0391	-
20000	c	0.5117,	0.0820,	0.0703,	0.7109,	-
		0.5273	0.9102	0.1758	0.0391	0.0742

$$\nabla \cdot \mathbf{u} = 0 , \quad (12)$$

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nu \nabla^2 \mathbf{u} + Pr \Theta \mathbf{k} , \quad (13)$$

$$\partial_t \Theta + \mathbf{u} \cdot \nabla \Theta = D \nabla^2 \Theta + D , \quad (14)$$

where $\nu = Pr Ra^{-0.5}$, $D = Ra^{-0.5}$, \mathbf{k} is the unit vector in the y -direction, Pr is the Prandtl number and Ra the Rayleigh number; t' , \mathbf{U} , P and Θ are the dimensionless time, velocity vector, pressure and temperature, respectively, as in Ref.[15].

The boundary conditions are taken to be $\mathbf{u}=0$ and $\Theta=0$ on all the four walls; Here we set $A=1$. At the beginning of the simulation, the fluid was considered at rest and isothermal, with mean temperature $\Theta=0$. Thus the initial conditions are set to be $\mathbf{u}=0, \Theta=0$ for all cases.

We first modify the ILBGK model described by Eqs.(1) and (6) for the velocity field by adding a term to the evolution equation:

$$f_i(\mathbf{x} + c\mathbf{e}_i \Delta t, t + \Delta t) - f_i(\mathbf{x}, t) = -\tau^{-1}[f_i(\mathbf{x}, t) - f_i^{(eq)}(\mathbf{x}, t)] + F_i \Delta t ,$$

where $F_i = \frac{Pr}{2c} \alpha_i \mathbf{e}_i \cdot \mathbf{k} \Theta$ such that $\sum_{i=1}^8 c e_i F_i = Pr \Theta k$, and $\alpha = \delta_{l2} + \delta_{l4}$. Other forms of F_i can also be used.

To solve Eq. (14), we utilize a D2Q5 lattice consisted of \mathbf{e}_i ($i = 0 : 4$) as used in Eq. (1), and the LBGK equation for Eq. (14) reads

$$\Theta_i(\mathbf{x} + c\mathbf{e}_i \Delta t, t + \Delta t) - \Theta_i(\mathbf{x}, t) = -\tau_\Theta^{-1}[\Theta_i(\mathbf{x}, t) - \Theta_i^{(0)}(\mathbf{x}, t)] + T_i \Delta t , \quad (15)$$

where $\Theta_i^{(0)} = (\Theta / 5)[1 + 2.5(\mathbf{e}_i \cdot \mathbf{u}) / c]$ is the equilibrium, and for the source term we take $T_i = (D / 5)[1 + 2.5(\mathbf{e}_i \cdot \mathbf{u}) / c]$ such that $\sum_{i=0}^4 T_i = D$. Θ and D are calculated by

$$\Theta = \sum_{i=0}^4 \Theta_i, \quad D = (2\tau_\Theta - 1)(\Delta x)^2 / (5\Delta t). \quad (16)$$

It should be noted that the other lattices could also be used for Eq. (14). However, the lattice introduced here is the simplest one with rest particle. Note that in Ref. [17] we successfully simulated the NC flow heated from side (without heat source in Eq.(14)) at Ra up to 10^{10} using a D2Q4 model for Eq. (15) without the source term. It was found that the TLBGK with D2Q5 has better numerical stability than that with D2Q4 for the flows considered here.

Similarly, for thermal boundary condition, we use

$$\Theta_i(\mathbf{x}_b, t) = \Theta_i^{(0)}(\mathbf{x}_b, t) + (\Theta_i(\mathbf{x}_f, t) - \Theta_i^{(0)}(\mathbf{x}_f, t)). \quad (17)$$

We simulate the flow with $Ra=10^6-10^{11}$ for $Pr=0.25$ and $Ra=10^6-10^{12}$ for $Pr=0.60$ respectively, based on a 256×256 uniform lattice. The corresponding relaxation parameter $\omega = \tau^{-1}$ is set to be 0.90-1.991 for the case $Pr=0.60$, and 1.30-1.987 for $Pr=0.25$, while τ_Θ is given by Eq. (16). In the simulations, steady solutions for $Ra=10^6$ and 10^7 were reached. For $Ra=10^8-10^{12}$, the flow becomes unsteady, and solutions were obtained from 10^5 to 4.0×10^5 time steps, corresponding to the dimensionless time $t' \approx 0.1, 0.5, 0.03, 0.015$ and 0.01 , respectively, as in Ref. [15]. It is well known that the numerical stability of the LBGK models is usually very poor as $\omega = \tau^{-1} \rightarrow 2$. While we found that our scheme was still stable and accurate even as $\omega = 1.991$, at which the computation blows up using other schemes for the velocity and temperature boundary conditions.

Isotherms of the flows are shown in Fig.3. From the figures, we can see that the main features of the flows are in agreement with those obtained in Ref. [15]. To quantify the results, the time-boundary-averaged Nusselt numbers (Nu_{ave}) obtained by the TLBGK and those in Ref.[15] are plotted in Fig.4-5 (\log_{10} - \log_{10} diagram). It can be found that our results for $Ra \leq 10^9$ agree well with those in Ref.[15], while the others are different a little, which is perhaps due to the different methods used.

4 Conclusion

We proposed the LBGK models with a robust boundary scheme for complex flows at large Re or Ra . The numerical results agree well with those of previous studies. Since little work on simulations of flows at large Re or Ra by LBM was performed before, our work is important for the development of LBM. The proper implementation of the boundary conditions is crucial for the LBGK simulation. Non-equilibrium extrapolation method has robust stability and the overall accuracy of distribution function is of second order. With the proposed boundary schemes, we can simulate the flow at very large Re or Ra . We found that other LBGK models with these boundary schemes have also better stability. It is also found that if a finer lattice is

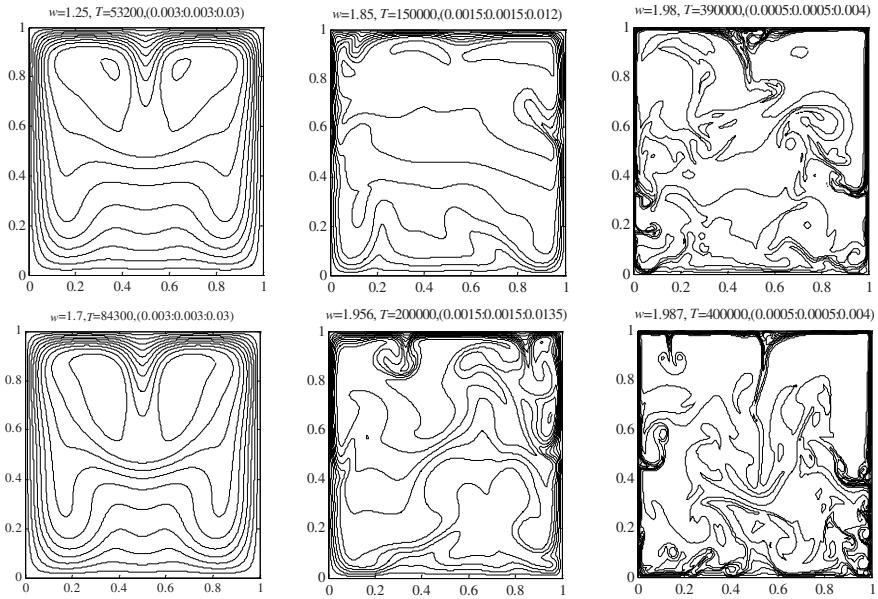


Fig. 3. Temperature isotherms: $Pr=0.6$ (top) and $Pr=0.25$ (bottom). $Ra=10^7, 10^9, 10^{11}$ from left to right. The data listed above each subfigure are the relaxation parameter ω , time steps T , and values of temperature isotherms

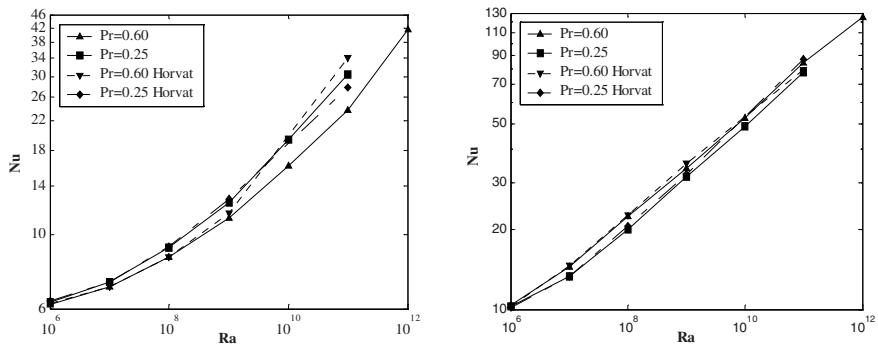


Fig. 4. Rayleigh number vs. time-boundary-averaged Nusselt number on the bottom(left) and side(right) boundaries

used, flows at larger Re or Ra than those here can be simulated using the present models. Moreover, our models can be easily extended to 3D problem. LBGK method is a relatively new approach for simulating complex flows. It is parallel in nature due to the locality of particle interaction and the transport of particle information, so it is well suited to massively parallel computing. Applying LBM to other complex flows is challenging work.

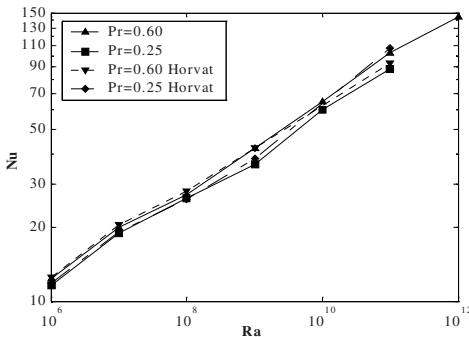


Fig. 5. Rayleigh number vs. time-boundary-averaged Nusselt number on the top boundary

Acknowledgments. This work is supported by the National Natural Science Foundation of China under Grant Nos. 60073044 and 70271069, and the State Key Development Programme for Basic Research of China under Grant No. G1999022207.

References

1. Frisch, U., d'Humières, D., Hasslacher, B., Lallemand, P., Pomeau, Y., Rivet, J.-P.: Lattice gas hydrodynamics in two and three dimensions, *Complex Syst.* 1 (1987) 649–707
2. Hou, S., Zou, Q., Chen, S., Doolen, G., Cogley, A.C.: Simulation of cavity flow by the lattice Boltzmann method, *J. Comput. Phys.* 118 (1995) 329–347
3. Chen S., Doolen G.: Lattice Boltzmann method for fluid flows, *Annu. Rev. Fluid Mech.* 30 (1998) 329–364
4. Luo, L.-S.: The lattice-gas and lattice Boltzmann methods: past, present, and future, *Proc. Int. Conf. Applied CFD*, (2000) 52–83
5. He X., Luo L.-S.: Lattice Boltzmann model for the incompressible Navier-Stokes equation, *J. Stat. Phys.* 88 (1997) 927–944
6. Lin, Z., Fang, H., Tao, R.: Improved lattice Boltzmann model for incompressible two-dimensional steady flows, *Phys. Rev. E* 54 (1997) 6323–6330
7. Zou, Q., Hou, S., Chen, S., Doolen, G.: An improved incompressible lattice Boltzmann model for time-independent flows, *J. Stat. Phys.* 81 (1995) 35–48
8. Guo, Z., Shi, B., Wang, N.: Lattice BGK model for incompressible Navier-Stokes equation, *J. Comput. Phys.* 165 (2000) 288–306
9. Maier, R.S., Bemard, R.S., Grunau, D.W.: Boundary conditions for the lattice Boltzmann method, *Phys. Fluid.* 8 (1996) 1788–1801
10. Guo, Z., Zheng C., Shi, B.: An extrapolation method for pressure and velocity boundary conditions in lattice Boltzmann method, *Chin. Phys.* 11 (2002) 366–374
11. Ghia, U., Ghia, K.N., Shin, C.T.: High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method, *J. Comput. Phys.* 48 (1982) 387–411
12. Schreiber, R., Keller, H.: Driven cavity flow by efficient numerical techniques, *J. Comput. Phys.* 49 (1983) 310
13. de V. Davis, G.: Int. J. Numer. Methods Fluids. 3 (1983) 249

14. Xin, S., Le Quéré, P.: Direct numerical simulations of two-dimensional chaotic natural convection in a differentially heated cavity of aspect ratio 4, *J. Fluid Mech.* 304 (1995) 87–118
15. Horvat, A., Kljenak, I., Marn, J.: *Int. J. Heat Mass Transfer.* 44 (2001) 3985–3995
16. Horvat, A.: PhD Thesis, University of Ljubljana, Slovenia, (2001)
17. Shi, B., Guo Z., Wang, N.: LBGK Simulations of Turbulent Natural Convection in a Cavity, *Chin. Phys. Lett.* 19 (2002) 515–517

Part III

Grid and Network

Design and Research of Strong-Mobile Agent Based Grid's Architecture

Zhirou Zhang and Siwei Luo

Dept. of Computer Science,
Northern Jiaotong University,
Beijing, China, 100044
zhangzhirou@hotmail.com
swluo@center.njtu.edu.cn

Abstract. Strong-Mobile Agent-Based Grid (SMAGrid) is based on mobile agents that support strong mobility. Because Tagent is Java-based agent system that supports strong mobility and MASIF Standard, we utilize it as underlying architecture. SMAGrid has a flexible structure and management mechanism that is suited for dynamic heterogeneous environments in WAN. In addition, SMAGrid is extensible and can provide user with safe and high-performance parallel application development environment. And it reduces developer's burden with a agent kernel library which are Java objects capable of performing the various tasks and all kinds of task assignment and scheduling algorithms. This paper emphasizes SMAGrid system's architecture and management mechanism.

1 Introduction

As the performance and number of computers and networks increase rapidly, how to harness the vast idle resource of computers in dynamic heterogeneous environments is a very valuable research direction. So it is necessary to connect geographically distributed heterogeneous computing resource, store resource, data resource with high-speed networks to become a system, referred to as a grid. All of members of this system cooperate in executing a large-scale application that a PC or workstation can't do independently.

Because grids can be utilized in various environments, different kinds of grids have been proposed: Computational grids [1], Access grids, Data grids, Datacentric grids [2]. Now a number of grid projects worldwide are actively exploring of grid technology. They include Legion[3], NASA Information Power Grid[4], Datagrid [5], AppLes[6], Nimrod/G[7], Unicore[8], Poznan Grid Computing[9], Globus [10]. In China, Institute of Computing Technology in Chinese Academy of Sciences, Tsinghua University, National University of Defence Technology are developing grid systems, such as VEGA Grid [11]. These systems have different advantages and features.

We design a grid system based on Java mobile agent, because mobile agents provide an efficient, flexible and asynchronous method for obtaining services in rapidly evolving networks and mobile agents support slow networks and lightweight

devices [9]. In addition, because as a carrier for task, strong-mobile agent make task continue to execute when it arrive at its destination, which save resource and time consumed by movement and reduce the burden of programmers, we design and develop a Strong-Mobile Agent-based Grid System, SMAGrid. And this system is achieved by Java, because Java is platform independent. Users can participate in this system to have their applications, which can't be accomplished only in their own computers, completed by all computers in system together. On the other hand, when computers are idle, they can be provided to others.

At present, there have been 60 mobile agent systems developed [12]. They have different functions and advantages. Some of them provide source codes free for non-commercial use. Tagent [13] is such a mobile agent system, which support strong mobility and MASIF (Mobile Agent System Interoperability Facility) Standard[14]. Comparing with performance, function and environment of all other mobile agent systems, we select it as underlying architecture to save development time and reduce work. Furthermore, we extend functions of Tagent system for grid, such as resource control and forced mobility, which improve efficiency of management and run of grid.

This paper emphasizes the design and architecture of SMAGrid. The rest of this paper is organized as follows. Section 2 presents a brief overview of SMAGrid's architecture. Section 3 presents the characteristics of Tagent system. Section 4 describes SMAGrid's implementation in details. In section 5, we conclude the paper and discuss future work.

2 SMAGrid Architecture

SMAGrid is based on mobile agents. According to different functions, there are 2 kind mobile agents: Master mobile agents (MMA) and worker mobile agents (WMA). MMAs cooperate each other with message delivery to manage SMAGrid system. Each MMA manage all the agents resided on computers, which are in the same LAN with the MMA. And WMAs execute assigned tasks for specific application. Furthermore, in our system, we generalize these two kind mobile agents by separating the mobile shell from the specific task code of the target application. We achieve this with the introduction of an agent kernel library, which consists of Java objects capable of performing the various tasks. Programmer can call Java objects in the library directly to reduce the work. Now we only implement some basic function, such as some simple task distribution and scheduling algorithm. User can extend the library for their specific application with Java objects developed by themselves. The shell provides general function for mobile agent, such as communication and movement.

The computers on which MMAs reside are referred to as Master. And the computers on which WMAs reside are referred to as Worker. In addition, the computers on which application submit task queue to SMAGrid are referred to as Clients.

Figure 1 is the architecture of SMAGrid, which has 2 kinds of hierarchies. One is management hierarchy, which is composed of all the MMAs. On the top of this hierarchy is Root Master Agents (RMA), which isn't mobile and lies on a fixed grid management computer. All MMA register to RMA or other MMA which take part in

the grid earlier to become a fork of the tree. How to construct the management hierarchy will be described in Section 4 in detail. The other is task diffusion hierarchy, which is formed along with task diffusion between nodes. The MMA to which Client submits task queue directly is called this task queue's Chief MMA (CMMA). CMMA firstly assigns tasks to Worker computers in the LAN that it manages and to the MMA on its lower layer, which assigns tasks in the LAN that it manages. Then CMMA submits the rest of queue to its upper MMA. In the view of task diffusion hierarchy, this upper MMA is the CMMA's descendent agent, and CMMA is this upper MMA's parent agent. This descendent MMA assigns tasks in the LAN that it manages and distributes rest of tasks to lower MMA, which distributes tasks in the LAN that it manages and becomes its descendent MMA. Tasks diffuse along with this order till all the tasks have been dispatched. In this course, because when the first computer in a LAN register to SMAGrid, it select the nearest (IP hops) MMA to register, this means a specific application's tasks firstly diffuse to nearer LAN from the Client that submit them. It assures that all tasks of this application would distribute on networks as near as possible to make communication distance between tasks and latency of message delivery smaller. This task diffusion mechanism is advantageous in low-bandwidth and high-latency WAN, especially in Internet. In Figure 1, the MMA layers connected by the bidirection thin arrowheads form management architecture, and the layers connected by the single-direction wide arrowheads form task diffusion architecture. Bidirection arrowheads between MMAs means these MMAs deliver messages each other. And single-direction arrowheads means the MMA only submit tasks queue along with the single direction.

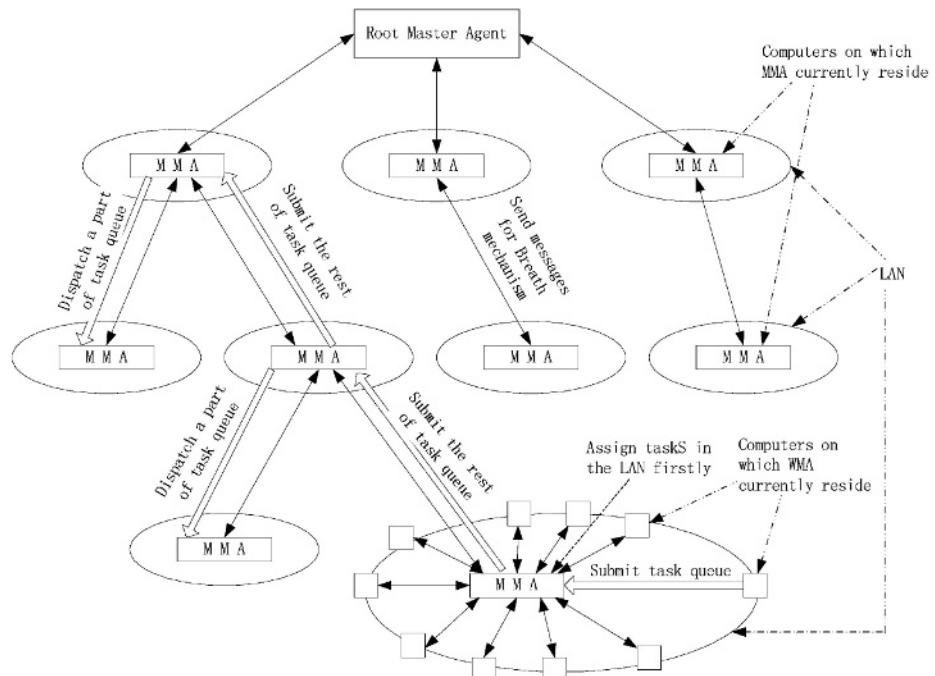


Fig. 1. The Architecture of SMAGrid

3 Features of Tagent System

Tagent System is a mobile agent system, based on the MASIF standard of the Object Management Group (OMG). Tagent is a new version of TAgents (Traveling Agents) System, which was redesigned and rewritten by Thomas Letsch for some new requirements, such as supporting more communication protocols, moving agents more easily, adapting standard and so on. So Tagent extend the old Tagents platform with plain socket communication, asynchronous messaging for RMI and plain socket.

The new system is a highly extendable system, which is implemented by the service-proxy mechanism. The new design of Tagent is independent to communication protocols, which are achieved by separating all the communication methods in an extra module and provide the system with a full interface to all communication needed. By providing the platform with the possibility to be extended with almost every kind of extension and resource access, Tagent can maintain resource.

Tagent separate the execution of an agent from the other parts of the system by intensively making use of threads and thread groups. Now an agent is totally separated from the system and with an appropriate security environment also from the standard libraries of Java. The design was built as secure as possible with the limitations of Java in relation to thread controlling [13].

In addition, Tagent implement MAFFinder interface of MASIF, which have it interact with other mobile agent systems that comply with the standard. MASIF standard uses CORBA, so it is defined as a set of IDL (Interface Definition Language) rules that have to be implemented by the Agent System. IDL rules for the MASIF, the MAF (Mobile Agent Facility) modules, are divided into two parts: The MAFAgentSystem and the MAFFinder. [14] MAFFinder is the main registry of a region. A region is a virtually defined place where one or more Agent System are running and which have one common region registry like the MAFFinder.

4 Design of SMAGrid in Detail

SMAGrid system has a dynamic architecture. Any participant can exit at any time and computer can join the system at any time. So this system needs a reliable management mechanism to hold system stable and needs a fault-tolerance mechanism to keep running application correct under abnormal conditions.

4.1 System Management Mechanism

When a computer intends to participate SMAGrid, it firstly broadcast in the LAN to enquire whether or not there has been a MMA in the LAN. If there has been a MMA, it registers to this MMA to become a member of this system. If there isn't a MMA, it sends message to RMA to obtain the IP address of all MMAs. Then it sends message to these MMAs and computes distances (IP hops) from these MMAs to itself with the data in communication packages. It registers to the nearest MMA and becomes next layer of this MMA. At last, this MMA creates a new MMA and make it move to the

new participator, which becomes a master of the LAN it lies on. In this way, SMAGrid forms a divide-and-conquer management hierarchy.

In the SMAGrid, MMA and WMA both can move autonomically and forcedly. When a WMA is executing on a computer, it measures the CPU and memory usage periodically to monitor the remaining resource on this computer. When the usage is higher than a threshold, the agent moves to other computer autonomically to avoid that current user of this computer feels inconvenient. Of course, external events also can force agents to move, for example, when the computer user desire to use it exclusively, he can force all agents that lie on this computer to move with interaction demand to SMAGrid system. So when a computer takes part in the system, it is still in good performance for user. Because mobile agents in Tagent don't have the resource monitoring and forced movement function, we extend them in the shell (See Section 4.2) of mobile agent in SMAGrid System to improve flexibility.

4.2 Mobile Agent Architecture

In SMAGrid, we separate the architecture of mobile agent into 2 parts: shell, which provide common functions such as migration, communication and registry; kernel, which provide specific functions for each agent. We build an agent kernel library that consists of kernel modules. Application developer can call the existed modules directly to reduce burden or extend the library with the special modules they need in their applications. So far, the library provides some basic functions and mechanisms for grid, such as simple task assignment and scheduling, maintenance of information in grid and so on. As grid applications have different characteristics and conditions, such as different size of tasks, communication frequency and size, request of runtime environments, we will develop different mechanisms for task assignment and scheduling.

4.3 Function of Agents

MMA's function is following:

- 1) Maintain a register and information list of SMAGrid member computers which is in the same LAN with the MMA. In this list, there are records about member's current state for task assignment. When the CPU usage of a member is lower than a threshold, its state is idle in the register and information list can be assigned new task.
- 2) Maintain an agent route table (agent's ID and computer address it currently reside on) for communication between agents.
- 3) Assign WMAs to computers in its LAN and communicate with them.
- 4) Create and dispatch MMAs.
- 5) Communicate with its upper and lower MMA and submit tasks to them.
- 6) Enquire current states of its lower MMA and WMA in its LAN with "Breath" mechanism.

- 7) Maintain a list of task queue information to manage all tasks (mobile agents) running in its LAN. The information includes: task queue's parent MMA, descendent MMA, and CMMA, which are used for task assignment and fault recovery.
 - 8) Monitor resource usage of the computers it lies on currently
- WMA's function is following:
- 1) Run autonomically on computers.
 - 2) Communication with other agents
 - 3) Create new WMA and submit to MMA
 - 4) Monitor resource usage of the computers it lies on currently

4.4 Faults Detection and Recovery Mechanism

In Internet environment, many failures may happen in computers and networks. So grid system must provide reliable fault detection and recovery mechanism. We use “Breath” mechanism, which is an extension of “heartbeat” mechanism, to detect faults. Root Master Agent (RMA) sends message to the lower MMAs and WMAs in its LAN at a fixed intervals to confirm whether or not they are normal. These MMAs in the next layer respond to these enquiries, and then enquire their lower MMAs and WMA in their LAN. In this way, “Breath” diffuses to all members of the grid. If a computer doesn't respond to the enquiry, it is thought as failure. There are 3 different failures and corresponding recovery methods:

- 1) When worker computer fails or disconnects from network, its MMA reassigns the worker's tasks to other computers according to task queue this MMA maintains.
- 2) When master computer fails or disconnects from network, the computer, which is in the same LAN with it and has copied management information of the MMA, become new master and inform its upper and lower MMAs and WMAs in the LAN.
- 3) The third fault is a LAN disconnects from WAN. Under this condition, if all the LANs whose MMA is in the next layer of the MMA in this disconnected LAN are thought to disconnect from the grid system, and all the tasks running on these LANs are discarded, a large amount of resource will be wasted. So we have MMAs in these LANs reregister to the upper MMA of this failed MMA to maintain stability of SMAGrid management architecture and management information to reduce loss caused by failure.

In Figure 2, the 2), 3) conditions are described. In 2), 3) conditions, the upper MMA is difficult to distinguish which fault happens, because they have the same phenomenon – MMA has no response to “Breath” enquiry. But we must deal them with different methods to reduce the loss caused by discarding tasks, so we utilize the participant which copy management information in the same LAN to distinguish them. When the upper MMA finds a disconnection to its next layer MMA, it does not deal with it immediately but waits several “Breath” intervals. If it is the second condition, in these intervals, the participant that copies management information, will displace the failed master computer and inform the upper MMA. If it is the third condition, because this LAN disconnect from WAN, the upper MMA can't receive message from displacer. So several intervals later, the upper MMA will think current the third condition happen and deal with it.

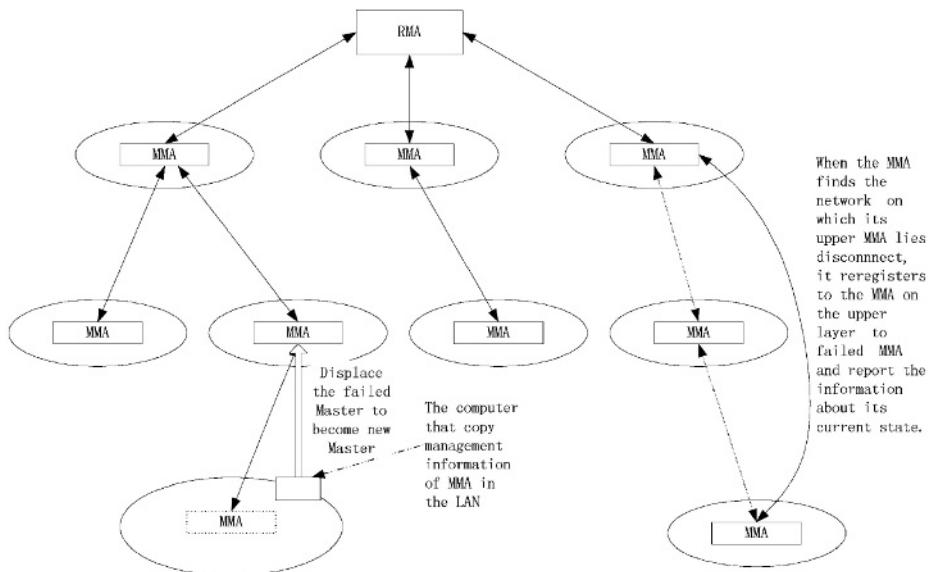


Fig. 2. Faults Recovery Mechanism

4.5 Load Balance

Now in the mobile agent kernel library, only simplest load balance algorithm has been implemented. Because in SMAGrid, tasks are packaged by mobile agent shell, task should be coarse-grained. Otherwise, task migration will cause too much additional cost. And SMAGrid is a flexible system in which available resource on participators may change at any time, so it needs dynamic scheduling mechanism for load balance and static mapping has few benefit to load balance. So we adopt even task mapping. Because an application doesn't be divided into too many small tasks, when there are many participators in SMAGrid, it is possible that tasks are many less than the participators that are idle currently. So only one task of a specific application will be mapped to a computer. Under this condition, because of task diffusion mechanism in SMAGrid, tasks of a specific application will distribute in a nearer scope to client that submit the application, which is beneficial to reduce communication cost. If more than a task is mapped to a computer, except even mapped tasks, the rest tasks will be assign to computers near the client. Dynamic scheduling is also implemented by MMA distributed in the whole system. When there isn't task running on a participator, it inform its MMA, and the MMA exchange information with other MMA to enquire whether there are more than a tasks lies on a computer or not. If there are, the task currently lies on nearer computer and isn't running will be scheduled to this idle computer to run. This is a basic scheduling algorithm. It is needed that more sophisticated dynamic algorithm to balance load, which is our future work.

5 Conclusion and Future Work

Based on Java mobile agent system, SMAGrid has a dynamic adaptability for dynamic heterogeneous environments and can make full use of a large number of idle resources of computers. Through it, computer owners can share their own resource with others, or complete their own applications together with others. This is convenient for computer owners or organizations who intend to run large parallel programs once in a while or whose computers are often idle. And the enterprises that want to lease a large number of resources for profit also can use SMAGrid.

Now the development of this system is in the first phase. There are many jobs to extend system, promote efficiency and flexibility, such as all kinds of task assignment and scheduling schemes, which are suited for different conditions. Appropriate authentication, authorization, delegation, accounting and payment are also important parts of grid system. These are our future research direction. In the next phase, we will develop all kinds of load balance schemes for different traditional parallel programs.

References

1. I Foster, C Kesselman. The Grid: Blueprint for a Future Computing Infrastructure. San Francisco, California: Morgan Kaufmann Publishers, 1999
2. D.B.Skillicorn, Motivating Computational grids, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.
3. S. chapin, The Legion Resource Management System, Porceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, April 1999
4. NASA IPG, <http://www.ipg.nasa.gov/>
5. Alexander Reinefeld, Volker Lindenstruth, How to build a high-performance compute cluster for the Grid, Parallel Processing Workshops, 2001.
6. F.Berman, R.Wolski, The AppleS Project: A Statics Report, Proceedings of the Eight NEC Research Symposium, Germany, May 1997
7. R.Buya, D.Abramson, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, International Conference on High Performance Computing in Asia-Pacific Region, Beijing, China, 2000
8. D.Erwin, “UNICORE and the Project UNICORE Plus”, Presentation at ZKI working group Supercomputing meeting on May 25, 2000
9. K.Kurowski, Multicriteria Resource Management Architecture for Grid, Proceedings of Fourth Annual Globus Retreat, 2000, Pittsburgh, PA
10. I.Foster and C.Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Application, Volume 11, No.2, 1997
11. Xu Zhi-Wei, Li Wei, RESEARCH ON VEGA GRID ARCHITECTURE, JOURNAL OF COMPUTER RESEARCH AND DEVELOPMENT, 2002.8
12. The Mobile Agent List,
<http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/mal.html>
13. Thomas Letsch, Redesign and Implementation of a Mobile Agent System compliant with the MAFFinder part of the MASIF Standard, 2000
14. Object Management Group, The Mobile Agent System Interoperability Facilities Submission Version 98-03-09

A Cost-Based Online Scheduling Algorithm for Job Assignment on Computational Grids

Chuliang Weng and Xinda Lu

Department of Computer Science and Engineering,
Shanghai Jiao Tong University,
Shanghai, 200030, People's Republic of China
{weng-cl,lu-xd}cs.sjtu.edu.cn

Abstract. The computational grid provides a promising platform for the deployment of various high-performance computing applications. Problem in implementing computational grid environments is how to effectively use various resources in the system, such as compute cycle, memory, communication network, and data repository. There are many effective heuristic algorithms for scheduling in the computational grid, however most scheduling strategies have no theoretical guarantees at all. This paper expands on the previous work, which considers CPU and memory with economic principle in the cluster environment. A cost-based online scheduling algorithm is presented for job assignment in the grid environment, and the performance of the algorithm is analyzed against the performance of the optimal offline algorithm.

1 Introduction

As a new infrastructure for next generation computing, computational grid enables the sharing, selection, and aggregation of geographically distributed heterogeneous resources for solving large-scale problems in science, engineering and commerce [1]. Many studies have focused on providing middleware and software programming layers to facilitate grid computing. There are a number of projects such as Globus, Legion, EcoGRID that deal with a variety of problems such as resource specification, information service, allocation and security issues in a grid computing environment involving different administrative domains. A crucial issue for the efficient deployment of distributed applications on the Grid is that of scheduling [2].

The computational grid consists of geographically distributed heterogeneous resources, such as CPU with different speed, and network with different bandwidth. Not only are these resources independent, but also they are not even directly comparable: they are measured in unrelated units. This may make it difficult to determine the optimal machine to which to schedule a given computational job.

In this paper, the concept of cost is introduced to the scheduling problem based on economic principles. The key of the method is to convert the total usage of different kinds of resources, such as CPU, memory and bandwidth into a homogeneous cost. According to the goal of minimizing the total cost, arrival jobs are scheduled in the computational grid.

The scheduling can be grouped into two categories: offline scheduling, and online scheduling. In the offline scheduling scenario, the sequence of jobs is known in advance, scheduling is based on information about all jobs in the sequence. However, a job is known only after all predecessors have been scheduled in the online scheduling scenario, and a job is scheduled only according to information of its predecessors in the sequence.

In this paper, we focus on the online scheduling problem. That is, when a new job arrives, it should be scheduled to a machine in the computational grid immediately. As customary, we evaluate the performance of the online algorithm in terms of makespan [3] and competitive ratio [4].

The paper is organized as follows. In Section 2, a brief overview is given for current research on scheduling for the computational grid and the difference of their works and ours. The model of the computational grid and assumptions are discussed in Section 3. The cost-based online scheduling algorithm is presented in Section 4. In Section 5, the performance of the algorithm is analyzed. Section 6 refers to the issue for applying the presented algorithm to practice. Finally, we conclude the paper in Section 7.

2 Related Work

There are many heuristic algorithms for scheduling on heterogeneous systems, and attempts to extend these heuristic algorithms to the grid environment have been made [6,7,8,9,10]. An extended version of Sufferage [5], Xsufferage is presented to schedule parameter sweep applications in grid environments [6]. A deadline scheduling strategy [7] is proposed that is appropriate for the multi-client multi-server case, and augmented with “Load Correction” and “Fallback” mechanisms, which could improve the performance of the algorithm. A distributed scheduling algorithm that uses multiple simultaneous requests at different sites is presented in [8]. A scheduling strategy for master-worker applications is discussed in [9], which dynamically measures the execution times of tasks and uses this information to dynamically adjust the number of worker to achieve a desirable efficiency, minimizing the impact on loss of speedup. Paper [10] addresses the potential benefit of sharing jobs between independent sites in a grid environment.

These heuristic algorithms are effective for scheduling in the computational grid, but most scheduling strategies have no theoretical guarantees at all. This paper expands on the work found in [11], which considers CPU and memory with economic principle in the cluster environment. We present a cost-based online scheduling algorithm for job assignment in the grid environment with theoretical guarantee, which is based on the work found in [12].

3 The Model

The topology of the computational grid is illustrated as Fig.1. Resources in the grid are organized as *resource domains* that are individual and autonomous administrative domains, which usually are in the range of Local Area Network (LAN). It is the

metascheduler or superscheduler that receives the requirement of users in the grid environment and assigns user's jobs to the geographically distributed resources. The metascheduler is connected to resource domains by Wide Area Network (WAN), and one data repository is available at the metascheduler so that the data needed by user's jobs are stored in the data repository in advance.

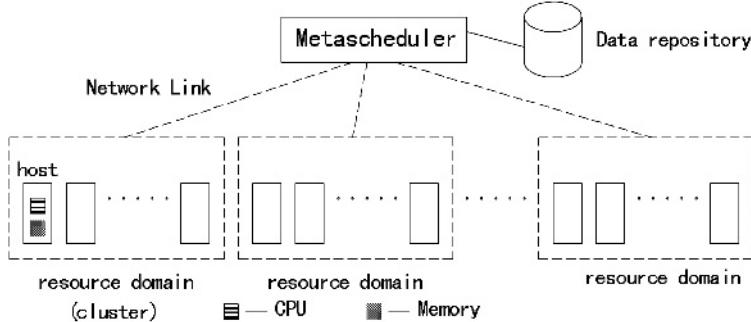


Fig. 1. Topology of the computational grid

3.1 Computational Grid Parameters

The work is intended to improve the performance in the computational grid, which consists of m resource domains that are geographically distributed, and resource domain rd_i and the data repository are connected with bandwidth $r_n(rd_i)$. Compared to communication overhead in WAN, the communication overhead in LAN is ignored. Host rd_{ij} in resource domain rd_i has a CPU resource of speed $r_c(rd_{ij})$ and a memory resource of size $r_m(rd_{ij})$, and for simplicity we assume that each resource domain has n computational hosts. All other resources associated with the host will be abstracted out, although the framework can be extended to handle additional resources.

There are many different factors that will influence the effective load of hosts in the grid and the execution time of jobs running there, so the *unrelated machines* model [13] is chosen for studying the scheduling algorithm. That is, the load of a given host added by a given job is known, but the load doesn't need to have any relationship to the load the job would add to another host. There are two other relevant machine models: *identical machines* and *related machines*. In *identical machines* model, all hosts are identical so that the completion time of a unit job on a host is determined only by the load of the host. In *related machines* model, all hosts are identical except that hosts' speeds differ from others so that the completion time of a unit job on a host just depends on the load and the speed of the host.

3.2 Job Parameters

Jobs arrive online, and each job has to be immediately assigned to one of hosts in the computational grid. We adopt *jobs arrive over time* [13] as our online paradigm, and each job jb_k is defined by following parameters:

- The number of CPU cycles required by the job, $t_c(jb_k)$.

- The amount of memory required by the job, $m(jb_k)$.
- The network bandwidth connected to the data repository it requires, $b(jb_k)$.

It is assumed that $m(jb_k)$, $t_c(jb_k)$, and $b(jb_k)$ are known upon job's arrival with the lack of knowledge of jobs arriving in the future. After assigned to a host in the computational grid immediately when it arrives, a job will not be reassigned to other hosts later. The data required by a job should be transferred to the assigned host during the course of the execution of the job.

3.3 Performance Metric

We adopt a classic performance metric: *makespan* [3], which is the length of the schedule, or equivalently the time when the first job starts running subtracted from the time when the last job is completed. The scheduling goal is to minimize the *makespan* of a given job sequence by developing an appropriate method for job assignment on hosts in the computational grid.

4 The Cost-Based Online Scheduling Algorithm

One of key issues when scheduling jobs based on economic principle is how to determine the cost of resources in the grid. There are many methods of determining the cost of resources in the grid [14]. In this paper, we just borrow ideas from the economics concept for converting the total usage of different kinds of resources, such as CPU, memory and bandwidth into a homogeneous cost. The *cost* of a resource is an exponential function of its *load* assigned on the resource since the first job arrives in the sequence of jobs.

4.1 System Load

$S(rd_i, jb_k)$ denotes the set of jobs in resource domain rd_i after job jb_k has been assigned on the computational grid, and $S(rd_{ij}, jb_k)$ denotes the set of jobs in host rd_{ij} after job jb_k has been assigned. Then they satisfy the following equation

$$S(rd_i, jb_k) = \bigcup_{j=1}^n S(rd_{ij}, jb_k) . \quad (1)$$

The *load* on a given resource equals its usage divided by its capacity, and then the load of the network connecting resource domain rd_i and the data repository after job jb_k was assigned is defined by

$$l_n(rd_i, jb_k) = \sum_{jb_p \in S(rd_i, jb_k)} b(jb_p) / r_n(rd_i) . \quad (2)$$

The memory load of host rd_{ij} in resource domain rd_i after job jb_k was assigned is defined by

$$l_m(rd_{ij}, jb_k) = \sum_{jb_p \in S(rd_{ij}, jb_k)} m(jb_p) / r_m(rd_{ij}) . \quad (3)$$

The CPU load of host rd_{ij} in resource domain rd_i after job jb_k was assigned is defined by

$$l_c(rd_{ij}, jb_k) = \sum_{jb_p \in S(rd_{ij}, jb_k)} t_c(jb_p) / r_c(rd_{ij}) . \quad (4)$$

4.2 Scheduling Algorithm

For simplicity we will abbreviate $l_n(rd, jb_k)$ as $l_n(i, k)$, $l_c(rd, jb_k)$ as $l_c(i, j, k)$, and $l_m(rd, jb_k)$ as $l_m(i, j, k)$. $l_n^o(i, k)$, $l_c^o(i, j, k)$, and $l_m^o(i, j, k)$ are the corresponding quantities for the scheduling produced by the optimal offline algorithm, which are the counterpart of $l_n(i, k)$, $l_c(i, j, k)$, and $l_m(i, j, k)$ produced by the cost-based online scheduling algorithm.

For normalizing the load, we introduce a positive parameter Δ , and the value of Δ will be discussed in Section 6. For variable x , we define:

$$\bar{x} = x / \Delta . \quad (5)$$

The *marginal cost* of a job jb_k assigned to host rd_{ij} in the grid system is denoted by $Cost(i, j, k)$, which consists of the marginal cost of network $Cost_n(i, k)$, the marginal cost of CPU $Cost_c(i, j, k)$, and the marginal cost of memory $Cost_m(i, j, k)$, and they are defined by:

$$Cost_n(i, k) = \alpha^{\bar{l}_n(i, k-1) + \bar{b}(jb_k) / r_n(rd_i)} - \alpha^{\bar{l}_n(i, k-1)} , \quad (6)$$

$$Cost_c(i, j, k) = \alpha^{\bar{l}_c(i, j, k-1) + \bar{t}_c(jb_k) / r_c(rd_{ij})} - \alpha^{\bar{l}_c(i, j, k-1)} , \quad (7)$$

$$Cost_m(i, j, k) = \alpha^{\bar{l}_m(i, j, k-1) + \bar{m}(jb_k) / r_m(rd_{ij})} - \alpha^{\bar{l}_m(i, j, k-1)} . \quad (8)$$

where α is a constant, and $\alpha > 1$.

Definition 1: Given a function $f: \mathbf{R}^n \rightarrow \mathbf{R}$

$$f(\arg \min_{\mathbf{x} \in \mathbf{R}^n} f(x)) = \min_{\mathbf{x} \in \mathbf{R}^n} f(x) . \quad (9)$$

The marginal cost method for assigning jobs puts a job on the host where its resource consumption has the minimum marginal cost. Given a sequence of jobs, jb_1, jb_2, \dots, jb_N , the *cost-based online scheduling* algorithm can be described as Fig.2.

```

Set  $\Delta = \max_{i,j} \{l_n^o(i, N), l_c^o(i, j, N), l_m^o(i, j, N)\};$ 
while (1) {
    when a new job  $jb_k$  arrives:
         $Cost(i, j, k) = Cost_n(i, k) + Cost_c(i, j, k) + Cost_m(i, j, k);$ 
         $(I, J) = \arg \min_{i,j} (Cost(i, j, k));$ 
        Assign:  $jb_k \rightarrow rd_{I,J};$ 
}

```

Fig. 2. The cost-based online scheduling algorithm

5 Performance Analysis

For any sequence J of jobs jb_1, jb_2, \dots, jb_N , the makespan of the schedule generated by online algorithm A is denoted by $A(J)$, and let $OPT(J)$ denote the makespan of the optimal offline algorithm for J . Online algorithm A 's competitive ratio is then

$$c_A = \sup_J \frac{A(J)}{OPT(J)}, \quad (10)$$

where the supremum is over all sequences of jobs.

Note that instead of makespan, the competitive ratio can be supremum of the maximum load of resources produced by the online algorithm divided by the maximum load of resources produced by the optimal offline algorithm. That is,

$$c_A = \sup_J \frac{\max_{i,j} \{l_n(i, N), l_c(i, j, N), l_m(i, j, N)\}}{\max_{i,j} \{l_n^o(i, N), l_c^o(i, j, N), l_m^o(i, j, N)\}}. \quad (11)$$

Theorem 1. The *cost-based online scheduling* algorithm is $O(\log(2n+1)m)$ competitive in the computational grid illustrated as Fig 1.

Proof. Define the potential function

$$\Psi(jb_k) = \sum_{i=1}^m (\alpha^{\bar{l}_n(i,k)} (\gamma - \bar{l}_n^o(i, k)) + \sum_{j=1}^n (\alpha^{\bar{l}_c(i,j,k)} (\gamma - \bar{l}_c^o(i, j, k)) + \alpha^{\bar{l}_m(i,j,k)} (\gamma - \bar{l}_m^o(i, j, k)))), \quad (12)$$

where γ is a constant, and $\gamma > 1$.

For jb_{k+1} ($k+1 \leq N$), the online scheduling algorithm chooses the host with the index (\bar{i}', \bar{j}') , while the optimal offline algorithm chooses the host with the index (\bar{i}'', \bar{j}'') , which will only influence the cost of CPU and memory of the two hosts respectively, and the corresponding network load. We then have

$$\begin{aligned}
& \Psi(jb_{k+1}) - \Psi(jb_k) \\
&= Cost_n(i^a, k+1)(\gamma - \bar{l}_n^o(i^a, k)) + Cost_c(i^a, j^a, k+1)(\gamma - \bar{l}_c^o(i^a, j^a, k)) + Cost_m(i^a, j^a, k+1)(\gamma - \bar{l}_m^o(i^a, j^a, k)) \\
&\quad - [\alpha^{\bar{l}_n(i^o, k)}(\bar{b}(jb_{k+1})/r_n(rd_{i^o})) + \alpha^{\bar{l}_c(i^o, j^o, k)}(\bar{l}_c(jb_{k+1})/r_c(rd_{i^o j^o})) + \alpha^{\bar{l}_m(i^o, j^o, k)}(\bar{m}(jb_{k+1})/r_m(rd_{i^o j^o}))] \\
&\leq \gamma Cost_n(i^a, k+1) + \gamma Cost_c(i^a, j^a, k+1) + \gamma Cost_m(i^a, j^a, k+1) \\
&\quad - [\alpha^{\bar{l}_n(i^o, k)}(\bar{b}(jb_{k+1})/r_n(rd_{i^o})) + \alpha^{\bar{l}_c(i^o, j^o, k)}(\bar{l}_c(jb_{k+1})/r_c(rd_{i^o j^o})) + \alpha^{\bar{l}_m(i^o, j^o, k)}(\bar{m}(jb_{k+1})/r_m(rd_{i^o j^o}))] \\
&\leq \gamma Cost_n(i^o, k+1) + \gamma Cost_c(i^o, j^o, k+1) + \gamma Cost_m(i^o, j^o, k+1) \\
&\quad - [\alpha^{\bar{l}_n(i^o, k)}(\bar{b}(jb_{k+1})/r_n(rd_{i^o})) + \alpha^{\bar{l}_c(i^o, j^o, k)}(\bar{l}_c(jb_{k+1})/r_c(rd_{i^o j^o})) + \alpha^{\bar{l}_m(i^o, j^o, k)}(\bar{m}(jb_{k+1})/r_m(rd_{i^o j^o}))] \\
&= \alpha^{\bar{l}_n(i^o, k)}[\gamma(\alpha^{\bar{b}(jb_{k+1})/r_n(rd_{i^o})} - 1) - \bar{b}(jb_{k+1})/r_n(rd_{i^o})] + \alpha^{\bar{l}_c(i^o, j^o, k)}[\gamma(\alpha^{\bar{l}_c(jb_{k+1})/r_c(rd_{i^o j^o})} - 1) \\
&\quad - \bar{l}_c(jb_{k+1})/r_c(rd_{i^o j^o})] + \alpha^{\bar{l}_m(i^o, j^o, k)}[\gamma(\alpha^{\bar{m}(jb_{k+1})/r_m(rd_{i^o j^o})} - 1) - \bar{m}(jb_{k+1})/r_m(rd_{i^o j^o})].
\end{aligned} \tag{13}$$

Since

$$\bar{b}(jb_{k+1})/r_n(rd_{i^o}) \leq \max_i \{l_n^o(i, k+1)\}/\Delta \leq 1, \tag{14}$$

and assume $\alpha = (\gamma+1)/\gamma$, it can be proved that

$$\gamma(\alpha^{\bar{b}(jb_{k+1})/r_n(rd_{i^o})} - 1) - \bar{b}(jb_{k+1})/r_n(rd_{i^o}) \leq 0. \tag{15}$$

In the similar manner, the other two items can also prove to be no more than zero. Hence, the potential function $\Psi(jb_k)$ is a nonincreasing function.

Since $\Psi(jb_0) = (2n+1)m \cdot \gamma$, after any job jb_k scheduled in the grid, we have

$$\sum_{i=1}^m (\alpha^{\bar{l}_n(i, k)}(\gamma - \bar{l}_n^o(i, k))) + \sum_{j=1}^n (\alpha^{\bar{l}_c(i, j, k)}(\gamma - \bar{l}_c^o(i, j, k)) + \alpha^{\bar{l}_m(i, j, k)}(\gamma - \bar{l}_m^o(i, j, k))) \leq (2n+1)m \cdot \gamma, \tag{16}$$

then,

$$\sum_{i=1}^m (\alpha^{\bar{l}_n(i, k)}(\gamma - 1)) + \sum_{j=1}^n (\alpha^{\bar{l}_c(i, j, k)}(\gamma - 1) + \alpha^{\bar{l}_m(i, j, k)}(\gamma - 1)) \leq (2n+1)m \cdot \gamma, \tag{17}$$

and then,

$$\alpha^{\max_{i,j} \{\bar{l}_n(i, k), \bar{l}_c(i, j, k), \bar{l}_m(i, j, k)\}}(\gamma - 1) \leq (2n+1)m \cdot \gamma, \tag{18}$$

hence,

$$\max_{i,j} \{l_n(i, k), l_c(i, j, k), l_m(i, j, k)\} \leq \Delta \cdot \log_\alpha \left(\frac{\gamma}{\gamma - 1} (2n+1)m \right) = O(\Delta \cdot \log(2n+1)m). \tag{19}$$

That is, $c_A = O(\log(2n+1)m)$. ■

6 Algorithm Application

In Fig.2, we determine Δ based on the optimal offline algorithm. In practice, it needs more operability for assigning appropriate value to parameter Δ before the cost-based

scheduling algorithm is applied to online schedule the arrival job. There are two manners to determine Δ :

1. According to existed result data produced by offline scheduling algorithms for the similar scheduling problem, an appropriate value is assign to the parameter Δ . For example, online parallel tomography [15] is an improvement for offline parallel tomography, therefore the parameter Δ can be determined according to the makespan of the previous similar offline parallel tomography.
2. Approximating Δ with the doubling approach.

Firstly, initialize Δ by

$$\Delta = \min_{i,j} \{b(jb_1)/r_n(rd_i), t_c(jb_1)/r_c(rd_{ij}), m(jb_1)/r_m(rd_{ij})\} . \quad (20)$$

After a new job arrives, we can determine host rd_{ij} with the presented algorithm, and then assign the job to the host under the condition: $\max\{l_n(I, k), l_c(I, J, k), l_m(I, J, k)\} \leq (\log(2n+1)m) \cdot \Delta$. If the inequality cannot be satisfied, Δ is doubled, and the marginal cost is recomputed for determining the new host to accomplish the job. It can be easy to see that this approach will not increase the asymptotic time complexity of the algorithm.

7 Conclusions

The work is inspired by economic principles, through which the total usage of different kinds of resources, such as CPU, memory and bandwidth are converted into a single cost. It is convenient for the metascheduler to assign jobs in the computational grid according to minimizing costs of all resources, and achieving the goal of minimizing the makespan of a sequence of online jobs.

Although there are effective heuristics for scheduling in the grid environment, most algorithms have no theoretical guarantees at all. In this paper, we present a method for determining cost of resource, and a cost-based online scheduling algorithm. Further, we theoretically analyze the performance of the algorithm against the performance of the optimal offline algorithm.

The further work includes applying the algorithm to schedule jobs in the campus grid testbed.

Acknowledgements. This research was supported by the National Natural Science Foundation of China, No. 60173031.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Int. J. of Supercomputer Appl.* 15(3)(2001) 200–222
2. Foster I., Kesselman, C. (ed.): *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, USA (1998)
3. Pinedo M.: *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall, Englewood Cliffs, USA (1995)

4. Bartal, Y., Fiat, A., Karloff, H., Vohra, R.: New Algorithms for an Ancient Scheduling Problem. *J. Comput. Syst. Sci.* 51(3)(1995) 359–366
5. Macheswaran, M., Ali, S., Siegel, H.J., Hensgen, D., Freund, R.F.: Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems. In: Proceedings of the 8th Heterogeneous Computing Workshop, IEEE, Los Alamitos (1999) 30–44
6. Casanova, H., Legrand, A., Zagorodnov, D., Berman, F.: Heuristics for Scheduling Parameter Sweep Applications in Grid Environments. In: Proceedings of the 9th Heterogeneous Computing Workshop, IEEE, Los Alamitos (2000) 349–363
7. Takefusa, A., Casanova, H., Matsuoka, S., Berman, F.: A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid. In: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing, IEEE, Los Alamitos (2001) 406–415
8. Subramani, V., Kettimuthu, R., Srinivasan, S., Sadayappan, P.: Distributed Job Scheduling on Computational Grids Using Multiple Simultaneous Requests. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, IEEE, Los Alamitos (2002) 359–367
9. Heymann, E., Senar, M.A., Luque, E., Livny, M.: Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In: Proceedings of the 1st IEEE/ACM International Workshop, Lecture Notes in Computer Science, Vol. 1971. Springer-Verlag, Berlin Heidelberg New York (2000) 214–227
10. Ernemann, C., Hamscher, V., Schwiegelshohn, U., Yahyapour, R., Streit, A.: On Advantages of Grid Computing for Parallel Job Scheduling. In: Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid, IEEE, Los Alamitos (2002) 39–46
11. Amir, Y., Awerbuch, B., Barak, A., Borgstrom, R.S., Keren, A.: Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster. *IEEE Trans. Parallel Distrib. Syst.* 11(7)(2000) 760–768
12. Aspnes, J., Azar, Y., Fiat, A., Plotkin, S., Waarts, O.: On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling. *J. ACM* 44(3)(1997) 486–504
13. Sgall, J.: On-line scheduling – a survey. In: Fiat, A., Woeginger, G. J. (eds.): *Online Algorithms: The State of the Art*. Lecture Notes in Computer Science, Vol. 1442. Springer-Verlag, Berlin Heidelberg New York (1998) 196–231
14. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic Models for Resource Management and Scheduling in Grid Computing. *J. Concurrency Comput. Practice & Experience* 14(13–15)(2002) 1507–1542
15. Smallen, S., Casanova, H., Berman, F.: Applying Scheduling and Tuning to On-Line Parallel Tomography. *Scient. Prog.* 10(4)(2002) 271–289

Composition and Automation of Grid Services

Zhihong Ren^{1,2}, Jiannong Cao¹, Alvin T.S. Chan¹, and Jing Li²

¹ Department of Computing,
Hong Kong Polytechnic University,
Hung Hom, Kowloon, Hong Kong
`{csjcao, cstschan}@comp.polyu.edu.hk`

²Institute of Software,
Chinese Academy of Sciences,
Beijing 100080, China
`{ren, lij}@otcaix.iscas.ac.cn`

Abstract. A grid service is a Web service that provides a set of well-defined interfaces and that follows specific conventions. Composition of grid services combines the offerings of two or more grid services to achieve the desired computing goals. Candidate standards have been proposed, providing a foundation for service composition specifications. However, at a higher level, there is no framework that supports service composition construction and automation. In this paper, we propose a framework that facilitates the visual composition as well as automation of service compositions. The framework is based mainly on Service Composition Graph (SCG), the underlying formalism for service compositions. Using graph grammar and graph transformation defined on SCG, the static topological structure of a service composition can be described and the automation of the constructed service compositions is also facilitated. We also outline the design and implementation of the prototype.

1 Introduction

Grid technologies have been widely adopted in scientific and engineering computing. By providing a set of services that allow a widely distributed collection of resources to be tied together into a relatively seamless computing framework, teams of researchers can collaborate to solve problems that they could not have attempted before. Current development of Grid infrastructures is aimed at supporting the sharing and coordinated use of diverse resources and services available from geographically distributed computers [10]. It is evolving toward an Open Grid Services Architecture (OGSA) in which a Grid provides an extensible set of services that virtual organizations (VOs) can aggregate in various ways. Examples of Grid services include security services, information services, job submission services, and co-scheduling services [11].

A grid service [19] is defined as a web service that conforms to a set of conventions (interfaces and behaviors) that define how a client interacts with a grid service. A web service [4] is a software application identified by a URI, whose interfaces and bindings are described and discovered by XML artifacts. It is capable of directly interacting with other software applications using XML-based messages via Internet protocols. Grid services offer a new and evolving paradigm for building

grid applications. They represent the logical evolution from object-oriented architecture to service-oriented architecture, exposing application capabilities as reusable services and structuring service directories and repositories [11, 13].

There have been initial building blocks for programmatic access to web services, through standards such as SOAP, WSDL, UDDI, and through distributed computing web services platforms such as .NET, WebSphere, and E-Speak. These results and experiences can be borrowed for the development of grid services, where the network-enabled entities are treated as services that provide some capability through the exchange of messages.

This paper is concerned with providing support for constructing and automating composite grid services. Composition of grid services combines the offerings of two or more grid services to develop a grid application that achieves the desired computing goals. Based on a specified computing process model, a service composition involves multi-VO interactions, in which grid services collaborate with each other in the way defined by the computing process. To our knowledge, there has not been much work in this important field.

Although approaches to providing support for specification and automation of process-based composite web services have proposed in the software engineering research [1], many issues remain largely unresolved, such as service composition model, service composition automation and the performance analysis of services composition. Typically, the specifications (or languages) of composition model are the key of most service compositions, which are supposed to support service dynamic composition, matching and message mapping. Flexible and effective execution management is the goal of service composition automation. Furthermore, new transaction protocol and exception handle are should be considered. The performance analysis includes service composition evaluation, monitoring, simulation and optimization in order to improve overall performance. Therefore, applications based on grid service, driven by business processes, also need a composition description model that covers the entire spectrum from purely abstract process design to automatic process execution.

In this paper, we propose the Service Composition Graph (SCG) as a high level visual model for constructing grid services. Due to the fact that graphs are a very natural way of describing complex interactions in an intuitive level, visual composition facilitates the interactive construction of running service based applications by the direct manipulation and interconnection of visually presented services. The compositions of the services are controlled by a set of rules defined by SCG grammar, which help eliminate errors such structural conflict and type mismatch. Combining with graph transformation, the execution of composite services described by SCG can be automated. The proposed approach is a first attempt of supporting composite services construction and automation by combining the graph grammar and graph transformation theory [6]. SCG complements the existing work on composite service specifications in the sense that it provides a higher-level, graphical modeling framework with executable semantics.

In the following, we focus on describing the SCG grammar, which defines construction method of service composition and the SCG transformations, which specifies the execution semantics of the service composition. The rest of the paper is organized as follows. Section 2 provides an overview of the framework for visual construction and automation of service compositions. Section 3 illustrates the concept of SCG. The meta-model of SCG for service composition construction is presented in

section 4. Section 5 details the automation of service composition based on SCG transformation. Section 6 presents a brief review of related works. Finally, section 7 concludes this paper with a summary and discussion on our future work.

2 Grid Service Composition Framework

The objective of the framework is to provide support for the whole process of Grid service composition ranging from composition specification, design, verification, and the ultimate concrete execution. As illustrated in Figure 1, the framework consists of the following components:

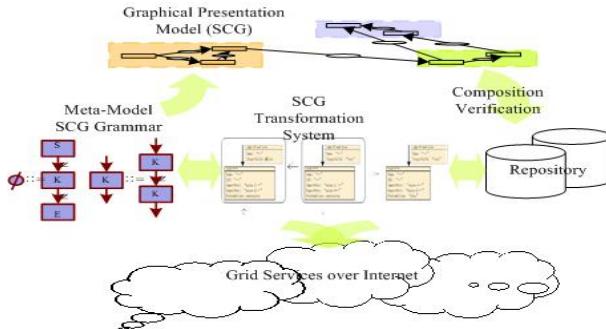


Fig. 1. Service Composition Framework

- 1) *Graphical Presentation Model.* Describing service compositions with text alone is too ineffective and error-prone. Service Composition Graph (SCG) provides a visual approach, in which an attributed, directed, and acyclic graph is used whose nodes represent participating services and whose edges describe the control and data relationships among the services.
- 2) *Meta-Model for Composition.* In composing services into ultimate service grids, rules are needed for handling sequential, concurrent and etc. behaviors. We use SCG grammar as an intuitive (graphical) means to describe how services are assembled during the design. The meta-model of SCG defined the grammar rules for service composition.
- 3) *Composition Verification.* Service composition verification ensures that the connections between the imported services are type safety and that overall structure of the composite service is well formed. The verification takes place at both the operational level and the service level [7]. At the operational level, input or output messages of the services must be matched with linked services; at the service level, the execution sequence should be a valid one. In the context of this paper, this means that a SCG satisfies the following three properties: reachability, liveness and deadlock-freedom.
- 4) *SCG Transformation System.* By assigning operational semantics to a SCG, a service composition based on the SCG is made executable. SCG transformation defines a set of productions that specify the SCG operational semantics. The SCG transformation system enables the interpretation and execution of a SCG.

- 5) *Repository.* The repository stores SCG specifications and other types of service composition specifications, as well as the instances of SCGs. At the design time, service composition specifications either composed by user or downloaded from public repository (UDDI center) will be put into repository. At run time, the repository is accessed by SCG transformation system, in order to get specifications and save instance data.

3 Service Composition as Attributed Graph

A composite service is composed of a set of services, which interact with each other for achieving a specified goal. To model a service composition, a SCG is defined as an attributed, directed and acyclic graph with labeled nodes and edges. Nodes represent constituting services and edges represent the interactions among the services. Labels can be either fixed or mutable. Fixed labels represent the *types* of nodes and edges used for structuring a SCG. Mutable labels are *attributes* used to store properties of a graph. A type can be a string and attributes are specified by several attribute tuples consisting of an attribute name, an attribute (data) type and an attribute value.

SCG has the following properties and restrictions:

- 1) Each node is typed by a vocabulary of node types. A node can hold exactly one type. Each node is attributed by signature. A node can hold multiple attributes. There are three types of nodes: *atomic*, *iterative* and *nesting* nodes.
- 2) Each edge is either a *control* edge or a *data* edge, or both. A control edge is attributed by transition conditions, while a data edge is associated with a part of message. Control edges are used to build a control structure of a composite service.
- 3) Each node may have an incoming port and an outgoing port for control edges, and multiple data ports for data edges. A data port is attributed by an input message or an output message, which is defined according to operations of corresponding service.
- 4) Control edges adjoin nodes through incoming and outgoing ports of nodes. There is only one control edge between an outgoing port of one node and an incoming port of another node. All data edges connect a data port of one node with a data port of another node. A data port can be the target of multiple data edges.
- 5) If a node has no incoming port, the node is called a source node; if a node has no outgoing port, the node is called a sink node.

Definition 3.1. Services Composition Graph (SCG). Let $\Omega = (\Omega_v, \Omega_e)$ be a pair of label alphabets for nodes and edges and a signature $Sig = (S, OP)$. A SCG $G = (G_v, G_e, s^G, t^G, lv^G, le^G, G_A, av^G, ae^G)$ is a Sig -attributed graph [6], where:

- 1) $\Omega_v = \{\text{atomic, nesting, iterative}\}$ and $\Omega_e = \{\text{data, atomic, and-split, and-join, xor-split, xor-join}\}$. We denote $\Omega = \Omega_v \cup \Omega_e$.
- 2) $S = \{\text{string, address, mesg, condition, validity, SCG}\}$. The *strings* are used for nodes and edges Names. The sort of *address* is used for service location (URI). The *mesg* is used for ImportMesg and ExportMesg. The *condition* sort is used for PreCondition, PostCondition and TransCondition, while the *validity* sort is

used for indicating TargetValid of edge transition condition. Finally, *scg* is a reference of subgraph of SCG.

- 3) G_v is a set of nodes, G_e is a set of edges, $s_g, t_g: G_e \rightarrow G_v$ are the source and target functions, and $lv_g: G_v \rightarrow \Omega_v$ and $le_g: G_e \rightarrow \Omega_e$ are the node and the edge labeling functions, respectively.
- 4) G_a is a Sig-algebra [6]. $av_g: G_v \rightarrow U(G_a)$ and $ae_g: G_e \rightarrow U(G_a)$ are the node and the edge attributing functions, respectively.

Figure 2 shows a node and an edge with their attributes in a SCG. The attributes URI, ImportMesg, and ExportMesg in a node are mapping to the corresponding WSDL descriptions of a web service.

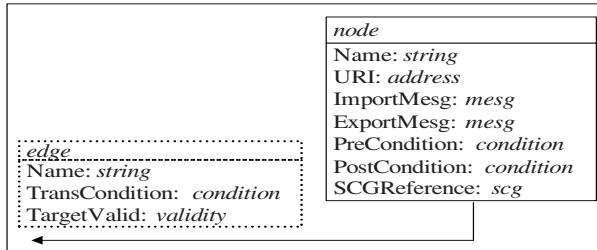


Fig. 2. Attributes of SCG Nodes and Edges

A *production* describes the transformation of a graph to another graph, through adding, deleting and preserving nodes and edges. A production comprises two graphs: a *left-hand side* graph and a *right-hand side* graph. A partial graph morphism [6] exists between the production's left-hand and right-hand side. The application of a production to a graph G requires a *match*, which is a total morphism [6] $m: L \rightarrow G$ from the production's left-hand side L to G (called the host graph - the graph to be transformed.) It is possible that more than one match exists. A match marks the part of graph that participates in the production application in the host graph. The remaining part of graph G is called *context*. With attributed and typed graphs, the attribute tuples of the production's left-hand side need to be matched as well. As a matter of fact, this depends on the attribute data type. In the transformed graph, the attribute values are evaluated depending on the productions' right-hand side, resulting in a constant value.

There are various ways to apply productions to graphs, and to control the process of iterated production applications. In this paper, we use an algebraic approach, which is relatively simple and easy to understand. In general, the application of a production to a graph defines a binary relation on graphs, which can be iterated arbitrarily yielding the derivation process. In this way, a set of productions defines an operational semantics. If one starts in a particular initial graph and collects all derivable graphs with only labels from a terminal label alphabet, one gets the notion of a *graph grammar* with its generated graphs. It is often the case that the application of a production depends on the context, i.e., it only takes place if additional nodes and edges are present or if a certain graph pattern is not present in host graph. Respectively, they are respectively named *positive* and *negative application conditions*. Arbitrarily context graphs may be attached to the left-hand side of a production, and their existence may be required or prohibited.

4 Service Composition Construction

The meta-model of service composition is set of SCG grammars, which form the guideline for constructing service composition represented as SCGs. The following illustrates the constructing and attributing grammars of SCG respectively.

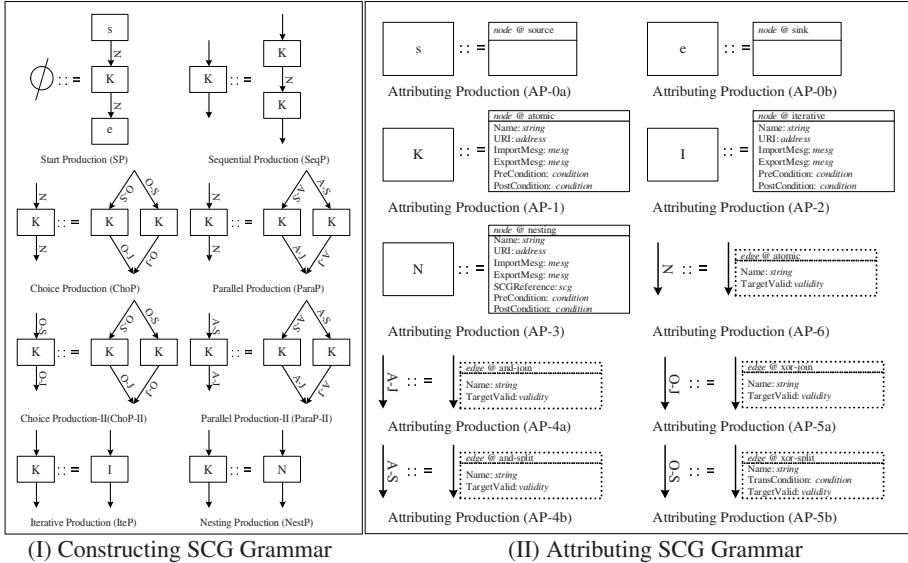


Fig. 3. Meta-model of Service Composition

Figure 3(I) shows a slice of the graph grammar for SCG, containing the productions used for SCG structure construction. The Start production replaces a \emptyset graph by two *terminal* nodes (start and end nodes, denoted by the lowercase letters s and e respectively) and one *nonterminal* node (denoted by the capital letter K , I and N), which are connected through directed edges. The Sequential production describes the basic flow structure and defines the sequential execution order of the service nodes occurring in the left-hand side, where edges without a label can match any labels. The Parallel production is used to describe concurrent flow within a SCG. The Choice production is used to build mutually exclusive alternative flow in SCGs, where only nonterminal nodes connected by two N -labeled edges can be replaced with a pair of nodes. The Iterative production replaces a K -labeled node with I -labeled node, which represents the repetition of a group of services until the exit condition is fulfilled. The Nesting production supports the notion of sub-flow, by which the execution of N -labeled nodes will trigger the execution of a subgraph of the SCG. We can also extend the Parallel (or Choice) production to support more than two parallel (or alternative) flows in a SCG.

Figure 3(II) shows the attributing SCG grammar containing the productions that mainly deal with the attributes of nodes and edges in a SCG. They ensure that, after being applied, a SCG will be well formed by the SCG definition.

There are five node attributing productions and edge attributing productions, respectively. Attributing productions AP-0a and AP-0b type the source node and sink node, respectively. Attributing production AP-1 replaces a nonterminal nodes labeled with K by an atomic node attributed with name, URI, ImportMesg, ExportMesg, PreCondition and PostCondition. At the same time, the nonterminal label K is discarded. Attributing production AP-2 is identical to AP-1 except that it replaces I -labeled node with a node typed as an iterative node. Attributing Production AP-3 replaces a N -labeled node with an additional attributed representing a sub-SCG. Five edge attributing productions are defined, one for each of the five types of edge labels. Especially, defined by P-5b, for an edge with an O - S label, the edge should be added an attribute of TransCondition. All other edges are attributed with name and TargetValid.

5 Services Composition Automation

In the previous sections, we have described how to model a service composition as a SCG. In this section, we study how to automate the execution of a service composition represented by a SCG. Generally, the execution of a service composition can be model as a sequence of SCG transformations. SCG transformation productions describe how to interpret the execution semantics of SCG.

Previously, we described transformations that can occur under positive application conditions, in terms of the existence of certain nodes, edges and their attributes. It is also possible to specify negative application conditions for transformation productions. The general idea of negative application conditions is to allow the left-hand side of a production to contain not only one, but several graphs, related by total

morphisms in the form $L \xrightarrow{l} \hat{L}$. Each such morphism is called a constraint, where L is the original left-hand side and $\hat{L} - l(L)$ represents the forbidden structure. A match satisfies a constraint if it cannot be extended to the forbidden graph \hat{L} .

For a typed and attributed SCG, its interpretation is defined by applications of the SCG productions that modify the attribute values and even the structure of the graph. Figure 4 shows all transformation productions for SCG interpretation.

The match of the left hand side of a production includes: the structure of SCG, node and edge types, and node and edge attributes' values. If one node (or edge) is typed as “*”, it means that this node (or edge) can be matched by any type of node (or edge). Transformation production TP-1a initiates SCG transformation by setting the outgoing edges' TargetValid value to “true”. On the other hand, transformation on a SCG instance is finished when transformation production TP-1b is applied.

The three productions TP-2a, 2b and 2c deal with all other incoming edges of a SCG instance. More attention should be paid to TP-2c, which contains a negative application condition, i.e. TP-2c can be applied to an and-join typed node only if this node does not contain any other incoming edge with “false” value TargetValid attribute. Indeed, application of TP-2c acts as a synchronizer for all and-join typed edges. TP-2a means when any type of node with an atomic typed incoming edge, its PreCondition attribute can be set to “true”. Same as TP-2a, TP-2b will be matched by xor-join typed edges. At this point, we need to point out that SCGs generated by a SCG grammar \hat{G} do not contain a node with different types incoming (or outgoing)

edges. This is why the productions in Figure 4 are adequate for interpreting these SCGs.

All of other outgoing edges are interpreted by TP-3a, 3b and 3c. If PostCondition attribute of a node is “*true*”, whose atomic (and and-split) outgoing edges’ TargetValid attributes will be updated by TP-3a (and 3b) with “*true*”. When the outgoing edge is typed with xor-split, however, the TargetValid attribute will be set according to the expression of the TransCondition attribute.

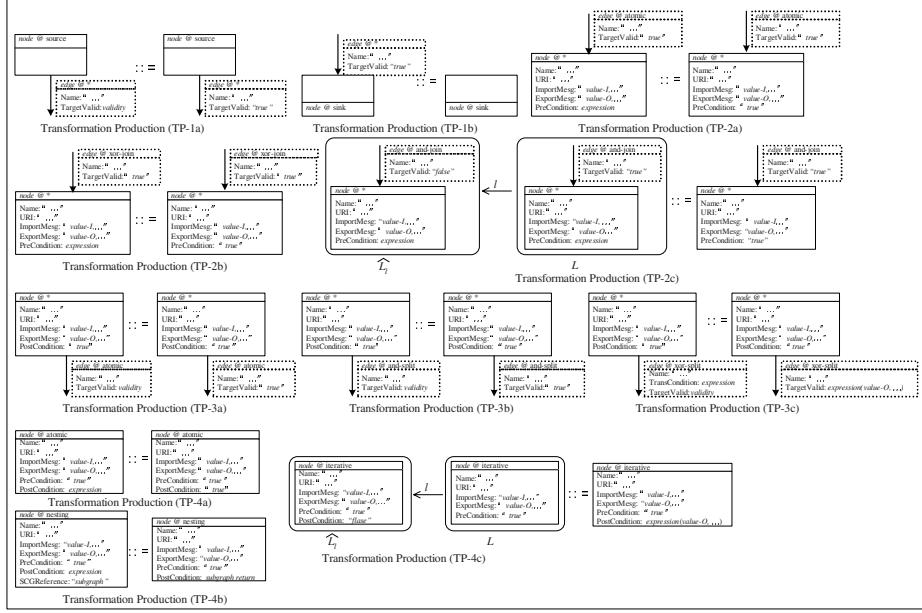


Fig. 4. SCG Transformation Productions

Finally, three types of nodes are interpreted by transformation production-4a, 4b and 4c. PostCondition attribute of a node can be set to “*true*” only if the execution of represented service is finished, i.e. result of these services is obtained. An atomic node’s PostCondition value is set to “*true*” after the corresponding service is performed, shown by TP-4a. If the node is typed nesting, its PostCondition value can be set to “*true*” after the completion of the corresponding SCG subgraph. TP-4c deals with iterative a node. When the expression of PostCondition is evaluated to “*true*”, the service will stop execution.

We are also developing a prototype of the proposed framework, including a visualization toolset (SCG Toolset). The prototype is built using off-the-shelf user-interface and graph editing software packages. The prototype allows us to study the feasibility of the proposed SCG approach to designing and automating composite services.

The facilities provided by the SCG Toolset include the following: The SCG editor provides a platform for SCG grammar directed graphical editing and visualization. The SCG verifier performs the SCG correctness check. The SCG transformation engine is the core of the total architecture, which consists of service invocator,

configuration manager, state manager and a generic graph transformation kernel, named AGG (Attributed Graph Grammar system) [8]. In our prototype, we only use the internal graph transformation engine of AGG in high-level service composition applications based on attributed graph transformation. As for the service invocator, it is a broker in charge of invoking services and receiving the execution results. The configuration manager determines what productions can be used by AGG. It allows us to easily implement composite service dynamic reconfiguration, because productions can be substituted dynamically at run time. The state manager, which is related to AGG and repository, stores the instances of SCG and related data. Figure 5 depicts a snapshot of SCG toolset.

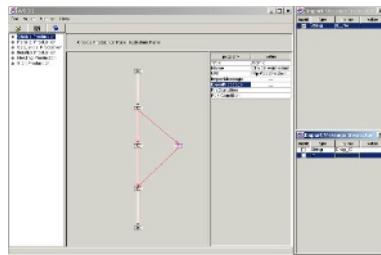


Fig. 5. SCG Toolset Snapshot

6 Related Work

Several specifications have been proposed for specification of service composition [17], including WSFL and XLANG, ebXML BPSS and BPML. However, these specification languages are of textual forms without adequate semantics constraints, and specifications written in them are difficult to understand, verify and visualize. Also, there is no provision for automatic execution based on the specifications.

There are several approaches proposed for supporting composite services at a higher level, including XL [9] and SWORD [16]. XL is an XML programming language for service specification and composition. It looks more like a traditional programming language, which is used in describing services and provides declarative constructs for service composition. XL requires the user to program with tedious XML text and does not consider much about main service related standards, such as WSDL and UDDI. Services in SWORD are represented by rules that express that given certain inputs, the services is capable of producing particular outputs. To create a composite service, the user specifies the input and output of the composite service. Using a rule-based expert system, SWORD will then generate a composition plan for the composite service. Although this automatic approach to services composition is attractive, its application is limited to specific application domains.

Graph grammars [15] have been used in a wide range of applications, including software specification and development, database design, parallel and distributed systems modeling, visual languages. In particular, the use of graph transformations techniques as formalisms of software architecture definition seems very promising [14]. Software architecture is defined as graphs and architecture style is specified by graph grammars. Furthermore, dynamic architecture evolution is defined as

conditional graph transformation. In [18], graph transformation is also proposed for visual modeling distributed object systems. In contrast to constructor-based component concepts for data type specification techniques, the component framework presented in [7] is based on a generic notion of transformation. Transformations are used to express intra-dependencies, between the export interface and the body of a component, and inter-dependencies, between the import and the export interfaces of different components. These works provide the sound basis of our work on SCG graph grammars and transformation for composite services.

Although Petri net is also used to describe and analyze distributed systems and there exists tools support for many different Petri nets, generally, graph grammar can be regarded as a proper generalization of Petri nets [2, 5], where the state of a system is described by a graph instead as by a collection tokens. A grammar-directed Petri net construction and transformable firing rules are not yet supported by many Petri net tools. On the contrary, we present a formal approach based on graph grammar and transformation supporting grammar-directed service composition and dynamic transformation productions (rules) chosen.

In this paper, we propose SCG as a graph based service composition model, combining the advantages of existing works on using flows for web service composition specification [12] and on generic process modeling techniques are proposed [20]. Using a formal, graph grammar based approach; SCG is assigned not only precise syntaxes but also semantics, which are lack in traditional workflow modeling languages.

7 Conclusions and Future Work

In this paper we have proposed SCG as a formalism for constructing and automation of composite Grid services. The graph-based approach has many advantages. Graphs are commonly used as a means to convey the general organization of an application. Their formal definition and its role in providing a clear separation between the computation of the individual entities and their coordination make it easier to check global properties of the system. In particular, properties about the control and data flows in a composite service can be derived from the composite meta-model. This is of prime importance for enforcing the imposed requirements

Although the work is still in progress, we have demonstrated the advantages of SCG. First, comparing with existing works, as a graphical approach, SCG is a more intuitive model for composite services. Second, we have developed a simple but powerful SCG grammar, which can be used to guide the process of composite service construction and to help eliminate some structural errors. In addition, with SCG transformation, the semantics of SCG can be interpreted precisely, which enables us to design a support system that not only provides graphical modeling facility but also automates the execution of composite services.

Our future work includes completing the prototype with the support for debugging the execution of SCG applications. We will also develop more complex example composite service applications using the prototype.

Acknowledgement. This research is partially supported by the Hong Kong Polytechnic University under the research grant H-ZJ80.

References

1. S. Aissi, P. Malu and Krishnamurthy Srinivasan, "E-Business Process Modeling: The Next Big Step", *IEEE Computer*, IEEE Press, Vol.35, No.5, (2002) 55–62.
2. R. Bardohl, C. Ermel and J. Padberg, "Formal Relationship between Petri Nets and Graph Grammars as Basis for Animation Views in GenGED", In the Proceeding of *International Conference on Design and Process Technologies* (IDPT 2002), Pasadena, USA, June 2002.
3. Z. Cheng, "Incorporating Agent Behavior into Web Services", Proceedings of the *40th Annual ACM SouthEast Conference*, ACM, 2002, pp. 87–96.
4. E. Christensen, F. Curbera, G. Meredith and S. Weerawarana, World Wide Web Consortium, "Web Service Description Language (WSDL1.1)", March 2001.
5. A. Corradini, "Concurrent Computing: from Petri Nets to Graph Grammars", *Electronic Notes in Theoretical Computer Science*, Vol. 2, Elsevier, Sept. 1995.
6. H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, A. Wagner and A. Corradini, "Algebraic Approach to Graph Transformation", In G. Rozenberg (Eds.): *Handbook of Graph Grammars and Computing by Graph Transformation*, World Scientific Publishing, 1997, pp. 247–312.
7. H. Ehrig, F. Orejas, B. Braatz, M. Klein and M. Piirainen, "A Generic Component Framework for System Modeling", In R.D. Kutsche and H. Weber (Eds.): Lecture Notes in Computer Science 2306 Springer 2002, pp. 33–48.
8. C. Ermel, M. Rudolf and G. Taentzer "The AGG Approach: Language and Tool Environment", in the *Handbook of Graph Grammars and Computing by Graph Transformation*, Volume 2, Applications, Languages and Tools, World Scientific, 1999.
9. D. Florescu, A. Grünhagen and D. Kossmann, "XL: An XML Programming Language for Web Service Specification and Composition", In the Proceeding of the *Eleventh International World Wide Web Conference (WWW2002)*, Hawaii, USA, May 2002.
10. I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke, "Grid Services for Distributed System Integration", *IEEE Computer*, June 2002. pp. 37–46.
11. D. Gannon, et al., "Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications", Technical Report, Department of Computer Science, Indiana University.
12. F. Leymann, "Web Service Flow Language (WSFL1.0)", <http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>, May 2001.
13. F. Leymann, D. Roller and M.T. Schmidt, "Web Services and Business Process Management", *IBM System Journal*, Vol. 41, No. 2, 2002, pp.198–211.
14. D. L. Metayer, "Describing Software Architecture Styles Using Graph Grammars", *IEEE Transactions on Software Engineering*, Vol. 24, No. 7, July 1998, pp. 521–533.
15. J. L. Pfaltz and A. Rosenfeld, "Web Grammars", In the Proceedings of the *1st International Joint Conference on Artificial Intelligence*, Washington, May 1969, pp. 609–620.
16. S. R. Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition", In the Proceeding of the *11th International World Wide Web Conference (WWW2002)*, Hawaii, USA, May 2002.
17. D. Riordan "Business Process Standards for Web Services", <http://www.webservicesarchitect.com/>, 2002.
18. G. Taentzer, "A Visual Modeling Framework for Distributed Object Computing", In V.B. Jacobs and A. Rensink (eds.): *Formal Methods for Open Object-based Distributed Systems* Kluwer Academic Publishers, 2002
19. S. Tuecke et al., "Open Grid Services Infrastructure v1.0 (Draft 29, April 5, 2003)", <http://www.ietf.org/ogsi-wg>
20. Workflow Management Coalition, Interface 1: Process Definition Interchange Process Model, Version 1.1, Document Number WfMC TC-1016-P, 1999.

Algorithmic Skeletons for Metacomputing

Martin Alt and Sergei Gorlatch

Technische Universität Berlin,
Germany

Abstract. Metacomputing means exploiting computer resources that are distributed over the Internet in a transparent, user-friendly way. It is motivated by the requirements of cooperative, computation-intensive, time-critical applications. The recent development of metacomputing systems has led to so-called *Computational Grids*, whose heterogeneity poses new challenges in software development. In this paper, we propose a novel approach to programming Grid systems using special software components called *skeletons*. We describe a prototypical implementation in Java and RMI and study the performance on a Grid consisting of multiprocessor clusters.

1 Introduction and Related Work

Metacomputing means exploiting computer resources that are distributed over the Internet in a transparent, user-friendly way. It is motivated by the requirements of cooperative, computation-intensive, time-critical applications. The recent development of metacomputing systems has led to so-called *Computational Grids* that combine different kinds of computational resources connected by the Internet and make them available to a wide user community.

Initial research on metacomputing and Grids focused on developing the enabling infrastructure, systems like Globus being a prominent example [5]. Other efforts have addressed important classes of applications and support tools for them, like NetSolve [3]. The aspects of algorithm and program development seem to have been given scant attention at this early stage of Grid research and are therefore not yet properly understood. Initial experience has shown that entirely new approaches to software development and programming are required for the Grid. The GrADS [4] project was one of the first to address this need. A common approach to developing metacomputing applications is to provide libraries on high-performance servers, which can be accessed by clients using remote invocation mechanisms (e. g. RPC/RMI). There are several systems, e. g. NetSolve [3] and Ninf [7], that adopt this approach.

We propose addressing the problem of programming the Grid by providing application programmers with a set of reusable algorithmic patterns, called *skeletons*, that are used as generic components, customizable for particular applications. Computational servers in the Grid may provide different, architecture-tuned implementations of the skeletons. Applications composed of skeletons can thus be targeted for execution on particular servers in the Grid with the goal of achieving better performance.

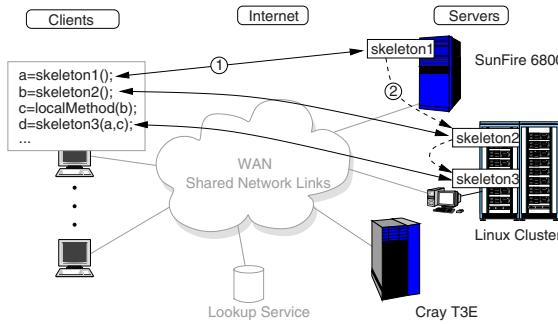


Fig. 1. Experimental Grid system: architecture and interaction

The particular contributions and organization of the paper are as follows:

- We present our prototype Grid environment, which serves as a proof-of-concept programming system and a testbed for experiments (Section 2).
- We give an overview of our implementation using Java RMI (Section 3).
- We describe the implementation of the Grid servers for different hardware architectures and report experimental results. (Section 4).
- We discuss our results in the context of related work (Section 5).

2 Programming Model and System Architecture

We have implemented a Java-based prototypical programming environment for a Grid system consisting of three kinds of components, shown in Fig. 1: clients (left) and servers (right), and a central entity called “lookup service”, used for resource discovery.

The challenge of Grid programming lies in the fact that the system is highly heterogeneous and dynamic, with servers of different architectures frequently leaving and joining the system. Programs should therefore be potentially executable on as many different servers as possible. Moreover, Grid programs cannot rely on a fixed set of servers: instead, they must be able to adapt to a changing environment and make use of the resources they find available at runtime.

In our programming model, each server provides a set of library methods accessible to clients via RMI. Libraries might be specifically developed to be available through RMI, or existing Java libraries could be “adapted” by writing remote wrapper classes.

In addition to problem specific libraries, more general algorithmic components called skeletons are provided by the system, to allow for executing algorithms that can not be implemented entirely using available library methods.

When a program is executed on the client, calls to resource intensive library methods or skeletons are delegated to the Grid-servers. The methods on the servers are invoked via RMI (① in Fig. 1). When the result of one skeleton is passed as argument to another one (e.g. **skeleton1**’s result is used as parameter of **skeleton3** in the figure), the result is sent directly from the first to the second server (② in Fig. 1), bypassing the client. This is achieved by means of so-called future references discussed briefly in the next section.

2.1 Skeleton Examples

In this paper, we confine our attention to *data-parallel* skeletons whose parallelism stems from partitioning the data among processors and performing computations simultaneously on different partitions. For illustration purposes, we provide two examples:

Map: Apply a unary function f to all elements of a list:

$$\text{map}(f, [x_1, \dots, x_n]) = [f(x_1), \dots, f(x_n)]$$

Reduction: Combine the list elements using a binary associative operator \oplus :

$$\text{reduce}(\oplus, [x_1, \dots, x_n]) = x_1 \oplus \dots \oplus x_n$$

In the next section, we use these skeletons for implementing an application case study.

2.2 Case Study: Maximum Segment Sum

The Maximum Segment Sum (MSS) Problem is defined as follows: given a list of numbers, function mss finds the contiguous list segment, whose members have the largest sum of all segments and returns this sum. For example: $mss[2, -4, 2 - 1, 6, -3] = 7$ being the sum of $[2, -1, 6]$. Function mss can be expressed using a composition of skeletons as follows (cf. [6]):

$$mss = \pi_1 \circ \text{reduce}(\otimes) \circ \text{map}(\text{tuple}) \quad (1)$$

with \otimes defined on four-tuples as follows:

$$\begin{pmatrix} m_1 \\ i_1 \\ c_1 \\ t_1 \end{pmatrix} \otimes \begin{pmatrix} m_2 \\ i_2 \\ c_2 \\ t_2 \end{pmatrix} = \begin{pmatrix} \max(m_1, c_1 + i_2, m_2) \\ \max(i_1, t_1 + i_2) \\ \max(c_1 + t_2, c_2) \\ ts_1 + t_2 \end{pmatrix} \quad (2)$$

Intuitively, each tuple represents a list, with m being the list's maximum segment sum, i the maximum initial segment (i.e. starting at the beginning of the list) sum, c the concluding segment sum and t the total sum of the list. Applying \otimes to two tuples representing lists, the result represents the list obtained by concatenating the two lists.

Functions tuple and π_1 are defined as $\text{tuple } a = (a, a, a, a)$ and $\pi_1(a, b, c, d) = a$. Thus, the algorithm in (1) works as follows: in the first step, a list of four-tuples is created from the input list, applying the tuple function using the map skeleton. The main work is done using the reduction skeleton with operator \otimes , resulting in a tuple for the entire input list. The final result is obtained by applying projection π_1 to the tuple.

In the following sections, we will demonstrate how the skeleton-based program for mss is implemented on our Grid system, and provide time measurements for it.

3 System Implementation

3.1 Overview

The system sketched in Fig. 1 was implemented in Java, using RMI for communication. We use Java because it has two important advantages in the metacomputing context.

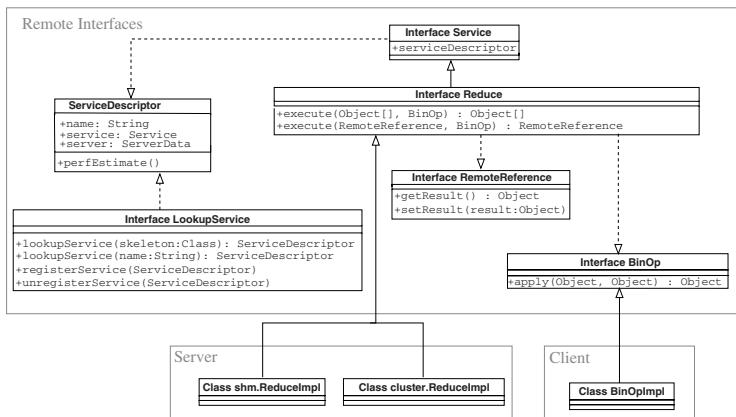


Fig. 2. Simplified class diagram of the implementation

First of all, Java bytecodes are portable across a broad range of machines. The method's customizing functional parameters can therefore be used on any of the server machines without rewriting or recompilation. Secondly, Java and RMI provide a simple, user-transparent mechanism for invoking methods remotely on the server.

The interaction between client, compute server and lookup server is realized in Java by implementing a set of remote interfaces known to all components. Fig. 2 shows a simplified class diagram for the most important classes and interfaces (solid lines connect interfaces and implementing classes, dashed lines denote the “uses” relationship).

Compute servers: For each skeleton or library a server provides, a corresponding interface is implemented. For example, in Fig. 2 the interface `Reduce` is shown for the reduce skeleton. Clients use this interface to call the skeleton on the server, where it is implemented by an object of class `ReduceImpl`. The implementation is highly architecture dependent as different servers may employ different implementations for the same skeleton (e.g. `shm.ReduceImpl` is used for a shared memory implementation and `cluster.ReduceImpl` for a cluster version).

The client provides the code for the operator to be used in the reduction, by implementing the interface denoted by `BinOp` in the figure. The necessary code shipping is handled transparently by the RMI mechanism.

The system is easily extensible: to add new libraries, an appropriate interface must be specified and copied to the codebase, along with any other necessary interfaces (e.g. functional parameters). The interfaces can then be implemented on the server and registered with the lookup service in the usual manner.

Lookup service: In our Java implementation, the lookup service has a list of `ServiceDescriptor`s (see Fig. 2), one per registered library/skeleton. A descriptor consists of the library's name and the implementing servers and a remote reference to the implementation on the server side. Clients and servers interact with the lookup service by calling methods of the `LookupService` interface shown in the class diagram: `registerService` is used by the servers to register their methods, and `lookupService` is used by the clients to query for a particular method.

3.2 Skeleton Invocation

Skeleton invocation is implemented using Java's RMI mechanism. This has the advantage that all parameter marshalling and unmarshalling as well as code shipping are handled transparently by the RMI system. One drawback, however, is the strict client-server model of communication that RMI provides. There are no means that allow two servers to communicate without involving the client. This results in a large time overhead for a composition of remote methods. For example, consider the following code:

```
r1=server1.skeleton1();
r2=server2.skeleton2(r1);
```

If the skeletons are called remotely, then the result of the first method would be sent to the client first and from there to the second server, even if both servers are the same.

To eliminate this overhead, we use RMI with so-called *future references*: an invocation of a skeleton on a server initiates the skeleton's execution and then returns immediately, without waiting for the skeleton's completion. As a result of the skeleton call, a remote reference is returned to the client, which provides two methods `setValue` and `getValue` (see Fig. 2). `setValue` is local to the server where the remote reference is created, and used to set the result as it becomes available. `getValue` is accessible remotely and used to retrieve the result. It blocks if the result is not yet available.

Thus, future references can be used as placeholder for the result and can be passed as an argument to the second skeleton. When the future reference is dereferenced (using `getValue`), the dereferencing thread is blocked until the result is available, i. e. the first skeleton actually completes. The result is then sent directly to the second server. After completion of `skeleton2`, the result is sent to the client. For further details about the future references used in the system, the reader is referred to [1].

4 Server Implementation

In our system, servers are not accessed by sending programs and data to the server for execution, instead servers can only be used through the services they provide. This has the advantage, that the application programmer does not have to consider the server hardware during program design, and the resulting programs are therefore hardware independent. This allows to choose servers based on performance and cost information, rather than on hardware considerations.

Using services to access a server allows to address many different hardware architectures in a uniform way. The only requirement that the server machine must fulfill is that it provides a Java virtual machine implementation. Besides, as the skeletons may contain functional parameters, the JVM must also support dynamic class loading (this part of the Java standard is not supported by all JVMs).

We have implemented a set of skeletons for two different hardware architectures: a shared memory SunFire server and a distributed memory Linux cluster. Our server implementations consist of two parts (cf. Fig. 3): a generic, hardware-independent *controller*, and a hardware-dependent *service object* for each available service (e. g. objects of class `ReduceImpl` as mentioned above).

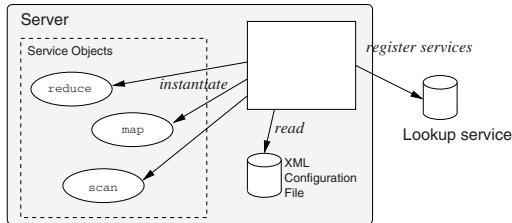


Fig. 3. Structure of the server implementation.

Controller: the controller is responsible for registering the available services (either libraries or skeletons) with the lookup service. The available services are described in a configuration file containing an XML entry for each service. The entries are added by hand when the service object is implemented. Each entry in the configuration file contains the name of the Java interface for the particular service, the name of the implementing service module and additional information such as documentation and performance information.

On startup, the controller parses the XML description of the available services, creates a set of service descriptors and registers these with the lookup service. In addition to the name of the service's interface and the information from the XML database, the service descriptors also contain the server's name and the IP address. When a client wants to use a specific service, it contacts the lookup service and receives an appropriate service request, containing the service's name. The client then contacts the controller of the server via RMI, requesting a remote reference to the corresponding service object. The controller creates a new instance of the service object and returns a remote reference to the client.

Service objects: The service objects provide remote access to the computational resources of the server. Each service module implements a Java RMI interface, providing remotely accessible methods to clients. For example, Fig. 2 shows the interface Reduce along with its corresponding service module `ReduceImpl`. Service objects can either implement the methods of the remote interface directly or delegate execution to appropriate local methods (e. g. to provide remote access to local library functions).

While the controller module is architecture independent, the service objects depend on the type of underlying hardware, so their implementations need to be tailored to the specific machine architecture. In the next two sections, we discuss our experimental implementations for two different architecture classes: shared memory and distributed memory. We outline the implementation using the reduce skeleton as an example.

4.1 Shared Memory Implementation

The only requirement for a particular implementation of services in our Grid system is that it implements the corresponding Java interface.

The shared memory version of the skeletons is implemented in a rather straightforward way, using Java's multithreading mechanism. When a service object is instantiated,

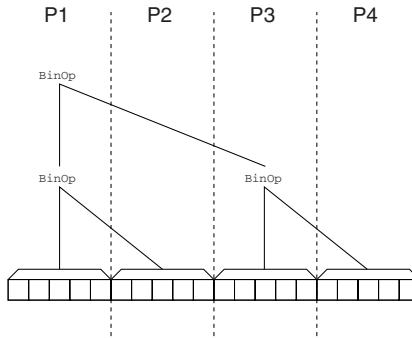


Fig. 4. Structure of the shared memory server implementation.

it first initializes a thread pool in order to avoid thread creation overhead when repeatedly using the skeleton. Besides, skeleton specific initializations are performed. For example, for the reduce skeleton, a barrier object is created which is later used to synchronize the threads. When the initialization is complete, a remote reference to the new service object is returned to the client.

When the service object's `execute` method is invoked, the method assigns work to the threads in the thread pool and coordinates their execution, until the result is available and returned to the client. For the reduce skeleton example, the `execute` method (cf. Fig. 2) first assigns each thread an equally sized part of the input list and starts their execution. It then blocks until the result is available.

Each thread performs the reduction on his local elements using the supplied `BinOp` operator. After that first local step, all threads synchronize using the barrier mentioned above. The partial results are reduced in parallel in a tree-like manner as shown in Fig. 4, until the result is available in one thread and is returned to the client.

Note that executing the `BinOp` operator provided by the client implies that the code for the operator is sent from the client to the server when the skeleton is invoked. This is handled transparently by Java's RMI mechanism.

Thus the shared memory implementation of skeletons is realizable easily using the means provided by the Java language. Only a few additional control structures such as synchronization barriers need to be implemented by the skeleton programmer.

4.2 Distributed Memory Implementation

For a distributed memory implementation, e. g. on a cluster, execution is divided between several nodes and thus between several Java virtual machines, each running in its own address space. The controller module remains the same as described above, running on the cluster frontend node that provides the connection between the clusters internal network (e. g. SCI) and the external network.

The implementation of the service objects, however, changes considerably as compared to the shared memory implementation. It is divided into two parts: a frontend module running on the frontend node that provides the interface to the rest of the system, and several node modules, executing computations on the cluster's nodes (cf. Fig. 5).

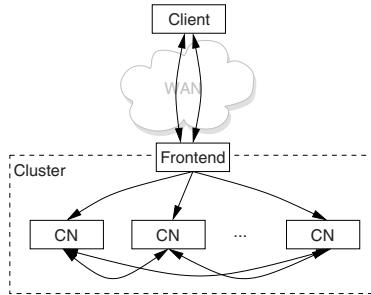


Fig. 5. Structure of the distributed memory server implementation.

The node modules for the different service objects are gathered in a single “node server”. The node servers are started by the controller before the server is registered with the lookup service. At startup, each node server opens a socket connection to the frontend to register itself; thereby the node becomes accessible for computations.

The task of the frontend is to communicate with the clients, distribute the received parameters to the server nodes, collect the results return them to the client. We will now provide more details using the implementation of the reduce skeleton as an example.

Frontend Module: The frontend provides a remote access to the reduction skeleton by implementing the Reduce Java interface as usual (see Fig. 2). The execute methods of the interface are used by the clients to start a reduction on the server. When execute is called, it divides the input list in sublists of equal length, one for each registered node server. The sublists are then sent to the nodes, along with a request to start execution of the reduction on the nodes. The method executed on the frontend then blocks until it receives the result from one of the nodes and finally returns the result to the client.

Node Server: After startup and registration with the frontend, the node server waits on the socket for a message from the frontend. When the parameters and the request for a specific skeleton is received from the frontend, the node server starts a local method corresponding to the called skeleton.

For the reduction example, a `reduceLocal` method is started, with the sublist and `BinOp` operator received from the frontend as arguments. The method performs a task very similar to the threads for the shared memory implementation. The first step is to reduce all local elements. Then the results are again reduced in a tree-like manner, with the node servers communicating results through sockets. When the final result is available in one node, this node contacts the frontend and delivers the result.

Class Loading: One problem of the distributed memory implementation of skeletons is that class loading and code shipping cannot be handled completely by Java’s RMI mechanism alone. When the parameters of an RMI call are sent from the caller to the remote server, Java also sends the URL of a so-called codebase. When an argument of the invoked remote method is an object of a class unknown to the server, the classloader automatically contacts this codebase URL to retrieve a class definition for the argument.

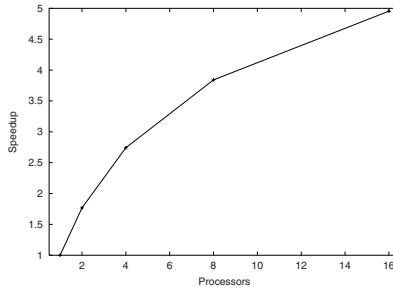


Fig. 6. Relative speedup for MSS with a problem size of 2^{15} .

While this mechanism works well for the shared memory implementation, it fails for the cluster version of the server for two reasons. First, RMI is only used for invoking methods of the service objects on the frontend, whereas communication between frontend and node servers is realized using sockets. Therefore, while any unknown classes are loaded automatically for the frontend node, that is not the case for the cluster nodes. However, even if RMI would be used for communication between frontend and node servers, the dynamic class loading would still not work on many clusters, because the cluster nodes usually do not have direct access to network address outside the cluster. Thus an attempt to contact the client's codbase URL from a cluster node would fail.

Therefore, we have implemented a distributed class loader running on both frontend and nodes. If a class is not found in the local classpath when a remote method is invoked on the frontend, the class loader contacts the codebase URL provided by the client to retrieve the class files for the unknown classes and stores them locally. In contrast, the standard RMI class loader would not store the class files locally, thus making it impossible to forward the class files to the nodes of the cluster. When a class definition is unknown in one of the node servers, the classloader of the node contacts the frontend to retrieve the classfile and load the class.

4.3 Experimental Results

For evaluating our concepts and implementations, we have measured the performance for the MSS example described in Sect. 2. We used a prototypical Grid system, structured as in Fig. 1. It consists of two university LANs – one at the Technical University of Berlin and the other at the University of Erlangen. They are connected by the German academic internet backbone, covering a distance of approx. 500 km. We use Berlin as the server side with a 16 node linux cluster (SCI interconnect and dual Pentium IV 1.7GHz nodes). The client-side role is played by Erlangen, where clients are running on SUN Ultra 5 Workstations with an UltraSparc-II processor running at 360 MHz.

The measured relative speedups are shown in Fig. 6. The obtained speedup decreases for larger processor numbers, because the measured runtime contains the communication times for sending the data and results from the clients to the server, and besides, for sending data between frontend and cluster nodes. Thus, the runtime contains a rather large non-parallelizable overhead, resulting in decreased speedup.

5 Conclusions and Future Work

Our work attempts to overcome the difficulties of algorithm design for Grids by using higher-order, parameterized programming constructs called skeletons. The advantage of skeletons is their high level of abstraction combined with an efficient implementation, tuned to a particular node of the Grid. Even complex applications composed of skeletons have a simple structure, and at the same time each skeleton can exemplify quite a complicated parallel or multithreaded structure in its implementation.

The experimental Grid system presented in our paper is still in an early stage and lacking many important services, such as authentication and resource allocation services. We plan to address these using sophisticated infrastructures, specifically developed for this purpose, e.g. as provided by the Globus toolkit ([5]).

Our current implementation of the skeletons on the servers provides ample room for optimization and improvements. One important topic of possible optimizations is load balancing. At present, work is distributed equally among the cluster's nodes. However, even for homogeneous clusters it can be advantageous to distribute different amounts of data on the nodes (cf. e.g. [2]). In the future, we intend to also use networks of (heterogeneous) workstations as servers, where load balancing is even more important.

Another possible area of improvement is optimization of intra-cluster communication. At present, all parameters and results are communicated between frontend and nodes for each skeleton called. However, for a composition of two skeletons on the same cluster, it can be advantageous to leave the results of one skeleton distributed across the nodes if it is an argument of the next skeleton call.

References

1. M. Alt and S. Gorlatch. Optimizing the use of Java RMI for Grid application programming. Technical Report 2003/08, Technische Universität, Berlin, March 2003. ISSN 1436-9915.
2. T. Altilar and Y. Paker. Optimal scheduling algorithms for communication constrained parallel processing. In *Euro-Par 2002*, LNCS 2400, Berlin Heidelberg, August 2002. Springer Verlag.
3. D. Arnold et al. Users' guide to netsolve v1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.
4. F. Berman et al. The GrADS project: Software support for high-level Grid application development. *Int. J. of High Performance Computing Applications*, 15(4):327–344, 2001.
5. I. Foster and C. Kesselmann, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.
6. S. Gorlatch and C. Lengauer. Abstraction and performance in the design of parallel programs: overview of the SAT approach. *Acta Informatica*, 36(9):761–803, 2000.
7. H. Nakada, M. Sato, and S. Sekiguchi. Design and implementations of Ninf: towards a global computing infrastructure. *FGCS*, 15(5-6):649–658, 1999.

Grid Services Performance Tuning in OGSA

Hechuan, Dubin, and Sanli Li

Grid Research Group.

CS Dept.

Tsinghua Univ.

hechuan97@mails.tsinghua.edu.cn

Abstract. In the next generation of Open Grid Service Architecture – OGSA, Grid has become a Service Oriented environment which is more open, more scalable and more flexible. However, the increase of flexibility will inevitably bring performance decreasing. And the performance problem increasingly constitutes a great challenge to the Grid Researching field. In this paper, we propose a time-based model in order to analyze the performance of Grid Services. On the foundation of this model, we give two methods – Grid Service Instance Pool and Caching on Grid Server – both of them can enhance the performance of Grid Services prominently. Moreover, we not only give theoretic explanation of those methods, but also carry out some experiments by using Globus 3.0 alpha 2 for verification.

1 Introduction

Grid, which is called the next generation of Internet, becomes more and more important in research realm[1]. OGSA[2], which combines the original Grid technology with current Web Services[3], is an open, scalable and flexible architecture of Grid. The most prominent idea of OGSA is “everything is a service”. In the framework of OGSA, various kinds of things including computers, programs, data and devices are all abstract to some kinds of Grid Services. Grid Services provide typical interfaces on such a high level that the services can be accessed and used more conveniently by the users or applications.

OGSA focuses on opening, scalability and flexibility, which will lead to an inescapable fall on the side of performance. The drop of performance is mainly caused by two reasons:

1) Grid Services are some kinds of Web Services who have Grid features [4], Grid Services also adopt SOAP-RPC which has been used in Web Services, and as we know, SOAP-RPC does not have a satisfactory efficiency.

2) The life time management in OGSA can solve the problem of resource releasing. However, that kind of management means when we call a Grid Service, a new instance of that Grid Service is created. If the service is used frequently, the creation of instances will sharply decrease the performance.

Performance tuning problem becomes a great challenge in Grid research. In this paper, we present a time-based model for Grid Services in order to analyze and solve the performance problem of Grid Services.

2 Time-Based Performance Model

There are many evaluation standards which can be used to measure the performance of Grid Services, such as: availability; accessibility; integrity/reliability; throughput; latency; regulatory; security and etc. In this paper, we mainly concern the latency.

2.1 Performance Model

Suppose a user called a Grid Service $Service_{[i]}$ through network, the Grid Service executes and returns result, the user receive the result in the end. We do not take into account how Grid Service life time is extended. During this period of time, the total cost of time – RT includes the following three parts:

1) Network delay: T_{tran}

The time used in transmission, including the transmitting time of both request and response. Network delay can be subdivided into four categories:

T_{syn} (*conn*): the time cost in establishing a connection to the server

T_{req} : the time cost from sending a request to receiving the request

T_{resp} : the time cost from sending a response to receiving the response

T_{ack} (*disconn*): the time cost in sending an ACK to the server and disconnecting from the server

$$T_{tran} = T_{syn}(\textit{conn}) + T_{req} + T_{resp} + T_{ack}(\textit{disconn})$$

2) Parsing messages: $Tparse$

T_{parse}^{req} : the time cost in parsing request XML messages

T_{parse}^{resp} : the time cost in parsing response XML messages

T_{dec} : the time cost in decoding request messages. For a Grid Service, it is the time cost in decoding SOAP messages.

T_{enc} : the time const in encoding response messages. For a Grid Service, it is the time cost in encoding SOAP messages.

T_{sche} : the time cost in flow control, this part of time is only useful when Grid Services contains transactions and coordination; while in other situations, $T_{sche}=0$ s.

T_{call} : the time cost in calling Grid Services. For a Grid Service, this part of time is the sum of the time cost in creating instances and the time cost in calling the instances.

$$T_{parse} = T_{parse}^{req} + T_{parse}^{resp} + T_{dec} + T_{enc} + T_{sche} + T_{call}$$

3) Executing time: $Texec$

Executing time of a Grid Service does not include the time of calling that service. This part of time is always the most time-consuming one.

Apparently, $RT=T_{tran}+T_{parse}+T_{exec}$. Fig 1 shows the time sequence when calling a Grid Service.

The Grid Services described in this time-based model is suitable for two situations:

- a) RT only changes in a threshold and the change is stable. Client can explicitly indicate the live time of that Grid Service and do not need to prolong the service's life time.

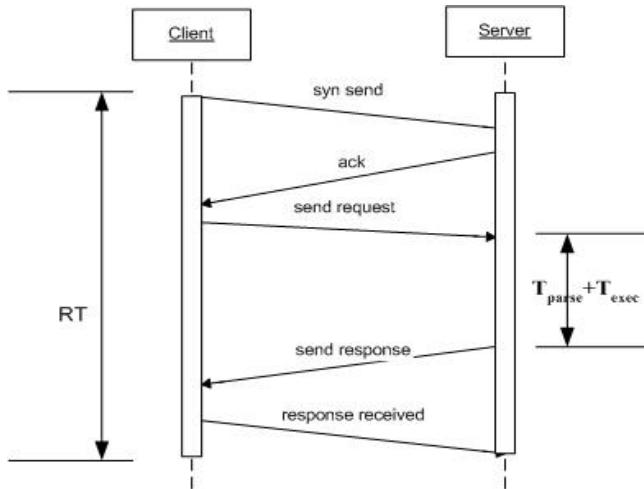


Fig. 1. Time sequence when calling a Grid Service

- b) T_{exec} has some kinds of relationships with input data such as $T_{exec} = f(\text{Input})$. Client can calculate or estimate RT by using that function, and then indicate the live time of that Grid Service.

Typically, those kinds of Grid Services include adjusting time, querying stock information, weather forecast and simple computing in science and engineering.

2.2 Evaluating Model

For a Grid Service $Service_{ij}$, we define following variables:

RT_{ij}^k : the response time of user k when accessing the service through network

N : the times of accessing the $Service_{ij}$ during a relatively long period – T

$Ave(RT_{ij})$: the average intervals of accessing the Service during T

$$Ave(T) = \frac{T}{N}$$

From users' aspect, the quality of $Service_{ij}$ can be measured by the parameters as follows.

- 1) $Ave(RT_{ij})$: The average response time of $Service_{ij}$

$$Ave(RT_{ij}) = \frac{\sum_{k=1}^N RT_{ij}^k}{N}$$

- 2) $Max(RT_{ij})$: The maximum response time of $Service_{ij}$

$$Max(RT_{ij}) = max(RT_{ij}^1, RT_{ij}^2, \dots, RT_{ij}^N)$$

3) $Normal(RT_{[i]})$

$$Normal(RT_{[i]}) = \frac{\sum_{k=1}^N (RT_{[i]}^k - Ave(RT_{[i]}))^2}{N}$$

$Ave(RT_{[i]})$ describes the response time of $Service[i]$. The smaller $Ave(RT_{[i]})$ is, the quicker is the response velocity. $Max(RT_{[i]})$ describes the worst response time of $Service[i]$. The more $Max(RT_{[i]})$ is similar to $Ave(RT_{[i]})$, the better $Service[i]$ is. And $Normal(RT_{[i]})$ describes fairness of response time. The smaller $Normal(RT_{[i]})$ is, the fairer $Service[i]$ is.

3 Methods for Optimizing the Performance

Based on the formula $RT = T_{tran} + T_{parse} + T_{exec}$, we can focus the concrete methods on decrease each of the three parts in order to optimize the performance.

3.1 Using Cache on Server to Reduce T_{exec}

Generally, the execution time of many Grid Services is very long. For example, the Grid Services contain large-scale computation or transforming a text to audio or video file. Here the cache on the Server Side is applicable in order to solve these questions. However, when the interval of accessing of a certain Grid Service, $Ave(T)$ is more less than $Ave(RT)$, it is not necessary to use cache. Using cache contrarily may make it even worse because of the cost of scheduling.

Fig 2 shows the process of a request from the client. The cache locates in the place where the instances of Grid Services are called. Using cache can reduce the executing time T_{exec} .

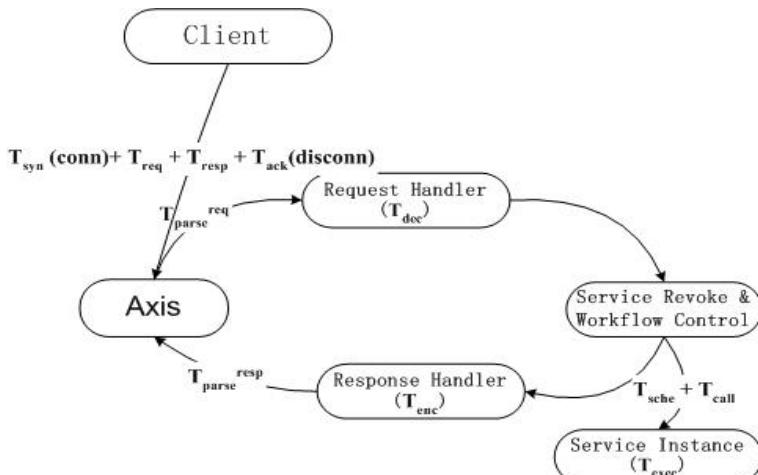


Fig. 2. The location of cache

We implement the cache by using a hash table. If the result of a client request is already in the cache, the result will return directly from cache. If the result is not in the cache, a new service instance will be created.

To guarantee the validity of caching information by using limited resources on the server, the cache must support some functions to clear the data which is old or not frequently used and keep the data which is visited frequently. Our cache is a self-managed cache. It has an upper limit. A background thread takes charge of maintaining the cache. Every item in the cache contains its expire time – T_{expire} . If there is no request for this item after T_{expire} , this item will automatically be destroyed by the managing thread. An item in the cache contains three parts: key (the index of that item), expire time, object (the content of that item).

3.2 Using Instance Pool to Reduce T_{parse}

It takes quite a long time for creating and destroying instances, for when an instance is created, it needs some system resources such as memory. The idea of using “Resource Pool” is avoiding creating instances frequently, especially for those instances whose creation needs a huge amount of resources. “Resource Pool” has already been used in many commercial products, for instance, in IBM’s WebSphere, in IONA’s Orbix 2000, in SUN’s Jini, in Microsoft’s MTS (Microsoft Transaction Server 2.0), COM+ and so on.

The ways of optimizing “resource pool” contains:

- 1) adding thread dynamically, like in Microsoft SQL Server
- 2) optimizing the amount of working threads, like in MTS
- 3) providing more than one thread pool, like in COM+
- 4) setting priority for every working thread
- 5) adding daemon thread in the thread pool

We adopt multi-pools strategy on the server for different kinds of Grid Services. For $Service_{ijp}$, we provide an instance pool – $Pool_{ij}$ – to serve it. There are some initialized instances whose amount is $Count_{ini}$ in every pool. And each pool has an upper limit $Count_{max}$ and a queue whose size is $Count_{queue}$.

Supposed that one user’s request only needs one service instance, the amount of all used instances is $Count_{used}$, we can describe the work flow of service instance pool as follow:

request instance

- 1) getting from pool directly $(0 \leq Count_{used} \leq Count_{ini})$
- 2) creating new instance and put into the pool $(Count_{ini} < Count_{used} \leq Count_{max})$
- 3) putting the request in to Queue $(Count_{max} < Count_{used} \leq Count_{max} + Count_{queue})$
- 4) unable to get instance, then notifying user $(Count_{max} + Count_{queue} \leq Count_{used})$

release instance

An instance will be put back into the pool when it is no longer used. If the amount of instances in the pool is equal to or bigger than $Count_{ini}$, then the instance should be released; otherwise, the instance will be held in the pool for other usages.

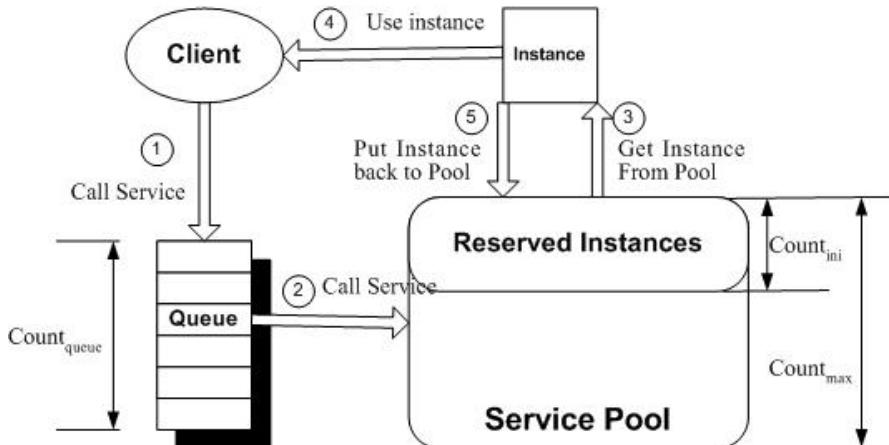


Fig. 3. Architecture of instance pool

In Fig 3, we present the architecture of our test instance pool and the work flow of that pool. Of course, maintaining instance pools will bring additional cost to the server, while experiments have showed the maintaining cost is far more less than the cost of creating and destroying instances. There are piles of papers which study instance pool such as [5] [6]. So we only give a relatively simple implementation of instance pools. We do experiments in the test instance pool, and study how the instance pool influences the efficiency of Grid Services.

4 Experiments and Analysis

4.1 Mini-Grid Test Bed

There are many toolkits which have already implemented some OGSA standards, such as Globus[7] – a toolkit mainly uses in developing computing grid and grid applications. Currently Globus has been applied in a lot of grid projects, such as TeraGrid[8], SFExpress[9] and Nimrod[10]. The next version Globus 3.0 is based on OGSA for developing the Grid Services. Another project, OGSI.NET[11], aims to build OGSI using Microsoft .Net. (OGSI refers to the basic infrastructure used to form OGSA whose kernel is “Grid Service Specification” which defines the interfaces and standards of deeds of Grid Service which is founded on the Web Service.)

As grid is consisted of diverse resources, we set up a mini-grid test bed environment which consisted of heterogeneous computers to do the experiments. There are three types of client running on Red hat Linux 8.0, Windows 2000 and Windows XP respectively. The server runs on Windows 2000 Server and has OGSA Alpha 2 environment.

4.2 Design of Experiments

Firstly, we define the running time of a certain Grid Service_{*ij*} as T_{exec} , and then we simulate the calls of clients which is according with Poisson distribution about N times per minute (N is respectively assumed as 10,20,30,...,100). Moreover, we design four different conditions as follows.

- 1) Condition A: no optimization at all.
- 2) Condition B: only introducing instance pools
- 3) Condition C: only introducing cache on server side(the hit probability of cache is about 30%)
- 4) Condition D: introducing both of instance pools and cache

In succession, we compare $Ave(RT_{ij})$, $Max(RT_{ij})$ and $Normal(RT_{ij})$ under the four conditions and draw charts to show the differences.

4.3 Result Analysis

According to Fig 4 and Fig 5, it can be concluded that both introducing instance pools and introducing cache both the efficiency of Grid Service, furthermore, the former is more effective on the optimization than the latter. However, it is also found from the charts that even though introducing cache could lead to performance improvement after introducing the instance pools, actually its effect is very limited. So it is questioned whether introducing cache on server side is necessary.

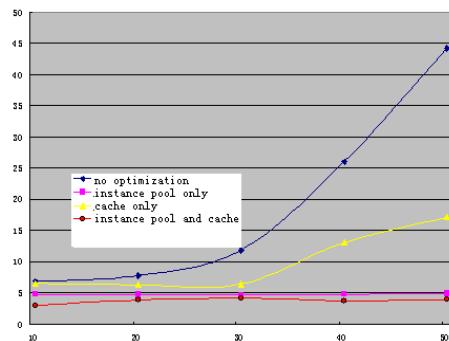
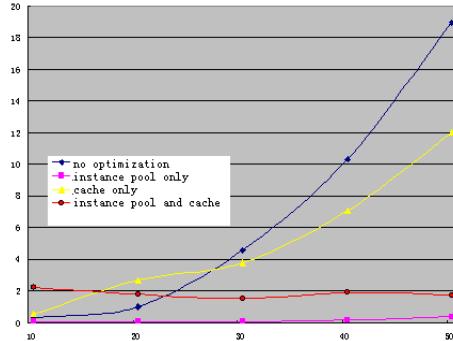
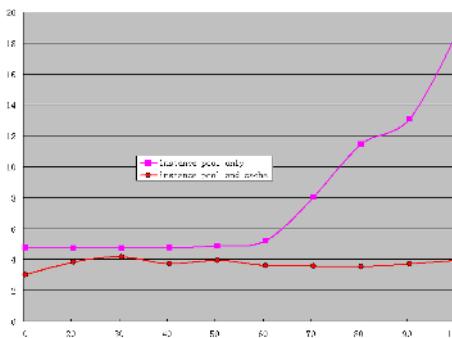
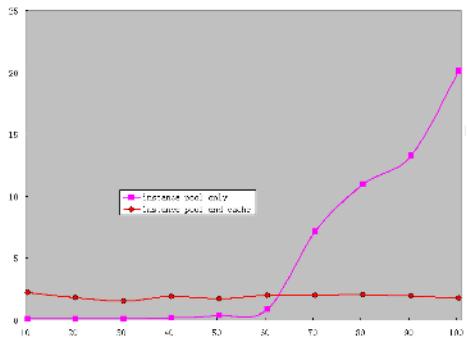
Fig 6 and Fig 7 are mainly about the comparision of the results under Condition B and Condition D along with clients' calls increasing. In last experiment, the frequency of the clients' calls is 50 times per minute, now it is increased to 100 times per minute. Then we compare $Ave(RT_{ij})$ and $Normal(RT_{ij})$ again and draw a conclusion that when the calls are few, introducing cache can not lead to great performance improvement. However, when the clients' calls increase greatly, introduction of cache can effectively optimize the performance. From the charts above, it is clear that in such conditions when the calls of services are raised, we get distinctive decreasing in $Ave(RT_{ij})$ and more stable $Normal(RT_{ij})$ by introducing cache.

5 Related Works

In this paper, we consider to improve the performance of Grid Services from software aspects. Because the performance of most applications does not show linear increase when we improve hardware ability. Currently, the research on improving the performance of Grid Services by using software techniques includes:

- 1) Optimizing in transporting

This kind of techniques mainly concerns how to decrease the transporting delay in the network— T_{tran} . Those techniques include improving traditional transmitting protocols and technology, compressing XML, using some special protocol which is faster, using asynchronous communication and etc.

**Fig. 4.** Optimization effect to Average**Fig. 5.** Optimization effect to Normal**Fig. 6.** Caching effect to Average**Fig. 7.** Caching effect to Normal

2) Optimizing for SOAP and XML

This kind of techniques mainly concerns how to decrease the time of parsing XML messages – T_{parse} . Those techniques include simplifying XML parser[12], improving SOAP or provide some methods without SOAP like SUN’s JAXB[13]. Globus provides high-throughput XML communication protocol using Grid interface, IBM’s Web Services Invocation Framework (WSIF)[14] provides the ability of dynamic invocation services.

3) Optimizing for frequently services call

This kind of techniques mainly concerns the problem of frequently services call. Those techniques include using cache on client side in order to avoiding frequent SOAP-RPC[15], establishing contract between clients and servers and etc.

6 Future Works

In the future, we want to consider the performance problem of Grid Services from other aspects such as server loading and do more experiments not only on Mini-Grid but also on Internet.

References

1. "The Grid: Blueprint for a New Computing Infrastructure", Ian Foster and Carl Kesselman, July 1998
2. <http://www.globus.org/ogsa/>
3. <http://www.w3.org/2002/ws/>
4. The physiology of the grid: An open grid services architecture for distributed Cysteros integrattn. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Technical Report, The Global Grid Forum. 2002 <http://www.globus.org/ogsa/>
5. Evaluating and Optimizing Thread Pool Strategies for Real-Time CORBA, Irfan Pyarali, Marina Spivak, Ron Cytron, Douglas C. Schmidt, Proceedings of the ACM SIGPLAN workshop on Optimization of middleware and distributed systems
6. Analysis of optimal thread pool size, Yibei Ling, Tracy Mullen, Xiaola Lin, ACM SIGOPS Operating Systems Review (April 2000), Volume: 34 Issue: 2
7. <http://www.globus.org>
8. The philosophy of TeraGrid: building an open, extensible, distributed TeraScale facility, Catlett, C. ,Cluster Computing and the Grid 2nd IEEE/ACM International Symposium CCGRID2002 , 2002
9. Data Logging and Visualization of Large Scale Simulations (SF Express) , Lucian Plesea and Laura Ekroot, Simulation Interoperability Workshop conference
10. Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid, Buyya, R.; Abramson, D.; Giddy, J. ,High Performance Computing in the Asia-Pacific Region, 2000. Proceedings. The Fourth International Conference/ Exhibition on , Volume: 1 , 2000 Page(s): 283–289 vol.1
11. <http://www.cs.virginia.edu/~humphrey/GCG/ogs1.net.html>
12. <http://www-106.ibm.com/developerworks/webservices/library/ws-quality.html>
13. Sun Microsystems. Java Architecture for XML Binding 0.7 Specification, 2002, JSR 31, <http://jcp.org/aboutJava/communityprocess/review/jsr031>
14. <http://www.alphaworks.ibm.com/tech/wsif>
15. <http://www.javaworld.com/javaworld/jw-03-2002/jw-0308-soap.html>

A Transaction Model for Grid Computing

Feilong Tang, Minglu Li, and Jian Cao

Department of Computer Science & Engineering,
Shanghai Jiao Tong University
Shanghai 200030, P.R. China
{tang-f1, li-ml, cao-jian}@cs.sjtu.edu.cn

Abstract. In Grid environment, transaction must have the abilities to coordinate both short-lived operations and long-lived business activities, which can not directly be supported by traditional distributed transaction mechanisms. This paper introduces a transaction model based on agent technologies to satisfy these requirements. The model can coordinate both Atomic transaction and Compensation transaction. The agents shield users from complex transaction process. For two types of transactions, we respectively analyze the coordination mechanisms and provide corresponding algorithms. We also discuss the implementation of this model and propose a security solution.

1 Introduction

The main goal of Grid computing is sharing large-scale resources, accomplishing collaborative tasks [1], where transaction will become more and more important. To date, however, few efforts have been performed for transaction in Grid environment.

In this paper, we present a transaction model for Grid computing. The contributions mainly include: (a) the architecture and algorithms for transaction in Grid environment; (b) abilities for users to execute transaction, without knowing of process.

2 Related Work

There have been some efforts about distributed transaction. [2,3] and [4], proposed by IBM, Microsoft, BEA and OASIS respectively, describe the transaction framework for Web services. However, they can not directly support for Grid transaction.

3 Architecture of Transaction Based on Agent

The transaction model, as shown in figure 1, is based on the services provided by Globus Toolkit [5,6,7]. What a user need do is only to start up a transaction and make desirable selection because the agent can create a TransactionManager (TM), start up or cancel transaction, manage transaction and inform the user of execution results.

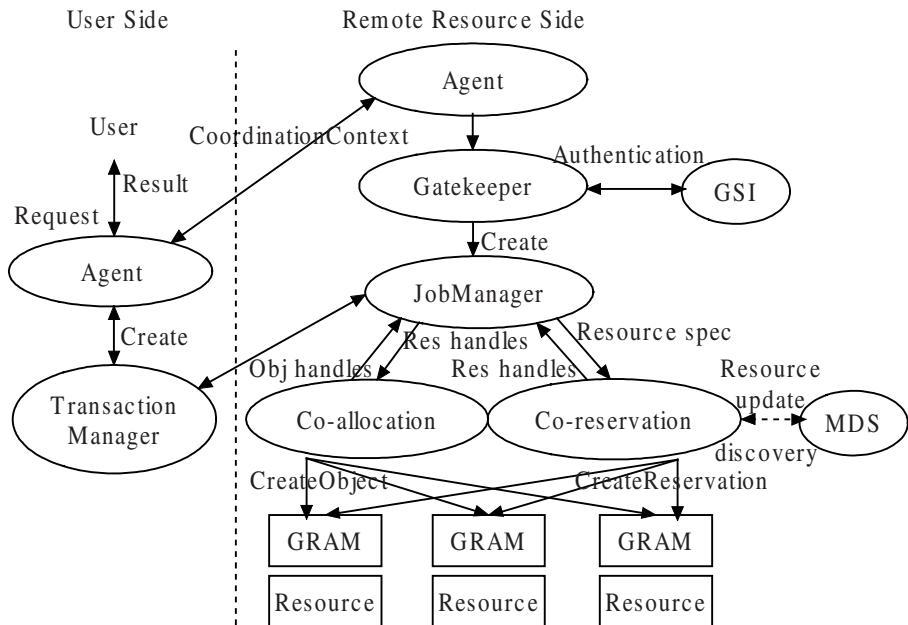


Fig. 1. An extensible transaction architecture for Grid computing

Comparing with transaction for Web services, where each service is statically created, our model adapts to the highly dynamic Grid environment. It can meet grid features and requirements because:

- It does not lock resources so as to improve the concurrency.
- It is robust in face of various failures and dynamic join and leaving of resource.
- It can coordinate both short-lived operations and long-lived business activities.

4 Coordination Mechanism

The model may handle two types of transactions, which can be selected by a user:

- Atomic transaction (AT). It is used to coordinate short-lived operations, where participants must commit synchronously and intermediate results are invisible.
- Compensation transaction (CT). It is used to coordinate long-lived business activities and allows some candidates to abort while others to commit.

4.1 Atomic Transaction Coordination

The agent responds the transaction request from a user by:

- Creating a TM, which serves as the coordinator.
- Sending CoordinationContext messages to all agents of participants.

We adopt reservation of resources to prepare for AT transaction. After receiving the Responses from participants, TM coordinates transaction in following way:

Phase1: TM sends Prepare messages to all participants P_i ($i=1, 2, \dots, N$). If P_i successfully reserves, the resource handles are returned to JobManager (JM), which reports Prepared message to TM. Otherwise, the NotPrepared message is sent to TM.

Phase2: Only if TM knows that all P_i have successfully reserved, it sends Commit to all P_i , which enables all JMs to request for allocating reserved resources, record the Commit information in log, monitor the execution of tasks and report result to TM. Otherwise, TM sends Abort to all P_i , making them cancel previous reservation. Within T_2 , if TM receives N Committed messages, it judges that transaction is completed and then returns final result to user. Otherwise, the TM reports failure to the user and sends Rollback to all P_i . The algorithm is as follows:

<pre>ActionOfParent begin agent creates TM; agent sends CoordinationContext to all agents of participants P_i; wait for Response from JM of P_i; if timeout exit; send Prepare to all P_i; while ($t \leq T_1$) and ($n1 < N$) wait & record incoming messages; if ($n1 = N$) and (all messages are Prepared) begin record commit in log; send Commit to all P_i; while ($t \leq T_2$) and ($n2 < N$) wait & record incoming message; if ($n2 < N$) or (not N Committed) begin send Rollback to all P_i; exit after receiving Rollbackeds; end else begin send Abort to all P_i; exit after receiving all Aborted; end end</pre>	<pre>ActionOfChild begin agent creates JM; send Response to TM; wait for Prepare from TM; if timeout exit; success:=JM reserves resources; if (success) begin send Prepared to TM; while ($t \leq T_3$) and (not Commit) wait for incoming message; if (message is Commit) begin allocate reserved resources; record transaction in log; commit transaction; send Committed to TM; end else begin cancel reservation; exit; end end end</pre>
--	--

In execution of a transaction, if any P_i itself contains sub-transactions, it will apply above mechanism recursively.

4.2 Compensation Transaction Coordination

After receiving the Response, the TM coordinates transaction in following steps:

TM sends Enroll messages, which contain timestamp T , to all candidates. The candidates reserved and allocated resources successfully record operations in log, then directly commit the transaction and return Committed, which contain execution results, to the TM. The candidates failed to reserve resources return Aborted message.

According to returned results, the user may do the following by means of the TM:

- For candidates committed successfully, he selects some and cancels the others by sending Confirm and Cancel messages to them respectively within interval T .

- For candidates failed, he needn't reply them and may renew to send Enroll requests to locate new candidates until success or attempting N times.
- Within T, the candidate that has committed successfully performs either:
- In case of receiving Confirm message, it responds a Confirmed message.
- In the event of receiving Cancel message or nothing, it automatically rollbacks the taken operations according its log record, which may be implemented by performing a compensation transaction, and then returns a Cancelled message.

ActionOfSuperior

```

begin agent creates TM;
  while (transaction doesn't complete)
    begin agent sends CoordinationCont-
      ext to agent of candidate;
      wait for Response from JM;
      send Enroll to all candidates;
      while ( $t \leq T$ ) begin
        wait & record incoming messages;
        if (message is Committed)
          if (user selects some candidates)
            begin
              send Confirm messages to them;
              wait Confirmed messages; end
            else begin
              send Cancel messages to them;
              wait Cancelled messages; end
            end
        end
      end
    end
end

```

ActionOfInferior

```

begin agent creates JM;
  send Response to TM;
  wait for Enroll from TM;
  if timeout exit;
  reserve and allocate resources;
  record transaction in log;
  commit transaction;
  if (commit successfully) begin
    send Committed to TM;
    while ( $t \leq T$ ) begin
      wait for incoming message;
      if (message is Cancel) begin
        send Cancelled; rollback;
        release resources;
      end else
        if (message is Confirm)
          begin send Confirmed;
          release resources; end
    end end end

```

5 Security Solution

We use GSI [8] to address security issues. The security solution works like this:

- Authentication. It first creates a proxy credential signed by user's private key by using a user proxy and then mutually authenticates the identity.
- Authorization. The Gatekeeper maps the proxy credential into local user name by using text-based map file, then checks operation power of the user. If the user is authorized, Gatekeeper allocates a credential C_p used to create a process.
- Delegation. It promulgates C_p for the user to access other remote resources. By tracing back along the certificate chain to check the original user certificate, processes started on separate sites by the same user can authenticate one another, enabling the user to sign once, run anywhere.
- Encryption. Secure communication is implemented by SSL.

6 Implementation Discussions

The agent must be installed in every machine to join the transaction. Its behavior depends on the type of request. If an agent is invoked to initiate a transaction to run on remote sites, it sends CoordinationContext to the remote agents and locally creates the TM. If an agent receives a Coordination message, it creates the JM.

For participants/candidates to join a transaction, the agent sends CoordinationContext to them. The message contains the necessary information such as transaction type, transaction identifier, address of TM and timeout parameters. JM responds TM by the CoordinationContext, and passes Prepare message to the Co-reservation module. After querying the MDS, the Co-reservation gains the handles of reserved resources.

In addition, the model uses Globus Toolkit to complete low-level operations, such as discovery, reservation and allocation of resources, communication and security.

7 Conclusion and Future Work

We have described a secure transaction model for Grid computing. It can handle both short-lived operations and long-lived business activities, for which we propose coordination mechanisms and provide the algorithms respectively. In addition, we discuss some implementation strategies and propose a security solution.

The model is extensible because it may conveniently incorporate new protocols. In near future, we will develop it into a transaction middleware for Grid applications.

References

1. I.Foster, C.kesselman, and S.Tuecke. The anatomy of the grid: enabling scalable virtual organizations. Int'l J. High-Performance computing Applications. March 2001.
2. F. Cabrera, G. Copeland, T. Freund, J. Klein, D. Langworthy, D. Orchard, J. Shewchuk and T. Storey. Web Services Coordination(WS-Coordination). August, 2002.
3. F. Cabrera, G. Copeland, B. Cox, T. Freund, J. Klein, T. Storey and S. Thatte. Web Services Transaction (WS-Transaction). August, 2002.
4. A. Ceponkus, S. Dalal, T. Fletcher, P. Furniss, A. Green and B. Pope. Business Transaction Protocol V1.0. June, 2002.
5. I.Foster and C.kesselman. The Globus project: a status report. Heterogeneous Computing Workshop (HCW98) Proceedings. March, 1998.
6. I.Foster and C.Kesselman. Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputer Applications. 1997.
7. I.Foster, C.Kesselman, C.Lee, B.Lindell, K.Nahrstedt and A.Roy. A distributed resource management architecture that supports advance reservation and co-allocation. Intl Workshop on Quality of Service, 1999.
8. R. Butler, V.Welch, D. Engert, I. Foster, S.Tuecke, J. Volmer and C. Kesselman. A national-scale authentication infrastructure. Computer, December, 2000.

An Overview of Research on QoS Routing

Yanxing Zheng¹, Wenzhua Dou¹, Jing Tian², and Mingyan Xiao³

¹School of Computer,
National University of Defence Technology,
Chang Sha, China

²School of Mechanical Engineering & Automation Control,
Chang Sha, China 410073

³Navy Submarine Academy,
Qing Dao, China

Abstract. The subject of providing QoS guarantees in packet-switched networks has been a major area of research over the past 10 – 20 years [1]. Some QoS-based frameworks have been proposed to help solve the IP quality problem. QoS routing algorithm is the critical component of these frameworks or has a good future of applying in them. To better guarantee QoS, a new QoS routing mechanism, QoS service routing (or QoSSR for the rest of the paper), appears recently. Then, the conventional QoS routing is classified as QoS data routing (or QoSDR for the rest of the paper). Several overviews on QoSDR have been published [2][3][4]. The paper differentiates itself from other overviews in that: (1) It pays much attention to future directions, such as applying QoSDR in DiffServ and MPLS, and presents a proposal for integrating QoSDR with MPLS. (2) The paper also addresses QoSSR of which there is still no overview at all.

1 Introduction

A distinguishing feature of multimedia applications is that the volume of data required is large which indicates a bandwidth requirement. The second feature is that many of them are interactive, which implies a requirement for the provision of bounded delays [5]. QoS requirements are far more than bandwidth and delay, jitter and reliability metrics are also included. Any satisfaction or dissatisfaction can be described by a metric, which definitely brings too much difficulty to QoS routing [6].

IETF has proposed several QoS-based frameworks to support traffic with special QoS requirements, e.g., Intserv, DiffServ and MPLS etc. The Intserv network can provide per-flow QoS guarantee (delay/bandwidth) but raises the important issue of scalability, due to the complexities of per-flow operations both in the data plane and QoS control plane; The DiffServ network offers coarse-grain relative QoS differentiation, i.e., per-class, to user traffic without per-flow complexity and its scalability is very good. But DiffServ focuses on how to divide the traffic and fast-forwarding. It still runs conventional best-effort routing algorithms and cannot guarantee end-to-end QoS requirements. Recently, the problem of applying QoSDR in DiffServ and MPLS has been a very active area of research.

Normally, routing involves two identities: the routing protocol and the routing algorithm [7]. The task of each identity has been addressed in [2]. The paper will discuss the second identity, i.e., routing algorithm.

2 QoS Data Routing(QoSDR)

2.1 QoSDR Model

Let $G(N,E)$ denotes a network topology, N is the set of nodes(host, router, switch, etc.), E is the set of links, C is a set of constraints.

Unicast QoSDR model can be described as: for a given $G(N,E)$ and constraint C , a source node S and a destination node D , finding a path, which is from S to D and satisfies the constraint C , based on the network information currently know.

Multicast QoSDR model can be described as: for a given $G(N,E)$ and constraint C , a source node S and a set of destination node D , finding a tree, which roots at the node S , ends at the elements of D and satisfies the constraint C , based on the network information currently known.

2.2 Classes of QoSDR Algorithms

We can view QoSDR algorithms from several points of view. A QoSDR algorithm can be (1) a optimization algorithm or a heuristic algorithms; (2) a source routing algorithms, a distributed routing algorithm or a hierarchical routing algorithm; (3) an algorithm that bases on accurate routing information or an algorithm that bases on inaccurate routing information; (4) a unicast or a multicast routing algorithms.

Heuristic algorithm [8] is encouraged in QoSDR for its efficiency and low time-complexity. Optimization algorithms [6] view QoSDR as an optimization problem with a set of constraints. In fact, most optimization algorithms in QoSDR change their convergence conditions. They end after they get the feasible solution instead of an optimal one due to the time-complexity. Normally, when an optimization is convergent, it usually maintains several feasible solutions that can be used in multipath routing issues.

The QoSDR algorithms based on accurate routing information simplify the routing problem in fact. The routing information is scarcely accurate because it is not updated in time. So much attention should be paid to QoSDR based on inaccurate routing information.

2.3 QoSDR Constraints

The metrics of QoS Constraints can be classified into three categories: additive, concave and multiplicative [2][9]. Next we present different constraints format for each category. Assuming that a path consists of n links, $m(l_i)$ represents some metric on the link, e.g., delay. Then constraint formats corresponding different metrics can be depicted as:

- (1) $\sum m(li) \leq \Delta$ or $\min(\sum m(li))$ or $\sum m(li) \geq \Delta$ or $\max(\sum m(li))$ for additive metrics;
- (2) $\min(\sum m(li))$ for concave metrics;
- (3) $\prod m(li) \leq \Delta$ for multiplicative metrics.

In [6], metrics are differentiated between links and nodes, and they are finally viewed as nodes metrics.

2.4 Basic QoSDR Problems and the Corresponding Constraints

(1) Unicast QoS Routing Problem

Shigang[4] divides unicast QoSDR into four basic problems. We give their possible corresponding constraints:

- (1) link-optimization routing: $\max(m(li))$ or $\min(m(li))$;
- (2) link-constrained routing: $m(li) \leq \Delta$ or $m(li) \geq \Delta$
- (3) path-optimization routing: $\max(\sum m(li))$, or $\min(\sum m(li))$, or $\max(\prod m(li))$ or $\min(\prod m(li))$;
- (4) path-constrained routing: $\sum m(li) \leq \Delta$, or $\sum m(li) \geq \Delta$, or $\prod m(li) \leq \Delta$, or $\prod m(li) \geq \Delta$.

Many composite routing problems can be derived from the above four basic problems.

(2) Multicast QoS Routing Problem

Multicast QoSDR problems can also be divided into four different classes: (1) link-optimization routing; (2) link-constrained routing; (3) tree-optimization routing; (4) tree-constrained routing.

There are several classes of routing problems are NPC, such as path-constrained path-optimization routing, etc. We emphasize it again here in that for a NPC problem, we should try to find the approximate solution, then fruitless research can be avoided.

2.5 QoSDR Algorithms

We discuss QoSDR algorithm from the point of whether it is based on accurate routing information or not.

(1) QoSDR Algorithms with Accurate Routing Information

QoSDR Algorithms with accurate routing information [6][8][10] assume that the network-state information is temporarily static and has been distributed throughout the network and is accurately maintained at each node. YanXing Zheng[6] uses a vector-constraint model to describe the multi-constraint routing problem, and presents

GAVCMA (GA for Vector Constraint Multicast Routing) algorithm that aims at resolving it. Xin Yuan[8] investigated two heuristic algorithms: the limited granularity heuristic and the limited path heuristic for k-constraint problem, k is a small integer. The two algorithms all based on extended Bellman-Ford algorithm (EBFA). EBFA extends the Bellman-Ford shortest path algorithm by having each node u to maintain a set $\text{PATH}(u)$ that records all optimal QoS paths found so far from source node to u . The idea of the two heuristic algorithms is to limit the number of optimal QoS paths maintained in each node to reduce the time and space complexity, while maintaining the effectiveness in finding paths that satisfy QoS constraints. Shuqian Yan [10] proposes a multicast routing protocol, i.e., QoSMIC, which provides QoS-sensitive paths in a scalable, resource-efficient, and flexible way. QoSMIC differs from the previous protocols in that the multicast participant joining a group is offered multiple paths and selects the one that best satisfies its QoS requirements.

(2) QoSDR Algorithms with Inaccurate Routing Information

There are several reasons for the inaccuracy of the routing information: (1) The state of the network is always changing; (2) We cannot update the routing information timely because of the overhead incurred by exchanging the information. The solutions for QoSDR with inaccurate information can be divided into three classes: (1) Algorithms that base on the inaccurate information [11][12]. (2) Algorithms that use local QoS state information, instead of global information that may be inaccurate [14], some distributed QoSDR algorithms are in this class [15]. (3) Fuzzifying the objectives, and solving the routing problem by fuzzy multi-objective techniques [16].

D. H. Lorenz [11] studies the problem of routing connections with QoS requirements across network, when the information available for making decisions is inaccurate. The main QoS requirement discussed is end-to-end delay guarantees. The work bases on the assumption that the delay guarantees that are advertised by each link are random variables with known distributions and the probability distribution on the links are independent. Then the delay guarantees represent the probability that a link can satisfy a QoS delay requirement. D. H. Lorenz [11] defines the routing problem and an important variant: OP-MP (Optimally Partitioned MP), and carefully researches the solutions to them in the context of shortest path problems.

Yanxia Jia [12] addresses the problem of link state based QoS routing which is how to get acceptable performance in networks with inaccurate link state information. With WFQ-like scheduling, the end-to-end delay and delay jitter depend on the requested bandwidth [13], so the only metrics discussed in [12] is bandwidth. Yanxia Jia presents two classes of algorithms: bandwidth-based and hop-based. The idea of them is that a router maintains k “shortest” source-destination paths. When a routing request arrives at the router, the node selects one from the routing table. A route check mechanism will determine if this route is feasible. If it is not the case, a second path will be selected. If none of the k paths is feasible, the connection is blocked. The simulation results indicate that small $K>1$ suffices to produce a significant improvement over the $K=1$ case. It means that the complexity of the routing can be well contained, because all it needs to do is to find a few (a small K , e.g., 3) alternative paths, which effort is only moderately bigger than finding a single path.

Srihari Nelakuditi [14] proposed a localized approach to QoS routing. Under this approach, source nodes infer the network OoS state based on flow blocking statistic

collected locally, and perform flow routing using this localized view of the network QoS state. The advantage is that the communication overhead involved is minimal, because no global QoS state information exchange among network nodes is needed.

Emad Aboelela [16] treats the QoSDR as a multi-objective optimization problem and solves it by fuzzy multi-objective optimization techniques. There are several benefits of using fuzzy optimization approach. The most important one is that it can compensate for the inaccuracy of routing information and increase the solution space. Furthermore, Fuzzy logic allows us to apply efficiently the heuristic algorithms widely used in the operating system field to dynamically allocate the memory in the computer system.

2.6 QoSDR with Other Networks Components

(1) Applying QoSDR in DiffServ Network

DiffServ was proposed by IETF in order to assure QoS. DiffServ is not a per-flow, but a per-class model. In addition to BE (best-effort) service, DiffServ mainly support two kinds of service: (1)EF (Expedited Forwarding) and (2) AF(Assured Forwarding). [17].

It may be desirable to perform traffic engineering at a per-class level instead of an aggregate level, in order to further enhance networks in performance and efficiency [17]. In fact, QoSDR can be effectively applied to DiffServ. The architecture of the router in DiffServ is as below:

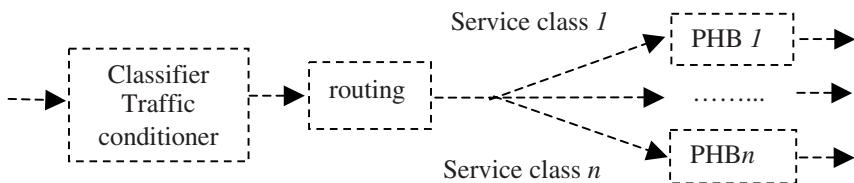


Fig. 1. Normal DiffServ router mechanism

After applying QoSDR to DiffServ, the router is as below:

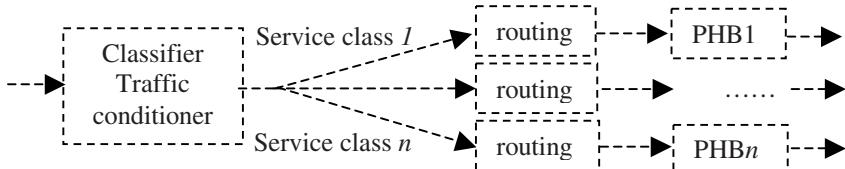


Fig. 2. Applying QoSDR to DiffServ

Fig.1. is the router model in normal DiffServ network. All packets run the same routing algorithm, and then they are sent to different PHBs to ensure different levels of QoS. While in Fig.2., routing module is disassembled into several sub-modules, which run different QoS routing algorithms according to different DHCP. As we known, multi-constrained QoSDR problems are almost NPC and time consuming. In

fact, by decomposing the routing module into several sub-modules simplifies the routing problem. The most complicate sub-module is the traditional QoSDR.

[17] presents a PERD(Per-class Routing Based on Per-class Dissemination) routing mechanism, based on the idea above. The PERD mechanism realizes routing differentiation for different forwarding class. A class in PERD can represent a PHB, or a group of PHBs in DiffServ network. Then traditional routing corresponds to the most complicate class and it is the special case of PERD. Future research should pay attention to the optimization of network as a whole.

(2) A proposal for Integrating QoSDR with MPLS

A FEC (forwarding equivalence class) is a group of IP packets that are forwarded in the same manner. In MPLS, each packet is assigned to a FEC, Which is encoded as a short fixed length value known as a "label". When a packet is forwarded to its next hop, the label is sent along with it; that is, the packets are "labeled" before they are forwarded. Each label identifies a FEC, which is usually of local significance.

MPLS allows (but does not require) the precedence or class of service to be fully or partially inferred from the label [18]. In this case, one may say that the label represents the combination of a FEC and a precedence or class of service. So in ingress router, we can disassemble the label into several "sub-labels" to providing different levels of QoS. Each sub-label represents a QoS level. Then the packets labeled by different sub-label will get different services.

The forwarding mechanism of normal (Fig. 3.) and proposed (Fig. 4.) MPLS can be depicted as follows:

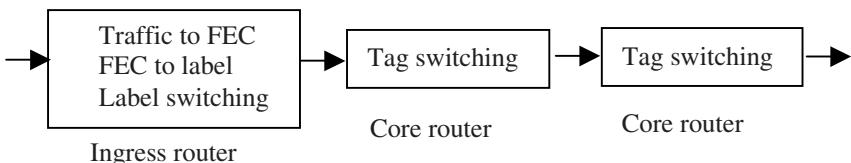


Fig. 3. Normal MPLS routing mechanism

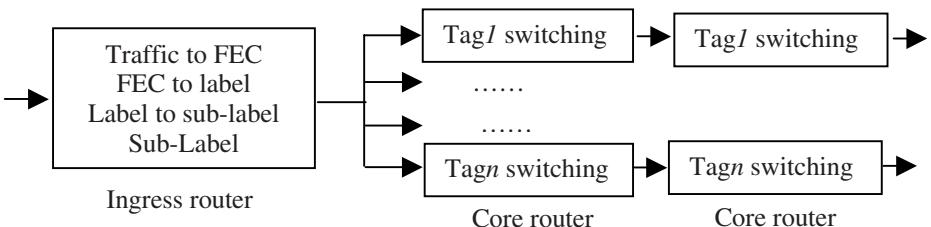


Fig. 4. Proposed MPLS routing mechanism

Strengths and Problem of the Proposal

The proposal has several strongpoints: (1) We can still label a path which is most important for fast forwarding. (2) It guarantees different level of QoS requirements, not only in the sense of fast switching but also in the sense of routing. (3) Label disassembling can be done according to the TOS octet of an IP header, just as the DSCP(Differentiated Service Code Point) in DiffServ.

There are also some problems: (1) We have to select several paths between the same pair of source end and destination end, which can be solved by multi-path routing algorithms or an optimizing algorithm. (2) We have to manage more labels than that in normal MPLS, because a label in MPLS has disassembled into several sub-labels.

2.7 Difficulty That QoSDR Algorithm Has to Deal with

For a efficient QoSDR algorithm, the following factors must be noticed: (1) **Complexity problem**: complexity, especially time complexity, is the first and most important problem that QoSDR must face up to, because most multi-constraints QoSDR are NP-complete problems, resolving which is much time consuming. Efficient heuristic algorithms are encouraged. (2) **Scalability**: Most existing algorithms does not scales well, and exhibits exponential runtime with respect to the size of networks. (3) **Inaccurate routing information**: the information of network state is changing all the time, which definitely will affect the validation of the routing algorithms based on them. (4) **Co-existence with best-effort routing**: the problem has been discussed in [3]. (5) **Implement problem**: given that we have solved the complexity problem (some multi-constraints algorithms do have tolerate computation complexity), there are still two issues need to be concerned: ① different traffics require different metrics, so a router has to applying particular algorithm according to the coming traffic; ② most routing algorithms assume that we have got the necessary routing information, i.e. routing protocols have done it for us. While in fact, for a multi-constraints routing, frequently getting the routing information for multiple metrics will bring too much overhead to the networks.

2.8 Future Directions of QoSDR Research

In addition to the directions pointed out in [4], the following directions call for much attention: (1) Applying QoSDR in some promising networks, e.g., in DiffServ and MPLS. (2) Solving QoSDR by fuzzy multi-objective techniques. The human feeling for the services is imprecise and it can be represented as a fuzzy concept. So it is natural that the requirements are viewed as a fuzzy goal. (3) Hierarchical routing algorithm [4]. We still emphasize hierarchical routing here in that it scales very well and can be used in large networks.

3 QoS Service Routing(QoSSR)

3.1 Background of QoSSR

Different from QoSDR(QoS data routing) problem, which describes network-level QoS routing, QoSSR(QoS service routing) problem addresses itself to application-level QoS routing. QoSSR is also the critical component of service delivery network

(SDN), which is the next generation data services platform for scalable data centers. Multimedia applications often have stringent QoS requirements. Therefore, service routing needs to be QoS-aware.

QoSSR based on two basic assumptions: (1) some multimedia services are composed of many sub-service, i.e., service is composable, multiple services can be applied in a determined order to deliver a complex service [9]; (2) multimedia service can be provided at strategic locations in the network. Then we can construct an overlay media service proxy network by proxies at strategic locations in the network. Every proxy in the proxy network provides specific services, and has its own capacity and bandwidth limit, such as cpu capacity and memory. Then the problem becomes finding a service path that contains all the necessary services in a determined order.

[19] specifies that to construct an end-to-end service path, we should know four independent information: (1) the media object itself; i.e., service requirement; (2) user configuration, i.e., how the services should be distributed over the network; (3) the node, with which services are installed, should know their own capacities; (4) the "path programmer", defines a service by giving the set of rules that govern how an end-to-end path is constructed.

The critical steps in QoSSR are: (1) Specifying service requirements; (2) Constructing the overlay proxy network; (3) Mapping the service requirements to the proxy network; (4) Routing algorithms. Next, we will discuss QoSSR, problems and possible solutions.

3.2 Service Requirements Description

We can describe a service as follows:

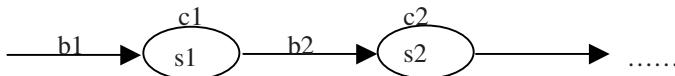


Fig. 5. A service requirement

Fig. 5. indicates that the service requirement is composed of service s1 and service s2, and service s1 requires that the proxy, at which s1 is to be handled, has at least c1 capacity and allow input bandwidth no less than b1. A service maybe is decomposed to other sub-service instead of s1 and s2.

3.3 Constructing the Overlay Proxy Network

A proxy in the overlay proxy network can always reach another proxy based on the fact that the underlying network does not partition [9]. There can be various kinds of topologies which satisfying above condition, such as trees, meshes and so on. Fig.6. depicts two kinds of topologies. Topology presents us that how the services are distributed in the proxy network, therefore, topology selection and maintenance is critical to the success of finding the service path. If the algorithm succeeds, a service path will be found. A service path connects a pair of communicating end points, i.e., source point and destination point, by a series of proxies in the proxy network.



Fig. 6. Possible topologies of overlay proxy network

[9] presents a proxy/service topology, which can be a tree, a mesh, etc. The node in proxy/service topology is a proxy that is installed with a set of services. The proxy maintains full service information such as its own capacity, the communication delay/bandwidth from itself to other neighbors. [20] maintains a changing mesh topology during the path finding process. A changing mesh topology can adapt itself to various kinds of services.

The topology problem, i.e., how to connect the proxies, is still an open problem. [20] keeps a highly connected mesh by a mesh augmentation method. Highly connect mesh will result in a shorter service path, which will be more safer to provide a service. A disadvantage of fully connected topologies is high routing information maintenances cost [9], while a mesh topology (not highly connected) will result in a long service path. A difference between QoSDR and QoSSR is that QoSSR must take the proxy capacity into account. Considering the proxy volatility, a long service path will have a high volatility. So there should be a good balance between a low service path volatility and a low routing maintenances cost.

3.4 Mapping

For a particular service requirement, such as $S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_n$, the mapping process can map the information of the service requirement into the overlay proxy network directly [20], or map service request information and proxy network information into a new graph [9]. Then the QoS routing algorithms will be executed on the mapped overlay proxy network or the new graph. Mapping process must resolve service availability and dependency issues. Service availability issue means that a proxy is not installed with all kinds of services. Service dependency issue means that the sub-service of a service request should be applied in a determined order, which differentiates QoSSR from QoSDR. Guidance for mapping mechanisms is still lacking. Here we will pay much attention to the routing algorithms, and leave the mapping process open.

3.5 Routing Algorithms

Once we get the topology that we run routing algorithm on, we can use some existing solutions in QoSDR., but there are still some issues that need to be dealt with.

The routing algorithms in QoSDR allow no loop in the routing path, while in QoSSR there maybe a proxy that is visited twice. So the routing algorithms in QoSDR are not readily applicable. The performance metric in QoSSR can also be classified into three categories: addictive, concave, and multiplicative, which have the same

meaning with that in QoSDR. But there are some differences between them. All metrics in QoSSR take their meaning in application-level services. Delay and hop count are additive metrics at proxy granularity. Bandwidth and proxy are concave metrics. Bandwidth is between two services, so it can be a CPU bandwidth if the two services locate at the same proxy. If the services are located in two separated proxies, then the bandwidth is bandwidth the two proxies. QoSSR views proxy volatility as a multiplicative metric, while QoSDR takes no proxy volatility into account.

As in QoSDR, there maybe several paths that connect source end and destination end. So we have to present a performance function that valuates the path selected currently. A performance function should try its best to optimize all service metrics at the same time, though some metrics usually conflict with each other.

As an evaluation criterion, performance function can be defined in various kinds of ways, and it partly relies on which metrics we care more about and achieving multiple-metric optimization simultaneously. The performance function in [9] includes: (1)the delay, hop count and bandwidth between two proxies; (2) the capacity of the proxies; (3) the volatility of the proxies, while the performance function in [20] includes capacity of proxies, the bandwidth and hop count between the two proxies.

3.6 Future Directions of QoSSR

(1) Profiting from QoSDR – “Portable Problem”: After we get the mapped topology, how to use the existing algorithms in QoSDR without getting them changed too much should be concerned. (2) Applying in SDN: SDN is the next generation data services platform, how to combining QoSSR with other mechanisms of SDN to provide a satisfying service delivery remains to be researched. (3) QoSSR brings itself with QoS Measurement issue. To keep the information maintained by a proxy updated timely, the proxy must perform some QoS measurement that is another research hotspot. (4) The need for scalable, secure, and high-performance overlay proxy topology. As we discussed above, a highly connected topology may result in frequently information exchanging, which will lead to large routing information measurement/maintenance cost, while a small degrees of proxies will result in a longer service path, which will lead to an unsafe service, considering the factors of delay and the volatility of proxies. (5) Mapping service request to the overlay proxy network. The mapping result can be a new topology and the previous overlay proxy network. The topology of the mapping result is important because the routing algorithms will finally run on it. Efficient mapping mechanisms are still missing.

4 Conclusions

QoS routing is a critical component of QoS framework. It has two sub-routing problems: QoSSR and QoSDR. QoSDR algorithms can be divided into two classes: the algorithms based on accurate routing information and the algorithms based on inaccurate routing information. DiffServ and MPLS are two promising frameworks and combining QoSDR with them has a good future. QoSSR aims to solve the application-level QoS routing problems, and it's also the critical component of service delivery network (SDN).

References

1. Martini Reisslein, Keith W.Ross and Srinivas Rajagopal, "a Framework for Guaranteeing Statistical QoS", IEEE/ACM TRANSACTIONS ON NETWORKING. VOL.10, NO.1, February 2002
2. F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "*A Review of Constraint-Based Routing Algorithms*", Technical Report, June 2002.
<http://www.tvs.et.tudelft.nl/people/fernando/papers/TRreviewqosalg.pdf>
3. Marília Curado, Edmundo Monteiro, "An overview of Quality of Service Routing Issues", in Proceedings the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001)
4. Shigang Chen, Klara Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions", IEEE Network, Special Issue on Transmission and Distribution of Digital Video, Nov./Dec. 1998.
5. Joseph C. Pasquale, George C. Polyzos, George Xyloomenos, The multimedia multicasting problem, ACM/Springer-Verlag Multimedia Sytems, January 1998. Vol.6, No.1, 43–59
6. YanXing Zheng Jing Tian, WenHua Dou, "GA for Vector Constraint Multicast Routing Problem", accepted by Chinese Journal of Computers, 2003
7. C. Huitema, routing in the Internet, Prentice Hall, Inc., 1995
8. Xin Yuan, "Heuristic Algorithms for Multi-constrained Quality-of-Service Routing", IEEE/ACM TRANSACTIONS ON NETWORKING, VOL.10.NO.2, APRIL. 2002
9. Jin, Jingwen, and Klara Nahrstedtm, "QoS Service Routing for Supporting Multimedia Applications", Technical Report, Department of Computer Science, University of Illinois at Urbana-Champaign, November 2002.
10. Shuqian Yan, Michalis Faloutsos, Anindo Banerjea, "QoS-Aware Multicast Routing for the Internet: The Design and Evaluation of QoSMIC", IEEE/ACM TRANSACTIONS ON NETWORKING, VOL.10.NO.1, FEBRUARY. 2002
11. D. H. Lorenz and A. Orda, " QoS routing in net works with uncertain parameters," in *IEEE INFOCOM'98*, 1998
12. Yanxia Jia, Ioannis Nikolaidis and P. Gburzynski. "Multiple path QoS routing." *Proceedings of ICC'01*, Helsinki, Finland, June 11–15, 2001, pp. 2583–2587
13. H.Zhang, "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks", In Proceedings of the IEEE, 83(10), Oct 1995
14. Srihari Nelakuditi, Zhi-Li Zhang, Rose P. Tsang, "Adaptive Proportional Routing: A Localized QoS Routing Approach", INFOCOM 2000: 1566–1575
15. S. Chen and K. Nahrstedt, "Distributed Quality-of-Service Routing in High-Speed Networks Based on Selective Probing", Technical Report, University of Illinois at Urbana-Champaign, Department of Computer Science, 1998. The extended abstract was accepted by LCN'98
16. E. Aboelela; C. Douligeris, "Fuzzy Optimization Model For QoS Routing and Bandwidth Allocation ", World Scientific Publisher, Advances in Informatics, edited by Dimitrios I Fotiadis & Stavros D Nikolopoulos, ISBN 981-02-4192-5, pp. 18–29, July 2000.
17. Peng Zhang, Raimo Kantola, Samuli Aalto, "QoS Routing for DiffServ Networks: Issues and solution", Technical Report, October, 2002
18. E. Rosen, A. Viswanathan, R. Callon, "Multiprotocol Label Switching Architecture", rfc3031, January 2001
19. Akihiro Nakao, Larry Peterson, and Andy Bavier, "Constructing End-to-End Paths for Playing Media Objects", *2001 IEEE Open Architectures and Network Programming, Proceedings*, pages 117–128, April 2001
20. D. Xu, K. Nahrstedt, "Finding Service Paths in an Overlay Media Service Proxy Network (submitted for publication)", 2002 ,
<http://www.cs.purdue.edu/homes/dxu/pubs/MMCN02-extended.pdf>

A Novel Model and Architecture on NMS – Dynamically Constructed Network Management*

Zhongzhi Luan¹, Depei Qian², Xingjun Zhang¹, Tao Liu¹, and Heng Chen¹

¹ Neocomputer Institute, Dept. of Computer Science,
Xi'an Jiaotong University,
Xi'an , 710049, China
rick710055@263.net

² National Lab on Software Development Environment,
BeiHang University,
BeiJing, 100083, China
depeiq@263.net

Abstract. A novel model and architecture of network management – Dynamically Constructed Network Management (DCNM) based on the active network technology is proposed in this paper. The network services and applications are treated as “soft equipment” managed by the NMS thus all the resources in the network can be managed under this model. As the function of managed objects, i.e., network services and applications etc., management functions will be dynamically constructed along with the development and deployment of variety of services and applications in the network. This makes management itself an adaptive component of the network. The architecture of dynamically constructed network management is composed of the management center, the middle management node, the data collection node, and the management client. A DCNM prototype has been developed based on our research experiences.

1 Introduction

Along with the sharp enlargement of network scale, the topology of network becomes more and more complexity. In company with the emerging technology such as intelligence network in telecom world and active network and grid in computer area, the services and applications are continual expanded and deployed dynamically on the network. To manage the network applications changed dynamic efficient is impendency demand to adapt those technology developments. The management and maintenance mechanism and function should expand and update along with the change of protocols, services and applications in the network system. We advance a novel network management model – Dynamically Constructed Network Management (*DCNM*) based on the active network technology with ability to manage network and network applications efficient, agility and scalable.

* This work was supported by the National Science Foundation of China (NSFC) under the grant No.90104022, and the National Key Basic Research Program of China (973 Project) under the grant No. G1999032710.

Dynamically constructed network management has three marked characteristic. Applications are treated as “soft equipment” and brought into the NMS. This makes network devices and network applications an overall management. As the function of managed objects, i.e., network services and applications etc., management functions will be dynamically constructed along with the development and deployment of variety of services and applications in the network. This makes the NMS a characteristic of dynamic scalability. Using active network technology to realize the dynamic distribution, deployment, storage and execution of management code makes the NMS a characteristic of dynamic deployment [1] [2] [3]. These characteristics make management itself an adaptive component of the network.

2 The DCNM Model

2.1 DCNM System Model

A DCNMS (dynamically constructed network management system) is formed of a six-element-set as followed:

$DCNMS = \{MO, MF, MG, MU, f, g\}$, in this six-element-set:

MO , sets of managed objects, is denoted as $MO = \{mo_i \mid i=1, 2, \dots\}$. Managed object sets include network resources such as network services, applications and protocols, topology, host and other hardware.

MF , sets of management functions that are constructed dynamically, is denoted as $MF = \{mf_j \mid j=1, 2, \dots\}$. mf_j is management function is constructed dynamically by the network management system.

MU , sets of management system users, is denoted as $MU = \{mu_k \mid k=1, 2, \dots\}$. We use $mu(mf)$ to denote user mu executing management function mf .

MI , sets of dynamically constructed management interfaces, is denoted as $MI = \{mi_l \mid l=1, 2, \dots\}$. mi_l is management interface be constructed dynamically by the network management system.

f , a function relation between management interfaces, management users and management functions, namely, a function between MI , MU and MF . It indicates that the management interfaces would changed dynamically along with the variety of management users and management functions and the executions of functions by users. That is to say, toward MI , MU and MF , $mi_l = f(mu_k, mf_j, mu_k(mf_j))$, in this equation, $mi_l \in MI$, $mu_k \in MU$, $mf_j \in MF$, $j, k, l \in N$.

g , a function relation between management functions and managed objects, namely, a function between MF and MO . It indicates that the management functions would changed dynamically along with the change of managed objects. That is to say, toward MF and MO , $mf_j = g(mo_i)$, in this equation, $mf_j \in MF$, $mo_i \in MO$, $i, j \in N$.

2.2 DCNM System Deployment Model

The deployment model of the dynamically constructed network management system that is described above is formed of a four-element-set $\{AN, EE, AM, h\}$, in this four-element-set:

AN , sets of active nodes, is denoted as $AN = \{an_i \mid i=1, 2, \dots\}$,

AM , sets of management modules that can be deployed dynamically, is denoted as $AM = \{am_j \mid j=1, 2, \dots\}$,

EE , sets of management module executive environments, is denoted as $EE = \{ee_k \mid k=1, 2, \dots\}$,

h , an operation relation inside the active network environment, namely, a relation between AN, AM and EE . It indicates that toward any management modules that can be deployed dynamically there do exist an EE on any possible active node. That is to say, $\forall am_j, an_i, am_j \in AM, an_i \in AN, \exists ee_k (ee_k \in EE), ee_k = h(am_j, an_i), i, j, k \in N$.

2.3 Model Signification

According to the descriptions of the *DCNM* model we can know two issues. On the one hand, through operation f , management interfaces become functions of management users and management functions. Management interfaces will be constructed dynamically along with the variety of the executions of functions by users. This incarnates the individuation and dynamic of management interfaces. On the other hand, through operation g , management functions become functions of managed objects. It means that management functions will be constructed dynamically along with the variety of the managed objects including devices, applications and protocols. This incarnates the association between management functions and managed objects and the scalability of management functions. The characteristics mentioned above are realized based on the deployment model. Through operation h , the management functions get the guarantee of be deployed and executed dynamically by active network technology support.

3 The *DCNM* Architecture

To realize the model described above, we bring forward a corresponding architecture. The architecture consists of management center, middle management node, and data collection node and management client. See Fig.1. The parts in broken line in Fig.1 are managed objects. The description of main composing components of the architecture is followed.

3.1 Management Center

Management center is the highest level in the network management system. It offers the management functions support to the whole system, such as creation, maintenance and expanding of management codes, application model and managed objects, creation of management interfaces and management of user registration etc. The codes distribution, management functions construction, management interfaces construction and system scalability can realize dynamically via the interaction between management center and management client and middle management node. Management center consists of management server, code server, application model base, managed object sets, interface creator and registration manager etc.

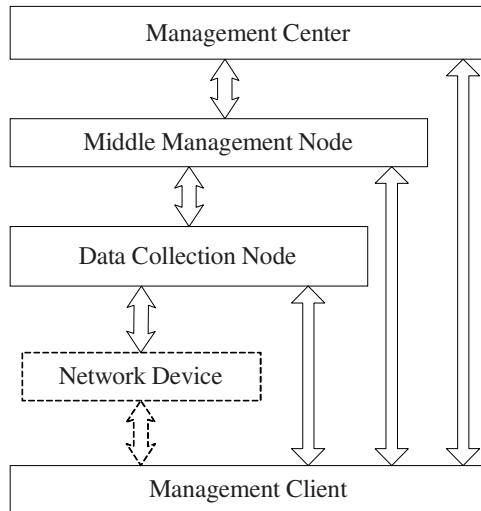


Fig. 1. Dynamically Constructed Network Management Architecture

3.2 Middle Management Node

Middle management node obtains correlation network state and management information of applications and services collected by data collection node. It realizes local management functions and offers local management interfaces. Under the indications from management center, middle management node possesses a certain function setting network parameter to change network state. This architecture realizes the dynamic characteristic and distributing characteristic of network management system. The main components of it are local node manager and active management agent execution environment.

3.3 Data Collection Node

According to the managed object sets supported by applications, data collection node capture information and send to middle management node by demand of management from devices, application servers or applications clients that are appointed by management function. The main components of it are collection node manager and management interface.

3.4 Management Client

After the managed applications' startup, management agents on management clients would send associated information to management center and request management center to organize associations between managed objects and management functions, executions of management functions and dynamically constructed of management interfaces. Management agents on management client would also register management functions information to middle management nodes according to

specific constitution. Management clients also offer various user interfaces to different users and by different management demands.

4 The DCNM Prototype

We realized a dynamically constructed network management system prototype on Linux and Windows using java technology. This prototype is consisting of the main composing components described in the architecture, such as management center, middle management node, data collection node, etc. The dynamically constructed network management system prototype is shown in Fig.2.

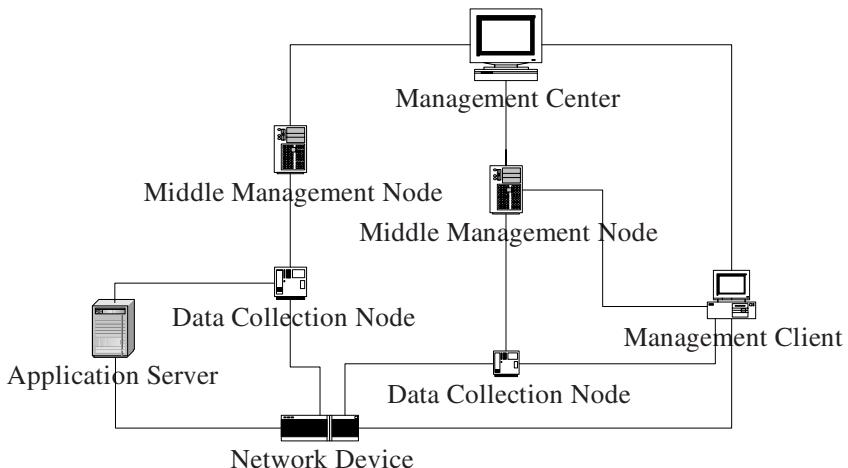


Fig. 2. Dynamically Constructed Network Management System Prototype Sketch

5 Conclusions

According to the status quo of network management and network development trend, a novel model and architecture of network management – Dynamically Constructed Network Management (DCNM) based on the active network technology is proposed in this paper. The network services and applications are treated as “soft equipment” managed by the NMS thus all the resources in the network can be managed under this model. As the function of managed objects, i.e., network services and applications etc., management functions will be dynamically constructed along with the development and deployment of variety of services and applications in the network. This makes management itself an adaptive component of the network. The architecture of dynamically constructed network management is composed of the management center, the intermediate management node, the data collection node, and the management client. We have studied some key technologies including association of the managing and managed objects, transfer, activation, and security mechanisms for the mobile active code, and dynamic construction of the management interface. A DCNM prototype has been developed based on our research experiences.

References

- [1] Xu Bin, Qian Depei et al. "The Research on Active Network Management Architecture". *Journal of Computer Research and Development*. Vol.39, No.4, Apr. 2002
- [2] D.Tennenhouse, Jonathan M Smith, W David Sincoskie et al. "A Survey of Active Network Research"[J]. *IEEE Communications Magazine*, 1998, vol35 (1): 80–86
- [3] Danny Raz, Yuval Shavitt, "Active Networks for Efficient Distributed Network Management", *IEEE Communications Magazine*, 38(3):138–143, March 2000

QoS-Driven Multicast Tree Generation Using Genetic Algorithm*

Xingwei Wang¹, Hui Cheng¹, Jiannong Cao², Zhijun Wang³, and Min Huang³

¹ Computing Center,
Northeastern University,
Shenyang, 110004, China
wangxw@mail.neu.edu.cn

² Computing Department,
Hongkong Polytechnic University,
Hong Kong, China

³ College of Information Science and Engineering,
Northeastern University,
Shenyang, 110004, China

Abstract. A QoS-driven multicast tree generation algorithm is discussed by considering QoS (Quality of Service) requirements and network resource constraints in multimedia group communication. Inspired by the successful application of optimization computing methods-GA (Genetic Algorithm) in other combinatorial optimization problems, an algorithm is proposed, which constructs a QoS-driven multicast routing tree based on genetic algorithm. By simulation research, some application parameters of the algorithm are given; meanwhile, the feasibility, effectiveness and robustness of the algorithm are also proved.

1 Introduction

Multicast is an important area of the research on high-speed IP network infrastructure and key technologies. It is also one essential capability for next-generation networks. There are many network applications nowadays, which need the support of QoS (Quality of Service) multicast, such as video conferencing and distance education, etc. Thus, multicast has been widely studied in packet-switched IP networks. Many efficient QoS multicast algorithms and protocols have been developed and in use for many years [1,2].

In this paper, an algorithm is proposed to generate a QoS-driven multicast routing tree with the lowest cost of network resources in multimedia group communication [3]. In the algorithm the requirements on CPU, buffer and bandwidth are considered.

* This work was supported by the National Natural Science Foundation of China under Grant No.60003006 (jointly supported by Bell labs) and No.70101006; the National High-Tech Research and Development Plan of China under Grant No.2001AA121064; the Modern Distance Education Key Technology and Supporting Service System Program of Ministry of Education of China; the Foundation of Shenyang Science and Technology Committee.

In addition, two QoS parameters, end-to-end delay and error rate, are also considered. It is proved that the problem of constructing such a routing tree is NP-complete problem [4]. The proposed algorithm can construct the minimum cost Steiner tree [5] based on GA (genetic algorithm) [6], and a destination node startup joining algorithm is designed to generate the initial optimum solution. Simulation shows that the proposed algorithm is feasible, effective and robust.

2 Model Description

2.1 Network Model

A network can be modeled as a directed graph $G=(V, E)$, where V is the set of nodes representing routers and E is the set of edges representing links that connect the routers. Graph $G=(V, E)$ contains $n=|V|$ nodes and $L=|E|$ edges. Every node $v_i \in V$ is marked by following parameters: available CPU resource $c(v_i)$, available buffer resource $b(v_i)$, error rate $\epsilon(v_i)$, queuing delay $t(v_i)$, sending delay $\tau(v_i)$. Every edge $e_{ij}=(v_i, v_j) \in E$ is marked by following parameters: available bandwidth $w(e_{ij})$, transmission delay $\delta(e_{ij})$, error rate $\xi(e_{ij})$, where $w(e_{ij})=w(e_{ji})$, $\delta(e_{ij})=\delta(e_{ji})$, $\xi(e_{ij})=\xi(e_{ji})$, $1 \leq i, j \leq n$.

2.2 Mathematical Model

In graph $G=(V, E)$, the nodes in $U=\{u_0, u_1, \dots, u_m\}$, $U \subseteq V$ communicate mutually.

The amount of network resources required by users is as follows: CPU resource C , buffer resource B and bandwidth resource W .

The problem is to find a tree $T = (X, F)$, $U \subseteq X \subseteq V, F \subseteq E$, satisfying:

- $\min_{v_i \in X} [c(v_i)] \geq C, \min_{v_i \in X} [b(v_i)] \geq B$;
- $\min_{e_{ij} \in F} [w(e_{ij})] \geq W$;
- Minimize $\left[\sum_{v_i \in X} h(v_i) + \sum_{e_{ij} \in F} H(e_{ij}) \right]$, i.e., minimize the cost of tree T , where $h: v_i \rightarrow R^+, H: e_{ij} \rightarrow R^+$ are heuristic cost functions defined on nodes and edges, respectively.

The key objective of optimization is that the cost of the tree is as small as possible. This objective does not necessarily guarantee that the resources on nodes and edges of the tree satisfy the requirements of users. However, we can adjust it by defining cost functions h on nodes and H on edges properly. For example, by defining heuristic cost functions, for the nodes and edges on which the available resources are not less than the requirements of users, the cost takes smaller value. Therefore, it can help to satisfy the multi-constrained QoS requirements. We define h, H as follows:

$h(v_i) = \frac{k_1}{c_i} + \frac{k_2}{b_i}$, $H(e_{ij}) = \frac{k_3}{w(e_{ij})}$, where k_1, k_2, k_3 are constants used to compute the network cost.

The delay D_{ij} and error rate R_{ij} of path $P(u_i, u_j)$ between any two different session nodes on tree T are stated as follows:

- $\forall u_i \in U, \forall u_j \in U \text{ and } u_i \neq u_j : D_{ij} = \left[\sum_{v_i \in P(u_i, u_j)} (t(v_i) + \tau(v_i)) + \sum_{e_{ij} \in P(u_i, u_j)} \delta(e_{ij}) \right];$
- $\forall u_i \in U, \forall u_j \in U \text{ and } u_i \neq u_j : R_{ij} = \left[1 - \prod_{v_i \in P(u_i, u_j)} (1 - \epsilon(v_i)) * \prod_{e_{ij} \in P(u_i, u_j)} (1 - \xi(e_{ij})) \right].$

When the multicast tree is generated, the algorithm continues to calculate D_{ij} and R_{ij} . If their values cannot satisfy user requirements, the construction of the multicast tree for the group communication fails. Then, the users are negotiated with QoS.

3 Statement of QoS-Driven Multicast Algorithm Using GA

3.1 Expression of the Solution

We denote the solution s by binary coding. Each bit of binary cluster corresponds to each node. The graph corresponding to s is $G' = (V', E')$. Let the function $bit(s, i)$ denote the i_{th} bit of s . If and only if $bit(s, k) = 1$, then $v_k \in V'$. For our problem, every solution s corresponds to a tree $T'_i(X'_i, F'_i)$, which is the minimum cost spanning tree of G' . T'_i spans the nodes set U .

G' may be unconnected. Thus, every subgraph of it has a minimum cost spanning tree, and s corresponds to a minimum cost spanning forest, denoted by $T'_i(X'_i, F'_i)$. If G' is unconnected, add penalty value to the cost of the solution.

Thus, every solution s corresponds to a graph G' , which corresponds to a minimum cost spanning forest T'_i (a tree is also a forest). Prune T'_i , the forest T_i corresponding to s is obtained.

3.2 Fitness Function

The fitness of solution s is obtained by fitness function f , which is determined by the cost $g(T_i)$ of the corresponding tree (forest) $T_i(X_i, F_i)$:

$f(s) = g[T_i] = \sum_{v_i \in X_i} h(v_i) + \sum_{e_{ij} \in F_i} H(e_{ij}) + [count(T_i) - 1]\rho$, where $count(T_i)$ is the number of trees in the forest T_i , ρ is a constant, $\rho > 0$.

Thus, for every solution, the smaller the fitness, the better the solution.

3.3 Generation of Initial Feasible Solution

A destination node startup joining algorithm has been designed, which can find a feasible solution in polynomial time. Then, the feasible solution is used as initial locally optimum one. The algorithm is stated as follows.

Let the group be $U = (u_0, u_1, \dots, u_m)$. Choose a node in U arbitrarily as the source node, e.g., u_0 .

Step 1: start ($i=1$). Let u_0 be the source node, u_i be the destination node, construct initial route using “one pass hop-by-hop algorithm” [7]. A tree $T_i(W_i, F_i)$ corresponding to the initial route is then generated. It is actually a path from u_0 to u_i . $U = U - \{u_0\}$

Step 2: $U = U - \{u_i\}$. If $U = \emptyset$, the algorithm stops, otherwise, go to Step 3.

Step 3: $i=i+1$. Let u_i be the destination node, u_0 be the source node, $T_{i-1}(W_{i-1}, F_{i-1})$ be the tree that has been generated. Implement “direct destination node initiated dynamic joining algorithm” [8]. Add the path from u_i to T_{i-1} given by the algorithm to T_{i-1} . New generated tree $T_i(W_i, F_i)$ is obtained. Go to Step 2.

When the algorithm stops, the generated tree $T_i(W_i, F_i)$ is a feasible solution. In the worst case, the total time complexity of the above algorithm is $O(|V|^3)$.

4 Design of QoS-Driven Multicast Algorithm Using GA

4.1 Generation of Initial Population

Every chromosome corresponds to a solution. The initial population is composed of the chromosomes generated randomly according to the population size. It's possible that some chromosomes are bad, because they are generated randomly. For example, for $U = \{1, 2, 5, 6\}$, the chromosome $s = 110110$ is bad.

To solve this problem, we can generate a special chromosome s . If $v_i \in U$, then $bit(s, i) = 1$, otherwise, $bit(s, i) = 0$. For the above example, the special chromosome is $s = 110011$. Let randomly generated s_i and s have the logical “or” operation, i.e., $s_i \vee s \rightarrow s_i$.

The initial population is generated as follows:

Step 1: start ($i=0$). Generate the special chromosome s .

Step 2: $i=i+1$.

Step 3: Generate chromosome s_i randomly.

Step 4: $s_i \vee s \rightarrow s_i$.

Step 5: Let k denote the population size. If $i < k$, then go to step 2, otherwise, stop. Thus, the initial population $S = \{s_1, s_2, \dots, s_k\}$ will be gained.

4.2 Crossover

We adopt single point crossover. The crossover point is chosen randomly. For example, chromosome $s_i = 101101$ and chromosome $s_j = 110101$ will have the crossover operation. Choose the crossover point randomly and assume it to be the point between the second position and the third position in the binary cluster corresponding to the chromosome. Then, after the crossover operation, $s_i = 100101$ and $s_j = 111101$.

4.3 Mutation

We have the mutation operation according to mutation probability. It must be pointed out that bad chromosomes may be generated when mutation. For example, if chromosome $s_i = 110111$ and the mutation position is the second position in the binary cluster, after the mutation operation $s_i = 100111$ and $\text{bit}(s_i, 2) = 0$. But node $2 \in U$, so the chromosome is bad. To solve this problem, we have two methods. One is to have the bad chromosome “or” (logical operation) the special chromosome defined above. The other is to add penalty value to the fitness of the chromosome when computing. We adopt the former.

4.4 Selection Policy

We adopt the roulette wheel selection [6] as the policy to choose the chromosomes. If the fitness is better, the probability of choosing the chromosome is higher. For our problem, the better is the chromosome, the smaller is its fitness. Thus, we adopt the following conversion function:

$$f'(s_i) = \max_{s_i \in S} [f(s_i)] - f(s_i) + \min_{s_i \in S} [f(s_i)], \text{ where } S \text{ is chromosomes set, i.e., population.}$$

Let the area of the roulette wheel be 1, then the area which corresponds to each chromosome in the wheel is the ratio of the converted fitness of the chromosome to the total sum.

The computation expression is $\text{Roulette}(s_i) = f'(s_i) / \sum_{s_j \in S} f'(s_j)$.

$$\sum_{s_i \in S} \text{Roulette}(s_i) = 1.$$

Thus, after generating a random number between 0 and 1 and finding the position of the number in the roulette wheel, we can decide to choose which chromosome.

5 Simulation Results and Discussion

5.1 Evaluation of Solution Quality

The following simulation research is based on NSFNET [9] topology. For ease of expression, we define the following notations to describe the parameters of GA: population size: n_p , crossover probability: ρ_c , mutation probability: ρ_m , generation number: n_G , choosing probability of chromosome genes: ρ_g . We have chosen the appropriate values for these parameters by simulation research. It is stated as follows: $n_p \in [20,30]$, $\rho_c \in [0.85, 0.95]$, $\rho_m \in [0.01, 0.05]$. n_G and ρ_g should be set according to the different network state, which is discussed in section 5.2.

In the simulation experiments, we set the CPU resource, the buffer resource, and the bandwidth resource at each node or edge are all evenly distributed random number between 100 and 1000; the error rate at each node or edge is evenly distributed random number between 0 and 0.0001; the queuing delay, the sending delay, and the transmission delay at each node or edge are all evenly distributed random number between 1 and 100. Set $k_1 = k_2 = k_3 = 1000$.

Table 1. Result of simulation

C1	C2	C3	C4	$\leq 1\%$
1	35.71	10000	34.0	99.96%
2	50.00	10000	52.0	100.0%
3	64.29	10000	64.0	100.0%

Table 1 is the analysis table of solution quality. By contrasting solutions obtained by the proposed algorithm with the optimal cost solution obtained by exhaustive search, quantitative analysis is made. Only the cost performance of the solutions is shown. C1 denotes Session No, C2 denotes the ratio of session nodes in network (%), C3 denotes the run times of the algorithm, C4 denotes the optimal cost given by exhaustive search. $\leq 1\%$ means that the ratio of the difference between the cost of the solution obtained by the proposed algorithm and the optimal cost (i.e., the deviation of this feasible solution) to the optimal cost is $\leq 1\%$. The value under the ratio interval means the ratio of the number of feasible solutions whose deviation ratio is in this interval to the total number of solutions.

The session nodes are chosen randomly from sparse mode to dense mode. We can see that for the actual topology of NSFNET, the cost accuracy degree of the solutions obtained by the proposed algorithm is very high.

5.2 The Time Complexity of the Algorithm

In the proposed algorithm, the primary computation part of GA is to compute the fitness of chromosomes. The primary computation part of the fitness is to compute the minimum cost spanning tree.

Denote the length of the chromosome as n (which equals the nodes number in the network) and the number of user nodes as m .

Now we begin to discuss the time complexity of the algorithm is affected by the application parameters.

1) The population size n_p is in direct proportion to the time complexity of the algorithm.

2) The larger the generation number n_G , the larger the time complexity of the algorithm. But its effect on the time complexity of the algorithm isn't as simple as the population size. It is also affected by other parameters as is explained later.

3) In the proposed algorithm, we must calculate the fitness of every chromosome for every subpopulation. This step is marked as *step**. But the chromosomes whose fitness needs to be calculated are only those new chromosomes generated by crossover operation and mutation operation. The mathematical expectation of the number of the chromosomes generated by crossover operation is $n_p \rho_c$. Hence, the mathematical expectation of the number of the chromosomes generated by reproducing operation is $n_p(1 - \rho_c)$. The mutation operation follows the crossover operation. As long as they have just experienced crossover operation, the chromosomes must recalculate their fitness. So, the number of chromosomes which need to recalculate their fitness after mutation operation are the number of chromosomes which are reproduced from their parental generation chromosomes and have just experienced mutation operation. For ease of expression, it is denoted by x .

Mutation probability means the probability that every bit will have mutation operation in the binary cluster corresponding to the chromosome. Hence the mutation probability of a chromosome is $1 - (1 - \rho_m)^{n_p}$. Then, the mathematical expectation of x is $E(x) = n_p(1 - \rho_c)[1 - (1 - \rho_m)^{n_p}]$. So in *step** the number of chromosomes whose fitness need to be calculated is $n_p \rho_c + n_p(1 - \rho_c)[1 - (1 - \rho_m)^{n_p}]$.

4) The time complexity of computing the minimum cost spanning tree is $O(y^2)$, where y is the nodes number in the network. For every chromosome s , the nodes number of the graph generated from s is the number of 1 in the binary cluster corresponding to s . y has relationship with choosing probability of chromosome genes and mutation probability. That y has relationship with mutation probability is because mutation means to change the genes in binary cluster from 1 to 0 or from 0 to 1. Thus, at many moments the number of 1 in the binary cluster will be changed by mutation operation.

Denote the mathematical expectation of the number of 1 in the binary cluster in the i_{th} generation population as $E(y_i)$, where y_i is the number of 1 in any binary cluster in this generation. Now generate the $i+1_{th}$ generation population from the i_{th} generation. First, no matter whether the chromosomes in the $i+1_{th}$ generation population are generated by crossover operation or reproducing directly, the mathematical expectation of the number of 1 in the binary cluster won't change. Only the mutation operation changes the mathematical expectation. In addition, the bits, which correspond to user nodes in binary cluster, should always be 1, i.e., they don't have mutation operation. Based on the above analysis, we can see that the

mathematical expectation of the number of the genes, which will change from 1 to 0 in binary cluster, is $[E(y_i) - m]\rho_m$, and that the mathematical expectation from 0 to 1 is $[n - E(y_i)]\rho_m$. Thus, the mathematical expectation of the number of 1, which is newly added to the binary cluster, is $[n - E(y_i)]\rho_m - [E(y_i) - m]\rho_m$ (1).

So the mathematical expectation of the number of 1 in binary cluster corresponding to the chromosome is $E(y_{i+1}) = E(y_i) + [n - E(y_i)]\rho_m - [E(y_i) - m]\rho_m$ after passing through one generation. Simplify this expression and get $E(y_{i+1}) = (1 - 2\rho_m)E(y_i) + \rho_m(n + m)$ (2).

For $E(y_i)$, we have the following statements.

- $E(y_1) = \rho_g(n - m) + m$;
- If $\rho_g = 0.5$, then $E(y_{i+1}) = E(y_i) = E(y_1) = \frac{1}{2}(n - m) + m = \frac{1}{2}(n + m)$;
- If $\rho_m = 0$, then $E(y_{i+1}) = E(y_i) = E(y_1) = \rho_g(n - m) + m$;
- If $\rho_m \in (0, 0.5)$, it has two cases:
 - if $\rho_g > 0.5$, then $E(y_{i+1}) < E(y_i)$, $E(y_1) > \frac{1}{2}(n + m)$;
 - if $\rho_g < 0.5$, then $E(y_{i+1}) > E(y_i)$, $E(y_1) < \frac{1}{2}(n + m)$;
- If $\rho_m \in (0.5, 1)$, it has also two cases:
 - if $\rho_g > 0.5$, then $\left|E(y_{i+1}) - \frac{1}{2}(n + m)\right| < \left|E(y_i) - \frac{1}{2}(n + m)\right|$, $\begin{cases} E(y_{2k}) < \frac{1}{2}(n + m) \\ E(y_{2k+1}) > \frac{1}{2}(n + m) \end{cases}$;
 - if $\rho_g < 0.5$, then $\left|E(y_{i+1}) - \frac{1}{2}(n + m)\right| < \left|E(y_i) - \frac{1}{2}(n + m)\right|$, $\begin{cases} E(y_{2k}) > \frac{1}{2}(n + m) \\ E(y_{2k+1}) < \frac{1}{2}(n + m) \end{cases}$;
- If $\rho_m = 1$, $E(y_{i+1}) + E(y_i) = n + m$.

Because of $E(y_1) = \rho_g(n - m) + m$, we have $E(y_2) = n - \rho_g(n - m)$, $E(y_3) = \rho_g(n - m) + m$, ...

Then, we get: $\begin{cases} E(y_{2k}) = n - \rho_g(n - m) \\ E(y_{2k+1}) = \rho_g(n - m) + m \end{cases}$.

We have mathematical limitation for expression (2):
 $\lim_{i \rightarrow +\infty} E(y_{i+1}) = \lim_{i \rightarrow +\infty} [(1 - 2\rho_m)E(y_i) + \rho_m(n + m)]$.

Because of $\lim_{i \rightarrow +\infty} E(y_{i+1}) = \lim_{i \rightarrow +\infty} E(y_i)$, we get $\lim_{i \rightarrow +\infty} E(y_i) = \frac{1}{2}(n + m)$.

The above discussion is useful for selecting the appropriate choosing probability of chromosome genes when the algorithm runs. If the nodes number of multicast tree corresponding to the solution of our problem is approximately k and $k > \frac{1}{2}(n + m)$,

then the choosing probability of chromosome genes ρ_g should satisfy $E(y_1) = \rho_g(n-m) + m > k$. If not, it's possible that feasible solutions are very hard to be attained in any case that we adjust other parameters. This is because the probability of attaining the feasible solutions is very small here.

If $k < \frac{1}{2}(n+m)$, then the selection for the choosing probability of chromosome genes is very easy. If $\rho_m > 0$, then we'd better set ρ_g to make that $E(y_1) = \rho_g(n-m) + m$ is a little less than or approaches k . Certainly, it's allowable that ρ_g is less than or equivalent to 0.5 because in such cases feasible solutions can be attained although the probability of attaining the optimal solution is small. But if setting ρ_g to be more than 0.5, it's bad for the algorithm because most solutions aren't good. And much machine time is wasted.

Therefore, if we can set the choosing probability of chromosome genes appropriately, we can get satisfactory solutions through a few generations of genetic operations and save much machine time simultaneously. If not, the solutions maybe aren't good and even it's possible to fail to attain feasible solutions.

6 Conclusions

A QoS-driven multicast tree generation algorithm is described. The algorithm can help satisfy the users requirements for QoS. It constructs a group sharing routing tree according to the user requirements for network resources, including CPU time or utilization ratio of router, the buffer size of router, and bandwidth of network link. It also considers the end-to-end delay and error rate of the generated routing tree. If the routing tree can't satisfy user QoS requirements, then the user is negotiated with QoS. We have also discussed the relationship between the time complexity and the GA parameters. Simulation shows that the proposed algorithm solves the QoS multicast problem in multimedia group communication efficiently.

References

1. Wang, B., Hou, C.J.: A Survey on Multicast Routing and Its QoS Extensions: Problems, Algorithms, and Protocols. *IEEE Network Magazine*, Vol. 14, No. 1. (2000) 22–36
2. Wittmann, R., Zitterbart, M.: *Multicast Communication: Protocols, Programming, and Applications*. 1rd edn. San Francisco, CA: Morgan Kauffman (2000)
3. Chockler, G.V., Keidar, I.: Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys*, Vol. 33, No. 4. (2001) 1–43
4. Wang, X.W.: Research on QoS Management and Group Communication Mechanisms in Distributed Multimedia System, PhD thesis, Northeastern University, China (1998) 115–119 (in Chinese)
5. Hwang, F.K., Richards, D.S.: Steiner Tree Problems. *Networks*, Vol. 22, No.1. (1992) 55–89

6. Gen, M., Cheng, R.W.: *Genetic Algorithms and Engineering Design*. 1rd edn. New York, NY: John Wiley & Sons Inc (1997)
7. Wang, X.W., Cai, G.Q., Liu, J.R.: A Quality-of-Service-Based Routing Algorithm for Point-to-Point Multimedia Communication. *Proceedings of WCC2000*, Beijing, China, Vol. 2. (2002) 1613–1616
8. Wang, X.W., Huang, M., Liu, J.R.: Research on Quality of Service Based Destination Node Joining and Leaving Algorithms for Multimedia Group Communication. *Chinese Journal of Computers*, Vol. 24, No. 8. (2001) 838–844 (in Chinese)
9. Saha, D., Purkayastha, M.D.: An Approach to Wide Area WDM Optical Network Design Using Genetic Algorithm. *Computer Communications*, Vol. 22, No. 2. (1999) 156–172

A Scalable Peer-to-Peer Network with Constant Degree*

Dongsheng Li¹, Xinxin Fang², Yijie Wang¹, Xicheng Lu¹, Kai Lu¹, and Nong Xiao¹

¹School of Computer,
National University of Defense Technology,
410073 Changsha, China
leedongsh@hotmail.com

²Department of Compute science and technology,
Tongji University,
200092 Shanghai, China
sabina_xin@hotmail.com

Abstract. Recently many distributed hash table (DHT) schemas have been proposed to build scalable peer-to-peer systems, in which degree and diameter are two important measures. In this paper, we propose *Fission*, a novel DHT-style peer-to-peer network, which is of constant degree and $O(\log N)$ diameter. Peers in Fission form an approximate Kautz topology and the “split large and merge small” policy is exploited to achieve load balance when peers join or depart. The performance of Fission is evaluated using both analysis and simulation. Formal proofs verify that the degree and the diameter of Fission are no more than $O(1)$ and $2^*\log N$ respectively and the join or departure of one peer requires only $O(1)$ peers to change their state. Simulations show that the load balance characteristic of Fission is good and the average path length of Fission is no more than $\log N$.

1 Introduction

In recent years, peer-to-peer computing has emerged as a novel and popular computation model and attracted significant attentions from both industrial and academic fields [1,2]. The core component of many proposed peer-to-peer storage systems [3,4,5,6,7,8] is the distributed hash table (DHT) schema that uses a hash table-like interface to publish and lookup data objects. In DHT schemas, each peer bears responsibility for a certain portion of the key space and uses a routing table to forward the query for data object. To maintain the DHT, the responsibility is reassigned between peers when peers join or depart.

Two important measures of DHT systems are degree, the size of routing table to be maintained on each peer, and diameter, the number of hops a query needs to travel in the worst case. In many existing DHT schemas, such as Chord [3], Tapestry [4] and Pastry [5], both the degree and the diameter tends to $\log N$. In this paper, we propose

* This work was supported by National Natural Science Foundation of China under the grant No. 69933030, 60203016 and 90104001, Excellent PHD Dissertation Foundation of China under the grant No. 200141 and National 863 High Technology Plan of China under the grant No. 2002AA131010.

Fission, a novel DHT-style peer-to-peer network with constant degree and $O(\log N)$ diameter, which exploits Kautz topology and uses the “split large and merge small” policy for maintenance. *Fission* is inspired by CAN [6], which allows peers re-positioning the space, and D2B [9], a DHT schema based on de Bruijn graphs. However, *Fission* can achieve better results than both. Proofs show that the degree and diameter of *Fission* is $O(1)$ and $O(\log N)$ respectively even in the worst case. Compared with *Fission*, CAN uses a d-dimensional torus topology with $2d$ degree, but the diameter of CAN is $O(dN^{1/d})$ and so it doesn't scale quite as well as *Fission*. D2B, which is based on de Bruijn graphs, provides constant expected degree and $O(\log N)$ diameter, but its degree could be at most $O(\log N)$ with high probability. That is to say, some peers in D2B will be likely to have $O(\log N)$ degree. Chord [3] uses a ring topology; Tapestry [4] and Pastry [5] utilize hypercube topology based on prefix-based routing. Their diameters are $O(\log N)$, but their degrees are $O(\log N)$ which is larger than that of *Fission*. Viceroy [10] is based on butterfly topology with constant expected degree and $O(\log N)$ diameter, but its degree and diameter could be $O(\log N)$ and $O(\log^2 N)$ respectively in the worst case.

The remainder of the paper is organized as follows. Section 2 introduces Kautz graph. Section 3 describes the design and properties of *Fission*. Section 4 presents the routing algorithm in *Fission* and proves its correctness. Section 5 evaluates the performance of *Fission*. Conclusions and future work is discussed in Section 6.

2 Kautz Graph

Fission exploits Kautz graph as its topology. Given two positive integers d and k , a Kautz graph [11,12], denoted by $K(d,k)$ is a directed graph with degree d and diameter k . The nodes of Kautz Graph are encoded with a string of length k where each symbol of the string is from the set $Z=\{0,1,2,\dots,d\}$ with the restriction that two consecutive symbols of the string are always different. That is to say, node u is encoded with $u_1u_2\dots u_k$ where $u_i \neq u_{i+1}$, $1 \leq i \leq k-1$, $u_i \in Z$. Obviously there are $N=d^k+d^{k-1}$ nodes in the $K(d,k)$ graph. Each node $u = u_1u_2\dots u_k$ has d outgoing edges: for each $\alpha \in Z$ and $\alpha \neq u_k$, node u has one outgoing edge to node $v = u_2u_3\dots u_k\alpha$ (denoted by $u \rightarrow v$), i.e., there is an edge from u to v iff v is a left-shifted version of u . Therefore, each node in $K(d,k)$ is of in-degree d and out-degree d . Figure 1 shows $K(2,3)$.

Routing in Kautz graph from node u to node v is accomplished by taking the string u and shifting in the symbols of v one at a time until the string u has been replaced by v . For instance, given two nodes $u = u_1u_2\dots u_k$ and $v = v_1v_2\dots v_k$, the routing path from u to v is

$$u = u_1u_2\dots u_k \rightarrow u_2u_3\dots u_kv_1 \rightarrow u_3u_4\dots u_kv_1v_2 \rightarrow \dots \rightarrow u_kv_1v_2\dots v_{k-1} \rightarrow v_1v_2\dots v_k \quad (1)$$

when $u_k \neq v_1$ or a path of length $k-1$ as below:

$$u = u_1u_2\dots u_k \rightarrow u_2u_3\dots u_kv_2 \rightarrow u_3u_4\dots u_kv_2v_3 \rightarrow \dots \rightarrow u_kv_2\dots v_{k-1}v_k = v_1v_2\dots v_k \quad (2)$$

Compared to de Bruijn graph [13] with degree d and diameter k , which has d^k nodes, Kautz graph $K(d,k)$ has $(1+1/d)$ times the number of nodes $B(d,k)$, i.e., with the same total number of nodes N and the same value of d , the diameter of Kautz graph is smaller than that of de Bruijn graph. Also it is known that there are obvious uneven loads of edges in a de Bruijn graph. Considering any node $u=i\dots i$, $0 \leq i \leq d-1$, the edge

from $ii\dots i$ to $ii\dots iia$ for any $0 \leq a \leq d-1$ is used only when the node u is the source node, i.e., node u never forward any query. That is because if there is a node $v=bii\dots i$ having an outgoing edge to u , it also has an outgoing edge to $ii\dots iia$. Further features [12] show that $K(d,k)$ is a better logical topology than $B(d,k)$, so Kautz graph is chose as the topology of the Fission.

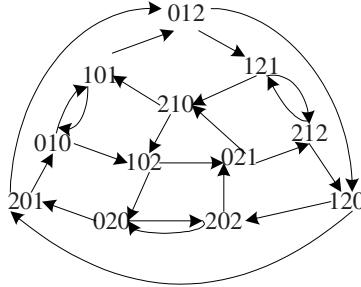


Fig. 1. Kautz graph $K(2,3)$

3 Fission Design

We use Kautz graph $K(2,k)$ as the topology of Fission. Initially, there is a virtual 2-dimensional Cartesian coordinate which is divided into three zones. Like CAN [6], the entire coordinate space is dynamically partitioned among all the peers in the system such that each peer “owns” its specific zone within the whole space. Unlike CAN, each zone in Fission has an identifier and zones are organized as an approximate Kautz graph $K(2,k)$ according to their identifiers. At the beginning the identifiers of the three initial zones are 0, 1 and 2 respectively, as showed in Figure 2.

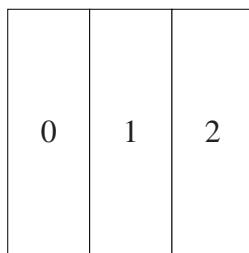


Fig. 2. Initial three Zones

Edges in Fission. The neighborhood of zones is based on their identifiers. The edges between zones can be categorized into three classes: in-edges, out-edges and brother-edges. Suppose the identifier of zone U is $u_1u_2\dots u_k$ (we denote it as $U=u_1u_2\dots u_k$), $0 \leq u_i \leq 2$ and $u_i \neq u_{i+1}, 1 \leq i \leq k$. To form a kautz graph, zone U has one out-edge to a zone whose identifier is $u_2u_3\dots u_k$, $i \leq k$ or U has out-edges to zones whose identifier is $u_2u_3\dots u_kq_1\dots q_m$ with $m \geq 1$. From Lemma 2 proved in section 3.3, we know that the in-edges are only come from zones whose identifiers are $au_1u_2\dots u_i$ ($a \neq u_i$) with $i \leq k$ to

zone U. For the split and mergence of the zones, each zone also has edges to its brothers. Brother-edge will be discussed in section 3.1. At the beginning each zone of the three initial zone has out-edges to the other two zones, and zone 0, 1, 2 are respectively the left brothers of zone 1, 2, 0. We refer to Zone U as the *neighbor* of zone V if there is any edge between U and V.

3.1 Peer Joins

When a new peer n joins in the system, it should know a peer (called *access entry*) currently in the system. As in most DHT systems, we can assume that it can be done with other mechanism, such as using some public available peers.

Spitting Large Zones. When new peer n joins in the system, it firstly chooses a random identifier $U=u_1u_2\dots u_{128}$ with $u_i \in \{0,1,2\}$, $u_i \neq u_{i+1}$, $1 \leq i \leq 127$. Then peer n sends a JOIN request from the access entry to its destination $u_1u_2\dots u_{128}$ according to the routing algorithm in section 4. From corollary 4 (also in section 4), the JOIN request will reach a unique zone W whose identifier is the prefix of U at last. Then start from zone W, if the current zone has a neighbor zone with larger area, it forwards the request to the neighbor (If there are more than one neighbors satisfying the condition, select a random one and forward the message to it). This process will not stop until the JOIN request reaches a zone V which has no larger neighbors and can't be forwarded any more. Thus zone V splits itself into two zones V1 and V2 and the original zone V disappears. The owner of zone V1 is set to peer m that was originally the owner of zone V before the split, and the owner of zone V2 is set to peer n. Suppose the identifier of zone V is $v_1v_2\dots v_k$, then the identifier of V1 is $v_1v_2\dots v_kx_0$ and the identifier of V2 is $v_1v_2\dots v_kx_1$ where $0 \leq x_0 < x_1 \leq 2$, $x_0 \neq v_k$ and $x_1 \neq v_k$. After splitting the zone V, both V1 and V2 should build brother-edges to each other: zone V1 is the left brother of V2, and zone V2 is the right brother of zone V1. Figure 3 shows an example of peer n joining into zone 01.

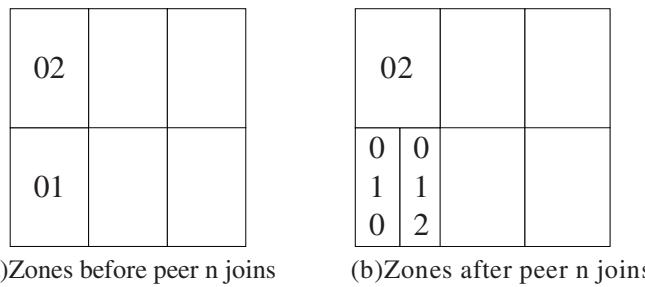


Fig. 3. Peer n joins in Fission

From the split procedure above, it's easy to know that the area of a zone is in proportion to 2^{-l} when the length of its identifier is l. The longer the identifier is, the less area the zone occupies.

Updating Edges. Once zone V is split, the edges between zones should be updated:

1. in-edge: For each zone U that has an out-edge to zone $V=v_1v_2\dots v_k$ before peer n joins, (From Lemma 3, $U=av_1v_2\dots v_j$ with $j \leq k$) zone U should delete its edge to zone V and adds one out-edge to zone V1 and one out-edge to zone V2.
2. out-edge: For each zone U to which zone V has a out-edge before peer n joins, the area that zone U occupies is no more than that of V because the JOIN message stops at V. Thus $U=v_2\dots v_kq_1\dots q_m$ with $m \geq 1$. Therefore zone V1 adds one out-edge to U if $q_1=x_0$ or V2 adds one out-edge to U if $q_1=x_1$.
3. brother-edge: Suppose the left brother of zone V is U and the right brother of V is Q. After peer n joins, the right brother of U is changed to V1 and the left brother of Q is changed to V2. At the same time, V1 is the left brother of V2 and V2 is the right brother of V1.

After all the related edges have been updated, the Fission network conforms to an approximate Kautz topology again.

3.2 Peer Departs

Merging Small Zones. When a peer n departs from Fission, the zone $V=v_1v_2\dots v_k$ it owns should be occupied by other peers. That will cause the mergence of multiple zones. For good load balance, we try to merge the small zones and maintain the Kautz topology at the same time. If peer n volunteers to depart from the system, it produces a DEPART messages. Start from zone V, if current zone has neighbor zones with smaller area, the DEPART request should be forwarded to one neighbor with smaller zone until a zone U is reached which has no neighbors with smaller area. It is easy to see that U has a brother W whose area is the same as that of U. Suppose $U=u_1u_2\dots u_{k-1}u_k$, $W=u_1u_2\dots u_{k-1}w_k$, and the owners of zone U, W are peer n1 and peer n2 respectively. Then:

1. if $U=V$, merge zone V and W into a new zone $V'=u_1u_2\dots u_{k-1}$ and assign the peer n2 as the owner of the zone V' . In this case, peer n1 and peer n are the same peer.
2. if $U \neq V$, merge zone U and W into a new zone $V'=u_1u_2\dots u_{k-1}$, then change the owner of the zone V to peer n1 and assign peer n2 as the owner of zone V' . In this case, the edges related to zone V don't need changing, and only the edges related to zone V' need to be adjusted.

Updating Edges. As discussed above, when zone $U=u_1u_2\dots u_{k-1}u_k$ and zone $W=u_1u_2\dots u_{k-1}w_k$ are merged into zone V' , the edges related to zone $V'=u_1u_2\dots u_{k-1}$ should be constructed to retain the approximate Kautz topology. For convenience, suppose $u_k < w_k$, i.e., U is the left brother of W. Then:

1. in-edges: if zone Q has an out-edge to zone U or zone W or both before the mergence, build an edge from zone Q to zone V' to replace it(them).
2. out-edges: if there is an edge from zone U or zone W or both to a zone Q before the mergence, build an edge from V' to Q to replace it (them).
3. brother-edge: if there is a right brother-edge from zone Q to U, build a right brother-edge from Q to V' . If there is a left brother-edge from W to a zone R, build a left brother-edge from V' to R.

Involuntary Departure. To deal with involuntary failure, each peer sends KeepAlive messages to all of its neighbors periodically. The deferred absence of a KeepAlive message from one neighbor indicates its failure. Once the failure of a peer U is detected by its neighbor P , peer P will produce one DEPART message for U . And the remainder procedure is the same as that in the case of voluntary departure.

3.3 Properties of Fission

In this section, we present some properties of neighborhood in Fission. Due to space limits, we just give the brief proofs and omit the complete details.

Lemma 1. *For each zone $U=u_1u_2\dots u_k$ in Fission, there are no zones V that satisfy $V=u_ju_2\dots u_kx_1\dots x_j$, with $j\geq 1$ and $U\neq V$. And each zone in Fission has no less than one out-edge.*

Proof. Initially there are three zones 0, 1, 2 and each zone has two out-edges to the other two zones. So initially lemma 1 holds. After a split or mergence, lemma 1 still holds. Thus lemma 1 is always true. Q.E.D.

Lemma 2. *Denote the length of the identifier of zone U as $|U|$. For each zone $U=u_1u_2\dots u_k$ in the Fission system, if there is one out-edge from zone U to zone $V=v_1v_2\dots v_m$, then $||U|-|V||\leq 1$, i.e., $|k-m|\leq 1$.*

Proof. Lemma 2 holds initially. We will show that if Lemma 2 holds at some time, Lemma 2 will also hold after a split or mergence. In the case of split, the large zone is divided into two zones. Assume after a split there are two zones U and V with $||U|-|V||\geq 2$ and there is one out-edge from U to V . Recall that before the split for each zone P, W in Fission, $||P|-|W||\leq 1$, thus $||U|-|V||\leq 2$. Therefore $||U|-|V||=2$ and either zone U or zone V is newly produced by the split.

If U is derived from U' in the split, then $||U'|-|V||\leq 1$ and $|U|=|U'|+1$. Obviously to achieve $||U|-|V||=2$, $|U'|-|V|$ must be 1 before the split. Recall that there is one out-edge from U to V after the split, thus U' must have one out-edge to V before the split. But if $|U'|-|V|=1$ (which means the area of zone V is larger than that of zone U') and U' has one out-edge V , then the “split large” policy would cause the JOIN message to be forwarded from U' to V and the zone U' would not have been split. Thus a contradiction occurs.

If V is derived from V' in the split, $|V'|-|U|$ must be 1 before the split. Then zone U is larger than zone V' , thus our “split large” policy would not have split the zone V' . Thus a contradiction occurs.

Therefore, after a split Lemma 2 remains true.

In the case of mergence, proof is similar and we omit it for the space.

Q.E.D.

Lemma 3. *For each zone $U=u_1u_2\dots u_k$, one and only one of the following two properties is true:*

1. *U has a unique out-edge to zone $Q=u_2\dots u_k$*
2. *For each $x_i \in \{0,1,2\}$ and $x_i \neq u_k$, either U has one out-edge to zone $Q=u_2\dots u_kx_i$ or U has two out-edges to zones whose identifiers are in the style of $u_2\dots u_kx_1x_2$ with $x_2 \neq x_1$ and $x_2 \in \{0,1,2\}$.*

Proof. From lemma 2, if there is one out-edge from Q to U, then $|U|-1 \leq |Q| \leq |U|+1$. And as the Fission system forms an approximate Kautz topology, thus the identifier of Q should be in the style of $u_2\dots u_k$ or $u_2\dots u_k x_1$ or $u_2\dots u_k x_1 x_2$. The proof is easy but long to make when considering the split and mergence procedure of zones. We omit it for space. Q.E.D.

The following two corollaries are direct conclusions from Lemma 3.

Corollary 1. If zone $U=u_1u_2\dots u_k$ has more than one out-edge, then for any string $S=s_1\dots s_m$, $s_i \neq u_k$ and $s_i \neq s_{i+1}$ ($1 \leq i \leq m-1$), U has an out-edge to zone $u_2\dots u_k x_j \dots x_j$ ($1 \leq j \leq 2$) with $x_j \dots x_j$ as the prefix of S.

Lemma 3 and Corollary1 show that the routing algorithm in section 4 could go on until the destination zone V is reached.

Corollary 2. The out-degree of any zone in Fission is no more than 4.

We can also prove that the in-degree of any zone is also no more than 4 in a similar way. The following Theorem 1 is a direct consequence of Lemma 1 and Corollary 2.

Theorem 1. In an N-peer Fission system, both the in-degree and the out-degree of each peer are between 1 and 4.

4 Routing Algorithm

Routing in Fission is somewhat similar to that in Kautz graph. Once a zone $U=u_1u_2\dots u_k$ receives a routing message $Routing(V, L, S)$ to destination $V=v_1v_2\dots v_m$ ($U \neq V$) with left path length L, U sends a new routing message $Routing(V, L-1, S)$ to Q if U has one out-edge to zone $Q=u_2u_3\dots u_k$ and U sends a new routing message $Routing(V, L-1, Sx_j \dots x_j)$ to Q if U has one out-edge to zone $Q=u_2\dots u_k x_1 \dots x_j$ ($1 \leq j \leq 2$) and $Sx_1 \dots x_j$ is the prefix of V. The initial value of L and S is set as below: Assume there is a message routing from source zone $W=w_1w_2\dots w_k$ to destination zone $V=v_1v_2\dots v_m$. Find S' that is the longest suffix of W and also appears as a prefix of V and the length of S' is denoted by s. Suppose $W \neq V$, then from Lemma 1, $s < k$. Thus $S'=w_{k-s+1}\dots w_k$ and the initial value of L is $k-s$ and the initial value of S=S'= $w_{k-s+1}\dots w_k$. Figure 4 shows the routing algorithm.

```
Procedure U.routing(dest V, pathlength L, commonfix S)
    //zone U=u_1u_2...u_k deal with the routing
    //message to destination V=v_1v_2...v_m
    {
        /*reach destination V*/
        if L=0 then return(true)
        else {
            if U has a out-edge to Q=u_2u_3...u_k then
                return(Q.routing(V, L-1, S))
        }
    }
```

```

if U has a out-edge to Q=u2...ukx1...xj (1≤j≤2) and
isprefix(Sx1...xj, V) then {
    S← Sx1...xj
    return(Q.routing(V, L-1, S))
} //then
} //else
} //Procedure

```

Fig. 4. Routing algorithm

Now we prove the correctness of the routing algorithm.

Lemma 4. *Considering routing from source zone $W=w_1w_2\dots w_k$ to any destination $V=v_1v_2\dots v_m$ ($W \neq V$) where the length of the longest suffix of W that is also the prefix of V is s , let the routing path from W to V is $U_1 (=W), U_2, U_3, \dots, U_q (=V)$, then U_i is of the form $w_i\dots w_{k-s}S$ and the routing message that U_i deals with is in the form of $routing(V, k-s, S)$ where S is the prefix of V .*

Proof. Let S_0 be the longest suffix of W that is the prefix of V , $S_0=w_{k-s+1}\dots w_k=v_1v_2\dots v_s$, then $U_1=W=w_1w_2\dots w_{k-s}w_{k-s+1}\dots w_k=w_1w_2\dots w_{k-s}S_0$, $V=v_1v_2\dots v_pv_{p+1}\dots v_m=S_0v_{s+1}\dots v_m$. The routing message that W deals with is $routing(V, k-s, S_0)$. Thus initially Lemma4 holds for U_1 .

Let current zone U_i is of the form $w_i\dots w_{k-p}S$ and the routing message that U_i deals with is $routing(V, k-s-i, S)$ where S is the prefix of V . From the routing algorithm in Figure 4, if U_i has one edge to $w_{i+1}\dots w_{k-s}S$, then $U_{i+1}=w_{i+1}\dots w_{k-s}S$ and the routing message that U_{i+1} deals with is $routing(V, k-s-i-1, S)$; or if U_i has one edge to $w_{i+1}\dots w_{k-s}Sx_1\dots x_j$ ($1 \leq j \leq 2$) with $S'=Sx_1\dots x_j$ as the prefix of V , then $U_{i+1}=w_{i+1}\dots w_{k-s}Sx_1\dots x_j=w_{i+1}\dots w_{k-s}S'$ and the routing message that U_{i+1} deals with is $routing(V, k-s-i, S')$. In both cases Lemma 4 holds for U_{i+1} . So Lemma 4 holds.

Q.E.D.

From Lemma 4, when $i=k-s$ ($L=0$), the routing messages reach a zone U_i and the routing process stops. And there is some S that is the prefix of V and $U_i=S$. If destination V is a zone identifier, then from Lemma 1 we can infer that $U_i=S=V$ and the routing reaches the destination zone correctly. The path length of routing from W to V is $k-s$ hops. Thus we get the following corollaries.

Corollary 3. *The path length of routing initiated by any source zone $U=u_1u_2\dots u_k$ is no more than k hops.*

From the proof of Lemma 4, we can get the following corollary 4 easily.

Corollary 4. *For any identifier $U=u_1u_2\dots u_{128}$ with $u_i \in \{0, 1, 2\}$, $u_i \neq u_{i+p}$, $1 \leq i \leq 127$, the routing from any source zone to destination U will arrive and stop at a unique zone V whose identifier is the prefix of U .*

Lemma 5. *In an N -peer Fission system, the largest zone U satisfies that $|U| < \log N$.*

Proof. Let $|U|=k$, then k is the smallest among the length of identifiers of zones in Fission. Peers in Fission form an approximate Kautz topology, thus $2^k + 2^{k-1} \leq N$. Then $k \leq \log N - \log 3 + 1 < \log N$. Q.E.D.

Lemma 6. *In an N -peer Fission system, the smallest zone V satisfies $|V| < 2 * \log N$.*

Proof. Suppose U is the largest zone in Fission system. Consider the routing path from U to V . From Corollary 3, we know that the path length is no more than $|U|$. From Lemma 2, we can infer that $||V|-|U|| \leq |U|$. Because $|V| \geq |U|$, thus $|V|-|U| \leq |U|$, $|V| \leq 2|U| < 2\log N$. Q.E.D.

The following can be derived from Lemma 6 and Corollary 3 directly.

Corollary 5. *In an N -peer Fission system, Let U and V be the smallest and largest zones in the system, then $|U|-|V| \leq |V| < \log N$.*

The following Theorem 2 is a direct consequence of Corollary 3 and Lemma 6.

Theorem 2. *The diameter of Fission systems is less than $2 * \log N$.*

Theorem 3. *When a peer joins in or departs from an N -peer Fission system, the JOIN and DEPART messages are propagated at most $3 * \log N$ and $\log N$ hops respectively, and only constant peers need to update their edges.*

Proof. Take the split procedure as an example: the JOIN message is first forwarded to the zone determined by hashing. From Theorem 2, in this phase the message is propagated less than $2 * \log N$ hops. Then the JOIN message is forwarded to neighbors whose identifiers are at least one less than that of current zone. From Corollary 5, in the phase the message is propagated at most $\log N$ hops. Therefore, the JOIN message is propagated at most $3 * \log N$ hops. When the JOIN message stops at one zone, from Theorem 1 the number of neighbors that should update edges due to the split is constant. Q.E.D.

5 Performance Evaluations

Firstly we analyze the performance of Fission by the measures below:

Degree: From Theorem 1, Fission system is of constant-degree, i.e., is $O(1)$.

Diameter: From Theorem 2, the diameter of Fission is less than $2 * \log N$, i.e., $O(\log N)$.

Scalability: Fission can support arbitrary number of peers. Peer can join in the system and depart from the system at its will.

Maintenance cost: From Theorem 3, the events of a peer joining in or departing from the Fission system induce $O(\log N)$ messages and require $O(1)$ peers to change their state.

We also built a simulator to evaluate the Fission algorithm. The simulator implements the joining and departing procedure in Section 3 and uses the routing algorithm in Section 4. We evaluate the average path length of Fission systems of different scales (from 256 peers up to 256K peers). For each scale, we build the network by adding nodes one by one. We select two random points from the 2-

dimension coordinate space and route a message from one point to the other, and get the average path length over 5000 such routings. Figure 5 shows the simulation results compared with CAN with $d=2$ or $d=3$. From Figure 5, we can infer that the average path length of Fission is about $\log N$.

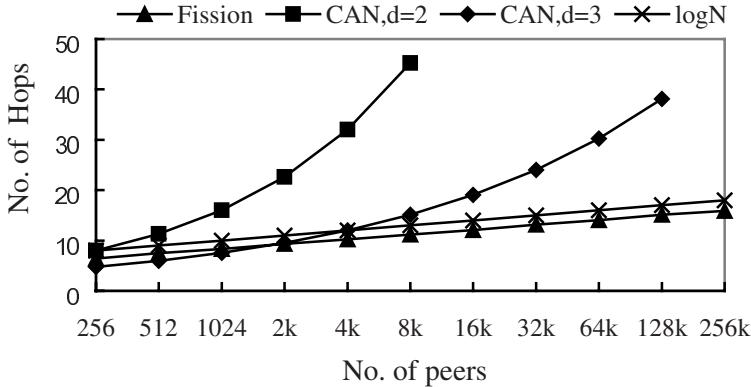


Fig. 5. Average path length of Fission

Then we observe the load balance characteristic of Fission. Let the total area of the entire 2-dimension coordinate space is S_r . We simulate Fission system with $N=4096$ peers and calculate the area each peer owns. Let $S=S_r/N$, Figure 6 shows the percentage of peers that own a particular area. From Figure 6, about 45% peers own the same area S and the percentage that owns area more than $4S$ or less than $S/4$ is almost zero. The number of keys stored on each peer is in direct proportion to the area the peer owns, thus the distribution of keys over peers is almost uniform.

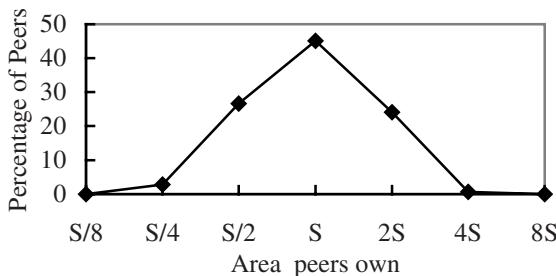


Fig. 6. Load balance characteristic of Fission

6 Conclusions and Future Work

Fission is a novel scalable DHT-style peer-to-peer network that can achieve $O(\log N)$ diameter with only $O(1)$ degrees. Beside its simplicity and low diameter as well as constant degree, Fission also has good load balance characteristic. These

characteristics of Fission suggest that Fission is a very promising peer-to-peer network. In our further work, physical topological information will be exploited in Fission to reduce latency. And the current design of Fission is based on Kautz graph $K(2,k)$ and will be extended to general $K(d,k)$ topology for flexibility.

References

1. Clark, D.: Face-to-face with peer-to-peer networking. *IEEE Computer*, Vol. 34, No.1, IEEE press (2001) 18–21
2. Schoder, D., Fischbach, K.: peer-to-peer prospects. *Communications of the ACM*, Vol.46, No.2, (2003) 27–29
3. Stoica, I., Morris, R., Karger. D. *et al.*: Chord: a scalable peer-to-peer lookup service for Internet applications. *Proc. of ACM SIGCOMM 2001*, ACM Press, New York (2001) 160–177
4. Hildrum, K., Kubiatowicz, J. D., Rao, S., and Zhao, B. Y.: Distributed object location in a dynamic network. *Proc. of 14th ACM Symp. on Parallel Algorithms and Architectures (SPAA)* (2002)
5. Rowstron, A. and Druschel, P.: Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. of IFIP/ACM Middleware'2001*, Heidelberg, Germany (2001) 329–350.
6. Ratnasamy, S., Francis, P., Handley, M. *et al.*: A scalable content-addressable network. *Proc. of ACM SIGCOMM 2001*, ACM Press, New York (2001) 149–160
7. Plaxton, C., Rajaraman, R., and Richa, A.: Accessing nearby copies of replicated objects in a distributed environment. *Proc. of ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, Newport, Rhode Island (1997).
8. Balakrishnan, H., Kaashoek, M. F., Karger, D., Morris, R., and Stoica, I.: Looking up data in P2P systems. *Communication of the ACM*, Vol. 46, No.2, (2003) 43–48
9. Fraigniaud, P., Gauron, P.: The Content-Addressable Network D2B. Tech Rept. 1349, CNRS University paris-Sud, France (2003)
10. Malkhi, D., Naor, M., and Ratajczak, D.: Viceroy: a scalable and dynamic lookup network. *Proc. of 21st AC M Symp. on Principles of Distributed Computing (PODC)*, Monterey, CA (2002)
11. Kautz, W. H.: The design of optimum interconnection networks for multiprocessors. *Architecture and design of Digital computer*. NATO advances summer Institute, (1969) 249–277
12. Panchapakesan, G. and Sengupta, A.: On a lightwave Network Topology using Kautz Digraphs. *IEEE Transaction on computers*, Vol. 48, No. 10, (1999) 1131–1138
13. Sivarajan, K. N. and Ramaswami, R.: Lightwave Networks based on de Bruijn Graphs. *IEEE/ACM Trans. Networking*, Vol.2 (1994) 70–79

Symmetric Distributed Server Architecture for Network Management System*

Xiaolin Lu

School of Information Technology,
Zhejiang University of Finance & Economics
310012 Hangzhou, China
luxiaolin@mail.hz.zj.cn
<http://www.zufe.edu.cn>

Abstract. Flexible, highly scalable, and low-cost unified network management system architecture is very important for top-level system design. Symmetric distributed server architecture is proposed to design a unified network management system for huge scale telecommunication network. The J2EE multi-tier and distributed technology is applied to the distributed systems allowing parts of the system on separate computers located in different location. The message bus is used to provide a powerful way to structure unified large system and control the complexity in network. Proposed symmetric distributed server architecture is implemented in a unified NMS system including front-end server and back-end server constructed a matrix of server nodes. In case of updating a managed object, front-end server will locate the back-end where this object is managed and send updating request to this back-end. And every back-end will execute the core tasks on the sub-network assigned to it. The experiments and application have demonstrated that the architecture has good performance, reliability and is feasible and scalable for operation in current and future telecommunication network management.

1 Introduction

In the large-scale computer and telecom networks there are heterogeneous systems, which consist of a combination of numerous different operating systems spreading across various computing systems such as mainframes, workstations, PCs and some other telecom devices. Distributed object-computing promises a distinct software engineering paradigm for developing a kind of software architecture that can unify computers available in the network and facilitate communication between them [1][2][3].

There are many problems compared to a small network management system. Network management data consist of millions up to hundreds of millions of records are commonplace. The system needs to deploy on a large and powerful mainframes with large relational database.

To be another choice, multi-tier, disturbed technology can perfectly apply to the large-scale network management system. It is possible to use a group of low cost

* The Foundation of Zhejiang Education Committee supported this work.

work-station servers instead of using single powerful mainframes. By using a group of lower cost workstation servers instead of large powerful workstation, the total cost of the system can be lowered. Also, the system is highly scalable. The network will not change the architecture of the system but increase another workstation server. In this paper, the symmetric distributed server architecture for the network management system using the multi layers, distributed technology is proposed.

2 Symmetric Distributed Server Architecture

Distributed computing allows an application system to be more accessible. Distributed systems allow parts of the system on separate computers located in different location[1][3][5].

Enterprise JavaBeans (EJB) is a comprehensive technology that provides the architecture for building enterprise-level server-side distributed Java components. The EJB technology provides a distributed component architecture that integrates several enterprise-level requirements such as distribution, transactions, security, messaging, persistence, and connectivity to mainframes. When compared with other distributed component technologies such as Java RMI and CORBA, the EJB architecture hides most the underlying system-level semantics that are typical of distributed component applications, such as instance management, object pooling, multiple threading, and connection pooling. Secondly, unlike other component models, EJB technology provides us with different types of components for business logic, persistence, and enterprise messages.

The EJB architecture specifies three types of beans – session beans, entity beans, and message-driven beans. A bean developer has to specify the home and remote interfaces and also he has to implement one of these bean interfaces depending upon the type of the bean. For instance, for session beans, he has to implement the javax.ejb.SessionBean interface. The EJB architecture expects him to implement the methods specified in the bean interface and the methods specified in the home and remote interfaces. The EJB container relies on specific method names and uses delegation for invoking methods on bean instances.

EJB container generates the proxy objects for all beans. EJB container for each bean implement a proxy object to the home interface and publishes in the JNDI implementation of the J2EE platform. One can use JNDI to look for this and obtain a reference. As this object implements the home interface only, he can use one of the creation methods of the home object to get a proxy to the remote interface of the bean. When one invokes a creation method on the home proxy object, the container makes sure that a bean instance is created on the EJB container runtime and its proxy is returned to the client. Once the client gets hold of the proxy for the remote interface, it can directly access the services of the bean.

Once the client decides to stop accessing the services of the bean, it can inform the EJB container by calling a remote method on the bean. This signals the EJB container to disassociate the bean instance from the proxy and that bean instance is ready to service any other clients.

Distributed computing allows business logic and data to be reached from remote locations at any time from anywhere by any one. Distributed systems allow parts of the system to be located on separate computers located in different location. The

different application architectures between single server architecture and distributed server architecture is explained as flowing. In single server architecture, all the tier of the system including the database is in a server. In single server application architecture, there is usually less than 10–20 clients will be exercising the server at the same time. The data collection, status polling and event processing requirements are modest. The following diagram gives an overview of the single server architecture. In the single server application architecture, the middle tier of the network management solution (servers) is deployed to run in one Java Virtual Machine (JVM) process. This could be deployed on a JRE (Java Runtime Environment) or in a J2EE server environment (like WebLogic or JBoss).

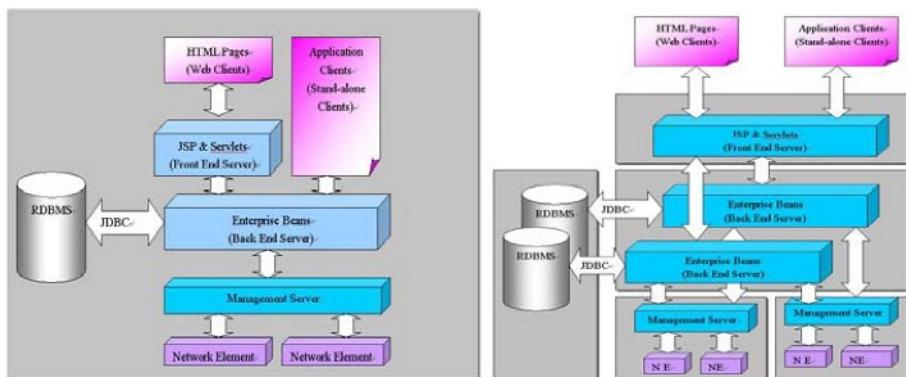


Fig. 1. Single-server and symmetric distributed server network management architecture

To build a highly scalable management solution for large-scale network management system where a large number of clients will be accessing the management data, the multi-tier and multi-servers can be used. The network management system should use the distributed server architecture. The following diagram gives an overview of the distributed server architecture. In the distributed server architecture, the middle tier of the management solution is distributed to run on multiple systems on the network. There can be multiple front-end servers to scale to a large number of clients. Based on scalability requirements, the back end server can also be run in a single server or multiple servers. There can be multiple management servers, based on the data collection, status polling and event processing capabilities, required out of the management solution. The servers support fail over and load balancing capabilities. There is a choice of rich, thin clients based on Java or thin clients based on HTML. The client session management services are deployed in the front-end servers and they access the back-end business logic APIs using RMI or messaging

When there exist multi-server architecture of the network management system and the different type of NMS, the communication between the different servers and different NMS is needed for integrated them together to a universal NMS. In the telecom network, the network management systems become increasingly large and complex. There exist different network management systems to management deferent devices.

How to integrate the different network management systems together are need to be considered in the architecture of a universal network management systems. The high degree of integration characteristic of programming at the class library level makes it difficult to construct heterogeneous applications that use modules from different systems. The core technologies needed to achieve this flexibility are the message bus, which provides flexible services and methods for distributed network management systems to communicate with one another and share data.

The message bus is a facility used to bind, at runtime, distributed objects (program components) to construct network management systems. Each network management system communicates at runtime via the message bus. Systems can dynamically connect to or disconnect from the message bus at runtime. In effect the message bus framework is a virtual machine, built from highly modular, interchangeable components based on an open architecture, with applications being the software available for this virtual machine.

3 Message Bus Communication

The message bus provide a powerful way to structure large network management systems to control complexity. Universal network management systems can be developed at a high level, relying upon sub network management systems for much of their functionality. Universal network management systems based on the message bus architecture are inherently modular and open, allowing very different sorts of systems to be intermixed. Since network management systems can be large and complex products, with few restrictions on how they are implemented internally, it becomes easier for a large number of people to make significant contributions to the universal network management systems, reducing the dependence on key personnel.

The message bus is also a means of structuring software, in that it defines system-level software architecture. A universal network management system is composed at a high level, relying upon network management systems for most of their functionality. The universal network management system can be large, while substantial NMS capable of independent execution. From the point of view of the message bus architecture they are merely interchangeable systems that share a standard interface to the message bus.

In addition to providing a range of messaging capabilities, the message bus provides facilities for service registration and lookup, data services, and distributed execution. Message bus clients (e.g., network management systems or services) can execute on any host computer connected to the message bus, allowing distributed network management systems to be easily constructed. A message bus console is used to examine and control the state of the message bus and monitor system activity.

At the simplest level the message bus is responsible only for classifying messages by type and handling message distribution and delivery. The actual content of a message, that is the data contained in a message, is determined by the messaging protocol used by the systems. Multiple messaging protocols are possible on the same bus although not necessarily recommended.

4 Implementation

We implemented the proposed a symmetric distributed server architecture for NMS. The whole network can be partitioned into small pieces each of which is monitored by a dedicated BES. In this way, we can manage a geographically distributed network or very large-scale network consisting of thousands devices in a reliable, good performance environments.

All the server tasks are distributed symmetrically without any server dedicated for one specific task. In this mode, the servers including front-end server and back-end server construct a matrix of server nodes. In case of updating a managed object, front-end server will locate the back-end where this object is managed and send updating request to this back-end. And every back-end will execute the core tasks on the sub-network assigned to it. This architecture has the best performance, reliability and scalability.

5 Conclusion

The symmetric distributed server architecture is a perfect solution to construct the large-scale, heterogeneous, and multi-tier network management system. By using a group of lower cost workstation servers instead of using large powerful mainframes, the total cost of the system will be lowered. Also, the system will be highly scalable. The message bus will provide flexible communication methods. Therefore heterogeneous NMS can be integrated together into a universal network management system.

References

1. Somers, F.: HYBRID: Unifying Centralized and Distributed Network Management using Intelligent Agents, In: Proc. IEEE (NOMS'96), IEEE Press, New York (1996) 34–43
2. Wang, W., Akyildiz, I. F.: Intersystem Location Update and Paging Schemes in Multi-tier Wireless Networks, In: Proceedings of IEEE MobiCom 2000, IEEE Press, New York (2000) 99–109
3. Kahani, M., Beadle, H. W. P.: Decentralized Approaches for Network Management, In: ACM Computer Communication Review, (1997) 36–47
4. Tennenhouse, D. L., Smith, J. M., Sincoskie, W. D., Wetherall, D. J., Minden, G. J.: A Survey of Active Network Research, In IEEE Communications Magazine, Vol. 35(1), IEEE Press, New York (1997) 80–86
5. Wies, R., Mountzia, M. A., Steenkamp, P.:A Practical Approach Towards a Distributed and Flexible Realization of Policies Using Intelligent Agents, In: Proc. 8th IFIP/IEEE Int. Workshop on Distributed Systems: Operations & Management (DSOM'97), Sydney, Australia (1997) 292–308
6. Yemini, Y.: The OSI Network Management Model, In IEEE Communications Magazine, Vol. 31(5), IEEE Press, New York (1993) 20–29

A Distributed Network Management Framework Based on NGI

Jinxiang Zhang, Jilong Wang, Jianping Wu, and Xing Li

Network Research Center of TsingHua University,
Beijing, 100084
jxzhang@cernet.edu.cn

Abstract. It is a difficult task to manage NGI (Next Generation Internet) efficiently. XML can allow for the interchange of data between management applications using different information models. This paper has put forward a novel network management framework using Independent Management Entity and XML technology in the management of NSFCNET(National Natural Science Foundation of China Network), which is a NGI testbed based on IPv6. The schemes for classifying the whole management system into independent management entities are presented in this paper, and the methods for access and definition of managed objects using XML are also given in this paper.

1 Introduction

As the network scale expands and the service provided by network becomes more complex, the traditional pattern of network management and information retrieval scheme become more and more difficult to meet the needs of network users. Furthermore, according to the new Moore Law, the bandwidth and traffic capacity can be doubled in six months. The number of managed objects increased exponentially[1,2]. Thus, the researches on NGI network management have been outspreaded from different aspects to obtain a network management system that is distributed, scalable, operational and information self-adaptive retrieval. In this background, as XML can allow for the interchange of data between management applications using different information models, network management technique based-on XML appears, which quickly becomes the hotspot in many fields, such as telecommunications network management technology, and distributed object technology etc., [3,4] Moreover, XML can interact with users by Web, So XML is increasingly used in the management of resource and service in NGI. Furthermore, The traditional network management way in a heterogeneous and wide area network is using a single network management system to achieve central-distributed management, which is not scalable and flexible. To solve the problem, we have classified the overall management system into various kinds of Independent Management Entity, which is an independent software entity able to run independently and manage some managed objects in a region or fulfill certain management function.

By combining merits of XML and Independent Management Entity technology, this paper has put forward a novel distributed network management framework

DNMN (Distributed Network Management of NSFCNET) based on NGI in the domain of the management of NSFCNET, which uses tree structure of managed objects arrangement, and the definition, implementation and finding schemes for managed objects using XML are also given by the tree structure, thus it can arrange and retrieve managed objects efficiently and has assured the performance of network management.

The remainder of this paper is organized as followings: section 2 presents the distributed network management framework; section 3 presents the implementation methods of managed objects based on XML; section 4 gives the integration scheme of Independent Management Entities; and section 5 gives the conclusion and future directions.

2 The Distributed Network Management Model Based on NGI

Several definitions are given as followings in order to explain our ideas.

Definition 1. *An XML Managed Object* is an object which is described by XML, can be accessed by management system and represent a few actual physical or logical objects. According to object oriented feature, it can be denoted as a triple $O_{mo} = \langle O_{name}, O_{attrs}, O_{opera} \rangle$, where O_{name} is name of the object, O_{attrs} is a group of attributes belonging to the object and O_{opera} is operations on the attributes..

Definition 2. *Related Logically Objects* are a group of managed object, they either belong to a common management function domain, or can be described by a managed object class, which means $B_{related} = B_{moc} \cup B_{func_dom}$, where $B_{related}$ is related logically objects, B_{moc} is the collection of objects belonging to a common object class and B_{func_dom} is the collection of objects belonging to a common management function domain.

Definition 3. *XML Cluster Object* is a collection of XML managed objects, which can be denoted as a duple $O_{cluster} = \langle C_{name}, C_{attrs} \rangle$, where C_{name} is cluster id, which can be a managed node name or managed system name, C_{attrs} are cluster attributes, which are the collection of all the managed objects. $C_{attrs} = \{O_{mo1}, O_{mo2}, \dots, O_{mon}\}$, where O_{mo} is object as defined in def. 1.

Definition 4. *Independent Management Entity* is an independent software entity able to run independently and manage some managed objects in a region or fulfill certain management function, which can be denoted as a duple $S = \langle B_{related}, S_{opera} \rangle$, where S_{opera} is the collection of the operations on all managed objects managed by the Independent Management Entity.

The distributed network management framework is illustrated in figure 1, the management of first level is user oriented manager, users can manage network by GUI(Graphical User Interface) and manager; the management of next level is Protocol-agent between Manager and Independent Management Entities. The third level is Independent Management Entities, which manage various management domain and function domain.

Thereinto, CP(Communication Protocol) represents the protocol used to integrated the Independent Management Entities, such as IIOP.

MP(Management Protocol) represents the protocol used to manage NGI, such as SNMP or CMIP/CMIS.

Protocol-agent is used to convert the MP information model to CP or vice versa.

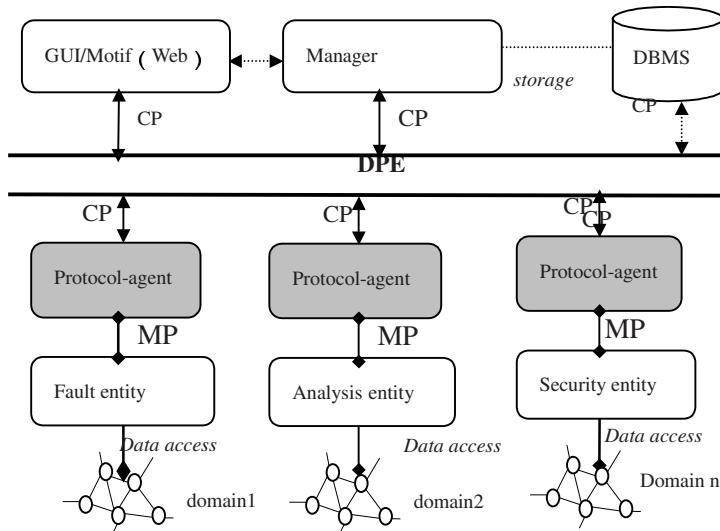


Fig. 1. DNMN Framework

3 Implementation Methods of Managed Objects Based on XML

A description template for a SNMP managed object belonging to Independent Management Entity using XML is shown as followings, which uses SMI/ASN.1 style definition of objects.

```

<OPERATION> Management Operation </OPERATION>
/*management operations on managed object such as Get, Get-next, Set etc.*/
< SMI/ASN.1_OBJECT>
<OBJECT-TYPE> OID of MO </OBJECT-TYPE>
<SYNTAX> Type for MO </SYNTAX>
<ACCESS> Access right for MO </ACCESS>
<STATUS> Status for MO </ STATUS>
<DESCRIPTION> Status for MO </DESCRIPTION>
< /SMI/ASN.1_OBJECT>

```

The managed objects belonging to Independent Management Entity can be arranged as a tree structure to provide efficient operation on managed objects. And we can use the operation algorithms for tree to fulfill the implementation and finding schemes for managed objects, and thus we can arrange and retrieve managed objects efficiently.

The description template using XML for a CMIP managed object which uses CMIS X.711 GDMO/ASN.1 style definition belonging to Independent Management Entity can be divided into several modules as the GDMO, such as template for managed object class, template for property, template for parameters and template for name-binding etc.

4 The Integration Scheme of Independent Management Entities

The traditional network management way in a heterogeneous and wide area network is using a single network management system to achieve central-distributed management, which is not scalable and flexible. To solve the problem, we have classified the overall management system into various kinds of Independent Management Entity, which is an independent software entity able to run independently and manage some managed objects in a region or fulfill certain management function. We have implemented an integrated environment that consists of different Independent Management Entities.

Each Independent Management Entity writes its own register information in ARF automatically at startup and removes its related information from ARF at shutdown. The network management system creates management interface as the ARF. The ARF is arranged as structural file and saved in MySQL. The portion of ARF is illustrated as followings.

```
<FUNCTION_HEAD> /* string, the name for the function module */
<INDEX_HEAD>
    /* string, the name for the sub function module if exist*/
</INDEX_END>
<LAYOUT_POS_HEAD>
    /*two integers for the position of function module in integrated environment*/
</LAYOUT_POS_END>
<INPUT_PARA_HEAD>
    /*string, for input parameters of function modules, such as IP address */;
</INPUT_PARA_END>
<RETURN_RESULT_HEAD>
    /*string, the return type for function module*/
</RETURN_RESULT_END>
<DIAGNOSIS_AIDED_HEAD>
    <APPLICATION_OBJ_HEAD>
        /*string, such as link or router*/
    </APPLICATION_OBJ_END>
    <LINK_INFO_HEAD>
        /*related with trouble-ticket, such as flow graph*/
    </LINK_INFO_END>
</DIAGNOSIS_AIDED_END>
<STATISTIC_INFO_HEAD>
    <REPORT_TABLE_HEAD>
        /*string, the report name provided by the function module*/
    </REPORT_TABLE_END>
    <REPORT_LINK_HEAD>
        /* the link with corresponding report name such as host IP address and directory*/;
    </REPORT_LINK_END>
</STATISTIC_INFO_END>
<HELP_INFO_HEAD>
    /* the link with corresponding help information such as host IP address and directory*/
</HELP_INFO_END>
<POSITION_INFO_HEAD>
    /*string, indicates the network address, description information, configure information
       and so on for function module*/
```

```
</POSITION_INFO_END>
<OBJECT_LIB_HEAD>
  /*the collection of the managed objects belonging to the function module, such as
   routers or links which are changeable dynamically */
</OBJECT_LIB_END>
</FUNCTION_END >
```

5 Conclusion and Future Works

The network management framework put forward in this thesis has combined the merits of Independent Management Entity and XML technology, and has been implemented to manage NSFCNET.

This research was supported by China post doctor science foundation and the National Natural Science Foundation of China under Grant Nos.69725003 and 90104002.

References

1. China Internet Network Information Center. Survey Report On Internet Development In China. China Internet Network Information Center, July,2002
2. JayantR, Harista, Michael O Ball et al. MANDATE: Managing networks using database technology. IEEE Journal on Selected Areas in Communications, 1993, 11(9):1361–1372
3. Jens-Peter Redlich et al., Distributed Object Technology for Networking. IEEE Communications Magazine, 1998, 36(10):100–111
4. Zhang Jinxiang, Wu Quanyuan, Wang Huaimin and Han Shu-ling. Design and implementation of a corba-based multi-hierarchy network management system. Journal of Computer Research and Development (in Chinese), 2001, 38(7):793–797
5. Zhang Jinxiang, WU JianPing, WANG JiLong. A Novel Network Management Model Based on XML for Service and Resource. Proceedings of Asia Pacific Advanced Network Conference. ShangHai, 2002:77–81

Performance Evaluation of Scheme Integrating Mobile IP and NHRP over ATM Networks

Tae-Young Byun¹, Min-Su Kim², and Ki-Jun Han²

¹ School of Computer and Electronic Engineering,
Gyeongju University,
Hyohyun-dong, Gyeongju, Kyungpuk, Korea
tybyun@gyeongju.ac.kr

² Department of Computer Engineering,
Kyungpook National University,
Sankyuk-dong, Pukgu, Taegu, Korea
mskim@comeng.ce.knu.ac.kr
kjhan@bh.knu.ac.kr

Abstract. In this paper, we present the operational procedures to integrate the NHRP and Mobile IP over ATM networks. The integration decreases the end-to-end path delay between a MN and CN by using the features of the ATM, which are fast switching and high scalability. We mathematically analyze the end-to-end path delay between end hosts in the integrated Mobile IP networks, also showing the improvement of delay by simulation.

1 Introduction

Currently there are proposals to incorporate IP-based technology into the core network of future wireless cellular systems [1][2][3][4]. Also, several solutions have been developed to support connectionless IP services over connection-oriented NBMA networks including the ATM networks [5][6][7][8]. In addition, as mobility of computing devices over IP networks becomes more important, integration technology of Mobile IP[11] and existing core network becomes a significant research area. Recently, integration of MPLS and Mobile IP also has been introduced to provide fast switching and scalability over ATM networks [9]. Since the NHRP(Next Hop Resolution Protocol) and NBMA(Non-broadcast Multiple Access) networks such as the ATM network are very closely related, it would be desirable to incorporate NHRP into these core networks too.

This paper is an extension of the previous study [10] to integrate the Mobile IP and NHRP. The previous work provided a solution for the incorporation of both the Mobile IP and NHRP into these future IP-based core networks, and also provides mobility support for NHRP. In addition, this paper includes the mathematical analysis of end-to-end path delay between CN and MN in mobile IP networks of two heterogeneous networks, which include the ATM network and the legacy LAN-based network. The organization of the rest of the article is as follows. Section 2 briefly presents the basics of NHRP and a short introduction to the Mobile IP basics, respectively. In Section 3, we present our scheme to integrate NHRP into the Mobile

IP in detail, and also show the mathematical analysis in terms of the end-to-end path delay between mobile node and correspondent node. Evaluation results by simulation are presented in Section 4. Finally, our conclusion is presented in Section 5.

2 Related Works

2.1 Next Hop Resolution Protocol

NHRP adds cut-through routing as an ATM service and hosts in LAN communicate with the NHRP clients through a router. This routing function allows the NHRP clients that are members of different LISs(Logical Internet Subnetworks) to establish a shortcut path through the ATM cloud. NHRP is a kind of address resolution service which maps IP address of destination hosts into the ATM addresses of the host, and provides shortcut routing between ATM hosts.

The NHRP's operation is initiated by the edge device and proceeds in a client-server manner. This can be caused by an explicit short-cut creation of an NHRP Resolution Request packet that is transmitted toward the destination along the default-routed path. The Resolution Request contains the source's Network layer address such as an IP address. The second phase of NHRP is the work done by the NHSs(NHRP Server) to find the ATM address belonging to determine if it is responsible for the target's IP-ATM mapping. If the present NHS does maintain the target's LIS, then it replies to the source with the target's ATM address. Also, if the path to the target is through a router or bridge placed at the egress of the ATM network, then that device's address is returned in the NHRP reply.

2.2 Mobile IP

Mobile IP is a protocol to support mobile computing over the Internet. A Mobile IP scheme has been adopted by the IETF for standardization in IPv4[7,8]. A Mobile Node(MN) is identified by the IP address it has when it is in its home network, called its home address. When a MN moves away from its home network to a foreign network, it obtains a temporary Care-of-Address(COA) from the Foreign Agent(FA) in the foreign network. The MN registers with a Home Agent(HA), which is typically a router, in its home network informing the latter of its COA. Any Correspondent Node(CN) wishing to communicate with the MN need not be aware that the MN has moved. It simply sends IP packets addressed to the MN's home address. These packets are routed via normal IP routing to the MN's home network, where they are intercepted by the HA. The HA encapsulates each packet into another IP packet which contains the MN's COA as its destination address. Thus, these packets are delivered to the MN's new location by a tunneling process.

3 Integration of NHRP and Mobile IP

Generally, routers are responsible for delivering IP packets to their destination. These routers exist on boundaries between two different networks, thus networks can be either LAN-based IP network or ATM-based IP network. Every packet forwarded by

the routers which are over the ATM-based IP networks has to undergo a comparably long processing delay due to the IP forwarding mechanism based on ‘store-and-forward’ in each router. We will improve the end-to-end path delay by utilizing a cut-through forwarding function that is an important capability of the ATM switch. With conforming the basic operation of the Mobile IP, AER(ATM Edge Router) or HA in the ATM networks can set a shortcut VC to FA. The shortcut VC has a role of tunnel partially or fully. Here, NHRP has a crucial role of setting a shortcut VC.

Table 1 enumerates four integration scenarios in which our scheme can be applied to reduce the end-to-end path delay between the CN and the MN. We assume that the CN, HA, FA and MN are distributed across LAN-based IP networks and NBMA-based IP networks. In Table 2, we define acronyms for the mathematical analysis of delay from CN to MN.

Table 1. Four Scenarios for integration of Mobile IP and NHRP

Scenario No.	Location of CN	Location of HA	Location of FA	Location of MN
1	LAN	LAN	ATM	ATM
2	LAN	ATM	ATM	ATM
3	ATM	LAN	ATM	ATM
4	ATM	ATM	ATM	ATM

Table 2. Acronym definitions

Acronym	Meaning
L_{R_i, R_j}^N	Link propagation delay from router R_i to router R_j over the IP network. N indicates an IP network, can be assigned to either LAN or ATM
$R_{R_i}^N$	Packet processing delay at router R_i over the IP networks. N indicates an IP network, can be assigned to either LAN or ATM
$S_{a,b}^{setup}$	Delay for establishing a shortcut VC from node a to node b over the ATM network
$ND_{i,j}^{NHRP-request}$	Delivery delay of NHRP-Request message from NHS_i to NHS_j over the ATM network
$ND_{i,j}^{NHRP-reply}$	Delivery delay of NHRP-Reply message from NHS_i to NHS_j over the ATM network
$ND_{i,j}^{NHRP-total}$	Total address resolution delay between NHS_i and NHS_j over the ATM network
$N_{NHS_s}^{NHRP-request}$	NHRP-Request message processing delay at NHS_s over the ATM network
$N_{NHS_s}^{NHRP-reply}$	NHRP-Reply message processing delay at NHS_s over the ATM network
LIS_s	Packet delivery delay in a LIS s over the ATM network
$\ell_{i,j}$	Propagation delay of the j^{th} ATM link in the i^{th} LIS over the ATM network
$s_{i,j}$	Cell switching delay of the j^{th} ATM switch in the i^{th} LIS over the ATM network
$D_{a,b}^{SN, R}$	Delay for delivering a packet from node a to node b in scenario SN, which is defined in Table 1. R indicates a packet forwarding mechanism that can be either a normal IP routing method based on the store-and-forward mechanism or a shortcut routing method based on cut-through mechanism.
R_{HA}^N	A packet processing delay at HA. N indicates an IP network, can be assigned to either LAN or ATM

3.1 Scenario 1: Shortcut VC between AER(ATM Edge Router) and FA

A typical network conforming to scenario 1 is illustrated in Fig. 1. Each entity consisting of a Mobile IP network is distributed across the LAN or ATM networks as shown in Table 1.

We obtain a packet delivery delay from the CN to HA as follows.

$$D_{CN,HA}^{1,IPRouting} = L_{CN,R_i}^{LAN} + R_{R_i}^{LAN} + \sum_{k=i}^{j-2} \{L_{k,k+1}^{LAN} + R_{k+1}^{LAN}\} + L_{R_{j-1},HA}^{LAN} + R_{HA}^{LAN} \quad (1)$$

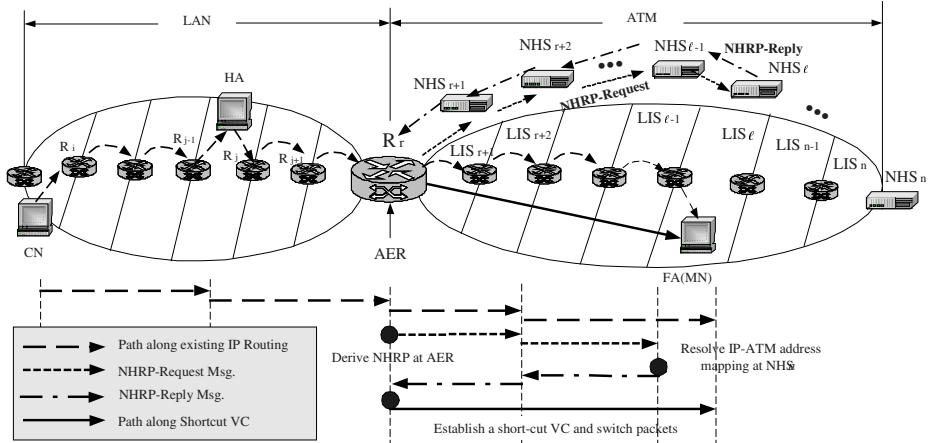


Fig. 1. An example of network conforming to scenario 1: CN and HA are located in the LAN-based IP network, FA and MN are located in a LIS over the ATM network.

Also, packet delivery delay from the HA to the AER(ATM Edge Router) is calculated as follows.

$$D_{HA,AER}^{\text{LIP Routing}} = L_{HA,R_j}^{LAN} + R_{R_j}^{LAN} + \sum_{k=j}^{r-1} \{L_{k,k+1}^{LAN} + R_{k+1}^{LAN}\} \quad (2)$$

Where, k is index of a router on path from the HA to the AER.

A packet delivery delay from the AER to the MN can be calculated by either an existing IP routing or a shortcut routing respectively.

- In an existing IP routing, we use a conventional IP forwarding that is based on the store-and-forward mechanism to deliver packets from the AER to the MN through FA. In this case, the packet delivery delay between the AER and MN can be obtained as follows:

$$D_{AER,MN}^{1, \text{IP Routing}} = \sum_{k=r}^{\ell-2} \{L_{k,k+1}^{ATM} + R_{k+1}^{ATM}\} + L_{R_{\ell-1},FA}^{ATM} + L_{FA,MN}^{ATM} \quad (3)$$

- In shortcut switching, we can fully utilize the cut-through switching capability of the ATM switch, thus it is possible to reduce the end-to-end path delay between the AER and the FA.

First, we should resolve the ATM address of the FA as the end of a tunnel that exists between the HA and FA. That is the reason that we use NHRP over the ATM network.

We can obtain total address resolution delay by adding a delivery delay of the NHRP-Request message and a delivery delay of the NHRP-Reply message as follows:

$$ND_{r,\ell}^{NHRP-request} = S_{R_r, NHS_{r+1}}^{setup} + \sum_{s=r+1}^{\ell-1} S_{NHS_s, NHS_{s+1}}^{setup} + \sum_{s=r+1}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=r+1}^{\ell} N_{NHS_s}^{NHRP-request} \quad (4)$$

$$ND_{r,\ell}^{NHRP-reply} = \sum_{s=r+1}^{\ell-1} S_{NHS_{s+1}, NHS_s}^{setup} + \sum_{s=r+1}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=r+1}^{\ell-1} N_{NHS_s}^{NHRP-reply} + S_{NHS_{r+1}, R_r}^{setup} \quad (5)$$

$$\begin{aligned} ND_{r,\ell}^{NHRP-total} &= ND_{r,\ell}^{NHRP-request} + ND_{r,\ell}^{NHRP-reply} \\ &= 2 \sum_{s=r+1}^{\ell-1} S_{NHS_s, NHS_{s+1}}^{setup} + 2 \sum_{s=r+1}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=r+1}^{\ell} N_{NHS_s}^{NHRP-request} + \sum_{s=r+1}^{\ell-1} N_{NHS_s}^{NHRP-reply} + 2S_{R_r, NHS_{r+1}}^{setup} \end{aligned} \quad (6)$$

After resolving the ATM address corresponding to the COA of the MN in which the IP address of the FA, AER establishes a shortcut VC to the FA by using ATM signaling procedures. Then, the AER correctly delivers packets to a MN through the FA along with pre-established short-cut VC. Packet delivery delay from the AER to the MN along with shortcut VC is expressed in equation (7).

$$D_{AER,MN}^{1, ShortcutRouting} = S_{AER, FA}^{setup} + \sum_{s=r+1}^{\ell} LIS_s + L_{FA,MN}^{ATM}, LIS_s = \sum_{k=1}^{m_s} \ell_{s,k} + s_{s,k} \quad (7)$$

Where, m_s indicates the number of ATM switches along with a shortcut path in s^{th} LIS, may be different in each LIS. ℓ_{ij} means the propagation delay of the j^{th} link along with shortcut path in the i^{th} LIS. Also s_{ij} indicates a cell switching delay of the j^{th} ATM switch in the i^{th} LIS.

3.2 Scenario 2: Shortcut VC between the HA and FA

An example network of scenario 2 is illustrated in Fig. 2. Different from scenario 1, the HA is located in the ATM. In this, we can fully establish a shortcut VC over a tunnel between the HA and FA. This indicates that our scheme can overcome tunneling by using existing IP routing in view of the tunnel path length. This possibility is due to a shorter tunnel in comparison with an existing tunnel using IP routing based on the store-and-forward mechanism. Additionally, cut-through switching of the ATM switch reduces tunneling path delay. Once a shortcut VC has been established, all packets along with the shortcut VC do not experience a store-and-forward delay that occur in each router. Instead, packets are directly delivered along with the VC shortcut to the FA not via a router.

Mathematical analysis of the end-to-end path delay in scenario 2 is very similar to that of scenario 1. The following equations are straightforward, we will explain briefly in order to avoid repetition.

We can obtain packet delivery delay from the CN to the HA as follows:

$$D_{CN,AER}^{2,IPRouting} = L_{CN,R_i}^{LAN} + R_{R_i}^{LAN} + \sum_{k=i}^{j-1} \{L_{k,k+1}^{LAN} + R_{k+1}^{LAN}\} \quad (8)$$

Also, packet delivery delay from the AER to the HA is calculated as follows:

$$D_{AER,HA}^{2,IPRouting} = \sum_{k=r}^{j-2} \{L_{k,k+1}^{ATM} + R_{k+1}^{ATM}\} + L_{R_{j-1},HA}^{ATM} + R_{HA}^{ATM} \quad (9)$$

Where, k is the index of a router on path from the AER to the HA.

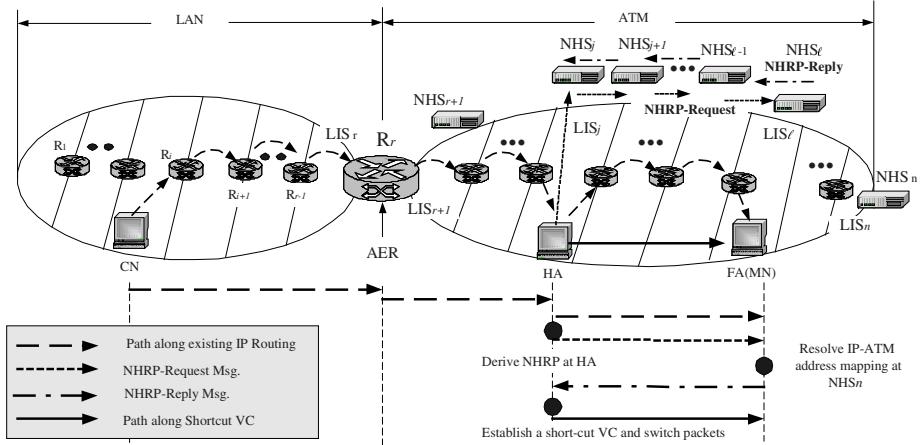


Fig. 2. An example of network conforming to scenario 2: the CN is located in the LAN-based IP network and the HA, FA and MN are located in the ATM network.

Packet delivery delay from the HA to the MN can be calculated by either an existing IP routing or a shortcut switching respectively.

■ The packet delivery delay between the HA and the MN by existing IP routing can be obtained as follows:

$$D_{HA,MN}^{2,IPRouting} = L_{HA,R_j}^{ATM} + R_{R_j}^{ATM} + \sum_{k=j}^{\ell-2} \{L_{k,k+1}^{ATM} + R_{k+1}^{ATM}\} + L_{R_{\ell-1},FA}^{ATM} + L_{FA,MN}^{ATM} \quad (10)$$

■ By using shortcut VC of cut-through switching, it is possible to reduce the end-to-end path delay between the HA and FA.

First, we obtain a total address resolution delay by adding a delivery delay of the NHRP-Request message and a delivery delay of the NHRP-Reply message as follows:

$$ND_{j,\ell}^{NHRP-request} = S_{HA,NHS_j}^{setup} + \sum_{s=j}^{\ell-1} S_{NHS_s,NHS_{s+1}}^{setup} + \sum_{s=j}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=j}^{\ell} N_{NHS_s}^{NHRP-request} \quad (11)$$

$$ND_{j,\ell}^{NHRP\text{-}reply} = \sum_{s=j}^{\ell-1} S_{NHS_{s+1},NHS_s}^{\text{setup}} + \sum_{s=j}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=j}^{\ell-1} N_{NHS_s}^{NHRP\text{-}reply} + S_{NHS_j,HA}^{\text{setup}} \quad (12)$$

$$\begin{aligned} ND_{j,\ell}^{NHRP\text{-total}} &= ND_{j,\ell}^{NHRP\text{-request}} + ND_{j,\ell}^{NHRP\text{-reply}} \\ &= 2 \sum_{s=j}^{\ell-1} S_{NHS_{s+1},NHS_s}^{\text{setup}} + 2 \sum_{s=j}^{\ell} \sum_{k=1}^{n_s} (\ell_{s,k} + s_{s,k}) + \sum_{s=j}^{\ell} N_{NHS_s}^{NHRP\text{-request}} + \sum_{s=j}^{\ell-1} N_{NHS_s}^{NHRP\text{-reply}} + 2 S_{NHS_r,NHS_{r+1}}^{\text{setup}} \end{aligned} \quad (13)$$

A packet delivery delay from the HA to the MN along with a shortcut path is expressed in equation (14).

$$D_{HA,MN}^{2,\text{ShortcutRouting}} = S_{HA,FA}^{\text{setup}} + \sum_{s=j}^{\ell} LIS_s + L_{FA,MN}^{\text{ATM}}, LIS_s = \sum_{k=1}^{m_s} \ell_{s,k} + s_{s,k} \quad (14)$$

3.3 Scenario 3: Shortcut VC between AER and FA

An example network of scenario 3 is illustrated in Fig. 3.

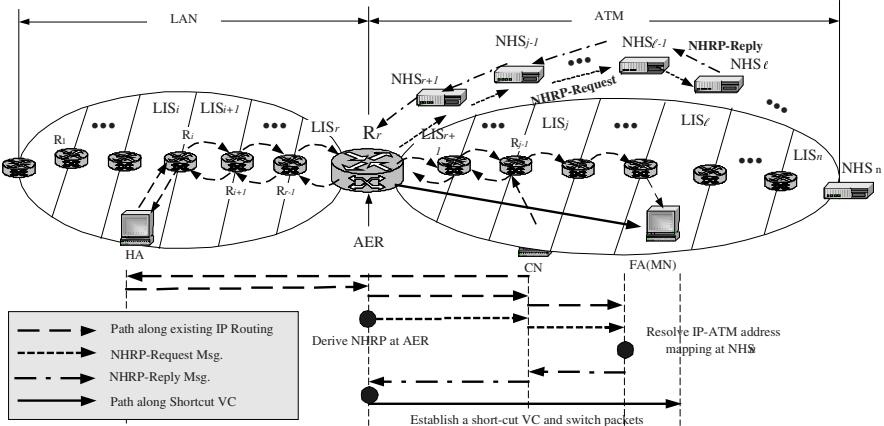


Fig. 3. An example of a network conforming to scenario 3: the HA is located in the LAN-based IP network and the CN, FA and MN are located in the ATM-based IP network.

Scenario 3 is similar to scenario 1 in that the HA is located in the LAN-based IP network, but the FA or MN is in the ATM-based IP network. So, we can establish a shortcut VC that covers partial tunneling paths between the AER and FA. This indicates that the partial tunnel path using a shortcut VC provides a greater decrease in path delay more than the tunnel path using an existing IP routing based on the store-and forward mechanism. A mathematical analysis of the end-to-end path delay between the CN and the MN is almost identical to that of the end-to-end path delay in scenario 1 and since it has been discussed previously it will not be included in this subsection.

3.4 Scenario 4: Shortcut VC between the HA and FA

An example network of scenario 4 is illustrated in Fig. 4. Scenario 4 is similar to scenario 2 in that both of the HA and FA exist in the ATM-based IP network. We, therefore, can establish a shortcut VC that fully covers tunneling paths between the HA and FA. This indicates that the full path of the tunnel along the shortcut VC provides a greater decrease in path delay than the existing tunnel path along the IP. A mathematical analysis of the end-to-end path delay between the CN and MN is almost identical to that of the end-to-end path delay in scenario 2.

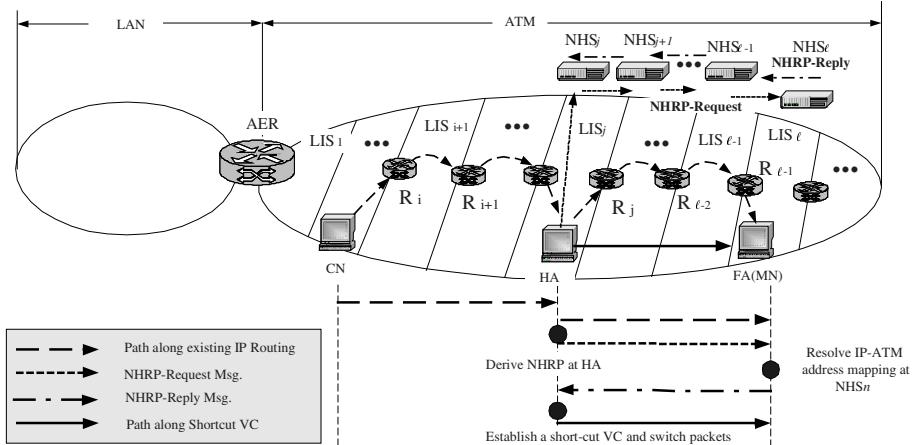


Fig. 4. An example of a network conforming to scenario 4: All entities are located in the ATM network.

4 Performance Evaluation

4.1 Simulation Models and Parameters

To evaluate the NHRP and Mobile integration performance, we designed two IP network models that conform to scenarios 1 and 2 respectively, as shown in Table 1. Each network model consists of LAN-based IP networks and an ATM-based IP network. In total, more than a hundred simulations were conducted meaning that more than fifty simulations per scenario were done. Two network models have been designed and evaluated by using COMNET-III, which is a powerful network simulator.

The simulation results reported in this paper are based on the average value of about fifty measurements per scenario. Major simulation parameters and values are shown in Table 3. We also streamed some traffic into two heterogeneous networks by two traffic generation nodes in order to obtain a stable end-to-end path delay between the CN and MN.

Table 3. Simulation parameters

	Parameter	Value
LAN-based IP network	The number of LANs	8
	Bandwidth (Mbps)	100
ATM-based IP network	The number of LISs	3
	The number of ATM switches in a LIS	19
	The number of NHSs	3
	Bandwidth (Mbps)	155
NHRP Control Messages	Message Length (byte)	1024
	Occurrence Frequency (sec)	Poisson(5.0)
Traffic Load (Tr)	Two nodes flow traffic in networks as a mount of Tr (10MBytes or 5MBytes) per 1 second respectively	

4.2 End-to-End Path Delay from the CN and MN

During this simulation, we increased the variable k that implies the degree of a reduced path length and varies from 0% to 50%. The variable k is defined as follows:

$$k = \left(1 - \frac{PathLength_{shortcutVC}}{PathLength_{ipRouting}}\right) \times 100 \quad (15)$$

$PathLength_{shortcut}$ meaning the path length of a shortcut VC which is established after a successful NHRP operation, exists between the AER(or HA) and FA. On the other hand, $PathLength_{ipRouting}$ means the length of the path which is established by normal routers. Generally, $PathLength_{shortcut}$ is smaller than $PathLength_{ipRouting}$. If we assume that $k=10\%$, this indicates that, the path length of a shortcut VC is equivalent to 90 percent of the path length by an IP routing, that shortcut VC decreased by about 10 percent compared with a path length by an IP routing.

To verify the enhancement of our scheme in comparison with existing IP routing, we measured the end-to-end path delays in three cases.

- Case 1: The end-to-end path delay from the CN to the MN by existing IP routing based on the store-and-forward method. In this case, we did not use a shortcut VC over the ATM network.
- Case 2: Different from Case 1, we used a shortcut VC between the AER(or HA) to the FA. Additionally, the end-to-end path delay includes a total address resolution delay by NHRP for the first packet that flowed into the ATM networks.
- Case 3: This case is similar to Case 2 in that it uses a shortcut VC between the AER(or HA) and the FA, but it does not include address resolution operations using NHRP. Since address resolution has already been completed for the first packet that flowed into the ATM network, the following packets do not need to resolve the ATM address of the FA, and are switched along the shortcut VC which was established in Case 2.

The measurements are plotted in Fig. 5 and Fig. 6. Each point on the graph was obtained by averaging 50 consecutive measurements. We find that the delay of Case 2 is slightly greater than that of Case 1 due to additional address resolution overhead. But, as shown in case 3 on the graphs, after either the HA or the AER has completed

address resolution using NHRP, the following packets are faster than the packets in Case 1. Case 1, therefore, reduces the delay in the amount of the difference between Case 1 and Case 3. The most important reason of that result is that the store-and forward processing delays on routers over the ATM networks are removed in Case 3.

Also, we note that as k grows, the delays in Case 2 and Case 3 decrease. That results from the reduced path length of the shortcut VC.

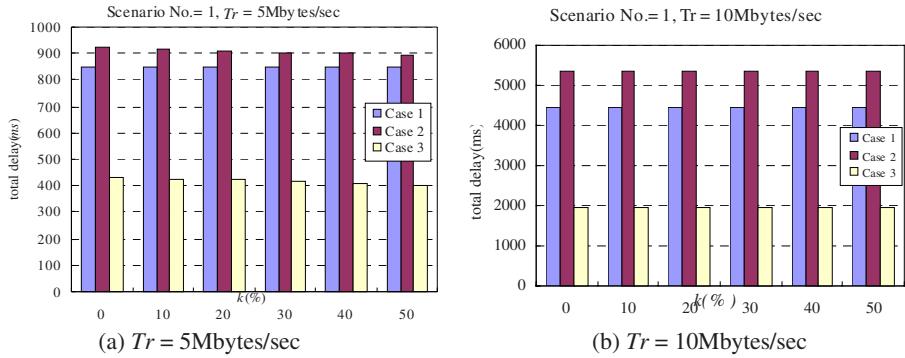


Fig. 5. Averaged end-to-end path delay from the CN to the MN in scenario 1

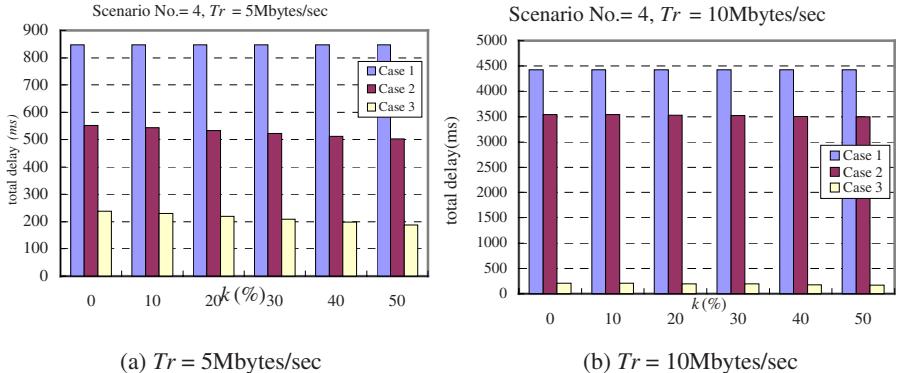


Fig. 6. Averaged end-to-end path delay from the CN to the MN in scenario 4

5 Conclusion

In this paper, we provided the signaling and control mechanisms to integrate the Mobile IP and NHRP. We use NHRP and established a shortcut VC to switch the packet. Switching is much faster than with the conventional IP forwarding, and the transmission delay and the packet-processing overhead is reduced.

By simulation, we showed that the integration of NHRP and Mobile IP overcome the existing Mobile IP operation based on the IP routers in view of the end-to-end path delay.

This work can easily be extended for optimized path routing in Mobile IPv6 and the proposed scheme also can be applied to other NBMA networks in addition to the ATM network.

References

1. P. C. Mason, J. M. Cullen, N. C. Loble, "UMTS Architectures," Mobile Communications Towards the Next Millennium and Beyond, IEE Colloquium, pp.401–411, 1996
2. R. H. Katz, et al., "ICEBERG: An Internet-Core Networks Architecture for Integrated Communications," IEEE Personal Communications.
3. A. T. Campbell, J. Gomez, A. G. Valko, "An Overview of Cellular IP," IEEE Wireless Communications and Networking Conference(WCNC'99), New Orleans, Sept. 1999
4. D. O'Mahoney, "UMTS: the fusion of mixed and mobile networking," IEEE Internet Computing. Volume:2, pp.49–56, 1998
5. N. Finn, T. Mason, "ATM LAN Emulation," IEEE Communication Magazine, pp. 96–100, June 1996
6. M. Laubach, "Classical IP and ARP over ATM," RFC 1577, Dec. 1993
7. ATM Forum, "Multi-protocol over ATM Specification, Version 1.1," af-mpoa-0114.0000, May 1999
8. J. Luciani, D. Katz etc., "NBMA Next Hop Resolution Protocol(NHRP)," RFC 2332, Apr. 1998
9. Z. Ren, C. K. Tham etc., "Integration of Mobile IP and Multi-Protocol Label Switching," ICC2001, June 2001.
10. T. Y. Byun, M. H. Cho, "Integration of Mobile IP and NHRP over ATM Networks," Lecture Notes in Computer Science, Vol. 2642, Apr. 2003
11. C. Perkins, ed., "IPv4 Mobility Support", RFC 2002, Oct. 1996

Improving Availability of P2P Storage Systems*

Shuming Shi, Guangwen Yang, Jin Yu, Yongwei Wu, and Dingxing Wang

Department of Computer Science and Technology,
Tsinghua University
Beijing 100084, P. R. China
{ssm01,yujin}@mails.tsinghua.edu.cn
{ygw,wuyw,dxwang}@tsinghua.edu.cn

Abstract. This paper discusses how to building high available storage systems on top of peer-to-peer infrastructure. We first demonstrate that traditional definitions of availability are not suitable for distributed systems. Then the application-specific availability is defined and analyzed in this paper. We observed by experiments that, in a P2P storage system with frequently node join and leave, availability can't be improved effectively by pure redundancy (replication or erasure coding). A method is proposed in this paper which combines peer auto-selection and redundancy to effectively improve the availability of P2P storage systems.

1 Introduction

Many P2P-based cooperative global storage systems have emerged [12, 3, 5]. Peer-to-peer systems typically lacks dedicated centralized infrastructure, but rather depend on the voluntary participation of peers to contribute their resources out of which the infrastructure is constructed.

Although P2P researches often treat all peers equally. There is actually a significant amount of heterogeneity in current deployed peer-to-peer systems. It has been observed [13] that bandwidth, latency, and availability vary between three and five orders of magnitude across the peers in Gnutella and Napster. In a P2P network, as peers are unmanaged volunteer participants, the availability of hosts can't be guaranteed. Bhagwan etc. [1] found that over 20% of the hosts in Overnet [8] arrive and depart every day. Saroiu etc. [13] observed in there experiments that the best 20% of Napster peers have an uptime of only 83% or more. As peer-to-peer networks are often comprise of hosts with relatively poor and highly variable availability, it is a challenge to design and implement an efficient, high available P2P storage system.

The traditional definition of availability is not suitable for distributed systems for two reasons: The first, a distributed system doesn't completely fail, but rather suffers performance degradation. The second, availability should not be defined as a simple average over a given time interval. In this paper, we give the definition of *application-specific availability*.

* This work is supported by the National Natural Science Foundation of China under Grant 60173007 and the National High Technology Development Program of China under Grant 2001AA111080.

Redundancy is a primary way to improve availability. There are two basic means of redundancies: replication and erasure coding. Existing storage systems use one or both of them to improve availability. But, by our experiments, neither replication nor erasure coding can *effectively* improve the availability of highly dynamic P2P storage systems.

It is intuitive that high available nodes should store more data than nodes which are always down. We design a double-layer DHT in this paper which combines peer selection with redundancy to achieve effective availability improvement. First, the availability of the whole system is improved by only storing data on relatively highly available hosts. Then, redundancy is adopted to improve availability further. Experiments showed that our double-layer DHT can improve the availability of P2P storage systems more efficiently than just redundancy.

The rest of the paper is as follows: Section 2 presents the definition of application-specific availability. Section 3 presents our method to effectively improve the availability of P2P storage systems. Some experiments are performed in section 4 to validate the merits of our methods. Related work is discussed in section 5. The final section is the conclusion and future work.

2 Application-Specific Availability

The research and industry communities adopt definitions for availability standards based on the tolerable amount of annual downtime. A system capable of Class-5 availability means it will experience less than five minutes of downtime per year or 99.999% uptime. The availability is defined as:

$$p = \frac{\text{uptime}}{\text{uptime} + \text{downtime}} \quad (2.1)$$

The traditional definition works well for measuring the availability of a single processor, disk or other device, where devices are either available or down. However, they fall short in measuring the availability of distributed systems (i.e. peer-to-peer networks), because a distributed system doesn't completely fail, but rather suffers performance degradation [4]. One choice is to use the service that one stream of customers sees to estimate system availability. But it is actually the availability of a part of the system, not the true availability of the whole system. Thus, we need to view availability at a single point of time as a spectrum instead of a binary metric 0[2, 4]. If define $A(t)$ as the availability of a system at a given instant of time t , then we should allow $A(t)$ to be any value from 0 to 1. Define $A(t_1, t_2)$ as the availability within a given time interval $[t_1, t_2]$, then the availability can be defined as:

$$p = A(t_1, t_2) = \left(\int_{t=t_1}^{t_2} A(t) dt \right) / (t_2 - t_1) \quad (2.2)$$

Sometimes it is inappropriate to define availability as a simple average over a given time interval. Consider the example of Figure 1. The availability functions of two systems are shown in this figure as $A_1(t)$ and $A_2(t)$ respectively. Compute availability according to Formula 2.2, we have: $A_1(0, T) = A_2(0, T) = 0.9$. That means

the two systems have the same availability. However, they are actually quite different. System 1 runs in a “perfect” state in 90 percent of time, but is completely “down” in the remaining time. While system 2 always runs in a 90% degraded state. For some applications (i.e. P2P file sharing, and distributed search engine), as 90% query result is enough for users, system 2 has more benefit than system 1 for its uninterruptibility. While for applications demanding for nearly “perfect” services, system 2 is unusable at all. So the requirement of applications must be considered in defining the availability of a distributed system.

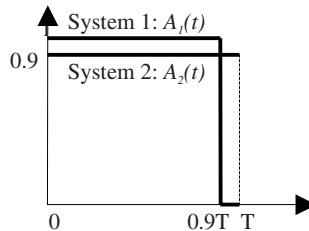


Fig. 1. The availability functions of two imaginary systems

Now we give the definition of **application-specific availability** as follows:

$$p = A(t_1, t_2) = \left(\int_{t=t_1}^{t_2} f(A(t)) dt \right) / (t_2 - t_1) \quad (2.3)$$

Here $f(x)$ is an application-specific function with the following attributes:

- 1). $f(0) = 0; f(1) = 1$
- 2). $\forall 0 \leq x_1 \leq x_2 \leq 1, f(x_1) \leq f(x_2)$.

Now we can satisfy the requirement of different applications by giving different definition of $f(x)$. Specially, by defining $f(x)=x$, we get the availability definition of Formula 2.2. We believe that the function f defined by the following formula can fulfill the need of a mass of applications:

$$f(x) = \begin{cases} \frac{p_1 x}{r_1} & \text{if } x < r_1 \\ \frac{(p_2 - p_1)x + (p_1 r_2 - r_1 p_2)}{r_2 - r_1} & \text{if } r_1 \leq x < r_2 \\ \frac{(1 - p_2)x + (p_2 - r_2)}{1 - r_2} & \text{if } x \geq r_2 \end{cases} \quad (2.4)$$

The above function comprise of three line segments (see Figure 2 (a)) connected one by one. The length and gradient of the line segments are controlled by four parameters: r_1, r_2, p_1, p_2 . Let $r_1=r_2, p_1=0, p_2=1.0$ (see Figure 2 (b)), we get the traditional definition in Formula 2.1. Let $(r_1, r_2, p_1, p_2) = (0, 0, 0, 0)$ (see Figure 2 (c)), we get the definition in Formula 2.2.

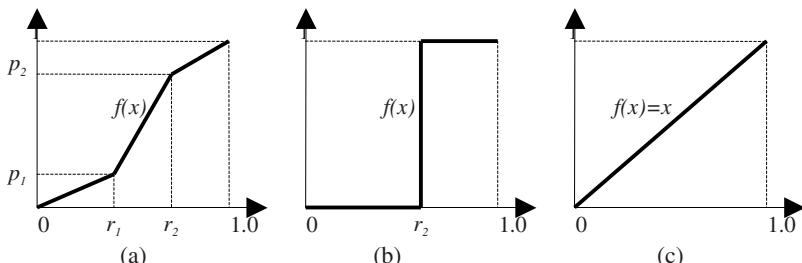


Fig. 2. A representative application-specific function and its two particular cases

To distinguish with the application-specific availability, we call the traditional definition of availability as absolute availability.

3 Improving Availability Effectively

This section discusses how to improve the availability of P2P storage systems effectively. We first demonstrate that only redundancy is not enough to improve availability.

3.1 Is Redundancy Enough to Improve Availability?

Redundancy is a primary way to improve availability. There are two basic means of redundancies: replication and erasure coding. Erasure coding is a process that treats input data as n fragments and transforms them into m ($m > n$) fragments to guarantee that any n (or slightly more) of the coded fragments are sufficient to construct the original data. Example of erasure codes includes Reed-Solomon codes [9] and Tornado codes [7].

To improve availability, PAST and CFS use replication, while OceanStore uses erasure coding to store archival objects. But how fine does redundancy work in highly dynamic P2P networks? Is it enough to guarantee high availability?

To answer the above questions, we do experiments using statistic data from real P2P networks. We use two distributions of host availability (see Table 3 and Table 4), as is explained in section 4. The setup and methodology of experiments are also illustrated in section 4. All data shown in this section are acquired from a P2P system containing 4096 hosts. We vary the number of hosts in the system, and similar results are obtained.

To test the effect of replication, we try different number of replicas ($k=1, 2, 4, 8, 16$), and the result is shown in Table 1. We can see that the availability of the system remains less than 99.9% even when 8 replicas are adopted. From Table 2, we can see that the 1/2-erasure coding behaves badly because of poor host availability. The 1/4-erasure coding also behaves not so good by the same reason. Thus, when a P2P storage system is comprised of hosts with relatively poor and highly variable availability, its availability can't be improved efficiently by just redundancy.

Table 1. The effect of replication on availability improvement (DHA1: distribution 1 of host availability; DHA2: distribution 2 of host availability)

K (Number of replicas)	1	2	4	8	16
Availability for DHA1	36.5	58.8	84.6	97.99	99.94
Availability for DHA2	53.2	77.6	95.5	99.74	99.999

Table 2. The effect of erasure coding on availability improvement

n/m	2/4	4/8	8/16	16/32	2/8	4/16	8/32
Availability for DHA1	48.2	33.9	19.1	9.4	87.0	88.7	94.70
Availability for DHA2	72.4	68.8	65.2	69.9	97.3	99.4	99.97

3.2 Our Scheme: A Double-Layer DHT

It is intuitive that high available nodes should store more data than nodes which are always down, in other words, the amount of data stored should be proportional to the availability of hosts. However, there are two challenges to achieve this: The first, how to store more data on high available nodes. As most peer-to-peer storage systems are built on top of DHTs (distributed hash tables [10, 11, 14, 17]) for efficient lookup, the locations of data items are judged by the hashing of fileId or blockId, resulting in nearly uniform distribution of data items on hosts. The second challenge is how to avoid the overloading of high available nodes if more data are stored on them.

We design and implement a double-layer DHT to meet these challenges. As shown in Figure 3, the double-layer DHT consists of two parts: an lower-layer DHT and an upper-layer DHT. The lower-layer DHT organizes all nodes of the system in an efficient and scalable way. A peer-selection module built on the lower-layer DHT is used to collect node availability information and to automatically choose high available nodes (called super-hosts) with which the upper-layer DHT is constructed. Data items are only stored in the upper-layer DHT. Thus, the availability of the whole system is improved by only storing data on relatively high available hosts. Replication or erasure coding is adopted by the upper-layer DHT to improve availability further. Experiments in the next section will show that the combination of peer-selection and redundancy can improve the availability of the system more efficiently.

A health monitoring module is built on the upper-layer DHT to automatically collect load and availability information of nodes. If the upper-layer DHT is overloaded, then more nodes are selected automatically by the peer-selection module from the lower-layer DHT to join the upper layer. When the load of the upper-layer DHT is light, then some nodes with low availability can be considered to be deleted and go back to the lower-layer DHT.

A key issue is how to make the peer-selection process automatic and scalable, which will be discussed in the following sub-section.

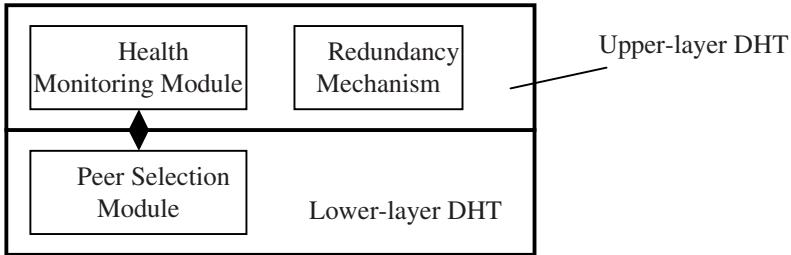


Fig. 3. Architecture of our double-layer DHT

3.3 Peer Selection and Health Monitoring

It is a challenge to collect availability information of hosts and choose the most highly available hosts from a large distributed system. A series of problems must be solved: How to determine the availability of a host? How to select highly available hosts in a scalable and efficient way? How to monitor the health of the upper-layer DHT?

Estimate host availability. Hosts in a peer-to-peer system are unmanaged volunteer participants. And some peers tend to deliberately misreport information if there is an incentive to do so [13]. So it is unpractical to report availability information of a host by itself. Since heartbeat messages are maintained between every pair of neighbors. The neighbors of a host know when it is “up” or “down”. So, one possible way is to have some or all neighbors of a host to estimate its availability by history availability information.

Select highly available hosts. SOMO [16]0 is a highly scalable, efficient and robust infrastructure built on any DHT to perform resource management. SOMO builds a soft-state tree on top of a DHT with each host maintains one or several SOMO nodes. Statistic information can be collected along with the tree from bottom to up, and as a result, the root of the SOMO tree contains the final statistics about the DHT. We use SOMO to perform peer selection in this paper. For any host, its availability is estimated by its neighbors and sent to a certain SOMO node. At the end, some most highly available hosts in the lower-layer DHT is appeared in the root SOMO node. These hosts then join the upper-layer DHT. This process is performed again and again until the upper-layer DHT has enough hosts to contain all the data items (files or blocks).

Health monitoring. It is necessary to monitor the condition of the upper-layer DHT to make it to have an appropriate size. Too small DHT size would make some hosts be overloaded; while too large size decreases system availability. Health monitoring can also be performed using SOMO. Free disk space instead of availability is collected along with the SOMO tree from bottom to up. As a result, total free space of the upper-layer DHT is computed.

4 Experiments

In order to demonstrate the merits of our design, we use two distributions of host availability to do our experiments. Saroiu etc. in [13] presents the IP-level uptime and application-level uptime (measured by the percentage of time a peer is reachable) of peers in Napster and Gnutella. We adopt approximately the statistic information for application-level uptime of Napster as our first host availability distribution (DHA1), as shown in Table 3. The second host availability distribution (DHA2, see Table 4) is from [1], an availability measurement on the Overnet file sharing network [8].

Table 3. Host availability distribution 1 (HDA1)

Percent of Hosts	20%	20%	20%	20%	10%	10%
Availability (%)	0.1 – 4.0	4.0 – 15.0	15.0 – 35.0	35.0 – 80.0	80.0 – 95.0	95.0 – 99.9

Table 4. Host availability distribution 2 (HDA2)

Percent of Hosts	10%	10%	10%	10%	10%
Availability (%)	0.1 – 15.0	15.0 – 22.0	22.0 – 30.0	30.0 – 40.0	40.0 – 50.0
Percent of Hosts	10%	10%	10%	10%	10%
Availability (%)	50.0 – 62.0	62.0 – 75.0	75.0 – 85.0	85.0 – 95.0	95.0 – 99.0

For each host availability distribution, we build a P2P storage system containing N (N is from 256 to 8192 in our experiments) hosts. The most highly available M hosts (super-hosts) are selected to construct the upper-layer DHT. We set M to be N/2, N/4, N/8 respectively. Totally W (we set W=1000N in our experiments) files are stored into the system with their location judged by hashing their fileIds. After inserting all these files, we test system availability by trying to retrieve the files. For any instant time t, we test the availability of each of the files. If there are V files can be retrieved, then we say the availability of the system at time t is V/W, that is, $A(t)=V/W$. We test for a long time interval T (long enough to guarantee any host in the system to “up” and “down” for many times), and then use Formula 2.2 to compute the availability of the system.

To achieve k -replication, each file is replicated and sent to k hosts by hashing. For testing the effect of erasure coding, we split each file into n fragments and encode them into m fragments. At any time t, we call a file can be retrieved if at least n of the encoded fragments can be retrieved.

Figure 4 shows the availability with different size of the upper-layer DHT, using DHA1. We can see from the figure that the availability can be improved greatly by reducing M (the size of the upper-layer DHT). That is easy to understand, for files are stored on relatively highly available hosts. But, using the same redundancy mechanism, small also M means more files averagely on each host, or heavy load on the upper-layer DHT.

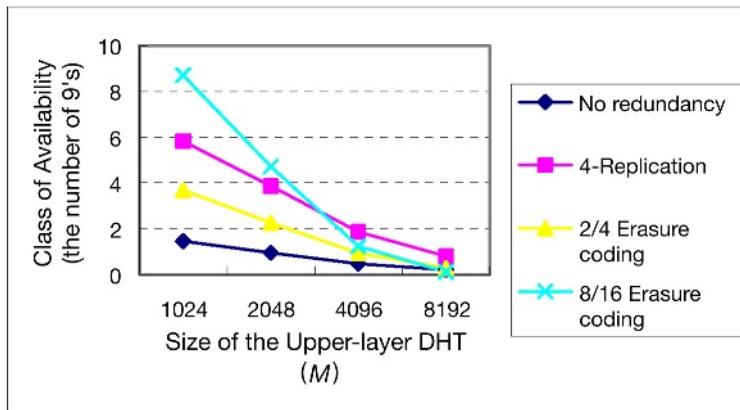


Fig. 4. Availability obtained with different size of the upper-layer DHT and different redundancy mechanisms. N=8192

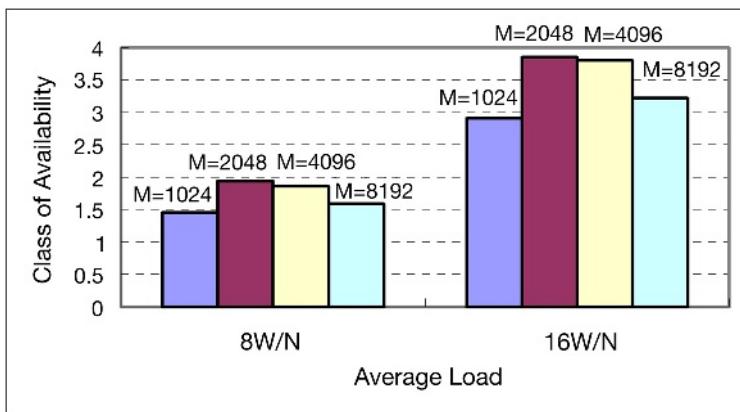


Fig. 5. Peer-selection plus replication improves availability without increasing load on hosts. N=8192. DHA1.

It is surprising that, even with the same load, peer-selection may still improve availability. In figure 5, we fix the average load on the upper-layer DHT, and then compare the effect of different M (and so different replication mechanisms). We can see from Figure 5 that 4096 super-hosts plus 4-replication achieves higher availability than 8192 hosts plus 8-replication. But when M decreases more, the equation doesn't hold, i.e. 1024 super-peers plus 2-replication achieves lower availability than 2048 super-peers plus 4-replication. Figure 6 shows the effect of peer-selection plus erasure coding, using DHA2 as the host availability distribution. We can see that, with the same average load, 4096 super-hosts plus 4/16 erasure coding achieves higher availability than 8192 hosts plus 2/16 erasure coding.

To sum up, these experiments demonstrate that peer-selection plus redundancy can improve availability of P2P storage systems more efficiently than pure redundancy.

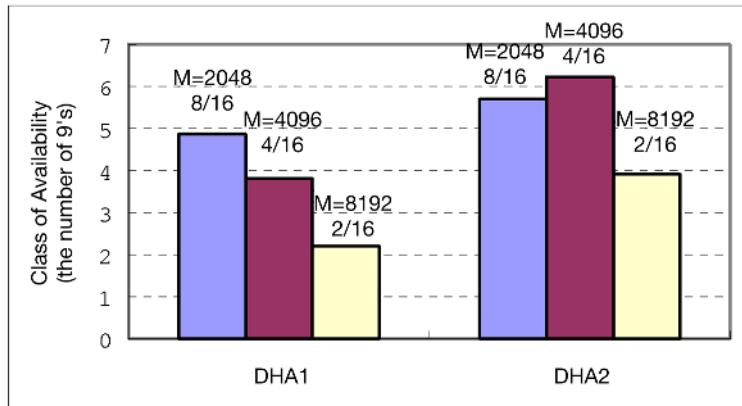


Fig. 6. Peer-selection plus erasure coding improves availability without increasing load on hosts. N=8192.

5 Related Work

There are several peer-to-peer storage systems in use. Among the most prominent are PAST [12], CFS [3] and OceanStore [5]. All of them provide mechanisms to improve the availability of their systems. PAST stores each file on the k nodes whose nodeIds are numerically closest to the 128 most significant bits (msb) of the file's fileId. CFS uses a similar way to replicate each block on k nodes. Objects of OceanStore exist in both active and archival forms. The archival versions are encoded with an erasure code and spread over the hosts. Besides availability, these systems focus on scalability, self-organization, security, etc. However, all these systems only use redundancy to achieve high availability, so they are not suitable for circumstances with considerable host turnover. In comparison, we combine peer-selection and redundancy to improve availability more efficiently.

A super-peer network [6, 15] is a P2P network that all supernodes (or called super-peers) are connected as a pure P2P network and each super-peer is connected to a set of clients. Our double-layer DHT can be seen as a special super-peer network. As we know, none paper in the literature has given a quantitatively study on the availability of super-peer networks. In addition, we give a scalable way to construct a self-tuning super-peer network to strike a good balance between suitable load and high availability.

Brown etc [2] and Dusseau etc [4] also observed the limitations of the traditional availability definitions. Both of them argued that availability should be examined as a function of the system's quality of service over time and represented as graphs. For easy to quantify and compare, [2] discusses the issues of representing availability as numerical summary statistics (mean, standard deviation, etc.). But none of these papers consider the needs of different kinds of applications. In contrast, we define and analyze application-specific availability which can apply for various kinds of applications.

6 Conclusion and Future Work

Application-specific availability is defined and analyzed in this paper to solve the limitation of the traditional availability definitions and to meet the need of different kinds of applications. We observed that, when a P2P storage system is comprised of hosts with relatively poor and highly variable availability, neither replication nor erasure coding behaves well in making the system highly available. We design a double-layer DHT to prevent storing data on hosts with too poor availability. By peer-selection and redundancy, we achieve high availability more efficiently than pure redundancy.

To improve availability, we exclude a portion of poorly available hosts from storing any data for simplicity. A better choice may be to allow the poorly available hosts to store a small amount of data. What is the “best” way to improve availability? This is left as future work.

References

1. R. Bhagwan, S. Savage and G.M. Voelker. Understanding Availability. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003
2. A. Brown and D.A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. In Proceedings of the 2000 USENIX Annual Technical Conference, San Diego, CA, June 2000.
3. F. Dabek, M.F. Kaashoek, D. Karger, etc. Wide-area cooperative storage with CFS. In Proc. ACM SOSP'01, Banff, Canada, Oct. 2001
4. R.A. Dusseau. Performance Availability for Networks of Workstations. Ph.D. Dissertation, U.C. Berkeley. December, 1999.
5. J. Kubiatowicz, D. Bindel, and Y. Chen, etc. OceanStore: An Architecture for Global-Scale Persistent Storage. In Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), 2000
6. KaZaA website. <http://www.kazaa.com>
7. M. Luby, M. Mitzenmacher, M. Shokrollahi, etc. Analysis of low density codes and improved designs using irregular graphs. In Proc. of ACM STOC, May 1998.
8. Overnet website. <http://www.overnet.com>
9. J. Plank. A tutorial on reed-solomon coding for fault tolerance in RAID-like systems. Software Practice and Experience, 27(9):995–1012, Sept. 1997.
10. S. Ratnasamy, et al. A Scalable Content-Addressable Network. In ACM SIGCOMM. 2001. San Diego, CA, USA.
11. A. Rowstron, and P. Druschel. Pastry: Scalable, distributed object location and routing for largescale peer-to-peer systems. in IFIP/ACM Middleware. 2001. Heidelberg, Germany.
12. A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In Proceedings of ACM SOSP'01, 2001.
13. S. Saroiu, P. K. Gummadi and S.D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In Proceedings of the Multimedia Computing and Networking (MMCN), San Jose, January, 2002
14. I. Stoica, et al. Chord: A scalable peer-to-peer lookup service for Internet applications. In ACM SIGCOMM. 2001. San Diego, CA, USA.
15. B. Yang and H.G. Molina. Designing a Super-Peer Network. ICDE'03, 2003.

16. Z. Zhang, S. Shi and J. Zhu. SOMO: self-organized metadata overlay for resource management in P2P DHT. In Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), 2003.
17. B. Y. Zhao, J.D. Kubiatowicz, and A.D. Josep. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Tech. Rep. UCB/CSD-01-1141, UC Berkeley, EECS, 2001.

Research and Implementation of Dynamic Web Services Composition

Haiyan Sun, Xiaodong Wang, Bin Zhou, and Peng Zou

School of Computer Science,
National University of Defense technology
shyfirst@sina.com

Abstract. In order to survive the massive competition created by the new online economy, traditional businesses are under the pressure to take advantage of the information revolution brought about by the Internet and the Web technology. Organizations of all sizes are moving their main businesses on the Web for more automation, efficient business processes, and global visibility. Web Services has gained a considerable momentum as paradigms for supporting both Business-to-Consumer (B2C) interaction and Business-to-Business (B2B) collaboration. As more Web services become available, it is important to share information between different Web services, and urgent to ally different services to provide newer, more powerful services quickly enough. Web Services composition technology is proposed to solve those problems. In this paper, we give an overall survey on Web Services composition. Key technologies in service composition and new challenges it faces are also researched. Based on those study, a dynamic Web Services Composition platform, Star Web Services Composition Platform (StarWSCoP) developed by us is introduced.

1 Introduction

In order to survive the massive competition created by the new online economy, traditional businesses are under the pressure to take advantage of the information revolution brought about by the Internet and the Web. Organizations of all sizes are moving their main businesses on the Web for more automation, efficient business processes, and global visibility. And now, Web Services has gained a considerable momentum as paradigms for supporting both Business-to-Consumer (B2C) interaction and Business-to-Business (B2B) collaboration.

Web Services are self-contained, self-describing modular applications that can be published, located, and invoked across the Web. Many businesses on the Web benefit from Web service's loose-coupled, Service-Oriented, and HTTP-based attributes.

To date, the development of B2B services is largely ad-hoc, time-consuming and often requires a considerable effort of low-level programming. There needs a way to develop B2B applications more efficiently. As more and more Web services become available, it is important to share information between different Web services, and it is urgent to ally different services to provide newer, more powerful services quick enough. Web Services composition technology is proposed to solve those problems[1].

2 Concept of Web Services Composition

Web Services composition is to compose autonomous Web services to achieve new functionality [1]. By Web Services composition, we can develop complex Web services more quickly and different Web services can ally to provide more powerful services. With more and more Web services appearing, Web Services composition is generating considerable interest in recent years in several computer science communities.

Web services include elementary services and composite services, developed through different method [2]. Elementary services are pre-existing services, whose instances' execution is entirely under the responsibility of some entity called the service provider. The provisioning of an elementary service may involve a complex business process, but in any case, its internals are hidden to the user. Composite services on the other hand, are recursively defined as aggregation of elementary and composite services, which are referred to as constituent services.

Services can be composed in two modes, static and dynamic. In a static composition, the services to be composed are chosen at design time by the designer. While in a dynamic composition, the services to be composed are chosen at run-time [3]. The composition mode, static or dynamic, depends on the type of business process being composed. If the process to be composed is of a fixed nature wherein the business partners/alliances and their service components rarely change, static composition will satisfy the needs. Microsoft's Biztalk server and Bea's WebLogic Integration are two of the most famous static service composition platforms. On the other hand, if the process has a loosely defined set of functions to perform, or it has to dynamically adapt to unpredictable changes in the environment, the static composition may be too restrictive. Dynamic composition is an important step in overcoming those problems. In dynamic composition, the Web services to be used for the process are decided at run-time by, for example, the process execution engine. Dynamic composition involves run-time searching of service registries to discover services. Stanford's Sword [1] and HP's eFlow [4] are two famous dynamic service composition platforms. Static service composition is the basis of dynamic service composition, and dynamic service composition is more complex and difficult to implement. In this paper, we mainly focus on dynamic Web Services composition.

The process of dynamic Web Services composition is showed in figure 1.

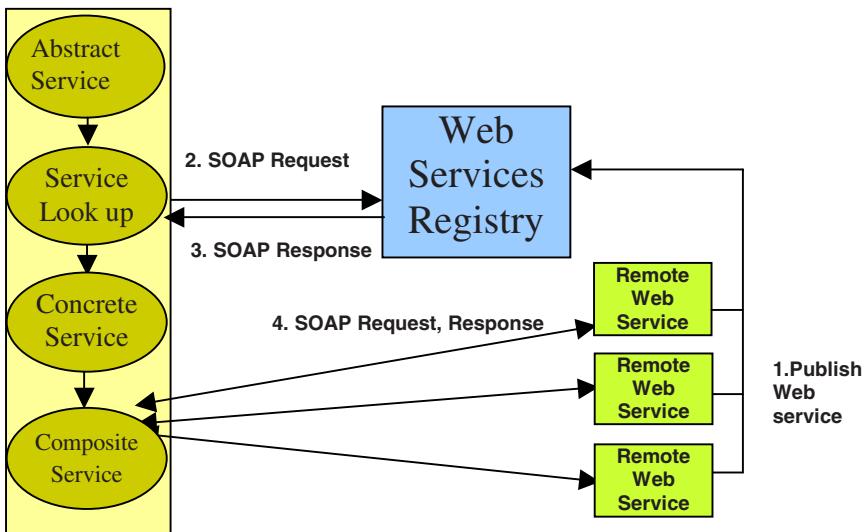
Dynamic Web Services composition is made up of three parts: remote Web service, Web Services registry and the service composition engine.

The composition procedure, as shown in the picture, consists of 4 steps as following:

1. Providers of remote Web services register their services at Web Services registry to publish those services.
2. The service composition engine sends soap requests to Web Services registry to find the proper Web services.
3. Web Services registry returns a set of proper Web services in soap response messages.
4. The service composition engine sends soap requests to the remote Web services and gets the soap responses.

Firstly, the engine decomposes the user's task into several simpler tasks, and the sequence of those tasks forms *Abstract Service*.

Service Composition Engine

**Fig. 1.** The process of dynamic Web Services Composition

After that, the engine sends *SOAP Request* to the *Web Services Registry* to find service providers who provide service for those simple tasks of the *Abstract Service* and the *Web Services Registry* responses with a *SOAP message*. Among several services suit for a certain simple task, the engine selects the most suitable one. Then the *Abstract Service* will be transformed to a *Concrete Service*.

At last, the engine schedules the *Concrete Service* to execute. The engine will send *SOAP requests* to remote Web services and get *SOAP responses* from them.

This is the basic process of dynamic service composition. In reality, it may be more complex due to for it may need to solve problems such as transaction, QoS, etc.

3 Research on Key Technologies in Dynamic Web Services Composition

Dynamic Web Services composition is a new research area in application integration. Compared with the former application integration technologies such as CORBA, EJB, it has some new characteristics:

1. The Web Services architecture is *loose-coupled*.
2. The Web Services architecture is *service-oriented*.
3. Web Services message is encoded in *SOAP style* and usually transferred on *HTTP protocol*.
4. The candidate Web services in composition may be developed *separately* and be deployed and management by *different providers*.

Due to those properties, Web Services composition is facing new challenges: How to *describe* a web service? How to *find* the proper service? How to solve the *interoperability* of services provided by different providers? How to *describe* the composite services? How to *execute* the composite services? How to *compose* web services according to user's *QoS* requirement? How to deal with the *transaction* in loose-coupled applications?

In this section, we will analyze the issues involved in dynamic Web services composition. To solve those problems, a new Dynamic Web Services Composition platform, StarWSCoP, will be proposed and introduced in the next section.

3.1 Web Services Description and Discovery

In dynamic Web Services composition, the most important problem is how to find the proper service. In Web Services architecture, the providers advertise their service descriptions in the public Web Services registries. The requesters search for services that match their requests in the registries, according to those services' descriptions.

The description of a Web service, should include syntactic (what does it look like), semantic (what does it mean) and QoS (how well does it perform) information, respectively. Quality of Service (QoS) attributes, such as timeliness, cost of service, and reliability, provide a description of the quality that can be expected from the service.

As a widely used language for describing Web services, WSDL (Web Services Description Language) is used to specify a Web service's interface [7]. It defines the syntactic information about a service such as the set of available operations, network end-points, etc. But it doesn't support QoS description of services.

Another Web Services description language, DAML-S (DARPA Agent Markup Language for Service), which is an ontology based interface description language, can describe the syntactic as well as the semantic content of a service. As a prototype expanded with some limited nonfunctional QoS related attributes of a service, the DAML-S is hard to implement.

The UDDI (Universal Description, Discovery and Integration) specifications offer users a unified and systematic way to find service providers through a centralized registry of services that is roughly equivalent to an automated online "phone directory" of Web services [9]. There are browser-accessible global UDDI Registries available for "public" access and, on the other hand, individual companies and industry groups are starting to use "private" UDDI Registries to integrate and access to their internal services. Now, only key words matching but complex semantic matching can be provided by the UDDI registry

3.2 Interoperability between Web Services

Interoperation between both internal and external Web services is critical because composite Web services are built on top of heterogeneous and independent Web services. Those Web services may be provided by *different* provider and may be developed *seperately*, and they may be not consistent in data type presentation, security policy, content, etc [5][6].

3.3 Web Services Composition Language

Service composition language is used to describe the execution sequence of concrete service, and describe the QoS, transaction, fault-tolerance requirements of the execution process.

Web Services composition is currently a hot area under research, with many service composition languages being proposed by academic and industrial research groups. IBM's Web Services Flow Language (WSFL)[10] and Microsoft's XLANG are two of the earliest languages to define standards for Web services composition. Both of them extend W3C's WSDL (Web Services Description Language). WSFL is proposed as an XML based language to describe complex service compositions. It supports both static configurations of services as well as run time searching of services in Web Services registries. Microsoft's service composition language, XLANG, extends WSDL with behavioral specifications to provide a model for composition of services. XLANG requires services to be configured statically. Business Process Execution Language for Web Services (BPEL4WS) is a recently proposed specification that represents the merging of WSFL and XLANG[11]. BPEL4WS combines the graph oriented process representation of WSFL and the structural construct based processes of XLANG into a unified standard for Web services composition.

Another service composition language is DAML-S. DAML-S supplies Web Services providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable forms. DAML-S markup of Web services will facilitate the automation of Web Services tasks including automated Web Services look up, execution, interoperation, composition and execution monitoring.

Most of those service composition languages haven't dealt with problems about *QoS, transaction and fault-tolerance*. There is still a long way for the Web Services composition language to be mature enough to describe dynamic Web Services composition.

3.4 Composite Web Services Execution

A composed Web Services can be executed via either a centralized approach or a distributed approach. The centralized approach is based on the Client/Server architecture, with a scheduler, which controls the execution of the components of the Web service. The controller (client) can reside either on the host where the composition is made or on a separate host to allow for a better load balancing. The controller/scheduler invokes a Web service, and gets its results. Based on the results and the Web Services design specification, the controller then invokes the next appropriate Web service. This is the easiest approach for executing composite Web services. HP's eFlow is such a system with a centralized execution engine [3].

The distributed approach for Web Services execution is more complex than the centralized one. In a distributed approach, the Web services involved are expected to collaborate and share the execution context to realize the distributed execution. Each of Web Services involved hosts a coordinator component that collaborates with other coordinators to realize the distributed execution. A slightly modified version of distributed execution involves coordinators that control a set of Web services. The

composite service is executed in a distributed fashion by these coordinators, but internally each coordinator implements a centralized model for executing the tasks controlled by it. SELF-SELV, developed by the University of New South Wales, is a dynamic Web Services composition platform with a distributed execution engine [2].

3.5 QoS Based Web Services Composition

QoS has been a major concern in the areas of networking, real-time applications and middleware. And in the composition of Web service, several distinct advantages will be brought by QoS technologies [12]:

- It allows the composition engine to design the composite services according to QoS metrics.
- It allows for the selection and execution of composite services based on their QoS to fulfill customer expectations.
- It makes the monitoring of composite services based on QoS possible.
- It allows for the evaluation of alternative strategies when adaptation becomes necessary.

To support QoS based dynamic Web Services composition, it need to:

- 3.1 propose a *Web Services description language* that supports QoS description.
- 3.2 build a *QoS estimation framework* of the composite Web service.
- 3.3 develop a *QoS monitor* of Web services.

3.6 Transaction in Web Services Composition

Traditionally, transactions are expected to satisfy the properties of *atomicity*, *isolation*, *consistency* and *durability*, known as ACID properties. These properties are used to define the notion of correctness of traditional transaction models and protocols. But, traditional transaction technologies cannot comply with the Web Services composition [13][14].

The traditional notion of *atomicity* means that the steps executed during a transaction are executed completely or not at all. Such strict notion of *atomicity* is not reasonable for Web Services environment. A composition engine must exploit the different transaction support provided by Web services as well as the existence of many semantically equivalent Web services. For instance, if a Web Services supports compensation, the composition must exploit it to deal with failures that may occur during its execution.

In traditional transaction mechanism, local subsystems must be monitored to identify service invocations that violate global dependencies to guarantee consistency. But such monitoring is unreasonable in a Web Services environment, where involved Web services are often developed separately and managed by different providers. Web services are strictly autonomous and there may be a huge number of services. For the above reasons, it is not feasible to guarantee *global consistency*.

Isolation is concerning with controlling the concurrent execution of transactions to provide the illusion that concurrent transactions are executed in a serial order. Generally, locking mechanisms are used to provide *isolation*. But in Web Services environment, the underlying system may not support or allow locking through Web

Services access. Moreover, *isolation* is also costly because transaction compositions may be long running, and some Web services may have a lot of concurrent clients.

In the process of Web Services composition, the only property that must be fully supported in transaction compositions is *durability*. The traditional notion of *durability* means that when a transaction completes its execution, all of its actions are made persistent, even in presence of system failures. In the same way, *durability* must be ensured in Web Services environments.

4 StarWSCoP – A Dynamic Web Services Composition Platform

Based on the research on key technologies of dynamic Web Services composition, we develop a dynamic Web Services composition platform, StarWSCoP (Star Web Services Composition Platform), shown in figure 2. StarWSCoP is developed on the Web Services platform, StarWS, which is implemented by our research group. StarWSCoP includes the following parts:

- *Intelligent System*. Intelligent System decomposes the user's requirement into simple tasks, and sequences those tasks into Abstract Service.
- *Service Registry*. Web services registry at the Service Registry and Service Discovery Engine looks up those services registered in the Service Registry.
- *Service Discovery Engine*. Service Discovery Engine looks up services in the Registry and selects the proper one that satisfies the requester's requirements.
- *Composition Engine*. Composition Engine schedules the composite Web services to be executed in order, keeps the composite Web service's trace information in Service Execution Information Library and deals with the events sent by the Event Monitors.
- *Wrapper*. Wrapper is used to conceal the difference of Web services and to achieve interoperability.
- *Service Execution Information Library*. It is used to store the trace information of composite Web Services execution.
- *QoS Estimation*. It is used to estimate the real-time QoS metrics of current composite Web service.
- *Event Monitors*. Event Monitors are used to monitor various events and notify the Composition Engine.

We extend WSDL to support QoS metrics, such as *Cost*, *Time* and *Reliability*. For example, the operation *getQuote* can be described as following:

```

<portType name="GetQuote">
    <operation name="getQuote" Cost="20" Time="50"
              Reliability="90%">
        <input message="tns:GetQuoteRequest"/>
        <output message="tns:GetQuoteResponse"/>
    </operation>
</portType>
```

An ontology-based semantic layer is added on UDDI. Based on ontology, we can achieve semantic match for Web service.

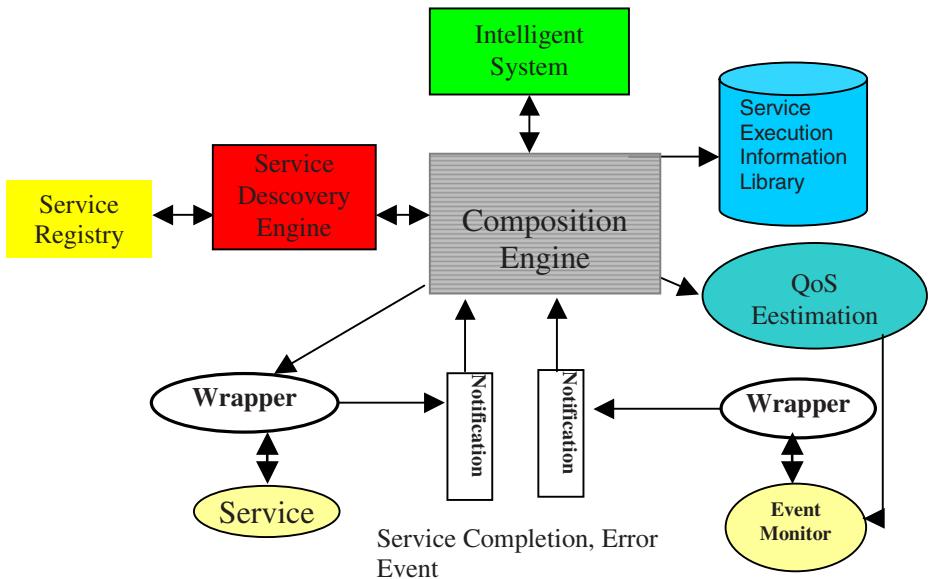


Fig. 2. Architecture of Dynamic Web Services Composition Platform StarWSCoP

The UDDI registry returns a set of services for each request. It is important for the requester to select the most proper one from the set. We proposed a filter based on QoS to satisfy the user's QoS requirements.

We develop a wrapper for each Web Services to conceal the difference of different Web services. The wrapper acts as a proxy of the Web service. Others communicate with the wrapper instead of the Web service. The wrapper will initiate, start, freeze and continue the Web Services according the requests sent by requester.

There are several managers, as listed in the following, in the wrapper to solve different problems:

- *Communication manager.* Web services may use different transport protocols, such as http, SMTP, etc. The *communication manager* is used to translate messages between those different protocols.
- *Security manager.* Web services may need to cross firewall and security systems in order to access partners' services. The purpose of this module is to handle security issues including authentication, access auditing and authorization, communication integrity, etc.
- *Content manager.* It is likely that Web services use disparate information formats and semantics. For example, if an internal application uses xCBL to represent business documents, and this application needs to interact with an external service, which expects documents in cXML, the conversion between these two formats should be handled by the *content manager*.
- *Conversation manager.* *Conversation manager* is concerned with the conversational interactions among Web services. For instance, a given service may require a login procedure to be performed prior to any access to the service's functionalities, while another service with similar capabilities, may provide access to some functionalities of the service without password

verification, and only require a full login procedure for accessing other functionalities.

- *QoS Monitor.* QoS Monitor is used to monitor the QoS metrics of Web services.

In StarWSCoP, we adopt BPEL4WS as Web service composition language. BPEL4WS is still in the elementray phase. In the current version of StarWSCoP, we extend the BPEL4WS to surport QoS. And we will do some work to suport transaction, fault-tolerance in future.

To support QoS in StarWSCoP, we have done the following efforts:

1. We extend WSDL to support QoS description.
2. We develop a *real-time QoS estimation* of the composite Web service. The Real-time QoS estimation is fired when the execution of each component service finished. The QoS Estimation estimates the real-time QoS of the composite Web and sends a QoS mismatch event to the engine when Real-time QoS estimation doesn't match user's requirements. The ability of Real-time QoS estimation is very important to design composite Web Services and to alter the composite Web Services during the execution.
3. The engine alters the composite Web Services definition when the real-time QoS estimation doesn't fit the user's QoS requirement. In the process of composite Web Services alternation, the consistency of the alternation must be considered.

In StarWSCoP, QoS description of Web services is realized by an extended WSDL. A semantic layer is added on UDDI to support semantic look up of Web services. Interoperability is also achieved through the implementation of a wrapper for each Web service, which can deal with the security, transport and data type mismatch. To satisfy the users' requirements more efficiently, a real-time QoS estimation and dynamic event manipulation are developed in our approach.

5 Conclusion

Web services composition is a new research area that combines Web services technology with process composition. In this paper, we give an overall survey of the Web Services composition technology, analyze the key technologies in Web Services composition and give out the new challenges of Web Services composition. We also introduce StarWSCoP, a dynamic Web Services composition platform, in which several problems on dynamic Web Services composition can be solved.

In the current version of StarWSCoP, we only support part of durability of the transaction properties. We will realize transaction in the later version of StarWSCoP.

References

1. Shankar R. Ponnekanti and Armando Fox. SWORD: A Developer Toolkit for Building Composite Web Services. Computer Science Dept Stanford University Stanford, CA 94305
2. Benatallah, B., Dumas, M., Marie-Christine Fauvet, Hye-Yong Paik. Self-Coordinated and Self-Traced Composite Services with Dynamic Provider Selection. Queensland University of Technology.

3. Senthilanand Chandrasekaran, John A. Miller, Gregory S.Silver, Budak Arpinar, Amit P. Sheth. Composition, Performance Analysis and Simulation of Web services
4. Casati, F., Ilnicki, S., Jin, L., Krishnamoorthy, V., and Shan, M. (2000) 'Adaptive and Dynamic Service Composition in eFlow', in *Proceedings of the International Conference on Advanced Information Systems Engineering*, Stockholm, Sweden
5. Benatallah, B., Dumas, M., Fauvet, M., and Rabhi, F. (2002) 'Towards Patterns of Web Services Composition', in *Patterns and Skeletons for Parallel and Distributed Computing*, Springer Verlag, UK.
6. B. BENATALLAH, M. DUMAS, M.-C. FAUVET, F.A. RABHI, QUAN Z. SHENG. Overview of Some Patterns for Architecting and Managing Composite Web Services. ACM SIGecom Exchanges, Vol. 3, No. 3, August 2002, Pages 9–16
7. Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001) 'Web Services Description Language (WSDL) 1.1', <http://www.w3.org/TR/wsdl>
8. DAML-S: Semantic Markup for Web Services. The DAML Services Coalition
9. UDDI Technical White Paper
10. Dr. Frank Leymann. Web Services Flow Language (WSFL 1.0). IBM Software Group.
11. Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S. (2002) 'Business Process Execution Language for Web Services'.
<http://msdn.microsoft.com/webservices/default.asp>
12. Cardoso, J., Miller, J., Sheth, A., and Arnold, J. (2002) 'Modeling Quality of Service for Workflows and Web Services Processes', in *The VLDB Journal*.
13. Thomas Mikalsen, Stefan Tai, Isabelle Rouvellou. Transactional Attitudes: Reliable Composition of Autonomous Web Services. IBM T.J. Watson Research Center, Hawthorne, New York, USA.
14. Paulo de Figueiredo Pires. WEBTRANSACT: A FRAMEWORK FOR SPECIFYING AND COORDINATING RELIABLE WEB SERVICES COMPOSITIONS. FEDERAL UNIVERSITY OF RIO DE JANEIRO

Call Admission Control for Multimedia CDMA Networks under Imperfect Power Control

Xiaoming Bo and Zujue Chen

Department of Communication Engineering
Jiangsu University,
Zhenjiang, China
xiaoming_bo@163.com

Abstract. Call admission control is one of the most important techniques in mobile networks. A call admission control scheme for code division multiple access (CDMA) networks under imperfect power control is proposed in this paper. The admission region is derived based on the optimal power allocation scheme which takes into account the bursty nature of multimedia traffic. The complete sharing call admission scheme is then described and evaluated using Markov analysis. A K -dimensional birth-death process is used to model the system, from which the balance equations and equilibrium state probabilities are obtained. Call blocking probabilities and resource utilization of user classes are also derived as performance measures. Numerical examples show that the proposed scheme considerably outperforms the conventional scheme in the sense of larger admission region and better system performance. Additionally, several valuable conclusions are also made by analyzing the numerical results.

1 Introduction

The next generation wireless networks are expected to support multimedia services with different traffic characteristics and quality of service (QoS) requirements based on the code division multiple access (CDMA) air interface. Radio resource management (RRM) plays a vital role in CDMA networks to efficiently utilize the limited radio resources while guaranteeing the required QoS performance for mobile users. Among the RRM techniques, call admission control (CAC) has become the focus of many studies on CDMA networks in recent years [1-5].

The call admission control is to make a decision about whether a user should be admitted into the system according to the current traffic load and the QoS requirements of the user. The objective of CAC is to maximize the utilization of system resource as long as the QoS requirements of all users are guaranteed. The CAC problem was considered for voice and data integrated CDMA networks in [1] and for homogeneous and heterogeneous services in [2]. An optimal admission policy is presented in [3] with the objective of maximizing the overall traffic carried by the system. And in [4], the authors proposed a CAC scheme based on signal-to-interference (SIR) measurement for multimedia traffic. But all these works are limited to the perfect power control case. However, imperfect power control is a more realistic environment in the practical CDMA system. The key problem under

imperfect power control in multimedia CDMA networks is to guarantee the differentiated outage probabilities of different traffic classes resulted from the uncertainty of received powers. Therefore, in addition to the traditional parameters such as rate requirement, SIR threshold, outage probability constraint should also be included in the QoS characteristics under imperfect power control environment. A CAC scheme using differentiated outage probabilities in multimedia CDMA networks is proposed in [5]. This scheme also considers user activity factors, but there is no theoretical model and analysis of the system performance.

In this paper, we present a CAC scheme to guarantee different users' QoS requirements while enhance the efficiency of resource utilization in multimedia CDMA networks under imperfect power control. The scheme utilizes the bursty nature of multimedia traffic to achieve better system performance. The rest of this paper is organized as follows. We describe the system model in Section 2. Then we define the admission region and propose the power allocation scheme in Section 3. In Section 4, we present the admission decision and evaluate the system performance by Markov analysis. Numerical results are shown in section 5, followed by the conclusions and future work in section 6.

2 Model Description

We consider the uplink of a variable spreading gain (VSG)-CDMA cellular system. All traffic classes with different data rates are spread over the whole bandwidth with different spreading factors. Due to the bursty nature of multimedia traffic, not all users are on their active periods at the same time. The total received power (or the interference) at the base station (BS) only depends on the number of active users, which is a random variable [5]. Suppose users in the system are divided into K classes. We characterize the class i traffic by two state Markov model with ON and OFF states: its ON and OFF periods follow exponential distributions with mean $1/a_i$ and $1/b_i$, respectively. Then we can define the activity factor of class i traffic as $\nu_i = b_i/(a_i + b_i)$. All the users in the same class have the same traffic parameters and QoS requirements. The QoS requirements of class i user is represented by a triple $(R_i, \gamma_i, \delta_i)$ [5], where R_i is the user's bit rate requirement during its ON state, γ_i is the specified SIR threshold and δ_i is the maximum acceptable outage probability. Considering the definition of outage, γ_i and δ_i should satisfy $P_r\{(E_b/I_0)_i < \gamma_i\} \leq \delta_i$, where E_b/I_0 is the bit-energy-to-interference density ratio.

When the power control is imperfect, the actual received power is lognormally distributed. The signal model used in this paper is similar to the imperfect power control case in [6]. According to [6], suppose the *target* received power for the user in class i is S_i , then the *actual* received power for the user is $S_i e^{\beta X_i}$ due to imperfect power control, where $\beta = \ln(10)/10$, and X_i is a normally distributed random variable (r.v.). If we assume that all the users use the same power control algorithm, the statistics of the channel transmission conditions are the same for all the users, and each user transmits independently, then we can consider that all X_i 's are independent and identically distributed (i.i.d.). The mean and variance of X_i are $E[X_i] = 0$ and $Var[X_i] = \sigma_x^2$, respectively, for all users. The standard

deviation, σ_x , represents the power control errors, and its unit is dB. It should be pointed out that X_i s in the same user class are not necessarily equal to each other, but this fact does not affect the correctness of our analysis because $E[X_i] = 0$ holds for all users.

3 Power Allocation and Admission Region

3.1 Analysis of Active User States

Since only active users generate interference at the BS, we should know the number of active users of each traffic class. Consider a network user state (k_1, \dots, k_K) , where k_i denotes the total number of users in class i . We introduce the active user state $\theta_k = (n_1, \dots, n_K)$, which explicitly describes the number of *active* users in all classes. Let $Q(\mathbf{k})$ denote the state space of all feasible active user states under the network user state (k_1, \dots, k_K) . Then

$$Q(\mathbf{k}) = \{\theta_k \mid 0 \leq n_i \leq k_i, \text{ for } 1 \leq i \leq K\}$$

Assume the state (ON or OFF) of a user is independent of the state of other class users, i.e. n_i ($i = 1, \dots, K$) are independent random variables, the active user state distribution is given by [5]:

$$\Pr\{n_1, \dots, n_K \mid k_1, \dots, k_K\} = \Pr\{n_1 \mid k_1\} \dots \Pr\{n_K \mid k_K\}$$

Obviously, the instantaneous number of class i users in the ON state is a binomial distributed random variable, so the probability that at arbitrary time there are n_i active users of traffic class i under the network user state (k_1, \dots, k_K) is given by

$$\Pr\{n_i \mid k_i\} = C_{k_i}^{n_i} v_i^{n_i} (1 - v_i)^{k_i - n_i}$$

The distribution of active user state is then given by

$$\Pr\{n_1, \dots, n_K \mid k_1, \dots, k_K\} = \prod_{i=1}^K C_{k_i}^{n_i} v_i^{n_i} (1 - v_i)^{k_i - n_i} \quad (1)$$

Thus for a given network user state (k_1, \dots, k_K) , the outage probability for traffic class i is derived as

$$P_{out_i} = \Pr\{(E_b / I_0)_i < \gamma_i\} = \sum_{\theta_k \in Q(\mathbf{k})} \Pr\{(E_b / I_0)_i < \gamma_i \mid \theta_k\} \times \Pr\{\theta_k\} \quad (2)$$

where $\Pr\{(E_b / I_0)_i < \gamma_i \mid \theta_k\}$ is the conditional outage probability of traffic class i under the condition of active user state $\theta_k = (n_1, \dots, n_K)$.

3.2 Analysis of Outage Probabilities

A power distribution scheme for multicode (MC)-CDMA systems under imperfect power control was proposed in [6]. However, it has been proved that VSG-CDMA systems are more stable and power efficient than MC-CDMA systems [7]. So we consider the VSG-CDMA systems in this paper. Given active user state (n_1, \dots, n_K) , at the BS receiver output, the E_b/I_0 for class i user can be expressed as

$$\left(\frac{E_b}{I_0} \right)_i = \frac{W}{R_i} \frac{S_i e^{\beta X_i}}{(1+f) \sum_{j=1}^K n_j S_j e^{\beta X_j} - S_i e^{\beta X_i} + N_0 W}$$

where W is the spread spectrum bandwidth; N_0 is the power spectral density (PSD) of the background additive white Gaussian noise (AWGN) and f is relative other-cell interference factor. Outage occurs when the condition $(E_b/I_0)_i < \gamma_i$ holds, i.e.,

$$\frac{W}{R_i} \frac{S_i e^{\beta X_i}}{(1+f) \sum_{j=1}^K n_j S_j e^{\beta X_j} - S_i e^{\beta X_i} + N_0 W} < \gamma_i$$

After define the spreading gain G_i as $G_i = W/R_i$, we have

$$\left[\frac{G_i + \gamma_i}{(1+f)\gamma_i} \right] S_i e^{\beta X_i} - \sum_{j=1}^K n_j S_j e^{\beta X_j} < \frac{N_0 W}{1+f} \quad (3)$$

According to the signal model described in section 2, the left-hand side of (3) is the sum of many independent lognormal random variables. Therefore, the sum also approximately follows a lognormal distribution [8]. Let

$$e^{\Psi_i} = \left[\frac{G_i + \gamma_i}{(1+f)\gamma_i} \right] S_i e^{\beta X_i} - \sum_{j=1}^K n_j S_j e^{\beta X_j}$$

Then, Ψ_i is a normally distributed r.v. with mean M_{Ψ_i} and variance $D_{\Psi_i}^2$ derived as

$$D_{\Psi_i}^2 = \ln \left\{ 1 + \frac{H_i (e^{\sigma_x^2 \beta^2} - 1)}{F_i^2} \right\}$$

$$M_{\Psi_i} = -\frac{1}{2} D_{\Psi_i}^2 + \ln F_i + \frac{1}{2} \beta^2 \sigma_x^2$$

where F_i and H_i are given as

$$F_i = \frac{G_i + \gamma_i}{(1+f)\gamma_i} S_i - \sum_{j=1}^K n_j S_j$$

$$H_i = \frac{(G_i - f\gamma_i)^2 - (1+f)^2\gamma_i^2}{(1+f)^2\gamma_i^2} S_i^2 + \sum_{j=1}^K n_j S_j^2$$

Then, condition (3) can be further rewritten as

$$e^{\Psi_i} < N_0 W / (1+f)$$

Now, we can express the conditional outage probability of class i user under the condition of active user state $\theta_k = (n_1, \dots, n_K)$ as

$$\Pr \left\{ \left(E_b / I_0 \right)_i < \gamma_i \mid \theta_k \right\} = \Pr \left\{ \Psi_i < \ln \left(\frac{N_0 W}{1+f} \right) \right\} = Q \left[\frac{M_{\Psi_i} - \ln [N_0 W / (1+f)]}{\sqrt{D_{\Psi_i}^2}} \right] \quad (4)$$

$$\text{where } Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt.$$

Combining (1)(2) and (4), for a given network user state (k_1, \dots, k_K) , the comprehensive outage probability of class i user is given by

$$p_{out_i} = \sum_{\theta_k \in Q(\mathbf{k})} \left\{ Q \left[\frac{M_{\Psi_i} - \ln [N_0 W / (1+f)]}{\sqrt{D_{\Psi_i}^2}} \right] \prod_{i=1}^K C_{k_i}^{n_i} v_i^{n_i} (1-v_i)^{k_i-n_i} \right\} \quad (i = 1, \dots, K) \quad (5)$$

3.3 Admission Region and Power Allocation

For a CDMA system supporting K user classes, an admissible state \mathbf{k} is a combination of the numbers of users in each class that can coexist in the system with guaranteed QoS under a specified power allocation scheme Γ , i.e.

$$\mathbf{k} = (k_1, \dots, k_K) \quad (6)$$

$$\text{s.t. } p_{out_i} \leq \delta_i \text{ & } k_i \geq 0 \text{ for } i = 1, \dots, K$$

$$\text{when } (S_1, \dots, S_K) = \Gamma(k_1, \dots, k_K)$$

The admission region of the system, \mathbf{K} , is the set of all possible \mathbf{k} (i.e. $\mathbf{K}=\{\mathbf{k}\}$) and the capacity of the system is defined as the boundary of \mathbf{K} .

It is indicated by (5) that the outage probability of class i user is closely related to the power allocation schemes. Thus the admissible region \mathbf{K} is also dependent on the power allocation schemes. By saying power allocation scheme, we mean to determine the average received user power vector (S_1, \dots, S_K) for a given network user state (k_1, \dots, k_K) [9]. The objective of our power allocation scheme is enhancing the efficiency of resource utilization and maximizing the admission region. CDMA network is an interference-limited system and any reduction of received interference will lead to an increase of capacity. In this context, the objective of maximizing the admission region is consistent with the minimization of received total powers.

Since the class i user's outage probability in (5) is a mono-decreasing function of S_i and a mono-increasing function of S_j ($j = 1, \dots, K, j \neq i$), we can prove that all outage probability constraints in (6) should be met with equality at the optimal power allocation scheme which minimizes the received total powers and maximizes the system admission region (the proof is not given due to the page limits). Thus the optimal power allocation scheme must satisfy the following equation set:

$$\sum_{\theta_k \in Q(\mathbf{k})} \left\{ Q \left[\frac{M_{\Psi_i} - \ln [N_0 W / (1+f)]}{\sqrt{D_{\Psi_i}^2}} \right] \prod_{i=1}^K C_{k_i}^{n_i} v_i^{n_i} (1-v_i)^{k_i-n_i} \right\} = \delta_i \quad (7)$$

By solving the above equation set, we can obtain the power allocation vector (S_1, \dots, S_K) . Because the mobile station is power-limited, the allocated power value for class i user, S_i , cannot exceed the predetermined user power constraint P_{\max} . Under this power allocation scheme, the admission region can finally be expressed as

$$\mathbf{K} = \{(k_1, \dots, k_K) | 0 < S_i \leq P_{\max} \text{ & } k_i \geq 0, \quad i = 1, \dots, K\} \quad (8)$$

Eq.(7) is a transcendental equation set and its accurate solution is hard to derive. So we will solve it approximatively by numerical methods in section 5.

4 Admission Control and Performance Evaluation

In order to satisfy the QoS requirements for all users, it is necessary to admit or block incoming users as a function of the current network user state in the BS. The CAC scheme used in this paper is called the complete sharing scheme. Under this scheme, an incoming user is always offered access to the network if and only if the BS has sufficient capacity to simultaneously guarantee the QoS requirements of all present users and the new user at the time of request. Assume the current network user state is $\mathbf{k} = (k_1, \dots, k_i, \dots, k_K)$. An incoming user of class i sends an admission request to BS, together with its traffic parameters and QoS requirements. The BS then updates the network user state to $(k_1, \dots, k_i+1, \dots, k_K)$ and checks if it can support the new network user state. If $(k_1, \dots, k_i+1, \dots, k_K)$ is within the system's admissible region, the user is admitted and the new network user state is maintained. Otherwise, the user is blocked and the previous network user state is maintained. Let $A(\mathbf{k}, i)$ denote the admission probability of the user belongs to the i th class when the system is in state \mathbf{k} , the admission control scheme can be expressed as

$$A(\mathbf{k}, i) = \begin{cases} 1, & \text{if } \mathbf{k} + \mathbf{e}_i \in \mathbf{K} \\ 0, & \text{if } \mathbf{k} + \mathbf{e}_i \notin \mathbf{K} \end{cases}$$

where \mathbf{K} is the admission region of the system and \mathbf{e}_i is a vector with the i th component equal to 1 and all the other ($K-1$) components equal to zero.

4.1. Assumptions and Approximation Model

We assume that the overall system is homogeneous in statistical equilibrium. In a homogeneous system, a cell is statistically same as any other cell. Thus, for each class, the mean handoff arrival rate to a cell should be equal to the mean handoff departure rate from the cell. Under this observation, we can decouple an arbitrary cell from the rest of the system and evaluate the system performance by only analyzing this cell [4]. So we focus on an arbitrary cell in the performance evaluation.

The following assumptions regarding memoryless properties allow us to model the system as multidimensional continuous time Markov chain.

New calls of class i arrive at the cell according to Poisson process with rate λ_i^n .

Handoff calls of class i arrive at the cell according to Poisson process with rate λ_i^h .

The channel holding time of a class i new call is exponentially distributed with mean $1/\mu_i^n$.

The channel holding time of a class i handoff call is also exponentially distributed with mean $1/\mu_i^h$.

It has been reported that the channel holding times for new calls and handoff calls are distinct with different average values. So, contrary to the common assumption that both new calls and handoff calls are distributed with the same distribution, we assume the channel holding time for new calls and handoff calls have different averages (i.e. $1/\mu_i^n$ is not equal to $1/\mu_i^h$). But this assumption may lead to an intensive theoretical mode in performance evaluation. An efficient approximation model is proposed in [10] for single-class service in mobile wireless networks. This model is based on the observation that the performance for each traffic class calls mainly depends on the traffic intensity. We generalize this model to handle the multi-class services networks and apply it in our analysis.

Let ρ_i^n and ρ_i^h denote the traffic intensity of i th class new calls and handoff calls respectively, then we have $\rho_i^n = \lambda_i^n / \mu_i^n$ and $\rho_i^h = \lambda_i^h / \mu_i^h$. We use the following approximate model: the new call arrival process of class i is Poisson with arrival rate ρ_i^n and with service rate (corresponding channel holding time for new calls) 1 (the unity). The handoff call arrival process is also Poisson with arrival rate ρ_i^h and service rate 1. Note that the proposed CAC scheme can be partly revised to give a handoff call higher priority than a new call, but we do not consider the priority in this paper.

4.2 Balance Equations and Performance Measures

The system supporting K traffic classes can be modeled as a K -dimensional birth-death process. The state space of the Markov chain is admission region \mathbf{K} , which we have defined in (8). Let $q(\mathbf{k}, \mathbf{k}')$ denote the state transition rate from state \mathbf{k} to state \mathbf{k}' . Under the above approximation model, we have:

$$\begin{aligned} q(\mathbf{k} - \mathbf{e}_i; \mathbf{k}) &= \rho_i^n + \rho_i^h, \quad \text{when } \mathbf{k} \in \mathbf{K} \& k_i > 0 \\ q(\mathbf{k}; \mathbf{k} - \mathbf{e}_i) &= k_i, \quad \quad \quad \text{when } \mathbf{k} \in \mathbf{K} \& k_i > 0 \end{aligned}$$

Let $p(\mathbf{k})$ denote the equilibrium probability for the admissible state $\mathbf{k} = (k_1, \dots, k_i, \dots, k_K)$. Then, for all $\mathbf{k} \in \mathbf{K}$, the global balance equations are:

$$\begin{aligned} p(\mathbf{k}) \left\{ \sum_{i=1}^K (\rho_i^n + \rho_i^h) I_{(\mathbf{k} + \mathbf{e}_i) \in \mathbf{K}} + \sum_{i=1}^K k_i I_{(\mathbf{k} - \mathbf{e}_i) \in \mathbf{K}} \right\} = \\ \sum_{i=1}^K (\rho_i^n + \rho_i^h) p(\mathbf{k} - \mathbf{e}_i) I_{(\mathbf{k} - \mathbf{e}_i) \in \mathbf{K}} + \sum_{i=1}^K (k_i + 1) p(\mathbf{k} + \mathbf{e}_i) I_{(\mathbf{k} + \mathbf{e}_i) \in \mathbf{K}} \end{aligned} \quad (9)$$

In (9), I_C is an indicator function of which value is one if the condition C is true; otherwise, the value is zero. From the Jackson's theorem [11], we know that the product-form expression holds for the equilibrium state probability distribution and the detailed balance equations are satisfied for the system. After applying the constraint $\sum_{\mathbf{k} \in \mathbf{K}} p(\mathbf{k}) = 1$ to the set of balance equations, the equilibrium probability for state \mathbf{k} can be obtained as

$$p(\mathbf{k}) = p(\mathbf{0}) \prod_{i=1}^K \frac{\rho_i^{k_i}}{k_i!}$$

and

$$p(\mathbf{0}) = \left\{ \sum_{\mathbf{k} \in \mathbf{K}} \prod_{i=1}^K \frac{\rho_i^{k_i}}{k_i!} \right\}^{-1}$$

where ρ_i is the traffic intensity of class i call and is given by $\rho_i = \rho_i^n + \rho_i^h$ for $i=1, \dots, K$.

The performance measures which we focus on are the call blocking probabilities (CBP) and the resource utilization for user classes. Similar to [12], the call blocking probability for the i th class user is given by

$$p_i^b = \sum_{\mathbf{k} \in \mathbf{K}} \{1 - A(\mathbf{k}, i)\} \cdot p(\mathbf{k}) = \sum_{\mathbf{k} \in \mathbf{g}_i} p(\mathbf{k}) = p(\mathbf{0}) \sum_{\mathbf{k} \in \mathbf{g}_i} \prod_{i=1}^K \frac{\rho_i^{k_i}}{k_i!}$$

where \mathbf{g}_i is the set of the states that can not allow any additional i th class user admitted in the cell after serving all the existing users, and is given by

$$\mathbf{g}_i = \{\mathbf{k} \mid \mathbf{k} \in \mathbf{K}, \mathbf{k} + \mathbf{e}_i \notin \mathbf{K}\}$$

The resource utilization for the i th class user is given by

$$C_i = \sum_{\mathbf{k} \in \mathbf{K}} k_i p(\mathbf{k})$$

5 Numerical Examples and Discussions

Consider the uplink of a VSG-CDMA system. The following parameters are used in the numerical calculation. The spread spectrum bandwidth W is 3.84 MHz, and the AWGN PSD N_0 is 10^{-8} W/Hz. The relative other-cell interference factor f is 0.5. Two traffic classes are considered, whose QoS requirements and traffic parameters are shown in Tab.1. For each class, the arrivals of calls follow the Poisson process and the channel holding time follows exponential distribution.

Table 1. Traffic classes parameters

	$R_i(kb/s)$	γ_i	δ_i	v_i
Class1	16	4	10^{-2}	0.5
Class2	64	6	10^{-1}	0.4

The CAC scheme in [9], which did not consider the bursty nature of multimedia traffic (i.e. activity factors of all traffic classes are equal to 1), is referred to as the conventional scheme in this paper. Admission region comparison between the proposed scheme (scheme 1) and the conventional scheme (scheme2) is plotted in Fig.1. It shows that much larger admission region is achieved by the proposed scheme. A larger admission region means that the system can admit more users, so the system performance under the proposed scheme is better than the conventional scheme. However, because the received signals under imperfect power control are random variables instead of constants as under perfect power control, the capacity gain does not increase as large as that under the perfect power control environment.

CBP for each traffic class under the two CAC schemes are shown in Fig. 2. It can be seen that, under each scheme, CBP for each traffic class increases as the traffic intensity increases. The figure also shows that class2 traffic has greater impact on the CBP performance than class1 traffic does. CBP for each traffic class increases more rapidly as the class2 traffic intensity increases than that as the class1 traffic intensity increases. This is because a class1 user requires less system resource, including both transmission rate and power, than a class2 user does. We also see that due to the smaller admission region, the fluctuation range of CBP curves under the conventional scheme is less remarkable than that under the proposed scheme.

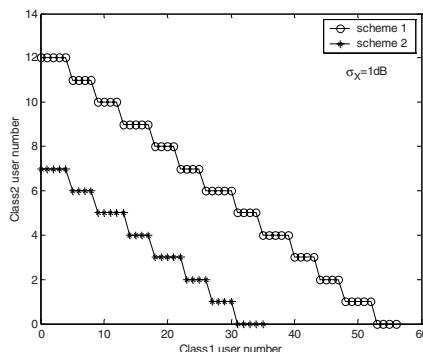
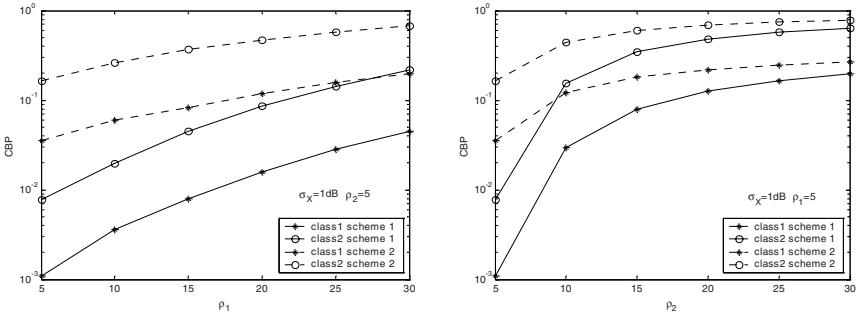
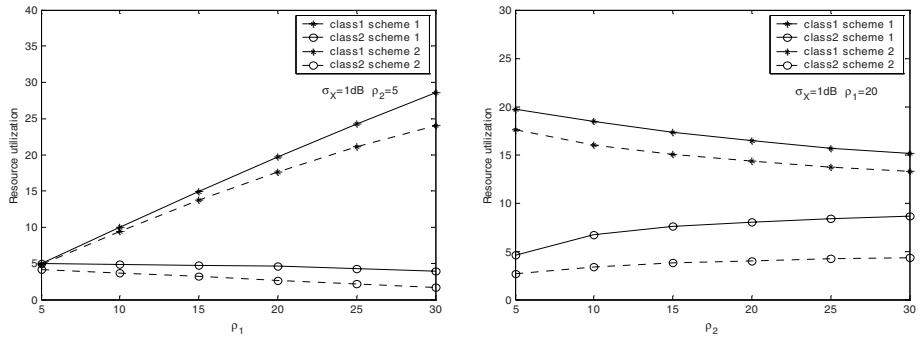


Fig. 1. Admission region comparison between two schemes

**Fig. 2.** CBP as function of traffic intensity**Fig. 3.** Resource utilization as function of traffic intensity

The resource utilization for each traffic class under the two CAC schemes are shown in Fig. 3. We can see that as the class1 traffic intensity increases, resource utilization for class1 user increases significantly, while resource utilization for class2 user decrease slightly. As the class2 traffic intensity increases, class2 traffic resource utilization increases significantly, while class1 traffic resource utilization decreases relatively slightly. These figures also show that the resource utilization improvement is still significant when the traffic load is higher, which indicates that the proposed scheme is efficient for high traffic load systems.

From the performance curves, we can conclude that the system performance under the proposed scheme is better than the conventional scheme in all cases, which verify the correctness of our analysis. Because the system resources are completely shared by all the users in the proposed scheme, traffic change in one traffic class affects the performance of all the traffic classes in the system. Additionally, the traffic classes which require fewer resources always receive better performance than those traffic classes which required more resources.

6 Conclusions and Future Work

We have developed and evaluated a call admission control scheme to guarantee differentiated users' outage requirements while sufficiently utilize system resource in

multimedia CDMA networks under imperfect power control. Since the scheme is based on the imperfect power control model, it is more appropriate to practical CDMA systems. The system performance is improved because the power allocation scheme is aiming at maximizing the admission region. The statistical feature of multimedia traffic utilized in the scheme brings capacity gains for the system and enhances the efficiency of resource utilization. Numerical results have verified the correctness of our analysis and showed that our scheme achieves a better system performance. In the future work, the priority for handoff calls and the priority mechanism among call classes should be adopted in the scheme. In addition, as future mobile networks will be based on IP technology, the CAC scheme and its performance evaluation must be reconsidered under the packet-switched mode.

References

1. S. Singh, V. Krishnamurthy, and H. Poor.: Integrated Voice/Data Call Admission Control for Wireless DS-CDMA Systems. *IEEE Trans. on Signal Processing*, Vol. 50, No. 6 (2002) 1483–1495
2. D. Shen and C. Ji.: Admission Control of Multimedia Traffic for Third Generation CDMA Network. In: Proc. of IEEE INFOCOM 2000. Vol.3, (2000) 1077–1086
3. D. Ayyagari and A. Ephremides.: Optimal Admission Control in Cellular DS-CDMA Systems with Multimedia Traffic. *IEEE Trans. on Wireless Commun.* Vol. 2, No. 1 (2003) 195–202
4. W. S. Jeon, and D. G. Jeong.: Call Admission Control for CDMA Mobile Communications Systems Supporting Multimedia Services. *IEEE Trans. on Wireless Commun.* Vol. 1, No. 4 (2002). 649–659
5. T. Shu and Z. Niu.: Call Admission Control Using Differentiated Outage Probabilities in Multimedia DS-CDMA Networks with Imperfect Power Control. *IEICE Trans. Commun.* Vol. E86-b, No.1 (2003) 16–24
6. D. Zhao, X. Shen and J. Mark.: Uplink Power Distribution in MC-CDMA Systems Supporting Heterogeneous Services. In: Proc. of IEEE Globecom'01. (2001) 604–608
7. O. Gurbuz, and H. Owen.: Dynamic Resource Scheduling Strategies for QoS in W-CDMA. In: Proc. of IEEE Globecom'99. Vol. 1, (1999) 183 –187
8. N.C. Beaulieu, A.A. Dayya and P.J. McLane.: Estimating the Distribution of a Sum of Independent Lognormal Random Variables. *IEEE Trans. on Commun.* Vol.43, No.12 (1995)
9. T. Shu and Z. Niu.: Capacity Optimized Power Allocation for Multimedia CDMA Networks under Imperfect Power Control. In: Proc. of IEEE Globecom'02 (2002)
10. Y. Fang, and Y. Zhang.: Call Admission Control Schemes and Performance Analysis in Wireless Mobile Networks. *IEEE Trans on Vehicular Tech.* Vol. 51, No. 2 (2002)
11. S.K. Bose.: An Introduction to Queueing Systems. Kluwer/Plenum Publishers (2001)
12. D. Zhao, X. Shen and J. Mark.: Performance Analysis for Cellular Systems Supporting Heterogeneous Services. In: Proc. of IEEE ICC 2000 (2000) 3351–3355

Efficient Data Consistency Schemes in 2-Tier Cellular Networks*

Seungbum Lim¹, Jai-Hoon Kim², and Young-Bae Ko²

¹ R&D Center, Curitel, South Korea
shawnux@curitel.com

² College of Information and Communication,
Ajou University, South Korea
{jaikim,youngko}@ajou.ac.kr

Abstract. Mobile computing raises many research issues due to the different characteristics of computing resources. In this paper, we present the concept of 2-tier data consistency schemes in which the messages to be transferred are handled by the appropriate MSS(Mobile Support Station). Instead of just relaying messages among MH's(Mobile hosts), MSS's maintain data and handle the messages together on behalf of the mobile hosts in its area. Also, different data consistency schemes can be applied for each different level (wireless or wired communication level) to reduced communication overhead.

1 Introduction

Mobile computing raises many new design issues to be considered, such as lack of stable storage, low bandwidth of the wireless channel, high mobility, and limited battery life. These new issues make traditional distributed schemes like data consistency and data replication algorithms unsuitable. Therefore, we need new schemes to deal with lack of resources such as computing power, memory, communication channel, and energy. Two conventional protocols are used to maintain data consistency between nodes[4]:

- **update scheme:** When a node accesses a data for the first time, a copy of the data is brought into the local memory of the node. This copy of the data is updated whenever another node modifies the data.
- **invalidate scheme:** Whenever a remote node modifies a data, the local copy is invalidated.

To take both advantages of the two schemes (update and invalidate schemes), many schemes are proposed(e.g., [1,5]). The basic idea of these schemes are to update those copies of a data that are expected to be used in the near future, while selectively invalidating other data. Particularly, when a mobile host(MH) accesses data shared between many MH's, considering data replication and consistency is essential to reduce communication overhead[2,3].

* This work is supported by the Ministry of Education of Korea(Brain 21 Project Supervised by Korea Research Foundation), and University Fundamental Research Program supported by Ministry of Information & Communication in Republic of Korea(2001-103-3).

In this paper, we propose an algorithm to reduce both the communication cost and the workload of MH's. 2-tier data consistency scheme can reduce the number of messages by utilizing two levels. In our schemes, the messages to be transferred are handled by the appropriate MSS's (Mobile Support Stations). Instead of just relaying the each individual message among MH's, MSS's handle a message once on behalf of all the MH's in its cell. Also, a different data consistency scheme can be applied for each level, wireless or wired communication level, to reduced communication overhead.

2 Related Works

Cost analysis models are proposed to measure the communication cost for maintaining data consistency in multi-computer systems. We use following two cost models to compare performance between the previous and proposed data consistency schemes for mobile computing:

- **segment model [1]:** Adaptive DSM(distributed shared memory) scheme is proposed based on cost comparison using “segment” model. Cost is analyzed for the shared memory accesses at each node. These accesses can be partitioned into “segments”. A new segment begins with the first access by a node following an update to the data by another node.
- **write-run model [7]:** Write-run model is proposed to predict the cache coherency overhead for the bus based multiprocessor system. The write-run is a sequence of local writes between two consecutive remote accesses.

The 2-tier distributed algorithm [8] is proposed to reduce the computation work of the MH and the overall message transfer. In this algorithm, most computation is performed on the fixed host(MSS: mobile support station) having enough computation power and the message transfer is relayed through the view point of wired network (high level) instead of the wireless network (low level). 2-tier networks can reduce communication overhead by making MSS maintain the data and handle messages together on behalf of MH's in its area and choosing an appropriate data consistency scheme for each levels.

3 2-Tier Data Consistency Schemes

In this section, we analyze the communication cost for the previous scheme and our proposed 2-tier scheme to maintain data consistency.

3.1 Cost Analysis

3.1.1 Previous Scheme (One-to-One Scheme)

We use following parameters for cost analysis:

- f : number of forwarding to find data copier
- C_c : cost for transferring control message

- C_b : cost for transferring data block
- C_{upd} : cost of sending update message
- w : ratio of wireless to wired communication cost
- s : the average number of mobile host which has copy in a cell
- n : total number of mobile host in a system
- m : total number of MSS in a system
- U : number of update from other nodes in a segment
- We analyze the cost in one segment in the view point of each node.
- **update scheme:** Let U be the number of updates to the local copy of data during a segment. Acknowledgement is sent (with cost C_c) for each update message received (with cost C_{upd}) in update scheme. Two wireless links (between MSS and MH) and one wired link (between two MSS's) are connected between two MH's. Therefore, the cost needed in one segment is $U(2w + 1)(C_{upd} + C_c)$.
- **invalidate scheme:** From the definition of a segment, each segment begins with a data access fault. We assume that data owner has a master copy and located in fixed host (e.g., MSS). To locate the owner one wireless links and f (number of forwarding) wired link are needed (with cost $(w + f)C_c$). Then, to fetch the data cost $(1+w)C_b$ is needed. An invalidation is needed for each segment which requires one update message and one for a negative acknowledgement to the sender of the update. Thus, $(2w+1)(C_{upd} + C_c)$ is required. Therefore, total cost is $(w + f)C_c + (1 + w)C_b + (2w+1)(C_{upd} + C_c)$.

3.1.2 2-Tier Scheme

We can also analyze cost for 2-tier scheme using the similar way as in subsection 3.1.1. However, a message to be delivered to MH's in the same cell is sent to MSS of receiving MH's via MSS of sending MH's. Then, the receiving MSS delivers the message for each MH. A message from sending MH can be shared until the message reach the receiving MSS. Thus, cost analysis for 2-tier scheme is as follows:

- invalidate scheme: $(w + f)C_c + (1 + w)C_c + (w + w/s + 1/s)(C_{upd} + C_c)$
- update scheme: $u(w + w/s + 1/s)(C_{upd} + C_c)$

3.1.3 Two-Level Scheme

In the 2-tier network, two levels use different communication channels; wired communication for high-level and wireless communication channel for low-level. As two communication channels have different characteristics, It had better choose appropriate scheme for each level individually. We use simpler cost model than those of 3.1.1 and 3.1.2 using the following parameters:

- R : number write-run in a segment (refer section 2 for definition of write-run)
- $C_{invalidate}$: cost for receiving invalidation and sending negative acknowledgement
- C_{update} : cost for receiving update and sending acknowledgement
- C_{bf} : cost for fetching data (including forwarding message to locate owner)
- U_i : the number of update from MH's of local cell in a segment
- U_r : the number of update from MH's of remote cell in a segment

In the analysis in subsections 3.1.1 and 3.1.2, we do not consider broadcast capability of MSS to the MH's in its cell. In the analysis for two-level scheme, we utilize the broadcast mechanism. Thus, we use different cost model for each level.

Segment model developed for shared data in distributed system is proper cost model for high-level while write-run cost model is proper for low-level. The costs are shown in Table 1. Cost analysis for high-level is similar to subsections 3.1.1 and 3.1.2. Cost analysis for low-level is as follows:

- **invalidate scheme:** When a new write-run starts writing MH fetches data with cost wC_{bf} and other MS's receive invalidation message (broadcast) with the cost $wC_{invalidate}$. Thus, $wR(C_{bf} + C_{invalidate})$ is required for a segment per MSS (cell)
- **update scheme:** For each update wC_{update} broadcast cost is required. Thus, cost for a segment per MSS(cell) is $w(U_r + U_l)C_{update}$.

Thus, cost for data consistency between MSS's is analyzed. Then, cost for low-level is analyzed

Table 1. Cost of consistency for two-level (per MSS in a segment)

Level Scheme(high + low)	High-level (between MSS's)	Low-level (MSS and MH)
Update + Update	$U_r C_{update}$	$w(U_r + U_l)C_{update}$
Update + Invalidate	$U_r C_{update}$	$wR(C_{bf} + C_{invalidate})$
Invalidate + Invalidate	$C_{bf} + C_{invalidate}$	$wR(C_{bf} + C_{invalidate})$

3.2 Performance Evaluations

We compare the performance of proposed 2-tier data consistency schemes to those of previous scheme. Parametric analysis is performed based on the analysis in subsection 3.1. Figure 1 shows the communication cost per segment. As shown in Figure 2, cost decreases as the value s increases. We also compare the cost for two-level scheme. As shown in Figures 3 and 4, no scheme is the best for all cases. Thus, we need to choose an appropriate two-level scheme for each case.

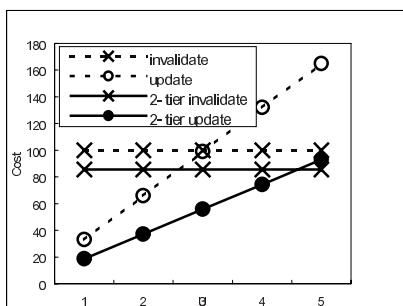


Fig. 1. 2-tier scheme by varying U ($w=5$, $f=2$, $C_c=1$, $C_{bf}=10$, $C_{upd}=2$, $s=5$)

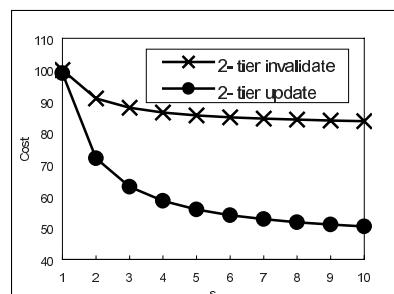


Fig. 2. 2-tier scheme by varying s ($w=5$, $f=2$, $C_c=1$, $C_{bf}=10$, $C_{upd}=2$, $U=3$)

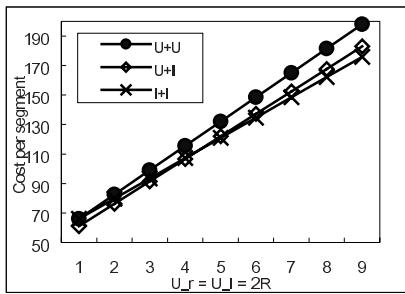


Fig. 3. Two-level scheme by varying U ($w=5$, $C_{\text{invalidate}}=1$, $C_{\text{bf}}=10$, $C_{\text{update}}=3$)

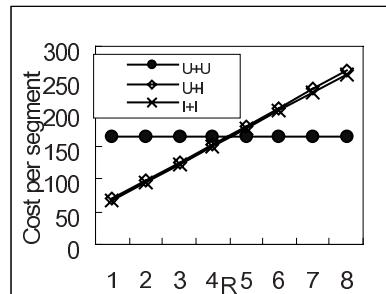


Fig. 4. Two-level scheme by varying R ($w=5$, $C_{\text{invalidate}}=1$, $C_{\text{upd}}=3$, $C_{\text{bf}}=10$)

4 Conclusions

In this paper, we propose an algorithm to reduce both the communication cost and the workload of the MH. A 2-tier data consistency scheme can reduce the communication cost by making MSS deliver the message once on behalf of MH's in the same cell, which can share the message between MH's. Also two-level schemes are proposed to choose an appropriate scheme for each level. Parametric analysis shows that 2-tier scheme and two-level schemes are viable approaches to reduce communication overhead in the 2-tier networks.

References

- [1] J. H. Kim and N. H. Vaidya, "A Cost-Comparison Approach for Adaptive Distributed Shared Memory," ACM International Conference on Supercomputing, pp. 44–51, May 1996.
- [2] A. Lubinski, A. Heuer, "Configured Replication for Mobile Applications," in Proc. of the BalticDB&IS'2000, May 2000.
- [3] E. Pitoura, B. K. Bhargava, and O. Wolfson, "Data Consistency in Intermittently Connected Distributed Systems," in Proc. of the Transactions on Knowledge and Data Engineering, 11(6), pp 896–915, Nov/Dec 1999.
- [4] M. Stum and S. Zhou, "Algorithms implementing distributed shared memory," IEEE Computer, vol. 23, pp. 54–64, May 1990.
- [5] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, "Using leases to support server-driven consistency in large-scale systems," International Conference on Distributed Systems, May 1998.
- [6] H. Yu, A. Vahdat, "Design and Evaluation of a Continuous Consistency Model for Replicated Services," in Proc. of the Fourth Symposium on Operating Systems Design and Implementation, Oct. 2000.
- [7] S. Eggers, "Simplicity versus accuracy in a model of cache coherency overhead," IEEE Transactions on Computers, vol. 40, pp. 893–906, Aug. 1991.
- [8] B. R. Badrinath, A. Acharya and Tomasz Imielinski, "Designing distributed algorithms for mobile computing networks", Computer Communications, vol. 19, no. 4, 1996.

iHOPE: An Intelligent Handoff Protocol for Seamless Multimedia Service in Wireless Network

Jang-Woon Baek and Dae-Wha Seo

Electrical Engineering and Computer Science,
Kyungpook National University,
1370 Sankyuk-Dong, Buk-gu, Daegu, Korea
kutc@palgong.knu.ac.kr
dwseo@ee.knu.ac.kr

Abstract. In this paper, we propose an intelligent hand-off protocol(iHOPE) to support seamless multimedia stream service with less handoff latency and packet loss. In iHOPE, neighbor base stations receive media data packets from a server and transcode it for a mobile host in advance. When the mobile host enters a new cell, the base station of the cell immediately forwards the transcoded media data packet to the mobile host. This paper describes the intelligent handoff protocol and the media data transcoding mechanism. We evaluate our approach with simulation results.

1 Introduction

In the development of wireless network and communication technologies, wireless internet users are dramatically growing and the demand of multimedia services are increasing. However, in mobile computing environment, the constraints of wireless links and user mobility make it difficult to provide seamless multimedia stream services for mobile hosts.

In this paper, we propose an intelligent hand-off protocol environment(iHOPE) to support seamless multimedia stream service in wireless network. In iHOPE, base stations, which are neighborhoods to the primary base station, receive media data packets from a server and transcode it for mobile host, in advance. When the mobile host enters the neighbor cell, the base station of the cell immediately forwards the transcoded media data to the mobile host. In the remainder of this paper, we describe the iHOPE in more detail. Section 2 discusses backgrounds for hand-off and transcoding. Section 3 describes the intelligent handoff protocol environment(iHOPE). Section 4 presents the simulation results. Finally conclusion remarks are given in Section 5.

2 Background

Mobile IP[1] is used for supporting the user mobility in wireless network. When mobile host moves to other networks, route update is performed in Mobile IP.

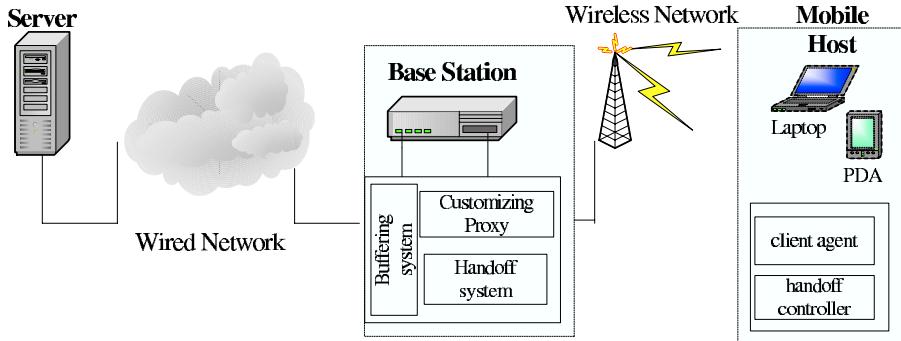


Fig. 1. intelligent Handoff Protocol Environments

While the route is set up, the packets are dropped or stored in buffer. Dropped packets cause the packet loss. After route update, the packet transmission brings about additional delay. These packet loss and delay degrade the quality of multimedia service. As a result, it is necessary for the handoff protocol to achieve low handoff delay and negligible data loss. Handoff protocol using multicast is proposed to overcome the problems caused by handoff [2].

Multimedia data has the attributes such as large volume and real time service. However, it is difficult to provide the multimedia stream services due to the narrow bandwidth and poor MN's device capacity in wireless network. To solve these problems, an intermediary has been located between wired network and wireless network. The intermediary plays a role such as filtering media data [3], transcoding web image [4]. But most of these methods only consider characteristics of a physical link without reflection of user mobility.

3 Intelligent Handoff Protocol Environment(iHOPE)

3.1 iHOPE Architecture

In this paper, we propose the intelligent hand-off protocol environments(iHOPE) that supports the seamless multimedia streaming services ensuring user mobility. Fig. 1. presents the iHOPE.

- Buffering Module: To store data that are modified by transcoding proxy,
- Transcoding Proxy: to transcode the media data according to the user profile and the bandwidth of network,
- Handoff Module: to support user mobility,
- Handoff control Module: to decide the route and control BS during handoff,
- Client module: to provide the user profile and control the transcoding proxy according to the change of host's environment.

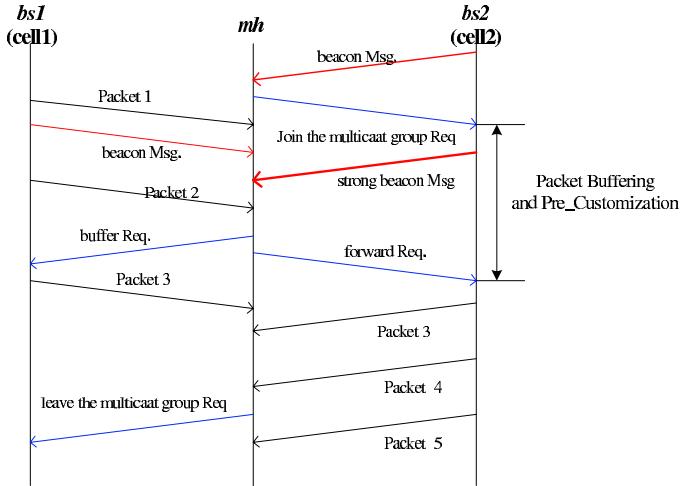


Fig. 2. iHOPE Handoff Flow

3.2 iHOPE Handoff Protocol

Fig 2. depicts a handoff protocol between BSs and a MH when a *mh* moves from the cell1 to the cell2. The *mh* periodically receives beacon messages from BSs(*bs1*, *bs2*). As the *mh* approaches to the cell 2, the *mh* gets more strong signals from *bs2* than before. If the *mh* receives the stronger signal than the threshold of signal strength, the *mh* name the *bs2* for buffering base station. At same time the *mh* sends the message that requests the *bs2* to join the multicast group. On joining the *mh*'s multicast group, *bs2* receives multicast packets from the home agent. The *bs2* decapsulates the received media packets, and transcodes media data through transcoding proxy, and then stores the transcoded media data packets in the buffer. If the *bs2*'s signal strength is stronger than the *bs1*'s signal strength, the *mh* decides the forwarding BS with the *bs2* and requests *bs2* the packet forwarding. The *bs2* immediately sends the transcoded packets when *bs1* enters the cell2. At this time, *bs1* receives the buffer request and becomes the buffering BS. If *mh* goes away from *bs1* and receive weaker signal than the lower threshold, *mh* sends a message that asks *bs1* to leave the multicast group. *bs1* no longer receives the message.

3.3 Transcoding Proxy

BS's Transcoding Proxy transcodes media data based on network bandwidth and user profile. Fig. 3. shows the internal architecture of TP. The policy manager makes a decision of transcoding policy on the basis of MH's user profile and network information. The transcoding policy means the set of transcoding parameters such as a encoding bit rate, a frame size, a frame format, a quantization table. The transcoding engine (TE) performs the transcoding of media data according to the transcoding policy which is provided by the policy manager. The

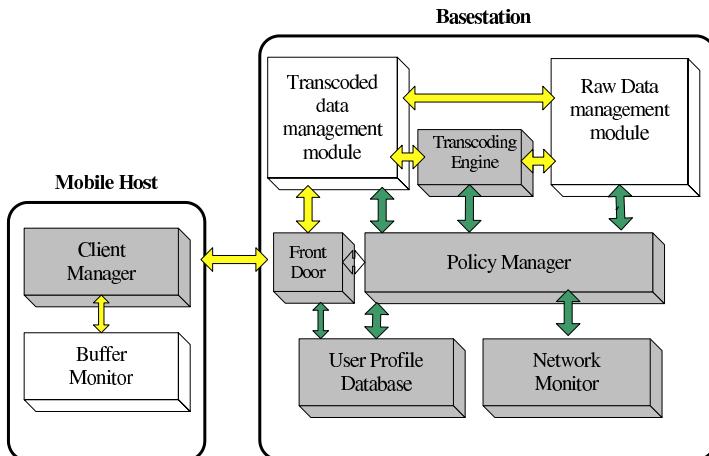


Fig. 3. Internal Structure of Transcoding Proxy

network monitor (NM) observes the change of network resources between MH and BS. The NM gives the network information to the policy manager through the query based interface. The User Profile is stored in the database. The User profile includes a user information for authentication, the device capacity and user preference of mobile host. Frontdoor(FD) is responsible for MH's authentication and intermediation between the server and the MH.

4 Simulation

We used the ns (network simulator)[5] to evaluate the iHOPE. We design the simulation model like Fig. 4.

We evaluate two cases of simulation. One is the thing which applies the iHOPE handoff protocol, another is the thing which uses general handoff protocol. Table 1 shows the result of simulation. We measured the handoff latency, number of packets lost and the average jitter. iHOPE handoff Protocol has lower handoff latency and jitter than the general handoff protocol. Also there isn't the packet loss in iHOPE. We know that the iHOPE Handoff protocol is better than the general handoff protocol.

Table 1. The Result of Network Simulation

Handoff Protocol	Handoff Latency	Number of lost packet	Jitter
General Handoff Protocol	12-13ms	5-6	0.017Mbps
iHOPE Handoff Protocol	5-8ms	0	0.01Mbps

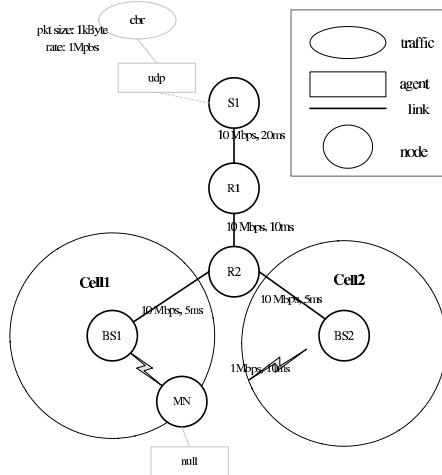


Fig. 4. Simulation Model

5 Conclusion and Future Work

In this paper, we proposed iHOPE to reduce handoff latency and packet loss, and to overcome constraint of wireless link and MH's device. In iHOPE each BS buffers the packet for MH and transcodes the media data before mobile host enter the cell. As a result, iHOPE is able to support seamless multimedia stream service for MH with less handoff latency and packet loss. The result of simulation shows the excellence of iHOPE in term of packet loss and handoff latency.

In the future, we study the media caching to reduce the response time for user. And we study transfer policy that considers the priority of application.

References

1. Charles E. Perkins, : "Mobile IP", IEEE Communication Magazine, (1997)
2. Bruce Zenel, : "A general purpose proxy filtering mechanism applied to the mobile environment", IEEE Wireless Networks 5 pp. 391–409, 1999
3. Srinivasan Sesha et al, : "Handoffs in Cellular Wireless Network: The Daedalus Implementation and Experience", Kluwer International Journal on Wireless Personal Communication, 1997.1
4. Armando Fox et al, : "Adapting to network and client variability via On Demand Dynamic Distillation", Proceeding. Seventh International Conference on Architecture. 1996.10
5. the network simulator, : ns-2, <http://www.isi.edu/nsnam/ns/>, (current April 2003)

Content and Cell Based Predictive Routing (CCPR) Protocol for Mobile Ad Hoc Networks*

Jörg Kaiser and Changling Liu

Department of Computer Structures

University of Ulm,

89069 Ulm, Germany

{joerg.kaiser, changling.liu}@informatik.uni-ulm.de

<http://www.informatik.uni-ulm.de/rs>

Abstract. With the popularity of mobile safety-critical applications, there is an increasing need for developing routing protocols with improved predictability and reliability for the mobile ad hoc networks. This paper proposes a Content and Cell based Predictive Routing (CCPR) protocol, which facilitates safety-critical applications in mobile environments. CCPR exploits the publisher/subscriber model in route discovery procedures to constraint the flooding of control packets. Cells are used in routing path construction instead of individual nodes. Together with proactive and local route maintenance, CCPR improve the predictability of packets forwarding.

1 Introduction

Mobile ad hoc networks (MANETs) are dynamically self-organized, rapidly deployable networks independent of any fixed infrastructure [1]. Compared to wired networks, MANETs have unique characteristics, such as the dynamic changing network topology, variable link capacity, the scarce bandwidth and the constrained energy. Because of the dynamic nature of MANETs, it is a tough challenge to design robust, scalable and effective routing protocols, which quickly adapt to un-predictive and frequent network topology changes.

Most of the existing MANET routing protocols work in a "best effort" way, and cannot provide adequate support for safety-critical applications needed in disaster search and rescue operations, cooperating autonomous robots, and traffic management. Safety-critical applications need high predictability and reliability and these requirements impose high challenges on MANET routing and message dissemination.

In this paper, we propose the Content and Cell based Predictive Routing protocol (CCPR) in order to improve the routing and dissemination predictability. CCPR is a transport layer protocol and to meet its goals, CCPR exploits support from the underlying MAC layer and benefits from the publisher/subscriber communication

* The work described in this paper was partially supported by the EC, through project IST-2000-26031 (CORTEX – CO-operating Real-time senTient objects: architecture and EXperimental evaluation).

model realized in the higher network layers. On the MAC-layer we base CCPR on the highly predictable TBMAC-protocol [6]. TBMAC imposes a cell structure to the MANET and provides a TDMA scheme for reliable communication. On the higher layer, the publisher/subscriber model offers anonymous, asynchronous communication that has many benefits in dynamic cooperative applications when compared to the synchronous point-to-point style of RPC/RMI communication [15, 16]. We particularly added event channels to the publisher/subscriber model [17], which are related to the contents of the messages they disseminate. The scheme allows us to assign QoS attributes to event channels. In CCPR we relate a routing path from a publisher to the subscribers to an event channel with a specific QoS guarantee.

The remainder of this paper is organized as follows. Section 2 gives a short introduction for TBMAC. Section 3 provides the basic operations of CCPR. In section 4, we present the related work, and section 5 gives the conclusion and provides an outlook on future work.

2 TBMAC

Without support from the MAC layer, it is hard to guarantee the network layer predictability and reliability for MANETs. TBMAC provides predictable access to the wireless medium for mobile nodes [6]. TBMAC is a location-dependent protocol and assumes that nodes can assess their geographical positions. The entire geographical area is statically divided into geo-cells. The size of the cells is related to the transmission range of the radio equipment. Orthogonal spreading codes are allocated to geo-cells to omit collisions and the hidden terminal problem. It should be noted that the geo-cells do not provide any infrastructure or access points as in other cell-based approaches. Rather, a geo-cell constitutes a well-defined area and a node can always determine in which of these areas it actually is. In TBMAC, communication is organized in rounds each of which comprises a Contention Free Period (CFP), a Contention Period (CP) and an Inter-cell Communication Period (ICP). The communication rounds are synchronous for all cells. Each period has a well-known duration and is divided into time slots. For intra-cell communication, the access to the medium is divided into time periods CFP and CP. The allocation and de-allocation of time slots in TBMAC are similar with the PCF (Point Coordination Function) in IEEE 802.11 standard, but fully distributed. The ICP is used for the message exchange between adjacent cells. Time slots in ICP are assigned to gateway nodes, one or more for each adjacent cell. The gateway nodes are dynamically elected. The gateway node election scheme extends the time slots allocation and de-allocation requests of TBMAC by including an extra field, which represents the power level or computing resource level information. Nodes decide whether they should act as gateway nodes according to this information.

3 Content and Cell Based Predictive Routing (CCPR) Protocol

In CCPR, nodes are organized in groups based on the geo-cells of TBMAC. To improve the robustness of the routing paths, geo-cells are used to construct routing

paths instead of individual nodes. Therefore, mobile nodes within a cell need to maintain some consistent information.

Obviously, in CCPR, nodes in a cell should maintain the same routing information. Therefore, movement of single node or disconnection of single link will not result in routing path disconnection. Consequently, the hop number is calculated based on the number of cells a packet traverses along the routing path, but not on a node-to-node style. When a routing information change occurs, all the nodes within a related cell update their routing tables according to the change. Because of the shared routing information, each node in a cell has the capability to act as a gateway to an adjacent node.

In CCPR, the cells are addressed reflecting their geographical relationship. Because the cell structure is static, the number of hops between any two nodes can be pre-calculated and remains unchanged as long as the respective cells on the route are populated. With the predictive MAC layer protocol and resource reservation, the time needed for packet transmission from one cell to another also can be calculated using the number of cells the packet will traverse.

In a publisher/subscriber protocol, routing of messages is performed on the basis of message contents rather than on source and destination addresses. We adopted subject-based routing [18] in which the content is represented by a short tag. In CCPR the subject is bound to a network address when it reaches the cell in which the respective publisher resides. Therefore, a Cell Resource Table (CRT) is introduced which maintains information about all publishers or, respectively, subscribers currently in a cell. During route discovery, the CRT binds subjects to the network identifiers of the publishers residing in the cell. Vice versa the binding is also performed when a message reaches the cell of a subscriber. Because a CRT is available in every node in a cell, it is possible to decide on the basis of the CRT whether nodes in this cell can provide the requested data. When a node leaves or moves in a cell, the content of the CRT will be updated accordingly.

CCPR has three phases, namely, the route discovery phase, the routing path construction phase and the route maintenance phase. The following subsections give descriptions of these phases.

3.1 The Route Discovery Phase

The route discovery procedure in CCPR is tightly coupled with the resource discovery of the publisher/subscriber model. A subscriber floods discovery requests to search for publishers belonging to the requested channel.

If a subscriber wants to get messages from an event channel, it firstly issues a route discovery (RDIS). The RDIS packet includes the cell address of the subscriber, the subject corresponding the channel, the sequence number, the maximum hop number, the search area, and the current hop number. Different RDIS packets issued by the same subscriber can be distinguished using the sequence number. The search area information, such as the maximum hop number, the proximity, the direction or cell addresses restricts the flooding of the discovery requests.

According to the cell division method used in CCPR, all gateway nodes of the subscriber residing cell will receive the RDIS. After receiving an RDIS packet, gateway nodes check the subject field and sequence number field of the packet. If this packet has been received recently, the gateway nodes drop the packet. Otherwise, the

gateway node increase the current hop number field in the RDIS by one, and selectively forwarding RDIS to the adjacent cells according to the search area field. The same operation will continue until the search area is covered or the current hop number equals the maximum hop number.

Flooding is a crucial limitation for the scalability of routing protocols. To reduce the overhead incurred by flooding, only the gateway nodes are involved in the route discovery procedure of CCPR.

3.2 The Routing Path Construction Phase

After receiving a new RDIS request, a gateway node checks the Cell Resource Table (CRT). If the CRT includes the channel specified in the subject field in the RDIS, the gateway node will check its local routing table. The routing table contains all information needed to route a packet from the source to the requesting destination. A routing table entry includes the subject of an event channel, the destination cell address, the maximum hop number to the destination cell, the current hop number to the destination cell, the next hop cell address, and the Time-To-Live (TTL) field. The maximum hop number is defined according to the channel's temporal requirements.

If there is no routing table entry yet for the respective subject and the requesting cell, a new path has to be constructed. In this case, the gateway node forwards a route reply (RREP) packet back to the cell of the subscriber. A RREP packet includes the cell address of the publisher, the cell address of the requesting subscriber, the subject of the channel, the maximum hop number, the current hop number, the QoS attribute used for routing path construction and a time stamp. During RREP forwarding procedure, a routing path is constructed. The proper adjacent cells are selected to construct the routing path based on the subscriber specified QoS requirements. The geographical information obtained from cell addresses is used to assist this procedure. A gateway node selects the next hop cell based on the QoS attributes and the geographical information and forwards the RREP packet to the selected cell. It creates a new entry in the local routing table using the information in the RDIS packet. Subsequently, a routing table consistency procedure among the nodes in the cell is initiated. Readers can get detail information of routing table consistency procedure from [19]. The operation described above will continue until the RREP arrives the subscriber or the hop number exceeds the specified maximum hop number.

The bandwidth, reliability or shortest path can be used as metrics when selecting the next cell. Generally, densely populated cells are more reliable, but have less bandwidth available per node compared to sparsely populated cells. Thus, when reliability is used as the metric for routing path construction, node density information will be exploited.

3.3 The Route Maintenance Phase

The main challenges of the routing protocol design for MANETs originate from the node mobility and the resulting dynamic network topology changes. CCPR reduces the impact from the node mobility by taking advantage of the redundant routing information available in a cell. Because all nodes within a cell maintain the same routing and channel information, all of them have the capability to act as gateway

nodes. Only in the following three cases, node movement will incur route maintenance procedures: 1.) a publisher moves out, 2.) a subscriber moves out, 3.) an intermediate cell along the routing path becomes empty.

The effect of node movement will be compensated by the local adaptation of the routing path if it is possible to connect the moving node to the already established routing path. Thus, the major part of the existing path can still be used. The alternative solution would be reconstructing a complete new routing path, which obviously creates a substantial overhead compared to the local solution. Moreover, because a node is aware of its position, speed and direction of movement, the local route maintenance can be performed proactively. When a node detects that it soon will move into an adjacent cell, it initiates the route maintenance protocol. This leads to an improved predictability of communication because now, the latency of the path in number of hops can either be preserved or it changes at most by one hop. This sharply contrasts to the temporal uncertainty of a complete route reconstruction. Only in cases of a very sparsely populated environment, i.e. most of the adjacent cells are empty; a complete reconstruction will be inevitable.

When a node, hosting a publisher, is leaving a cell, it sends a maintenance request to the adjacent cell that has been identified as its target cell. A gateway node in the target cell then checks its routing information to determine whether the cell is already part of the path or not. If there already exists a routing table entry that can be used, there is not much to do because the cell already was part of the path. If there is no such routing table entry for this publisher, the gateway node must create a new routing table entry to connect the cell of which the publisher is moving out. In fact, this operation constructs a local routing path segment and connects it to the existing routing path. The changes of the route path need to be sent explicitly along the routing path or be piggybacked with the next data packet to the cells along the routing path and the subscriber's cell.

A similar approach is used for a subscriber movement. Before a subscriber moves across a cell boundary, it proactively sends a route maintenance request to the cell it moves in. The request includes the routing information currently used by the subscriber. As in the previous case, the maintenance operation is confined to the cell that the subscriber is leaving and the target cell. The routing path update message will be sent explicitly to the nodes along the routing paths of the event channel.

The mobility of the nodes may lead to the situation that an intermediate cell long a multi-hop path becomes empty. In CCPR, an alternate routing path proactively is constructed before the last node moves out of the cell. This requires two steps. Firstly, a node must be able to detect that it is the only node in a cell. This is achieved through the time slot assignment vector that always comprises all nodes in a cell. Let's call the cell with only one node inside an unstable cell. The second step is the local adaptation of the path that has an unstable cell. The node in the unstable cell tries to find an alternate routing path segment in the adjacent cells. In principle, this proceeds like the maintenance protocols described above. However, there are two differences that can be seen as optimizations for this special case. Firstly, because the last node in a cell acts as the gateway node for each adjacent cell, it knows their bandwidth and reliability information. Secondly, there is not only one target cell as in the cases described above, but there are more options. Thus route maintenance requests will be sent to all target cells that can provide enough QoS support for the routing path. After receiving the request, the nodes in the selected adjacent cells update their routing

table entries respectively. As the final part of the maintenance operation, the routing information in the unstable cell is deleted.

It is possible that an alternative routing path segment cannot be found because the requirements of the subscriber, such as the maximum hop number, the bandwidth or the reliability cannot be satisfied. In this case, the full path route maintenance is needed, and a new routing path construction procedure will be performed. As been pointed out previously, this happens only in the case when no adequately populated cells are in reach. We assume that this will be a rare case.

4 Related Work

A variety of MANET routing protocols have been proposed in recent years. Generally, they can be classified into proactive routing, reactive routing and hybrid routing. CCPR is different from the proactive routing protocols such like Destination Sequenced Distance-Vector routing protocol [2] and Adaptive Distance Vector Routing Algorithm [9]. In a proactive routing protocol, each node maintains routing information for all reachable nodes in the network by continuously exchanging topology or link information. Proactive routing protocols have scalability problem when used in large or even medium size networks. CCPR maintains topology information in an on-demand style like the reactive routing protocols such like Ad hoc On-demand Distance Vector Routing [11] and Dynamic Source Routing Protocol [3]. Routing paths are constructed only when needed, and a route query flooding is exploited to search for the destination. Reactive routing protocols have better scalability characteristic compared to the proactive routing protocols. However, route discovery and routing path construction before data transmission can introduce delay and time uncertainty. Additionally, the flooding operation in route discovery procedures may incur heavy traffic overhead. These substantially affect predictability of packet transmissions. To alleviate above problems, CCPR introduces cell structure and try to incorporate merits both of proactive routing and reactive routing. To achieve the same objective, hybrid routing protocols, such as the Zone Routing Protocol [5], utilize appropriate routing schemes for nodes in different space scales.

Most of existing routing protocols do not maintain redundant routing information. Single node mobility and single link breakage may cause routing path disconnection. Consequently, frequent routing path re-discovery or maintenance procedures exacerbate the time uncertainty of routing. The cluster-based routing protocols, such as the Clusterhead Gateway Switch Routing protocol (CGSR) [8] and Cluster Based Routing Protocol (CBRP) [7], are proposed to improve robustness and scalability of MANET routing. For each cluster, a clusterhead is elected to manage membership and maintain routing information for the member nodes. However, cluster based routing protocols cannot provide satisfying predictability guarantees because of unpredictable cluster merging and splitting caused by the clusterhead movement, context-sensible temporal property of the clustering algorithms, and single point of failure and bottleneck problems. Using CCPR, nodes share routing information in a distributed and consistent way. So, CCPR is expected to have improved predictability and robustness without introducing single failure and bottleneck problems.

Obtaining geographical information from GPS or similar equipment, the Location-Aided Routing (LAR) [4] and Distance Routing Effect Algorithm for Mobility

(DREAM) [10] reduce the amount of control packets for the routing path discovery by exploiting location information, but they do not support network predictability and reliability. CCPR uses location information mainly for cell division and maps location information to addresses of cells.

Content based routing scheme is also proposed for the mobile ad hoc networks. The Content Based Multicast (CBM) routing model [12] is proposed for military and emergency applications. In CBM, the content of the multicast data determines the receiver set for the data. In CBM, the timeliness and proximity are used as QoS metrics. However, mobile nodes are organized in cluster-like structures as cluster-based routing protocols. So, CBM has same drawbacks as cluster-based routing.

Paper [17] proposes a scheme to construct an explicit publish/subscribe tree from a publisher to the corresponding subscribers. However, generally a tree structure is fragile and performs badly in highly dynamic mobile networks.

QoS routing protocols are mainly proposed for multimedia applications in MANETs. In CEDAR [13], the route discovery operation utilizes broadcasting along the “core” nodes to find routing paths that meet bandwidth requirement. In [14], a ticket-based scheme searches multiple routing paths in parallel and tickets are used to constraint the route discovery packets flooding. Some cluster-based routing protocols, such as CGSR, also provide QoS support. Unlike these protocols, CCPR provides QoS support by exploiting publisher/subscriber model, cell structure and support from TBMAC.

5 Conclusion and Future Work

Safety-critical applications are an important and emerging application domain of MANETs. Their high predictability and reliability requirements add tough challenges for the routing protocols. The Content and Cell based Predictive Routing (CCPR) protocol is proposed to support safety-critical applications in mobile ad hoc environments.

To improve the predictability of the routing paths in CCPR, mobile nodes are organized by cells. Cells are exploited as the basic units for multi-hop routing construction instead of individual nodes. The scheme has advantage that it allows calculating the latency of a path by simply adding the latency in the cells and preserves this latency of the path in the presence of a constantly changing node population inside the cells. Of course, this requires that cell latency is predictable and secondly, that the cells are stable, i.e. there is enough redundancy in a cell to assure stable routing in spite of the mobility of individual nodes. Replication of routing information and proactive route maintenance strategies are used to meet this goal. If possible, a route is actively reconfigured to preserve the specified properties. For route discovery, the subject based addressing of the publisher/subscriber model is exploited. Only a fraction of nodes are involved in control packets flooding, and the search areas of route discovery packets are constrained.

Our future work includes investigation of redundant routing information to increase the robustness of the routing paths, the information aggregation for the content based routing in mobile ad hoc networks, the publisher/subscriber model in QoS routing for mobile ad hoc networks, and the performance simulation work.

References

1. IETF Manet charter. <http://www.ietf.org/html.charters/manet-charter.html>
2. Perkins, C. E., Bhagwat, P.: Highly dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for mobile computers. *ACM Comput. Commun. Rev.*, Vol.24, No.4, (ACM SIGCOMM'94) Oct. 1994, pp.234–244
3. Johnson, D., Maltz, D. A.: Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, Kluwer Acad. Publ., 1996
4. Ko, Y. B., Vaidya, N. H.: Location Aid Routing (LAR) in mobile ad hoc networks. In *Proc. ACM/IEEE MOBICOM*, Oct. 1998
5. Haas, Z. J.: The Zone Routing Protocol (ZRP) for ad hoc networks. Internet draft
6. CORTEX project deliverable: D4: Preliminary definition of CORTEX system architecture [<http://deliverables/WP3-D4.pdf>]
7. Jiang, Mingliang, Li, Jinyang, Tay, Y. C.: Cluster Based Routing Protocol (CBRP), Internet draft
8. Chiang, C.C., Tsai, T. C., Liu, W., Gerla, M.: Routing in clustered multihop, mobile wireless networks with fading channel. *The Next Millennium, IEEE SICON*, 1997
9. Boppana, R.V., Konduru, S.P.: An adaptive distance vector routing algorithm for mobile ad hoc networks. *INFOCOM 2001 Proceedings*, IEEE, Volume: 3
10. Basagni, S., Chlamtac, I., Syrotiuk, V., WoodWard, B.: A Distance Routing Effect Algorithm for Mobility (DREAM). Proc. 4th MOBICOM, 1998
11. Perkins, C.E., Royer. E.M.: Ad hoc On demand Distance Vector routing, mobile computing systems and applications, 1999, Proceedings, WMCSA '99. Second IEEE Workshop on, 1999
12. Zhou, Hu, Singh, S.: Content Base Multicast (CBM) in ad hoc networks. In *Workshop on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Aug. 2000
13. Sinha, P., Sivakumar, R., Bharghavan, V.: CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm. *IEEE INFOCOM*, March, 1999
14. Chen, Shigang, Nahrstedt, K.: Distributed quality-of-service routing in ad hoc networks. *IEEE Journal on Selected Areas in Communications*, 17(8), August 1999
15. Gugola, G., Nitto, E. D.: Using a publish/subscribe middleware to support mobile computing. *Advanced Topic Workshop Middleware for Mobile Computing*, Nov., 2001, Heidelberg, Germany
16. Huang, Yongqiang, Garcia-Molina, H.: Publish/subscribe in a mobile environment. In *Proceedings of the Second ACM International Workshop on Data Engineering for Wireless and Mobile Access*, 2001
17. Kaiser, J., Mock, M.: Implementing the real-time publisher/subscriber model on the Controller Area Network (CAN). *Proceedings of the 2nd Int. Symp. on Object-oriented Real-time distributed Computing (ISORC99)*, Saint-Malo, France, May, 1999
18. Oki, B., Pfleeg, M., Siegel, A., Skeen, D.: The Information Bus – an architecture for extensible distributed systems. *Operating Systems Review*, 27.5:58 68, 1993.
19. CORTEX project deliverable: D5: Preliminary specification of basic services and protocols [<http://deliverables/WP3-D5.pdf>]

A Multiple Access Scheme Using Split and Merge Algorithm for Contention/Reservation-Based Wireless MAC Protocols

Min-Su Kim¹, Tae-Young Byun², and Ki-Jun Han¹

¹Department of Computer Engineering,
Kyungpook National University
mskim@comeng.ce.knu.ac.kr
kjhan@bh.knu.ac.kr

²School of Computer and Electronic Engineering,
Gyeongju University
tybyun@gyeongju.ac.kr

Abstract. Split Algorithm (SA) is one of the most prominent collision resolution algorithms that have been proposed recently. Our Split and Merge Algorithm (SMA) moderates the number of colliding packets that should be resolved adaptively to the traffic load to enhance the performance of SA. The predicted results indicate that the SMA offers a better performance than the SA. The analytical results are compared with simulated data and good agreement is reported.

1 Introduction

MAC protocols for wireless communication have been investigated extensively over the past year [1]~[4]. These MAC protocols can be classified into two categories, contention-free and contention based. Contention based wireless MAC (WMAC) employs random access for transmitting short message such as station's registration and bandwidth requests [4]. The use of random access is the most appropriate for such a purpose but has an inherent drawback, that is, when two or more stations happen to transmit at the same time, then a collision occurs. Since collision resolution consumes resources, the throughput of the random access system decreases. In addition, the registration delay and the bandwidth request delay also increase. Hence, a collision resolution algorithm plays a key role in the random access system and constitutes one of the major differences among various contention/reservation based MAC protocols.

Split Algorithm (SA) is one of the most prominent solutions that have been proposed recently [1]. The idea behind the SA algorithm for collision resolution is to divide collided users into Q sub-groups (where the variable Q may range from two to any required/reasonable integer value) and process each sub-group recursively [1]. SA is applicable to the MAC protocols based on TDMA and multi-channel MAC protocols because each group split by the SA is processed independently. [1-4] Performance of the SA depends on the initial group size, that is, the number of colliding packets that should be resolved. In this paper, we present an SMA, which

keeps the optimum value of initial group size adaptively to traffic load using the access status information (collision, success or empty) on the previous frame. We show that theoretical results are agreed with simulation results and the SMA operates adaptively to traffic load on contention time slots.

The related works and research background are explained in Section II. Section III describes the SMA, and section IV presents performance evaluation. Finally, conclusions are included in Section V.

2 Related Works and Research Backgrounds

The key idea of SA is to divide collided users into Q sub-groups. Each station involved in collision throws a die, which has Q faces. The stations flipping 1 will transmit first. Those flipping 2 will transmit in the next slot after the collision (if any) once those who flipped 1 have been resolved. Since this process of splitting collided users can be easily visualized through a Q-ary tree (especially the blocked access), or stack (especially the free access), these operations are also identified by names. All new calls arriving during the Collision Resolution Period (CRP) are blocked and then deferred to the next contention period. Fig. 1 represents the logical tree structure of SA algorithm when Q=2. In this figure, S_1 , S_2 and S_3 mean the initial groups and are assumed to have been split due to collisions.

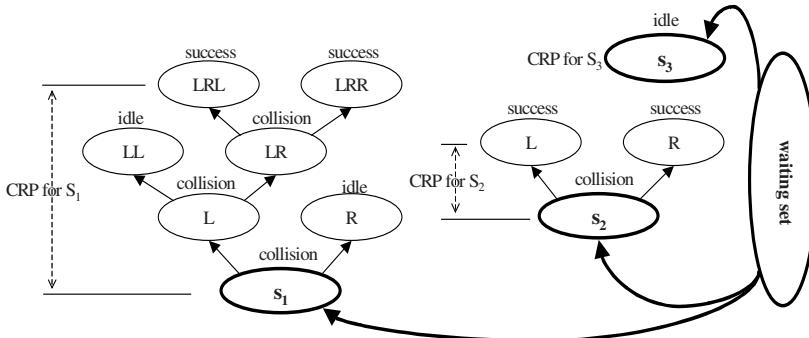


Fig. 1. Logical tree structure for collision resolution phase of SA

Let k be the number of colliding stations, and n be the allowable maximum depth of the split step. Then, in the case of ($k \geq 2$), the probability, denoted by $Q(n, k, k_l)$, that k_l terminals choose the left subset and the remaining $k_r = k - k_l$ terminals choose the right subset is:

$$Q(n, k, k_l) = \frac{\binom{2^{n-1}}{k_l} \binom{2^{n-1}}{k - k_l}}{\binom{2^n}{k}} \quad (1)$$

The probability that l periods are required for successful transmission of a packet is:

$$p(n, k, l) = \frac{1}{k} \sum_{i=0}^k Q(n, k, i) (i \cdot p(n-1, i, l-1) + (k-i) \cdot p(n-1, k-i, l-1)) \quad (2)$$

The size k of initial group, $0 \leq k \leq 2^n$, in the collision resolution phase follows a binomial random distribution with a mean value of \bar{k} .

$$p_b(n, k, \bar{k}) = \binom{2^n}{k} \left(\frac{\bar{k}}{2^n} \right)^k \left(1 - \frac{\bar{k}}{2^n} \right)^{2^n - k} \quad (3)$$

The average length for resolving k collisions \bar{l}_s is computed by (4).

$$\bar{l}_s(n, k) = \sum_{l=0}^{n+1} l \cdot p(n, k, l) \quad (4)$$

The average delay of the SA is the same as the length of CRP. Therefore, using these equations, the normalized average delay $\hat{\tau}_{SA}$ can be obtained by (5).

$$\hat{\tau}_{SA}(n, \bar{k}) = \frac{1}{\bar{k}} \sum_{k=0}^{2^n} (p_b(n, k, \bar{k}) \cdot k \cdot \bar{l}_s(n, k)) \quad (5)$$

Fig. 2 depicts $\hat{\tau}_{SA}$ versus \bar{k} .

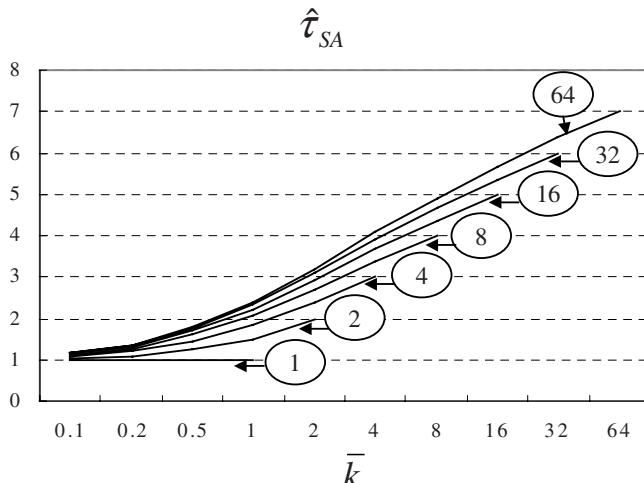


Fig. 2. Normalized average delay of SA

The number in each circle means the number of stations, and the offered load during CRP is denoted by \bar{k} . Therefore, when there are 64 stations and $\bar{k} = 64$, λ_s becomes 1, and so the average delay will be 7 frame times. In other words, 64 packets can be transmitted successfully in 7 frame times on the average. The average throughput over this period will be 1/2 because the number of collisions during this period is $1+2+4+\dots+16+32 = 63$. In a similar way, we can predict that the average throughput is 2/3 when $\bar{k} = 2$ with two stations. This result indicates that we need to reduce the initial group size (\bar{k}) because the performance is degraded when many stations are processed in one group. In fact, performance for 32 mini-groups (each group has 2 stations and $\bar{k} = 2$) is better than that for one group (one group has 64 stations and $\bar{k} = 64$) in terms of time slot consumption (127 for the former, 96 for the latter) and the average delay. However, the base station does not know the value of \bar{k} , so we need to predict them. Let λ_s denote the traffic load imposed to a time slot. The value of \bar{k} is dependent on λ_s . The small value of λ_s indicates underutilization of channel, and the large value of λ_s means the high traffic load. In particular, when the value of λ_s exceeds 1, there will be a collision.

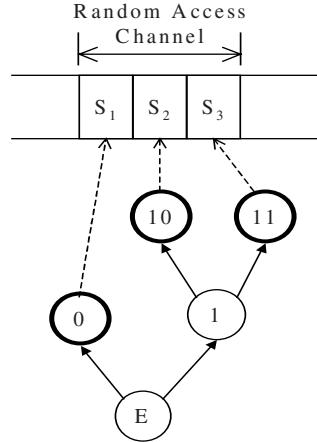
In this paper, we propose SMA algorithm, which tries to keep the value of λ_s as close as to 1 on each time slot. In the SMA, the status of each time slot is classified into one of empty, success or collision, depending on the value of λ_s . Empty state means that $\lambda_s = 0$, that is, no terminals access to the time slot. In this state, merge operation is performed to create a larger contending group in the next frame, which will increase λ_s as a result. Success state means that λ_s is close to 1, and the SMA keeps its current state using the stay operation. In case of collision, split operation is carried out to reduce the λ_s of the time slot in the next frame.

3 SMA Algorithm

The base station performs the SMA algorithm which consists of split, stay and merge operations using the value of λ_s which can be guessed by the uplink access information, and, the base station broadcasts GAP (Group Access Permission) for each time slot in random access channel.

3.1 Group Access Permissions (GAP)

GAP is an identifier of split group. The length of this identifier, denoted by l_{GAP} , is smaller than or equal to $\log n$, where n is the number of terminals. Each terminal makes decision for channel access using its own identifier and the GAP of each time slot. Fig. 3 describes an example of GAP. In this figure, GAP is represented by the sequence of binary numbers in each circle.

**Fig. 3.** Example of the GAP

In this example, GAP for time slot S_1 , S_2 and S_3 is 0, 10 and 11, respectively. Each terminal compares GAP with the first l_{GAP} bits of its own identifier and accesses the time slot with the correspondent value.

3.2 Split, Stay, and Merge Operation

Split, stay and merge operations are performed using the uplink access information in the contention time slot. Three operations are illustrated in the form of procedures. In the procedures, each parameter has IN (pass by value) and OUT (pass by reference) derivatives. Split operation is described in Fig. 4.

```
operation split (IN  $GAP_o$ , IN  $l_{GAPo}$ , OUT  $l_{GAPn}$ , OUT  $GAP_L$ , OUT  $GAP_R$ )
   $GAP_L := (GAP_o \ll 1) + 0$ 
   $GAP_R := (GAP_o \ll 1) + 1$ 
   $l_{GAPn} = l_{GAPo} + 1$ 
```

Fig. 4. Split operation

When there occurs a collision on a contention time slot, terminals accessed the time slot will be split into two groups, and each group will access the different time slot in the next frame. Two split groups need GAP (GAP_L and GAP_R) for two time slots. For this, GAP_L and GAP_R are computed at the base station, and the result is broadcasted to all terminals. In Fig. 4, l_{GAPo} means the length of GAP, and ' \ll ' indicates bit shift operator.

```

operation merge (IN  $GAP_L$ , OUT  $GAP_R$ , IN  $l_{GAP_0}$ , OUT  $l_{GAP_N}$ , OUT  $GAP_N$ )
  if ( $GAP_L = GAP_R$ )
     $GAP_N := GAP_L \gg 1$ 
     $l_{GAP_N} = l_{GAP_0} - 1$ 
    return true
  return false

```

Fig. 5. Merge operation

Fig. 5 describes merge operation. Merge operation is used for merging two groups that accesses no time slot in the previous frame. In this case, GAPs of two time slots (GAP_L and GAP_R) must be assigned a new value, denoted by GAP_N , for the merged group. If the base station can not merge two groups because of the different values of GAP_L and GAP_R , a false value is returned.

```

operation stay (IN  $GAP_o$ , OUT  $GAP_N$ , IN  $l_{GAP_0}$ , OUT  $l_{GAP_N}$ )
   $GAP_N := GAP_o$ 
   $l_{GAP_N} := l_{GAP_0}$ 

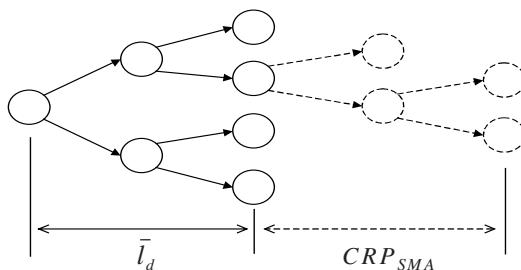
```

Fig. 6. Stay operation

Fig. 6 describes stay operation. Stay operation is performed when the access to the previous time slot has been successful.

4 Performance Evaluations

We evaluate our algorithm via a simulation. The delay of SMA was considered in steady split state. Once the SMA enters the split state, it remains there even though the collision is resolved. Intuitively, since the SMA begins collision resolution phase in the steady split state, the CRP becomes shorter comparing with SA, so the average delay could be reduced. Specifically, in SA, the initial set of CRP begins from n for the collision resolution. On the other hand, in SMA, the collision resolution begins from $n - \text{the depth of steady split state}$. Fig. 7 illustrates CRP of SMA at a steady state.

**Fig. 7.** CRP of SMA at a steady state

The average split depth, denoted by \bar{l}_d , is defined as the depth at which the SMA enters the steady state.

\bar{l}_d can be obtained by (6).

$$\bar{l}_d(n, \bar{k}) = \sum_{l=0}^{n+1} l \cdot p_c(n, \bar{k}, l) \quad (6)$$

Let $p_c(n, \bar{k}, l)$ be the state probability at each depth. Then, it can be obtained by the following state transition diagram.

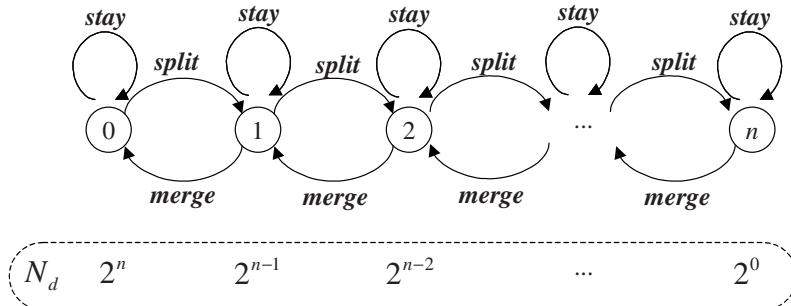


Fig. 8. State transition diagram

The number of the terminals that could transmit in each state, denoted by N_d , are reduced by 1/2 in each step. So, the collision probabilities in each state can be obtained by (7).

$$p_{\text{split}}(N_d, \bar{k}) = \sum_{i=2}^{N_d} \binom{N_d}{i} \left(\frac{\bar{k}}{2^n}\right)^i \left(1 - \frac{\bar{k}}{2^n}\right)^{N_d-i} \quad (7)$$

The merge probabilities and the stay probabilities in each state can be obtained by (8) and (9), respectively.

$$p_{\text{merge}}(N_d, \bar{k}) = 2 \cdot \left(\binom{N_d}{0} \left(\frac{\bar{k}}{2^n}\right)^0 \left(1 - \frac{\bar{k}}{2^n}\right)^{N_d} \right) \quad (8)$$

$$p_{\text{stay}}(N_d, \bar{k}) = 1 - (p_{\text{split}}(N_d, \bar{k}) + p_{\text{merge}}(N_d, \bar{k})) \quad (9)$$

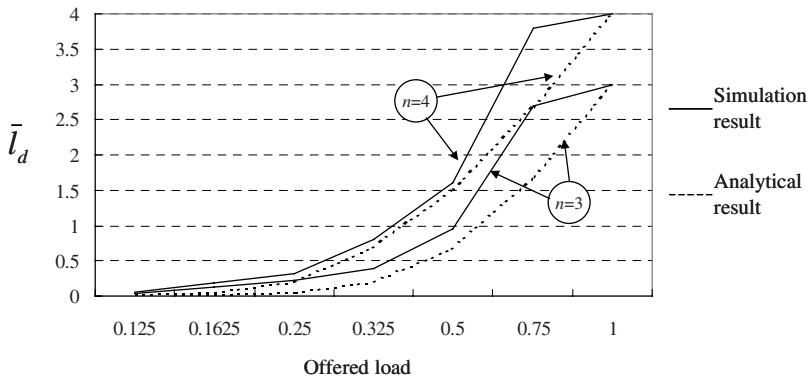


Fig. 9. Average split depth

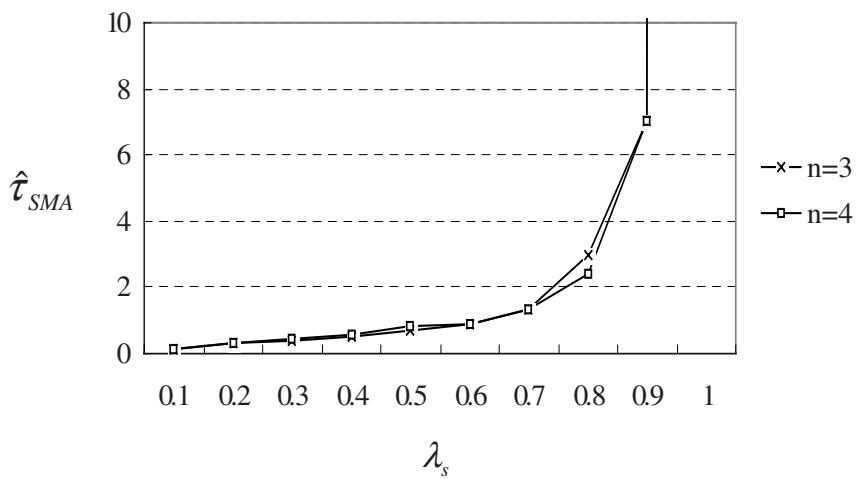


Fig. 10. Delay of SMA

Fig. 9 depicts the theoretical and simulation values of \bar{l}_d versus the offered load. We can see the simulation result is in a sound agreement with the analytical model.

Fig. 10 illustrates the simulation results of SMA delay versus λ_s . In this figure, the delay of SMA when $n = 3$ is almost the same with the one when $n = 4$. It indicates that the delay of SMA is independent on the value of n . It is only λ_s that affects the delay of SMA because the SMA starts collision resolution at a steady split state.

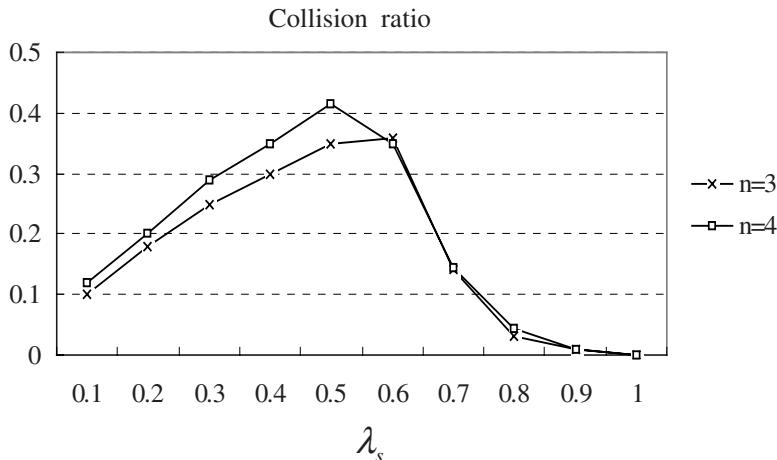
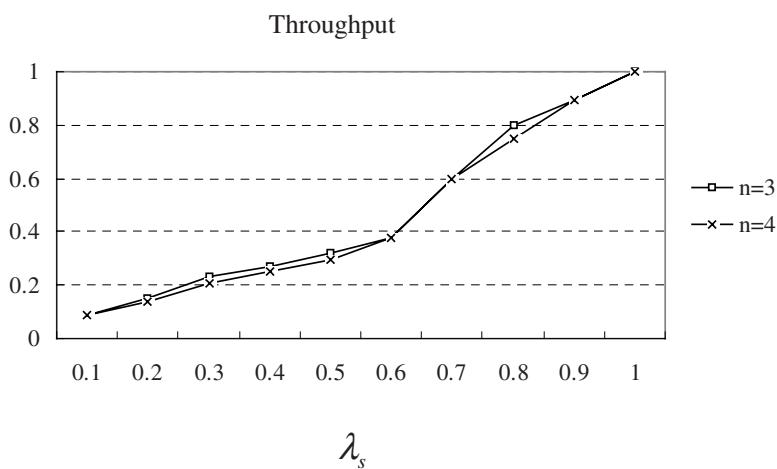
**Fig. 11.** Collision ratio of SMA**Fig. 12.** Throughput of SMA

Fig. 11 illustrates the collision ratio of the SMA. In this figure, the collision ratio of the SMA is also affected by λ_s , not by n . The collision ratio of the SMA begins to decrease when the value of λ_s goes over 0.5, it reflects that the average split depth becomes longer.

Fig. 12 illustrates the throughput of SMA. We can see that the throughput of SMA continuously increases because the collision ratio of the SMA decreases as the traffic load increases.

5 Conclusions

The SMA enhances the performance of SA using split, merge and stay operations for keeping the appropriate initial group size. The SMA utilizes the previous time slot access information (COLLISION, SUCCESS and EMPTY) to predict the offered load on a time slot. The SMA performs a split operation when a collision occurs and a merge operation when the split subgroups remain idle. Through the analysis, we find out the dependency of initial group size on the performance of SA, and proposed a strategy to reduce the initial group size. The simulation result indicates that predicted results of SMA sound agree with analysis and the performance of SMA is independent of the number of stations.

Reference

- [1] D. PETRAS, KRÄMLING.: “Fast Collision Resolution in Wireless ATM Networks”, in 2nd MATHOD, 1997.
- [2] VERIKOUKIS, CH.V., OLMOS, J.J.: “Up-link performance of the DQRUMA MAC protocol in a realistic indoor environment for W-ATM networks”, Communications and Vehicular Technology, 2000, 2000, 4, pp. 1650–1655
- [3] JEYHAN KARAOGUZ.: “High-Rate Wireless Personal Area Networks,” IEEE Communications Magazine, 2001, 39 (12), pp.96–102
- [4] FLIKKEMA, P.G.: “On multiple access and capacity in frequency-selective channels’, Information Technology: Coding and Computing, 2001. Proceedings. International Conference on, 2001, pp. 178–182

Part IV

Applied Technologies

Reconfigurable Cipher Processing Framework and Implementation

Jingfei Jiang, Xiaoqiang Ni, and Minxuan Zhang

School of Computer,
National Univ. of Defense Tech.
Changsha, Hunan, P. R. China
jingfeijiang@yahoo.com.cn

Abstract. Cryptographic algorithms are computing-intensive. Reconfiguration techniques can change hardware by software and implement multi-grain parallelism to adapt different applications. By studying many cryptographic algorithms, Reconfigurable Cipher Processing Framework (RCPF) is proposed using the method of induction and then the values of framework parameters for RCPF is achieved. According to the framework, a reconfigurable, pipelined architecture RCPFI-g for cipher processing is implemented on Xilinx Virtex II device. 17 cryptographic algorithms have been mapped on the RCPFI-g and simulated. The simulation results show the performance and the flexibility of the implementation. The results prove the framework RCPF and the implemented architecture RCPFI-g is efficient for cipher processing.

1 Introduction

Reconfiguration techniques can change hardware by software and implement multi-grain parallelism to adapt different applications. The silicon after manufacture can vary on user's requirements. It has the flexibility of general-purpose processors and the speed of ASIC. Such circuit is more suitable for special algorithms and has demonstrated significant performance gains in several classes of applications.

Reconfigurable architecture has lots of computing units to accomplish parallel processing. For simple, regular and computing-intensive applications that need to process large amounts of data but have small data width, such as signal and image processing, ATR, cipher processing, video and audio compressing etc., reconfigurable architecture has significant speedup. Most modern security protocols are algorithm-independent and the protocol standards support a variety of algorithms. Crypto standards are updating fast. The key length and computing complexity of cryptographic algorithms are increasing greatly. Two departments who communicate secretly may change the cryptographic algorithms and the keys at times to enhance security. Algorithms' implementations must respond to these demands. Reconfigurable computing offers high performance yet flexible solutions for cryptographic algorithms that have similar architectures but different functions.

Block cipher, hashing and stream cipher are widely used in cryptography. These algorithms usually have regular data widths, simple operation sequences and little branches. The operation set these algorithms refer is small. Coarse-grain

reconfigurable architecture can respond to these characteristics, which reduce the amount of reconfiguration data, the synthesis time and the complexity of mapping algorithms. The operation sequences of these algorithms must be processed for many rounds. Read After Write data dependences exist among adjacent operations and rounds. Pipelined architecture with proper parallelism is feasible to implement them.

The Reconfigurable Cipher Processing Framework (RCPF) is proposed in this paper. Its parameters generalize the reconfigurable granularity, the operation set of unit, the parallelism and the amount of buffers. The template technique to map algorithms and segment dynamic reconfiguration mode to reconfigure algorithms is used. Based on RCPF, the Reconfigurable Cipher Processing Framework Implementation with granularity g (RCPFI- g) is designed and many cryptographic algorithms have been mapped on it.

2 Related Works

The main idea of creating RCPF is induction. Firstly, many applications in cryptographic field are analyzed. Secondly, the common characteristics of these applications are summarized. At last, the conclusion is made about how the architecture is organized to process these applications efficiently. Katherine Compton et al [4] have studied architecture of one dimension data path to process computing-intensive tasks such as DSP, scientific computing, image processing based on RaPiD. Their architecture generator customized a data path of one dimension automatically based on the netlist RaPiD compiled. The generator accelerates the design of reconfiguration architecture.

Pipelined structure is commonly used when implementing cryptographic algorithms. It is also reviewed intensively in the field of reconfiguration computing. The pipelined structure of PipeRenCh [5] [6] is suitable for multimedia applications. To use hardware resources efficiently, the concept “virtual hardware” is proposed in PipeRenCh design. It is the combination of dynamic reconfiguration method and modularization. The reconfigurable units each of which has a granularity of 2 in Garp [7] are organized in rows. Many rows of units form fine-grain reconfigurable architecture. Kiran Bondalapati et al [8] offered mapping techniques of loop pipeling and optimization on FPGA.

3 Reconfigurable Cipher Processing Framework (RCPF)

3.1 Parameters of RCPF

Following the pipelining characteristic of cipher processing, RCPF has pipelined structure of two dimensions. Plaintexts enter the pipelined structure in sequence. One or more plaintexts are processed in parallel in one pipelined segment. RCPF has a lot of pipelined segments. Each segment has two pipelined stages. The first stage is the interconnection unit. The second has some basic processing units. The basic processing unit has fixed granularity. It can execute basic operations. The parameters for RCPF are defined as follow:

g : Basic granularity. It represents the data width of basic processing unit and interconnection lane.

PU : Processing unit. Its property is $P_{PU} = \{F_o, F_i, \dots\}$, F_i is some operation with granularity g .

S : The amount of buffers. It indicates one PU must buffer at least S data. That is, one PU must offer S outputs, each with granularity of g . all PU s' S are assumed identical.

P : Parallelism of one pipelined segment. It indicates one pipelined segment can process P data in parallel, each with granularity of g .

3.2 Induction Method of RCPF

The values of the parameters for RCPF are induced from the Task Directed Acyclic Graph (TDAG) of some example algorithms. The TDAG of algorithm A is defined as node set:

$$T^{(A)} = \{T_0^{(A)}, T_1^{(A)}, \dots\}, \quad T_i^{(A)} = (N_{Ti}^{(A)}, Ty_{Ti}^{(A)}, I_{Ti}^{(A)}, O_{Ti}^{(A)});$$

Items in $T_i^{(A)}$ represent the number of task, task type, task input set and task output set respectively. $I_{Ti}^{(A)} = \{i_0^{(A)}, i_1^{(A)}, \dots\}$, items in it are $i_q^{(A)} = (N, g, t, f)$. items in $i_q^{(A)}$ represent the number of input, the granularity of input, the time when the input is used in TDAG and the frequency of input respectively ($f=0$ means the input is fixed during algorithm execution; $f=1$ means the input is fixed during round execution; $f=2$ means the input is fixed during one segment execution). The value t in $i_q^{(A)}$ is an integer. Its minimum is 0. Its value depends on its task location in TDAG. Different input may have identical number. The numbers of initial inputs are tagged specially. The final outputs are nodes in TDAG. Inputs and outputs that have identical number imply they are joined.

The induction method of RCPF is described below. The symbol “ A ” represents arbitrary algorithm.

1) Induce g

$$I^{(A)} = \{\forall i, q, T_i^{(A)}, I_{Ti}^{(A)}, i_q^{(A)}\},$$

$$\forall i \in I^{(A)}, g^{(A)} = \max f(i_q^{(A)}), \quad g = \min(g^{(A)})$$

// maxf can find the item value which appears most frequently.

2) Educe PDAG

The oPeration Directed Acyclic Graph (PDAG) is educed from TDAG using template matching technique according to g . The template form is shown below.

if $Ty_{Ti} = shif_{ng}$, *then* $T_i = \{Ts_{i0}, Ts_{i1}, \dots, Ts_{i(n-1)}\}$, Ts . $Ty = [AND \bullet shif \bullet or]$

// Symbol “ \bullet ” indicates the join of operations;

if $Ty_{Ti} = perm_{ng}$, *then* $T_i = \{Tp_{i0}, Tp_{i1}, \dots, Tp_{i(n^2-1)}\}$,

Tp . $Ty = [AND \bullet permu \bullet or]$

// Figure 1 shows the first part of the template "permutation";

if $Ty_{Ti} = add_{ng}$, then $T_i = \{Ta_{i0}, Ta_{i1}, \dots, Ta_{i(n-1)}\}$, $Ta. Ty = [add \bullet (+carry)]$;
 if $Ty_{Ti} = sub_{ng}$, then $T_i = \{Tb_{i0}, Tb_{i1}, \dots, Tb_{i(n-1)}\}$, $Tb. Ty = [sub \bullet (-carry)]$;
 if $Ty_{Ti} = multi_{ng}$, then $T_i = \{Tm_{i0}, Tm_{i1}, \dots, Tm_{i(n^2 + \frac{n}{2}(n-1)-1)}\}$,
 $Tm. Ty = [multi] or [add] or [add \bullet (+carry)]$;
 else $T_i = \{T_{i0}, T_{i1}, \dots, T_{i(n-1)}\}$;

TDAG is transformed into PDAG after template matching. In PDAG, inputs and outputs are transformed according to g and their operations during template matching. All inputs and outputs of the operations have the same granularity.

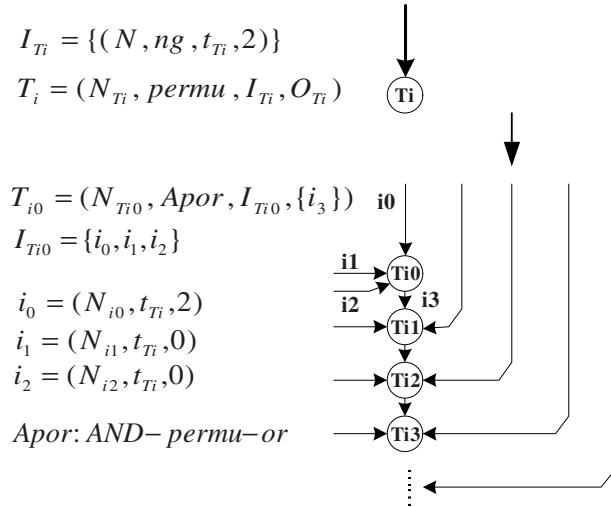


Fig. 1. First part of the template “permutation”

PDAG of algorithm A is defined as node set, the items’ meaning are similar to TDAG:

$$P^{(A)} = \{P_0^{(A)}, P_1^{(A)}, \dots\}, \quad P_i^{(A)} = (N_{Pi}^{(A)}, Ty_{Pi}^{(A)}, I_{Pi}^{(A)}, O_{Pi}^{(A)}), \\ I_{Pi}^{(A)} = \{i_0^{(A)}, i_1^{(A)}, \dots\}, \quad i_q^{(A)} = (N, t, f);$$

3) Decide according to PDAG

$$P_{PU} = \{\forall A, \forall i, P_i^{(A)}. Ty_{Pi}^{(A)}\};$$

4) Decide P

The P of A is decided by the number of initial inputs which f is equal to 2. The inputs that have identical number are treated as one input. $\bigcup i_q$ denotes counting the number of inputs which accord with the condition.

$P^{(A)} = \{\bigcup i_q \mid \forall j, q, I_{Pj}^{(A)}. i_q^{(A)}. N \text{ is tagged as initial input,}$
 $\text{and } I_{Pj}^{(A)}. i_q^{(A)}. f = 2\},$

$P = \max(P^{(A)});$

5) Induce S

$I^{(A)}_{tmp} = \{\forall j, \forall q, I_{Pj}^{(A)}. i_q^{(A)} \mid I_{Pj}^{(A)}. i_q^{(A)}. f = 2\} // \text{the set of inputs which } f=2;$

$I^{(A)} = \{i_p^{(A)} \mid \forall q, i_p^{(A)}, i_q^{(A)} \in I^{(A)}_{tmp}, i_p^{(A)} \neq i_q^{(A)}\} // \text{get rid of the repeated items;}$

$\forall q, i_q^{(A)}. N \text{ is tagged as initial inputs,}$

if $(r = \min(i_q^{(A)}. t)) > 0$, *then*

$$I^{(A)} = I^{(A)} + \{i_w^{(A)}(i_q^{(A)}. N, 0, 2)\} + \{i_{w+1}^{(A)}(i_q^{(A)}. N, 1, 2)\} + \\ \dots + \{i_{w+r-1}^{(A)}(i_q^{(A)}. N, r-1, 2)\};$$

//if the minimum of t of the input i_q is greater than 0, create r new inputs, w is the new suffix of the inputs;

for $t = 0, 1, \dots, [\forall q, \max(i_q^{(A)}. t)] // \text{maximum } t \text{ of all inputs.}$

$$S_t^{(A)} = \sum i^{(A)}, i^{(A)} \in I^{(A)}, i^{(A)}. t = t;$$

$$S^{(A)} = \max(S_t^{(A)}) / P_A;$$

$$S = \max(S^{(A)});$$

3.3 Parameters' Value for RCPF

Some example algorithms have been used to induce the values of the parameters, such as DES, IDEA, Blowfish, SERPENT, Rijndael etc. The values of parameters for RCPF are:

$g=8; P_{pu} = \{\text{add, sub, multiply, XOR, OR, AND, shift}\}; P=16; S=3.$

The example algorithms represent different kinds of cryptographic algorithms. Some of them focus on substitution of S box and bit-level permutation (DES, Rijndael). Some focus on arithmetic functions (IDEA). Others focus on logic functions (SERPENT). The parameters imply that RCPF is composed of a lot of pipelined segments. Each segment is composed of an interconnection unit and 16 processing units, each unit's granularity is 8. The processing units of adjacent segments can transfer three data, each with granularity of 8.

4 Design and Implementation of RCPFI-g

4.1 Hierarchical Interconnection Architecture

Multi-grain matching is one of the primary demands for reconfigurable computing. It has two keys:

1. Basic processing units can be combined to form larger-grain function unit efficiently (e.g. 2g, 3g, 4g etc.);

2. Multi-grain alignment can be achieved by interconnection (8, 16, 32 and 64).

Based on key 1 and cost-efficiency tradeoff, every 4 adjacent processing units within one pipelined segment are grouped. Function optimization is made in a group (e.g. lookahead carry chain [9] and dynamic shift control), so the units in a group can be combined to form function units of granularity g-4g very efficiently.

The delay and complexity of interconnection between adjacent segments are commonly very high. The interconnection in RCPFI-g can transfer S ($=3$) outputs from the processing unit in the segment above to the processing unit in the segment below. To decrease interconnection cost and satisfy key 2 at the same time, the RCPFI-g defines that one output can go into multiple units in the segment below according to the principle of granularity alignment. The other two can only connect with their corresponding unit in the segment below. The interconnection architecture is described in detail as follow:

Assume: PU_{i,p,j} is j_{th}unit of p_{th}group in i_{th}segment, p, j ∈ {0,1,2,3},

PU_{i,p,j}.a, PU_{i,p,j}.b, PU_{i,p,j}.c represent 3 outputs of j_{th}unit,

Then: the input of j_{th}unit of q_{th}group in (i+1)_{th}segment is:

$PU_{i+1,q,j} \cdot In = [PU_{i,q,j}.a, PU_{i,q,(j+1) \bmod 4}.a, PU_{i,q,(j+2) \bmod 4}.a, PU_{i,q,(j+3) \bmod 4}.a,$

$PU_{i,(q+2) \bmod 4,j}.a, PU_{i,(q+1) \bmod 2+2(q \bmod 2),j}.a, PU_{i,q,j}.b, PU_{i,q,j}.c];$

4.2 Processing Unit

Figure 2 shows the processing unit structure of RCPFI-g. Each segment is composed of two pipelined stages. The first stage routes the outputs from above (Input_mux), splits/joins the data (AND&Shift) and multiplexes data to next stage (AND&Shift). AND&Shift can convert data of which granularity is equal to or less than g into g-grain data. The inputs of this stage can be changed into new operands through conversion and enter the stage registers. Some of them can also enter the stage registers directly. The second stage takes arithmetic and logic operations (ALU). ALU functions include addition, subtraction, multiply, XOR, AND, OR and pass through. The ALU results with other data (data from memory etc.) are multiplexed (ALUO_mux) to form 3 outputs. These outputs pass through the inter-stage registers to next stage.

4.3 Global RAM

Global RAM is the main component to support substitution of S box. To lookup S box in parallel, all units in one segment should access global RAM at the same time. That is, the access requests from each segment are P ($=16$). For $g=8$, the memory capacity for every access request is $2^8 \times g$ (256B). The capacity of global RAM is $256B \times P = 4KB$. Global RAM is divided into P blocks. $PU_{i,P,j}$ ($i=1 \sim \text{segment amount}$) share one 256B RAM. To support simultaneous access from different segments,

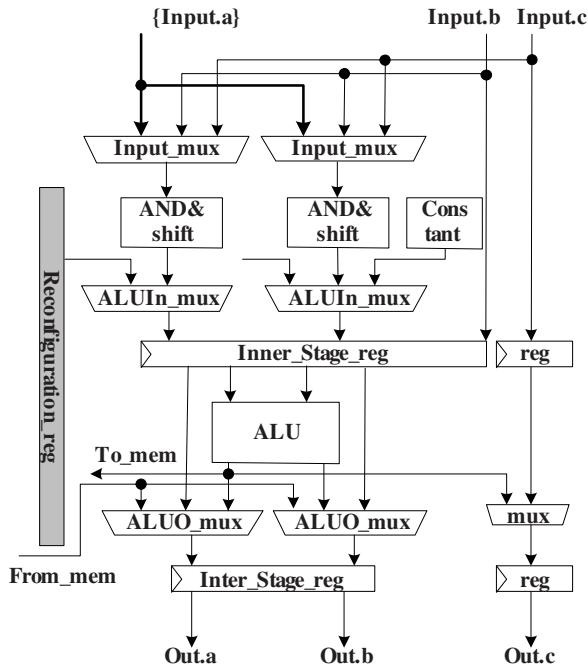


Fig. 2. Processing unit architecture

multi-ports RAM is used. Address multiplexers and configuration control do the selection of addresses from multiple segments.

For multi-grain access, global RAM use inner address multiplexers to support four combinations of access requests: 8 bits address in, 8 bits data out; 8 bits address in, 16 bits data out; 8 bits address in, 24 bits data out; 8 bits address in, 32 bits data out.

4.4 Reconfiguration Mode

Most of cryptographic algorithms have many rounds. Some of the configuration data in these rounds are algorithm-level fixed. Some of them are round-level fixed. Not all the operations can be mapped onto the device entirely due to the lack of capacity of device. RCPFI-g employs mixed reconfiguration modes to map some part of rounds every time on the device. For algorithm-level fixed reconfiguration data (e.g. S Box, algorithm constant), RCPFI-g employs component level reconfiguration mode. For round-level fixed reconfiguration data (e.g. round function), RCPFI-g employs segment dynamic reconfiguration mode.

Each processing unit of RCPFI-g requires 8 bytes reconfiguration data. One stage requires 4 bytes. RCPFI-g uses 3 ways to decrease reconfiguration time and match it with execution time, as described below.

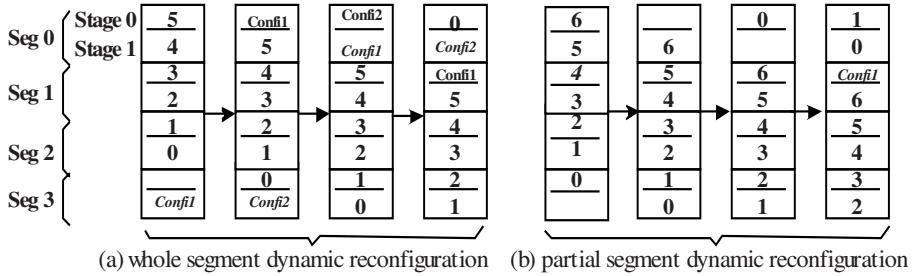


Fig. 3. Two conditions of segment dynamic reconfiguration mode (4-segments RCPFI-g for example)

1) Reusing reconfiguration registers: the reconfiguration registers in one processing unit can be used for two kinds of functions that do not operate at the same time (e.g. one register is shared by constant reconfiguration and spilt function reconfiguration; another register is shared by permutation reconfiguration and ALU type reconfiguration).

2) Decreasing reconfiguration cycle: the delay of reconfiguration is small. RCPFI-g has reconfiguration cycle that is one-half of the execution cycle.

3) Increasing reconfiguration bandwidth: Each column of processing units ($PU_{i,P,j}, (i = 1 \sim \text{segment amount})$) correspond to two initial reconfiguration ports, each with granularity g . They can reconfigure 4 bytes of arbitrary $PU_{i,P,j}$ and $PU_{k,P,j}, (k \neq i)$ in two reconfiguration cycles.

There are two conditions in segment dynamic reconfiguration mode of RCPFI-g. For algorithms of which not all operations in a round can map into the device at a time, whole segment dynamic reconfiguration is used. In this condition, every segment is reconfigured during execution. For algorithms of which one or more rounds can map onto the device entirely, partial segment dynamic reconfiguration is used. In this condition, only special reconfiguration registers (e.g. constant) need to be changed during execution. An example is given as Figure 3. Partial segment dynamic reconfiguration achieved the same performance as whole segment dynamic reconfiguration, but the amount of reconfiguration data decrease greatly.

4.5 Implementation

RCPFI-g is implemented in Verilog. It is synthesized, placed and routed using XilinxISE5.1 tools on Virtex II devices. ModelSim of Mentor Inc. is used for simulation. RCPFI-g has 22 segments. The outputs of the last segment turn around and connect to the inputs of the first segment through multiplexers. Thus, data can circle through RCPFI-g to process the whole algorithm. One unit of RCPFI-g takes about 200 CLB slices; each group of units with group optimization logic takes 820 CLB slices. The levels of logic in key data path is 9; the key data path delay is 9.79ns; the clock period is 102MHz.

5 Algorithm Mapping

5.1 Mapping Techniques

The main idea of algorithm mapping is mapping the PDAG of algorithm to RCPF. In RCPF, the outputs of the segment above cannot connect to arbitrary inputs of the segment below due to the lack of interconnection. They must be aligned according to their granularities. Algorithms mapping is based on the hierarchical interconnection restriction. The main techniques are:

- 1) Key data path finding and mapping. The PDAG of an algorithm can be created according to the second step in section 3.2. One or more key data paths will be found in PDAG by retrospecting from the outputs. Then the operations on the key data paths are mapped first (e.g. In DES, retrospection from the left ciphertext can get its key data path).
 - 2) If there are more than one key data paths in an algorithm, P columns of RCPF were divided into groups. Paths are mapped in different groups (e.g. In IDEA, we divide RCPF into 4 groups. Each group is composed of adjacent columns and two modular multiplies are mapped into two groups respectively).
 - 3) Operations mapping in PDAG use mapping templates which are created by hand. Mapping templates indicate the amount of units, the amount of segments, locations and the interconnection among units the operation used (e.g. In Twofish, the matrix multiply in G function uses 20 units in 3 segments in total).
 - 4) For operations that are not in the key data paths, their mapping locations are decided by their relationship to the operations in the key data path through retrospection from outputs to inputs. When there are not enough resources to map the operations, additional segments are added.
 - 5) Assume: one operation A has mapped to the RCPF and another operation B needs A's output, but A's outputs can not go directly to B's inputs due to interconnection constraints. If there are empty units in A's segment and its outputs can go directly to B, then operation A can be copied to the empty units to satisfy B's inputs requirement.
 - 6) Splitting and joining of data proceeds on the first stage of a segment, while accessing and ALU operations proceed on the second stage. Thus, split and join operations can combine with other operations in one segment.
 - 7) When memory address is not power of 2 (e.g. In MARS, the address width is 9), it can split into several sub-addresses that are power of 2. Then the memory outputs can compound final access output through logic and arithmetic operations (e.g. In MARS, the lower 8 bits of 9-bits address is treated as two 8-bits addresses and access two blocks of global RAM. The two access results multiply the 9th bit of the address respectively and then add together to form the final result).
- Reconfiguration mode is decided after PDAG has been mapped. Then, Reconfiguration Generating and Input Organizing tool is used to organize reconfiguration data and plaintexts automatically. Finally, RCPFI-g is configured and algorithms can be executed on it.

5.2 Mapping Results

17 cryptographic algorithms [1] [2] [3] have been mapped on RCPF using the mapping techniques mentioned in the previous section. The resources each algorithm used are listed in table 1. The estimated performances of RCPFI-g implementations on XCV8000 device are listed on the last column of table 1.

When mapping Twofish, the computation of function q is transformed into a substitution operation of S box that is 8-bits address in, 8-bits data out, thus saved about 5 segments. The throughputs of all algorithms implemented in RCPFI-g are much greater than software implementation.

RCPF uses hierarchical interconnection architecture rather than crossbar that is mentioned in [10]. It uses common multiply logic in processing unit. More algorithms on RCPFI-g have mapped than those mentioned in PipeRench [10]. It can be seen from table 1 that most algorithms archived the same or more speedup mentioned in PipeRench [10]. If RCPFI-g is manufactured in ASIC, its clock frequency and throughput would be improved further.

Table 1. Mapping results and performance

Algorithm	Segments/ round	Memory/ round	Groups/ round	Rounds	Total segments	Throughput (MB/s)
DES	8	512B	2	16	128	256.3
IDEA	23	/	2	8	184	171.1
NSSU	5	128B	2	32	160	192.4
Blowfish	8	4KB	2	16	128	256.3
TEA	8	/	2	32	256	128.4
SAFER	7	2KB	2	10	70	381.3
FEAL8	6	/	1	8	48	1018.7
RC6	10	/	4	20	200	154.0
SERPENT	8	4KB	4	32	256	128.4
Twofish	15	8KB	4	16	240	140.1
MARS	10	2KB	4	16	160	192.4
Rijndael	6	4KB	4	14	84	383.1
MD5	6	/	4	64	384	107.1
SHA-1	5	/	4	80	400	101.5
RC4	6	3KB	1	1	/	/
SEAL	4	2KB	4	8	/	/
WAKE	13	4KB	4	1	/	/

6 Conclusion

RCPF is induced from some mainstream cryptographic algorithms. Its flexibility is at the cost of hardware comparing to the ASIC implementation of some fixed algorithms. The simulation result proved the high performance and the flexibility of the framework RCPF and the implemented architecture RCPFI-g. Future works will focus on automatic mapping tools and advanced templates.

References

1. Lu Kai-Cheng: Computer Cryptology: Data Secrecy and Security in Computer Network. 2nd ed. Tsinghua University Press, Beijing (1998)
2. B Schneier author, Wu Shi-Zhong et al, translator: Applied Cryptography: protocols, algorithms, and source code in C. 2nd ed China Machine Press, Beijing (2001)
3. Si Qing-Han: Cryptology and Computer Network Security. Tsinghua University Press and Guang Xi Science and Technology Press, Beijing (2001)
4. Katherine Compton, Scott Hauck: Totem: Custom Reconfigurable Array Generation. IEEE Symposium on FPGAs for Custom Computing Machines, California (2001).
5. S C Goldstein, H Scbmit, M Budiu, S Cadambi, M Moe, R R Taylor: PipeRench: a reconfigurable architecture and compiler. IEEE Computer (2000) 33(4):70–76
6. S.C. Goldstein et al. : PipeRench: A Coprocessor for Streaming Multimedia Acceleration. In: Proc. 26th Ann. Int'l Symp. Computer Architecture, IEEE CS Press, Los Alamitos, Calif. (1999) 28–39
7. T J Callaban, J R Hauser, J Wawrzynek: The Garp: architecture and C compiler. IEEE Computer (2000) 33(4):62–69
8. Kiran Bondalapati, Viktor K.Prasanna: Loop Pipelining and Optimization for Run Time Reconfiguration. In: Proceedings of 7th Reconfigurable Architectures Workshop, Cancun, Mexico (2000).
9. John F. Wakerly: Digital Design: Principles and Practices, 3rd ed. Higher Education Press, Beijing (2001)
10. R. Reed Taylor, Seth Copen Goldstein: A High-Performance Flexible Architecture for Cryptography. In: Proceedings of the Workshop on Cryptographic Hardware and Embedded Systems 1999 (CHES), Worcester, MA. (1999).

A Dynamic Reconfiguration Platform Based on Distributed Component Technology CCM

Lei Dou, Quanyuan Wu, Yan Jia, and Weihong Han

School of Computer Science,
National University of Defense Technology, Changsha,
Hunan, 410073, China
fancydou@sina.com

Abstract. Dynamic reconfiguration research aims to solve the problem of the evolution of the long running or mission critical systems at runtime. CCM, as one of the distributed component technology, makes it very easy to implement dynamic reconfiguration through its assembly, deployment and container mechanisms. Instead of designing a general dynamic reconfiguration algorithm which is really hard to be general and efficient, we have defined a group of basic dynamic reconfiguration mechanisms by extending CCM. The people familiar with the semantic of the application, will utilize these mechanisms to describe specific dynamic reconfiguration algorithm in XML file exactly. Obviously, this dynamic reconfiguration platform based on CCM provides generality and adaptability.

1 Introduction

With the development of distributed computing technology, dynamic reconfiguration is in great need to change the structure of the long-running and mission-critical distributed systems at runtime, like bank business, e-commerce solutions and telecommunication switches, at the cost as little as possible to catch up with the development of technology or to satisfy the requirement of the customer and QOS[1].

Actually, the research of the dynamic reconfiguration is started at 70s. The reconfigurable entity changes from process to object. However it's hard to implement the dynamic reconfiguration and the functionality is limited, since each part of the system is coupled closely without explicit connection and there are few mechanisms to get the inner state and to control the behaviour of the system. The distributed component technology[3,4], especially the CCM (CORBA Component Model), makes this research area active for its strong support for dynamic reconfiguration:

- Assembly Mechanism and Deployment Framework: From the view of software architecture, the application is assembled with the components through the connections according to some rules. Thus the fundamental of the dynamic reconfiguration is a process to reassemble and redeploy the application at runtime. Different from the object model, the CCM component model defines the bidirectional interfaces for services they provide and require from the other components. While deploying the assembly to the distributed environment through the CCM deployment framework, the components will be created and connected

- explicitly according to the dependency relations. Obviously, CCM provides the most natural support for the dynamic reconfiguration.
- Container Model: In component technology, containers as the running environment of the components can get the state of the components and control its behaviour easily. Benefit from the CORBA, the CCM containers based on POA do more help to the dynamic reconfiguration, such as hold the request to the components.

So we tend to implement a general dynamic reconfiguration platform based on CCM. However, the close relation between the dynamic reconfiguration and the semantic of the application leads to the great difficulty to achieve the generality which is usually got at the price of the performance and flexibility, such as the pessimistic general algorithm of Jeff[5]. Moreover, the numerous applications and various reconfigurations even make it difficult to define the consistency of the system.

Consequently, instead of providing a general algorithm and by extending the CCM, we design the reconfiguration platform that only provides the basic mechanisms for reconfiguration as enough as possible aiming at allowing the reconfigurator who is familiar with the semantic of the application to compose these mechanisms at will to design and describe his specific reconfiguration algorithm in XML file. The extended CCM platform will parse it and implement the reconfiguration. Additionally, the ORB will be extended to replace the IOR(Interoperable Object Reference) with the IOGR(Interoperable Object Group Reference) introduced in the Fault Tolerant CORBA Standard to settle the problem of different object reference while updating the component[7]. Obviously, the correctness of the algorithm is guaranteed by the reconfigurator. In the future, we will try to assist and automate the validation of the algorithm through the formal tool.

The paper is further structured as follows. Section 2 introduces the basic mechanisms. Section 3 demonstrates the reconfiguration process through a case study. Finally, section 4 presents the conclusion and future work.

2 Dynamic Reconfiguration Mechanism

According to the requirements and the issues of the dynamic reconfiguration[2], we have designed six basic mechanisms by extending the CCM.

2.1 Software Architecture Description Mechanism

Only if the components composed of the application and the connections between them are described correctly and clearly, could the reconfigurator specify their reconfiguration requirement and could the scope of the reconfiguration be defined[6]. Furthermore, we can make some reason of the performance and the validation of the system according to the software architecture description with the use of formal tool and mathematic model.

According to the structure of the distributed application based on CCM, we proposed a group of architecture description objects in three layers structure. There is one architecture object lies on the root layer providing the interfaces for the lifecycle management of the home object on the middle layer, including creation, removal and find. Same as the architecture object, the home object manages the component object

on the bottom layer. The home and component object which can take part in the connection, also provide the functionality to get the information of the connections they involved. Additionally, since the component can initiate and disconnect the connection, the component object provides the interfaces for connection creation and removal. It's the responsibility of container to call these interfaces and construct these architecture description objects while it detects creation or removal of the homes, components and connections accordingly. Obviously, we can get the system architecture information through query these objects.

2.2 Component State Detecting Mechanism

Before taking any action to reconfigure the system dynamically, we must make sure the system is safe to be reconfigured for its consistency. Obviously, the system will be in abnormal state if the connection is removed while it is in use. Combined with the condition and branch sentences, the detecting result will determine the sequence and manner of the reconfiguration. Based on CCM abstract component model, the state detecting mechanism is designed as follows:

- Judge whether the component is responding to the request through the specific interface or method
- Judge whether the component is using the specific connection
- Judge whether the component has come to the specific state indicated by the specific attribute value

2.3 Component Behaviour Control Mechanism

The dynamic reconfiguration is a sequence of the component behaviour. So the control mechanism is required undoubtedly and is designed as follows:

- Hold the specific methods, interfaces or connections to drive the components into the safe reconfiguration state.
- Unhold the specific methods, interfaces or connections to recover the systems to normal state.
- Set the attribute of the components to influence the running of the components and assist the reconfiguration.
- Control mechanisms supported by CCM originally, including creation and removal of the home, component and the connection and the configuration_complete operation that drive the components from the deployment state to the running state.

2.4 Component State Transfer Mechanism

During the replacement of the component, the state transfer between the components in different version is needed. This mechanism can be implemented through extending the activate and passivate mechanisms of the CCM for memory management originally. And the difficulty of transferring the state between components with different state representation is avoided by passing this duty to the component.

2.5 Transaction Mechanism

To guarantee the atomicity of the dynamic reconfiguration operation sequence, transaction mechanism is adopted to rollback while some operations failed to be completed. In the description of the reconfiguration algorithm, the transaction boundary should be declared. By extending the callback interfaces implemented by the components and called by the container, the rollback mechanism is provided.

2.6 Dynamic Reconfiguration Description Mechanism

Familiar with the semantic of the application, the reconfigurator is required to design and describe the reconfiguration algorithm utilizing the above mechanisms in XML file to make use of the existed deployment framework of CCM. The algorithm is a sequence of operations combined with the condition and branch sentences, clarifying the pre-requirement, reconfiguration operation, post-requirement and the transaction demarcation. So the research of this mechanism focuses on how to allow the reconfigurator to describe the algorithm in XML file.

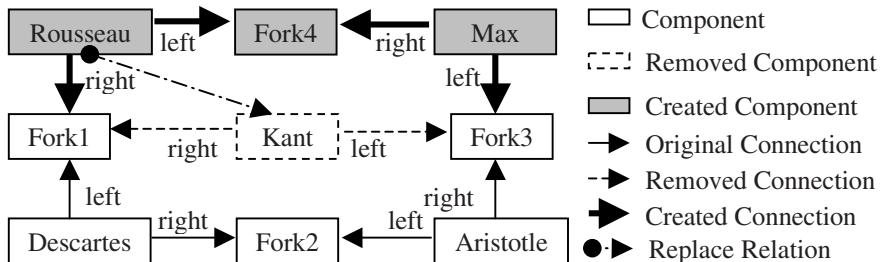
3 Case Study: Dining

Now we demonstrate how to implement the dynamic reconfiguration using the above mechanisms through the dining problem: Philosopher Descartes, Kant and Aristotle sit around a table with one fork besides each one. Only if the philosopher caught both the left and right forks, could he begin eating. The philosopher will repeat the process of eating, sleeping and thinking. To avoid deadlock, if the philosopher must wait for one fork, he must put down the other fork if he has caught one. Now we want to add a new philosopher Max between Kant and Aristotle and replace the Kant with Rousseau at runtime. The original structure and its change are demonstrated as fig.1 shows. Based on CCM, the philosophers and forks are modeled as components. The philosopher uses the left and right forks through receptacles separately named left and right. Using the above mechanisms, the reconfiguration algorithm is designed as follows:

```

Begin Transaction
if the value of attribute "state" of component Kant is "Sleeping"
{
    hold connections through the receptacles named left and right of Kant;
    remove connections through the receptacles named left and right of Kant;
    passivate the component Kant;
    remove the component Kant;
    create the component Rousseau, Max, Fork4;
    activate the component Rousseau with the state from component Kant;
    connect Rousseau with Fork1 and Fork4 through the receptacles right and left;
    connect Max with Fork4 and Fork3 through the receptacles right and left;
    configuration complete components Fork4, Rousseau, Max in turn;
}
Commit Transaction

```

**Fig. 1.** Dining Application

4 Conclusion

Distributed component technology CCM not only simplifies the development but also provides strong support for the dynamic reconfiguration. By extending the CCM, we design a general dynamic reconfiguration platform to allow the reconfigurator familiar with the semantic designs and describes the specific reconfiguration algorithms through the basic reconfiguration mechanisms provided by the platform. This solution provides the generality, flexibility and adaptability. In the future, the ability of validating and analysis and the automatism will be improved with the help of the formal tool and mathematical model.

Reference

1. Christine R. Hofmeister: Dynamic Reconfiguration of Distributed Applications. PH.D. Thesis. Department of Computer Science University of Maryland College Park. (1993)
2. Peyman Oreizy: Issues in the Runtime Modification of Software Architectures. Technical Report, UCI-ICS- 96-35, University of California, Irvine. (1996)
3. Sun Microsystems, Inc.: Java™ 2 Platform Enterprise Edition Specification, v1.4. Proposed Final Draft 2-11/8/02. (2002)
4. OMG: Corba Component Model Specification V3.0. Formal/2002-06-65.(2002)
5. J. Kramer and J. Magee. The evolving philosophers' problem: dynamic change management. IEEE Transactions on Software Engineering 16(11), pp. 1293–1306. (1990)
6. Peyman Oreizy and Richard N. Taylor. "On the Role of Software Architectures in Runtime System Reconfiguration", Proceedings of the International Conference on Configurable Distributed Systems (ICCDs 4). Annapolis, Maryland, May 4–6. (1998)
7. Online Upgrades. Draft Adopted Specification. OMG Document ptc/2002-07-01. (2002)

TMO-Based Object Group Framework for Supporting Distributed Object Management and Real-Time Services

Chang-Sun Shin, Myoung-Suk Kang, Chang-Won Jeong, and Su-Chong Joo

School of Electrical, Electronic and Information Engineering,

Wonkwang University,

Korea

{csshin, gnb, mediblue, scjoo}@wonkwang.ac.kr

Abstract. In this paper, we present a TMO-based object group framework that can support the distributed object group management and the real-time scheduling services on distributed real-time computing environments. These environments have some difficulties for managing lots of distributed objects and providing the timing constraints to real-time objects. For simultaneously solving these problems, we design a TMO object group framework that can manage as a grouping unit of the distributed TMO objects in order to reduce their own complicated managements and interfaces among individual objects without modifying the ORB itself. The TMO object as real-time object, defines the object having real-time property developed from Dream Laboratory at UC at Irvine. The TMO object group we suggested contains several components reflected the object grouping concepts and real-time service requirements analyzed by referring OMG CORBA specifications. To construct our TMO object group framework, we designed the TMO object group structure, and described the functional class diagram with representing relationships among components. We also explained the detailed functional definitions and interactions between the components from the following 2 points of views; object management service by the Dynamic Binder object for selecting an appropriate one out of objects with the same property, and the real-time scheduling service by the Scheduler object and the Real-Time Manager object. We finally verified the results produced by using the known algorithms like the Binding Priority algorithm and the EDF algorithm to see whether a distributed object management service and a real-time service can adapt on the suggested framework.

1 Introduction and Related Works

The modern computing environments have been changing toward the distributed real-time object computing environments with growing the real-time service requirements of practical applications. Though the existing real-time applications largely run as simple real-time constraints on a single processor system, nowadays most real-time applications, like avionics, widely distributed defense systems and so forth, are required to complex time constraints for distributed real-time services. That is, the distributed real-time applications might be executed by one or more logically distributed objects and required the exacting results by interacting among the distributed objects and the timely operation satisfying the real-time constraints[1,2,3].

As one of the representative researches that efficiently manage individual objects at point of view for distributed object oriented applications, the TINA-C (Telecommunications Information Networking Architecture-Consortium) defined the TINA[4]. In the TINA specification, distributed applications can be logically executed as unit of associated objects, called object group, on multiple systems. According to their specification, they have defined only the management specification of object group and the distributed functional components. Nevertheless, they have not defined the detailed specification about real-time services in a distributed environment yet.

For compensating the TINA's weakness, the OMG(Object Management Group) specified with CORBA (Common Object Request Broker Architecture) providing the standard software specification for improving the flexibility, scalability, reusability and etc via objects' implementation of the distributed environments[5], but it is impossible to support the real-time properties for the distributed real-time applications. After this, the RT-SIG(Real-Time Special Interest Group) organized by the OMG suggested the development of the CORBA specification having the real-time extensibility(CORBA/RT) for adding the real-time property to CORBA specification[6]. It made the distributed real-time environments that depended on the special system and/or the operating systems for the real-time services by modifying or extending the ORB that is the core of CORBA. In these same times, the Dream Laboratory at UCI reported the TMO(Time-triggered Message-triggered Object) scheme[9,10]. The TMO object is defined as an object having real-time property itself. The TMO object scheme is syntactically a simple and natural but semantically powerful extension of conventional object structuring approaches. However, the TMO scheme cannot support the concept of object group for real-time scheduling service and dynamically an appropriate object selection mechanism from replicated TMO objects with the same service property in an object group.

With following up the TINA and CORBA specifications, and providing both the object group management and real-time service on independent framework based on CORBA, the Real-Time Object Group(RTOG) model in the distributed environments has been researched by our researches[2,7,8]. As you referred our papers, the RTOG is developed for supporting the distributed real-time services based on the TINA's object group concepts without modifying the ORB on the standard CORBA. But this RTOG model is not enough to address problem of a dynamic object binding service among replicated objects that are called objects providing the same service property.

In this paper, for solving some problems mentioned above which did not studied in point of view of the object group, we suggest the TMO object group framework that can manage the TMO objects as a unit of the object group on COTS(Commercial Off-The-Shelf) middleware and provide the execution power of the guaranteed real-time services. With given pre-requirements of distributed real-time services, we defined the concepts of the TMO object and a structure of the TMO object group. We also explained the detailed functional definitions and interactions between components from the following 2 points of views; the object management service by the Dynamic Binder object for selecting an appropriate one out of objects with the same property, and the real-time scheduling service by the Scheduler object and the Real-Time Manager object. We finally verified the results executed by using the known algorithms like the Binding Priority algorithm and the EDF(Earliest Deadline First) algorithm to see whether a distributed object management service and a real-time service can adapt on the suggested framework. These algorithms might be altered to

others for improving adaptation of our framework. In this paper, the viewpoint of performance is out of boundary, because we are interested in mapping a physical environment to logical one, our framework, directly at aspect of adaptability.

2 TMO Object Group

This section explain the whole of overview of the suggested TMO object group for managing individual objects in an object group and guaranteeing the real-time service in the distributed systems. The TMO object scheme[9,10] we used in paper developed from Dream Laboratory at UC at Irvine. The TMO objects, as service objects contained in an object group, are implemented by using this scheme.

2.1 TMO Object Scheme

The TMO object is defined as a real-time object having real-time property itself. This object scheme is extending the concept of existing service object. Be different from the generic object concept, TMO object has additionally an SpM(Spontaneous Method) that can be spontaneously triggered by the defined time in an object. Figure 1 is shown its structure. The TMO object contains its name, an ODS, EAC, AAC, SpMs, and SvMs as follows. As stated in [10], the role of each component is described like below.

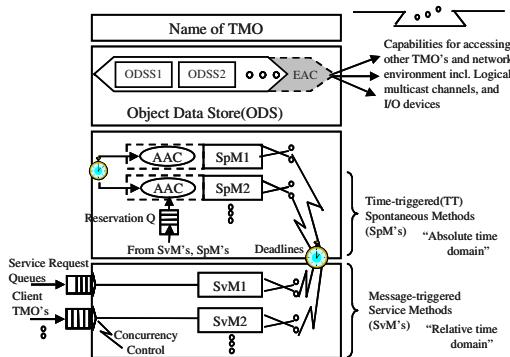


Fig. 1. Structure of TMO object scheme

- ① ODS(Object Data Store) : Storage of properties and states of the TMO object.
- ② EAC(Environment Access Capability) : List of gates to objects to providing efficient call-paths to remote object methods, logical communication channels, and I/O device interfaces.
- ③ AAC(Autonomous Activation Condition) : Activation condition for an SpM, which defines the time window for the execution of that SpMs.
- ④ SpM(Spontaneous Method) : A time triggered method which runs in real-time a periodic manner.
- ⑤ SvM(Service Method) : A message triggered method which responds to external service requests.

In details, an ODS is a common object data store being accessed by the SpM and the SvM, and both methods could not be accessed it simultaneously. When SpM and SvM access the ODS at the same time, the SpM's priority is higher than the SvM's one. That is, the TMO object is triggered by the BCC(Basic Concurrency Constraints). The EAC is responsible for the interface call of the communication channels and the I/O devices. The SpM and the SvM are the list of methods, which are clearly separated from the existing object. The TMO object can have several SpMs and SvMs. In the AAC(Autonomous Activation Condition) that located in the first clause of the SpM, we specify the activation time of the SpM, and implement the TMO object as the real-time object. The TMO object scheme is going on the lively research in the real-time simulation as the military or the transportation applications. But it is impossible to check the security for the object access, manage the replicated TMO objects with the same property and support the distributed scheduling service of several TMO objects globally in a given object group. We intended to solve these problems by taking advantage of the TMO object group model we suggested.

2.2 TMO Object Group Framework

The TMO object group framework proposed in this paper is a new structure that can apply the object group concept being suggested by TINA on COTS middleware. The TMO object group is represented by a logical unit that consists of a set of objects to manage an object group and a set of TMO objects to execute real-time services.

Let us explain the components and their functionalities in the TMO object group. The major roles of these components are categorized into two kinds of services; the object management service and the real-time scheduling service. For supporting the object management service, our framework contains several components, such as the Group Manager(GM) object, the Security object, the Information Repository object and the Dynamic Binder object. And for supporting the real-time scheduling service, our framework contains several components, such as the TMO objects, the Real-Time Manager(RTM) objects and the Scheduler objects. And the TMO object group may contains the replicated TMO objects with the same property and the Sub-TMO object groups as a nested inner object group. A nested object group allows encapsulation and hierarchical organization. Figure 2 shows the structure of the TMO object group framework.

From this framework, at point of view of a support of the object management service, the GM object is totally responsible for managing of all of objects being in an object group and returning the unique reference of the requesting TMO object to a client. The Security object checks access rights of an object requested by referring the access control list(ACL). The Information Repository object stores information such as service properties and their references about all of TMO objects existing in an object group. The GM object is also responsible for maintaining this information, and this will be used for selecting an arbitrary object or an appropriate one out of replicated TMO objects. In this procedure, the Information Repository object sends their references to the Dynamic Binder object. After then, the Dynamic Binder object selects an appropriate object that will be invoked by a client, after referring each system's load information and deadline. For selecting an appropriate one out of the replicated TMO objects, we will adopt the known sample algorithm like the binding priority algorithm considering system's workload and network traffic information and

the request deadline as inputs to the Dynamic Binder object for calculating binding priorities of the replicated TMO objects. This algorithm assigns the binding priorities to the replicated objects individually. The GM object finally returns the reference of the selected object with the highest priority to a client. In case of binding with a non-replicated object, it is trivial.

At point of view of a support of the real-time scheduling service, the RTM object takes the calculated service deadline of the TMO object requested over the Scheduler object. This basic flow procedure occurred from clients' request to getting results is divided into 3 detailed steps; a service requesting step, a service processing step and a result returning step. Each step should be defined the timing constraints itself. Considering given timing constraints, the Scheduler object assigns the priority to the requesting tasks according to scheduling sequences. For verifying whether the Scheduler object can schedule or not in our model, we adopt the EDF scheduling algorithm, as an algorithm, to the Scheduler object for deciding the requests' priority. The Figure 3 shows the functional class diagram of the components that organized in the TMO object group using the Object Modeling Technique(OMT).

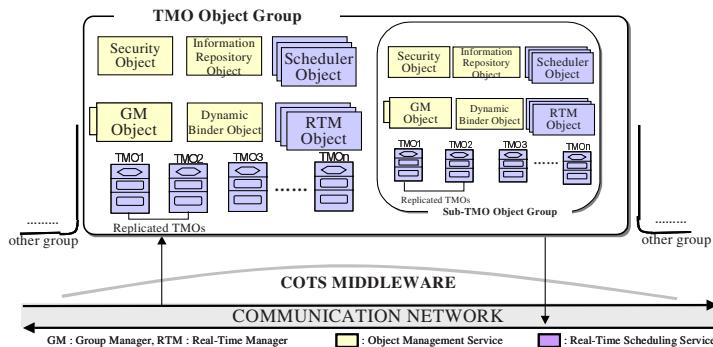


Fig. 2. Structure of the TMO object group framework

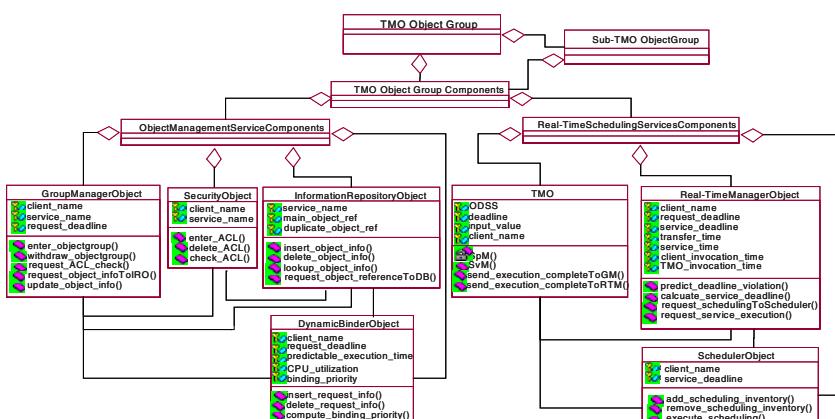


Fig. 3. Functional class diagram of components in a TMO object group

2.3 Timing Constraints

To support the object management and the real-time scheduling service at our TMO object group, the binding priority algorithm and the EDF algorithm adapting to the Dynamic Binder object and the Scheduler object respectively are requiring the timing constraints for real-time services. The basic flow procedure obtaining the result returned from a request is divided into 3 steps; the service request step, the service process step and the result returning step. Each step must have individual timing constraint and guarantee the timeliness by explicitly outlining the definitions of timing constraints. Therefore, we define five timing constraints with showing Figure 4 in our study as follows;

- An invocation time(IT) constraint specifies CIT(Client's Invocation Time), as a time that a client sends a request message to a TMO object and SIT(Service TMO's Invoked Time), as a time that a service TMO object received a request of a client.
- A service time(ST) constraint specifies the relative time for the service execution of a TMO object.
- A transfer time(TT) constraint specifies the relative time required for transferring a request from a client to an TMO object(SIT-CIT), or inversely.
- A service deadline(SD) constraint specifies the absolute time that a TMO object completes the service requested.
- A request deadline(RD) constraint specifies the absolute time that a client received a result returning from a TMO object after a client requested a TMO object. Here, a RD is the timing constraint that must be guaranteed in a distributed real-time application.

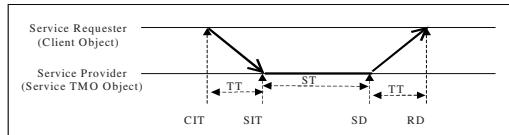


Fig. 4. Definition of timing constraint

From this figure, the timing constraint of each step can be expressed by following equations;

$$\begin{aligned} RD &\geq CIT + ST + (2 \times TT) + \text{slack time}, \\ SD &\leq RD - TT, \text{ where } TT = SIT - CIT \text{ and } ST = SD - SIT. \end{aligned}$$

The deadline for service execution(RD) is the sum of the client's invocation time(CIT), the service time(ST), total transfer time($2 \times TT$) and slack time. Here, the *slack time* means a constant as a factor to decide the time of the adequate RD. This service deadline(SD) must be smaller than and equal a time value that subtracts transfer time(TT) from a client's request deadline(RD).

3 Object Management Services

As we shown in Figure 2, the GM object is a representative of all of objects in a given object group. For requesting a service, a client should firstly obtain a reference of the

GM object existing in an object group via a Naming Server, and then, request the desiring TMO object's reference to the GM object. To do so, the GM object checks access rights of the requesting TMO object from the Security object, and if it is possible to be accessed, it continuously requests to the Information Repository object for getting TMO object's reference. The Information Repository object searches a TMO object's reference from the object table managing objects in a group itself, and returns it to the GM object. At this time, if the TMO objects requested are being replicated in a given object group, the GM object requests the Dynamic Binder object to obtain the reference of an appropriate TMO object by way of the Information Repository object. The Dynamic Binder object will be selected an appropriate one out of the replicated TMO objects using an arbitrary algorithm. Otherwise, that is, if the replicated TMO objects are not existed in a given object group, this algorithm will not be needed. Through these procedures, finally the GM object receives the TMO object's reference, and returns one to a client inversely. Figure 5 shows an Event Trace Diagram(ETD) for representing the whole management procedures mentioned above. We note that the binding priority algorithm is only used as a sample algorithm for implementation of the Dynamic Binder object.

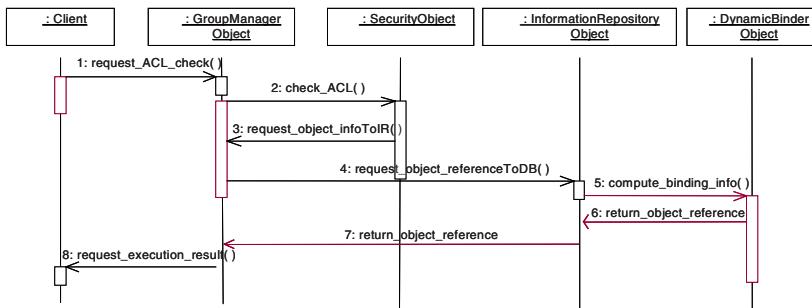


Fig. 5. The ETD for object management service

3.1 Dynamic Selection and Binding Service from Replicated TMO Objects

When two or more TMO objects with the same service property, called replicated target object, exist in an object group for supporting service, the least one out of replicated TMO objects should be selected and this selected object have to bind to the client object. For this reason, the Dynamic Binder object is implemented as an algorithm for selecting an appropriate one among replicated TMO objects. Here, we explain an example algorithm, called the binding priority algorithm, which describes with calculating each object's binding priority and then selecting an appropriate object's reference. As input parameters of this algorithm, we use load information of systems that objects are located on and the request deadline information of client's task. You do not care of adopting another algorithm as an alternative. In this section, we only use the binding priority algorithm for verifying whether the Dynamic Binder object in our object group may operate correctly, but not showing the improved performance of our framework itself. Let us consider the calculation of binding-priority. The $binding_priority_k$ for the client's $request_k$ can calculate by the following expression[1,11].

$$binding_priority_k = \frac{1}{\sum_{i=0}^k request_deadline_i} + \frac{c}{CPU_utilization}$$

where

request_deadline : client's request deadline, *CPU_utilization*: CPU utilization rate, *c* : rate constant(0.01)

From above expression we have defined, we can obtain the binding priorities of all of replicated TMO objects with the same property. In the dynamic object selection and binding procedure, we just allocate client's request to the TMO object with the highest binding-priority out of replicated ones. That is, the client will be received the reference of the selected object for binding between a client object and a server object. To verify the dynamic selection and binding service in our object management services, we show an appropriate example as follows;

In our framework environment, Let us assume that a sequence of 8 clients' requests continue to arrive to the GM object in an object group for invoking replicated TMO objects(TMO1 or TMO2) with the same property and they have already had their client names, request deadlines, CPU utilizations of systems in which TMO1 and TMO2 are located, and binding-priorities for taking TMO1 or TMO2 services, that are the client's status information. Here, we will not numerically describe the client's status information, like request-deadline(RD), CPU-utilization, and so on. The binding-priority will be decided through competitions of clients waiting in Ready Queues of TMO1 and TMO2 using the clients' status information. After finishing the dynamic selection and binding service, we can get the results that client c1, c3, c5, c6 and c8 are bound to TMO1, and c2, c4 and c7 are bound to TMO2. Here, if each CPU's utilization may change due to system overloads, the binding priorities should be also changed. Figure 6 shows the binding priorities and the selected binding references as results executed by the Dynamic Binder object.

```

C:\WInprise\Wbroker\bin\wbj.exe
client_name:c1, object_reference:TMO1 -->TMO1's priority:0.197, TMO2's priority:0.188
client_name:c2, object_reference:TMO2 -->TMO1's priority:0.147, TMO2's priority:0.184
client_name:c3, object_reference:TMO1 -->TMO1's priority:0.148, TMO2's priority:0.138
client_name:c4, object_reference:TMO2 -->TMO1's priority:0.132, TMO2's priority:0.138
client_name:c5, object_reference:TMO1 -->TMO1's priority:0.131, TMO2's priority:0.122
client_name:c6, object_reference:TMO1 -->TMO1's priority:0.123, TMO2's priority:0.122
client_name:c7, object_reference:TMO2 -->TMO1's priority:0.119, TMO2's priority:0.122
client_name:c8, object_reference:TMO1 -->TMO1's priority:0.118, TMO2's priority:0.114

```

Fig. 6. Results executed by the Dynamic Binder object

The each component in the TMO object group was designed to object-based technology, and implemented by using VisiBroker running on Windows 2K and Linux for independently supporting heterogeneous distributed computing environments.

4 Real-Time Services

For supporting real-time service from the TMO object group framework, we use the client's status information described above section. On coming client TMOs' requests into an TMO object group including the desiring target TMO object, each client TMO object transmits its own status information to the Real-Time Manager(RTM) object. After calculating the service deadline(SD) which a target TMO object should be served for a client, the RTM send the calculated SD and the client's status information toward the Scheduler object. The Scheduler object schedules the client's task, i.e. client request by using the Earliest Deadline First(EDF) algorithm. In this section, we only use the EDF algorithm for verifying whether the Scheduler object may be scheduled correctly, but not interesting in the improved performance of our framework itself. The below Figure 7 showed the ETD described interactions among the relevant objects for supporting the real-time scheduling service in the TMO object group.

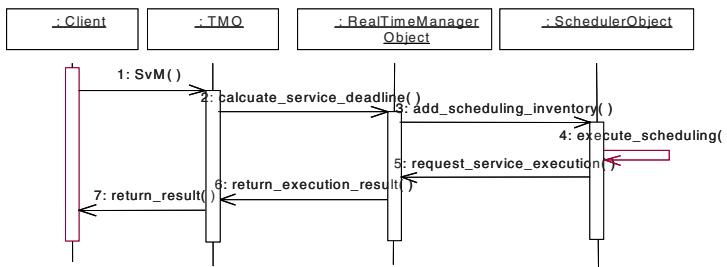


Fig. 7. The ETD for real-time scheduling service

According to real-time scheduling procedures, the Scheduler object have to immediately execute a task of a client's request whenever the TMO object stays in the idle state. Otherwise, the successive client's requests have to be piled on the Scheduler object's ready queue and waited for taking the guaranteed service until the executing task is finished. The Scheduler object is implemented to the EDF algorithm with non-preemptive property. According to this algorithm, the highest priority task is defined the client's request with minimum service deadline. The task execution priority list will be renewed whenever tasks are arrived or finished. The following expression describes making a condition deciding a task priority(TP), given two tasks' service deadlines[12].

$$\text{if } SD_i < SD_{i-1} \text{ then } TP_i > TP_{i-1}$$

Let us consider the same framework environment given at chapter 3. After finishing the dynamic TMO object selection and binding service, like Figure 6. Client c1, c3, c5, c6 and c8 are assigned to TMO1, and client c2, c4 and c7 are assigned to TMO2. Here, to verify the real-time scheduling service on our object group framework, we show the all of scheduling procedures of TMO1 that are requested by c1, c3, c5, c6 and c8. For real-time scheduling, we considered the service deadline(SD) which subtracted the transfer time(TT) from client's request deadline(RD) . That is, the RTM object calculates the service deadline(SD) from following expression; $SD = RD - TT$. The RTM object invokes the Scheduler object to

decide the priority level. As a result, the Figure 8 is shown a sequence of their executions on window screen, as scheduling execution results(c1, c3, c5, c8 and c6) of the Scheduler object on our framework, when TMO1 is requested by the coming sequent clients(c1, c3, c5, c6 and c8).

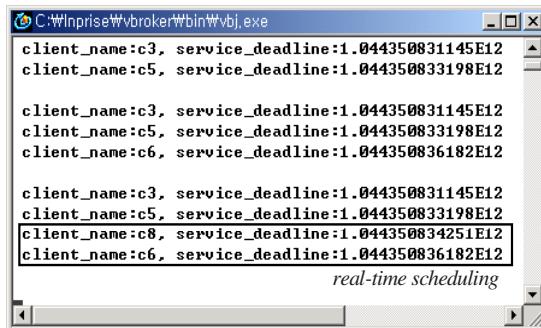


Fig. 8. Scheduling results of the Scheduler object for the target object TMO1

5 Conclusions

The TMO object group framework proposed in this paper is a logical distributed structure that can provide the object group management and the TMO-based real-time services on COTS middleware without restricting the real-time CORBA or operating systems. For achieving this framework, we described whole of overview of the TMO object group, such as the concepts of the TMO object, structuring our model, designing the functions, interactions among the components in the object group, and implementing the Dynamic Binder object and the Scheduler object for supporting dynamic binding and real-time scheduling services. We also described the ETD for conveniently showing procedures of the object group management and real-time service in our framework. After then, in order to verify executions of the framework we constructed, we adopted the known algorithms as implementation of these objects described above, while implementing Dynamic Binder object and Scheduler object respectively. The algorithms we used are the binding priority algorithm for dynamic binding service and the EDF algorithm for real-time scheduling service. For the reason we used them, we only use these algorithms for verifying whether our framework may operate correctly, but not showing the improved performance of the framework itself. With the execution results obtained from our framework using above algorithms, we showed that our framework can be being supported not only the dynamic selection and binding service of replicated or non-replicated TMO objects from client's request, but also real-time scheduling service for an arbitrary TMO object requested from clients.

In future, for applying this framework to practical fields, we have a plan to develop a prototypical framework that can variously adopt new dynamic binding and real-time scheduling strategies to Dynamic Binder object and Scheduler object in an object group. After then we will verify the execution power of the distributed real-time application on the TMO object group framework via various simulations for improving real-time services and convenient distributed object management services.

Acknowledgements. The authors wish to acknowledge helpful discussions of the TMO Programming Scheme with Professor Kane Kim in UC at Irvine DREAM Lab. This work reported here was supported in part by Research funds of Wonkwang University and the Brain Korea 21-Project of KRF, 2003.

References

1. M. Takemoto: Fault-Tolerant Object on Network-wide Distributed Object-Oriented Systems for Future Telecommunications Applications. In IEEE PRFTS (1997) 139–146
2. W.J. Lee, C.W. Jeong, M.H. Kim, and S.C. Joo: Design and Implementation of An Object Group in Distributed Computing Environments. Journal of Electronics & Computer Science, Vol. 2, No. 1 (2000)
3. E.D. Jensen, C.D. Locky, and H. Tokuda: A Time-Driven Scheduling Model for Real-Time Operating Systems. In Proc. 6th IEEE Real-Time System Symposium (1985) 112–122
4. L. Kristiansen, P.Farley, R.Minetti, M. Mampaey, P.F. Hansen, and C.A. Licciardi: TINA Service Architecture and Specifications. <http://www.tinac.com/specifications>
5. Object Management Group: The Common Object Request Broker: Architecture and Specification 2.2. <http://www.omg.org/corba/corbaCB.htm> (1998)
6. OMG Real-time Platform SIG: Real-time CORBA A White Paper-Issue 1.0. http://www.omg.org/real_time/real-time_whitepapers.html (1996)
7. C.S. Shin, M.H. Kim, Y.S. Jeong, S.K. Han, and S.C. Joo: Construction of CORBA Based Object Group Platform for Distributed Real-Time Services. In Proc. 7th IEEE Int'l Workshop on Object-oriented Real-time Dependable Systems (WORDS'02) (2002) 229–302
8. C.S. Shin, M.S. Kang, Y.S. Jeong, S.K. Han, and S.C. Joo: TMO-Based Object Group Model for Distributed Real-Time Services. In Proc. IASTED Int'l Conference Networks, Parallel and Distributed Processing, and Applications(NPDPA'02) (2002) 178–183
9. K.H. Kim: Object-Oriented Real-Time Distributed Programming and Support Middleware. In Proc. 7th Int'l Conf. on Parallel & Distributed System (2000) 10–20
10. K.H. Kim, Seok-Joong Kang, and Yuqing Li: GUI Approach to Generation of Code-Frameworks of TMO. In Proc. 7th IEEE Int'l Workshop on Object-oriented Real-time Dependable Systems(Words'02) (2002) 17–25
11. V. Kalogeraki, P.M. Melliar-Smith, and L.E. Moser: Dynamic Scheduling for Soft Real-Time Distributed Object Systems. In Proc. IEEE 3rd Int'l Symp. on Object-Oriented Real-Time Distributed Computing (2000) 114–121
12. John A. Stankovic, Marco Spuri, Krithi Ramamirthm, Giorgio C. Buttazzo: Deadline Scheduling for Real-Time Systems. Kluwer Academic Publishers (2002) 31
13. G.M. Shin, M.H. Kim, and S.C. Joo: Distributed Objects Grouping and Management for Supporting Real-Time in CORBA Environments. Journal of The Korea Information Processing Society, Vol. 6, No. 5 (1999)
14. B.T Jun, M. Kim, and S.C Joo: The Construction of QoS Integration Platform for Real-Time Negotiation and Adaptation Stream in Distributed Object Computing Environments. Journal of The Korea Information Processing Society, Vol. 7, No. 11S (2000)

Extendable and Interchangeable Architecture Description of Distributed Systems Using UML and XML

Changai Sun^{1,2}, Jiannong Cao¹, Maozhong Jin², Chao Liu², and Michael R. Lyu³

¹ Department of Computing, Hong Kong Polytechnic University
Hung Hom, KLW Hong Kong

{cscasun, csjcao}@comp.polyu.edu.hk

<http://www.comp.polyu.edu.hk/people/csjcao.html>

² School of Computer Science and Engineering

Beijing University of Aeronautics and Astronautics

Xueyuan Road 37, Haidian district, 100083 Beijing, P.R. China

{jmz, liuchao}@buaa.edu.cn

³ Department of Computer Science and Engineering

The Chinese University of Hong Kong, Shatin N.T. Hong Kong

lyu@cse.cuhk.edu.hk

Abstract. Software Architecture can help people to better understand the gross structure and, with powerful analysis techniques, to evaluate the properties of a software system. To accommodate the dynamic changes and facilitate interoperation of tools, an architectural description of the distributed system should be extensible and interchangeable. In this paper, we utilize the built-in extension mechanism of the Unified Modeling Language (UML) to describe the architectures of distributed systems, with the underlying architectural metadata represented in XML. In particular, the approach has been applied to describe the architectural model of distributed software in the Graph-Oriented Programming framework. The proposed approach has many desirable features, characterized by being visual, easily extendable and interchangeable, and well supported by tools.

1 Introduction

Software Architecture (SA), as a bridge between requirements and design [1,2], is a high level abstraction of system structure. It is composed of a set of components with independent functions and explicit interfaces and interactions between the components. Software architecture description is the representation of abstract architectural model in Architecture Description Language (ADL). It provides a blueprint for system construction and composition, and permits designers to make early design decisions based on the architectural documentation. Various ADLs have been proposed in the community of SA, such as Darwin, C2, Rapide, Aesop, Wright, SADL, Acme[3].

Software architecture description can be characterized by various features, including composition, abstraction, reusability, configuration, heterogeneity, and analysis [6]. Furthermore, it has been argued that software architecture description of distributed systems should be dynamic and reflective [7]. With the development of

many well-known ADLs, some additional issues need to be addressed. For example, how to decide the tradeoff between visualization and rigorousness is still a key issue to design ADL. Another problem is how to exchange description between the architectural models described by different ADLs. It is our opinion that architecture description of a distributed system should also be:

Extendable: The architecture description should be able to be extended as required. For example, the support of time performance analysis is a key requirement of the architecture model of Real-time systems, while it may be neglected in early systems.

Interchangeable: The architectural representation should be able to be exchanged between heterogeneous environments, including different languages and platforms. This property is particularly important for distributed systems.

There are some attempts to address the interoperability, including: (1) The development of an architectural interchange language, e.g. Acme [4]. (2) The establishment of methods to integrate architecture-based tools [2]. (3) The definition of ADL based on XML schema, such as xADL [5]. The first approach requires that ADLs involved in architecture description interchange be homogeneous. This is hard to imagine because any individual ADL is designed for special types of applications or features. The second approach leaves the task of translation to the tools, which is impossible in some cases. The third approach utilizes XML, which makes the description easily extensive, and more the physical model easily exchanged. For this approach to be successful, however special graphic notations for architectural elements must be redesigned for the feature of visualization, and thereby some corresponding supporting tools are also developed.

This paper proposes an approach to develop description of architecture of distributed systems, which is extendable, interchangeable and well supported by standard graphic notation. The core idea of the approach is to incorporate UML and XML. In particular, rather than translating an ADL to UML, we will extend UML based on its built-in extension mechanism, to directly describe an architectural model for distributed systems, namely the Graph-Oriented Programming (GOP) Model [7].

The rest of this paper is organized as follows. The next section describes the proposed framework of describing the GOP architecture. Section 3 proposes an approach to extending UML to describe the GOP architectural model based on UML extension mechanism and XML and provides an example of GOP demonstrating how the proposed approach can be used. Section 4 concludes this paper by offering some conclusions and plans for future works.

2 The Framework of Describing the GOP Architecture

In this section, we introduce the underlying concepts of GOP model and present the framework of describing the GOP architecture.

2.1 The GOP Model

The Graph-Oriented Programming Model (GOP Model) was originally proposed for distributed programming [7]. Under the GOP model, the components of a distributed program are configured as a logical graph and implemented using a set of operations defined over the graph. In GOP, a distributed program is defined as a collection of

local programs (LPs) that may execute on several processors. Each LP performs operations on available data at various points in the program and communicates with other LPs. The GOP model is shown as Fig.1:

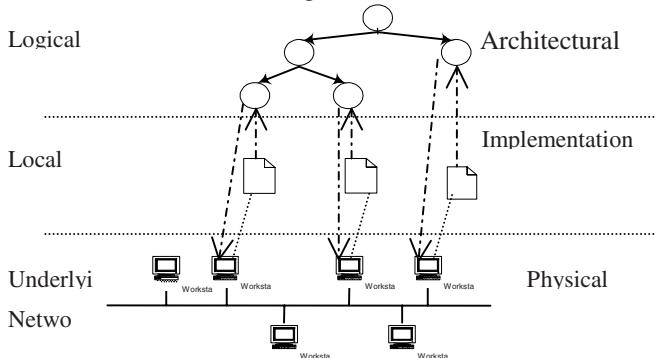


Fig. 1. The GOP Conceptual Model

As illustrated in Figure 1, the GOP model consists of:

- a **logical graph** (directed or undirected), whose nodes are associated with local programs (LPs), and whose edges define the relationships amongst the LPs.
- a set of **local programs** (LPs) performing the computation tasks and cooperating with each other using the graph-oriented programming primitives.
- a **LPs-to-nodes mapping**, which allows the programmer to bind LPs to specific nodes,
- an optional **nodes-to-processors mapping**: which allows the programmer to explicitly specify the mapping of the logical graph to the underlying network of processors.
- a library of language-level graph-oriented **programming primitives**.

It is noted that the logical graph in Fig.1 is very similar to the architecture of distributed program, according to the definition of Software Architecture discussed in Section 1. Further, this *architecture layer* of GOP Model relates to the *implementation layer*, which consists of a set of *Local Programs*, and the *physical Layer*, which consists of a set of processors connected by the underlying network.

When the distributed programs following the GOP Model are implemented, the Logical Graph is used as a type object, implementing the expected functions together with all the local programs. That is, the graph-oriented architecture model has been incorporated into the runtime system. Therefore, the architecture of distributed program can be dynamically *reconfigured* with *explicit reflection*, by invoking the behaviors provided by a logical graph object. It is a highly desirable feature in building dynamic distributed system.

We have leveraged the GOP Model as the architectural model. All processors will need to share the architecture representation of GOP Model, so the architecture description should be easily interchangeable between the different processors. It is especially true, as a result, for the heterogeneous distributed systems. Next, we will focus on extending UML to describe the GOP-based architecture.

2.2 The Framework

The principle of extending UML to describe the architecture model of GOP is illustrated in Figure 2. The central idea is to describe the architectural level information and semantics (including concepts and properties) of the GOP Model, a kind of architecture model, based on UML infrastructure (standard modeling notations and built-in extension mechanism) and XML.

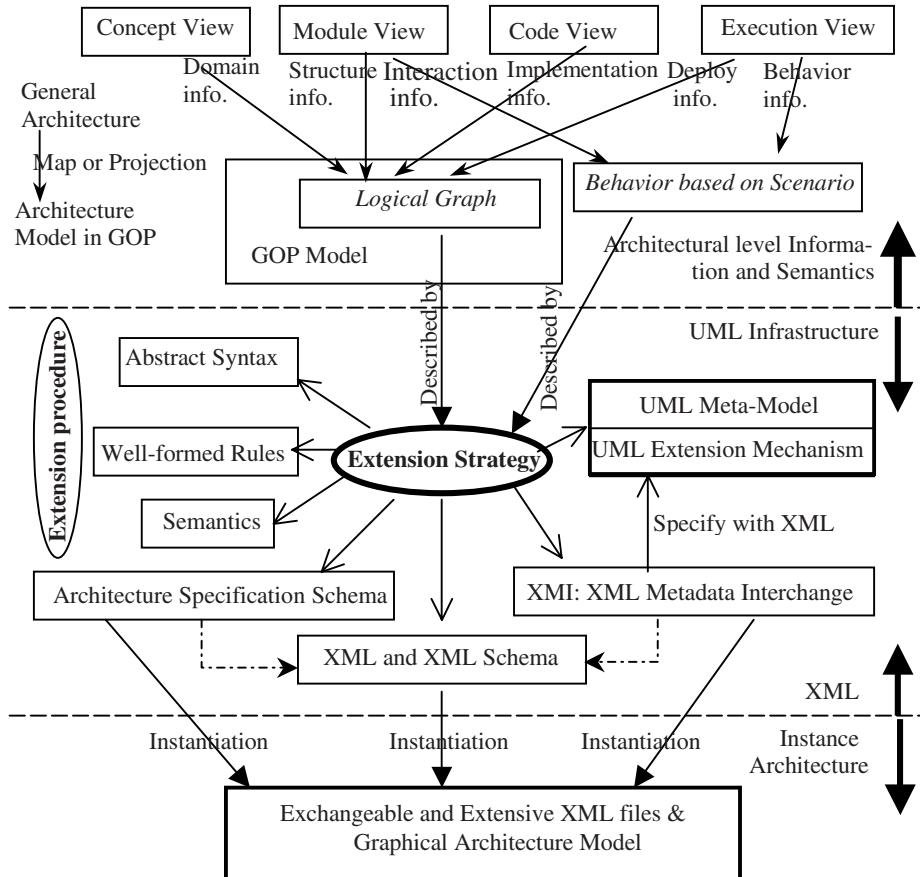


Fig. 2. The Framework for Describing GOP Architecture Model with Extensive UML and XML

The framework of our approach is divided into three layers. The top layer is *Architectural information and semantics*, which is concerned with what should be described for software architecture. Four common recognized architecture views are mapped or projected into the *logical graph* of the GOP model. The middle layer is *UML infrastructure and XML*. *Extension strategy*, the key element of this layer is concerned with issues, such as which Meta-model of UML is selected and extended to describe logical graph and behavior based on Scenario, how to extend them, and what are the steps to follow for extension. The bottom layer is *Instance Architecture*. An architecture instance can be described as a group of graphical architecture views or a set of interchangeable and extendable XML files.

3 Extending UML to Describe GOP Model

In this section, we propose an approach to extending UML to describe the architecture model in GOP.

3.1 UML Built-in Extension Mechanisms and XMI

UML [8] is a standard object notation for industry. A UML Model of a software system generally consists of several partial models, each of which addresses a certain set

of issues from different angles and at different levels. UML is graphical language with a fairly well defined syntax and semantics. The syntax and semantics of the underlying model are specified semi-formally via descriptive text and constraints, such as OCL expression. The linguistic architecture of UML is a four-layer meta-model, and the *meta-model layer* is a focus for *extension* and the *development of tools*. It is noted that the *meta-model* of UML itself is organized in the form of package.

UML is an extendable language in that new constructs may be added to address new issues in software development. Three mechanisms are provided to allow extensions without changing the existing syntax or semantics of the language: 1) *Stereotype* allows groups of constraints and tagged values to be given descriptive names and applied to other model elements, 2) *Tagged Value* allows new attributes to be added to model elements, 3) *Constraint* allows new semantics to be specified linguistically for a model element.

To enable model representations generated by the different supporting tools to be exchanged between the tools, XML Metadata Interchange (XMI) Specification is also provided in version 1.4, and later version.

3.2 Extension Strategy

As discussed in Section 3.1, UML provides a set of extension mechanism to satisfy new modeling requirements. Some standard stereotypes are also provided in the meta-model of UML for general extension. The process of extending UML to describe the architecture can be summarized as follows:

- 1) Select an appropriate meta-model package to extend. The meta-model selected should be the one closest to the architectural view from the perspective of semantics.
- 2) Decide the Meta-Class to use as the base-class of standard element in the UML meta-model.
- 3) Associate architectural semantics to the elements of the architectural view, including some extended attributes or specific constraints. Here, extensions for different architectural views and architecture properties can differ greatly.

The architectural model using UML can be described with different extension strategies. David Garlan et al. proposed that different extension strategies for architecture structure should be evaluated by using three evaluation criteria: *Semantics-Match*, *Legibility*, and *Completeness* [9]. Extension strategies should also be: 1) *Extensible and well-structured*, meaning that architectural description should be able to be enhanced further to support the description and analysis of new properties.

2) *Efficient and Effective*, meaning that the extension should as far as possible reuse the features of UML supporting tools to serve the requirements of the architectural description.

3.3 A Reference Extension for the Logical Graph

As an illustration, we propose a reference extension for the logical graph. First, we must choose the meta-model package closest to the logical graph in semantics. Comparing UML Component and UML Class, it's more suitable to select the meta-model of Class to extend for describing the logical graph, based on the extension criteria discussed in Section 3.2 and the following observations:

- 1) UML Component focuses on physical implementation, while logical graph focuses on logical structure.
- 2) As the core element of OO, UML Class can specify systems from several abstract levels, such as concept, specification and implementation.
- 3) The relationships between UML Classes are also richer than those between UML Components. This can make it easier to depict the interaction of components.

Once we decide to choose UML Class to extend for describing the node of logical graph of GOP, the next thing is to decide the Extension Procedure. In order to maintain consistency with the UML manual, we take the following steps: *Abstract Syntax*, *Well-formed Rules*, *Semantics* and *specification*. A reference extension to the Core Package of Foundation package of UML meta-model is illustrated as Figure 3.

Step 1: Abstract Syntax

In Figure 3, <<LGcomponent>> is used to describe the components of the architectural model from three perspectives, namely *context*, *specification*, and *implementation*. We'll focus on the specification of a component, which consists of:

Name: is inherited from ModelElement::Class

isOption: indicates whether the component takes part in the current configuration

isComposite: indicates whether the component is a composite component

Type: abstracts the function of the component described in certain kind of specification language

Version: is used to identify the version information of the component

Vocabulary: indicates special type or customary component, such as Filter or Pipe. Preserved for Architecture style or Pattern.

LGComponent assembles Interface, Context and Implementation, by means of the association.

Implementation: the extension used to indicate the implementation of components. It can be one or more artifacts, such as binary, executable, or script files.

Context: the extension specifying the scenario in which the component can operate as expected.

Interface: the extension that names the points through which the component can interact with the environments, including asking for and providing services.

CompositeComponent: the extension used to indicate the compositecomponent in which the component takes part.

LGConnector, LGConfiguration and LGCompositeComponent can be specified in the similar ways. But they will not be further discussed here.

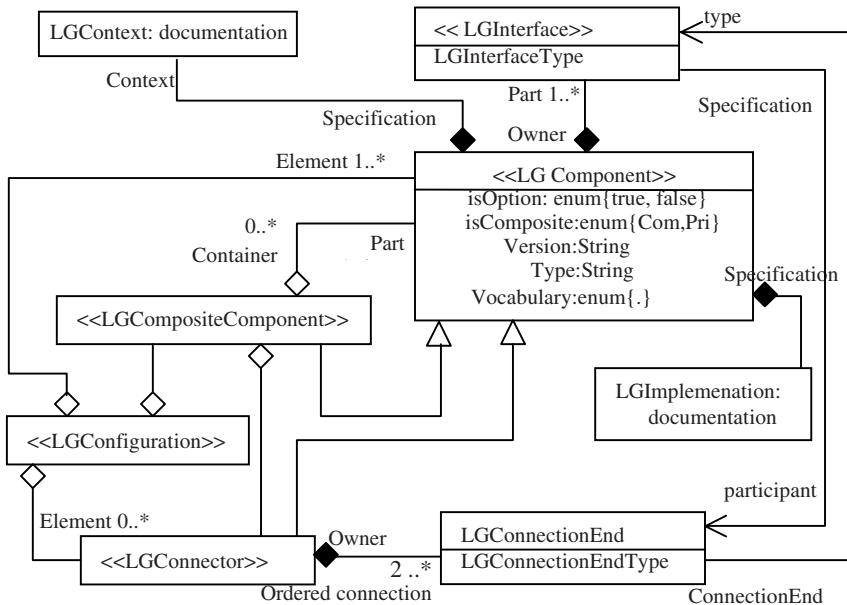


Fig. 3. Meta Model for the Logical Graph of GOP

Step 2: Well-Formed Rules

Some well-formed rules must be followed when using the extended UML model to describe the logical graph of the GOP model. Otherwise, the extension will not be interpreted appropriately. Once again, for brevity, we will demonstrate only a few important well-formed rules.

LGInterface is an instance of a meta-class Interface. It defines the interface of *LGComponent* and must satisfy following constraints:

- 1) The contents of *LGInterface* must be public or protected operation (Interfaces may not have Attributes, Associations, or Methods).

Self.allFeatures->forAll(f| f.visibility=# Public or f.visibility=# Protected)

- 2) Only two interface types are allowed in the *LGComponent*.

LGInterfaceType: Enum{Request, Service}

LGComponent is an instance of meta-class Class. It incorporates many tagged values for describing architecture attributes. Some enhanced constraints must be satisfied in our architectural model, including:

- 1) A *LGComponent* must have at least one Service Interface.

self.allInterfaces->exist(i| i.LGInterfaceType=#Service)

- 2) Any attribute of a *LGComponent* can be accessed only by the *LGInterface*.

Self.OclType.feature->forAll(f| f.oclIsKindOf(Operation))

Step 3: Semantics

LGComponent is a basic construct of the logical graph. A *LGComponent* consists of at least an interface to communicate with external environments. Communication with *LGComponent* can only occur through the *LGInterface*. *LGComponent* can be categorized as Atomic and Composite component, the latter is composed by many

LGComponents and LGConnectors. A composite component can also be a part of one or more larger composite components.

Step4: Specification

The results of a graphical architectural description must be translated into the textual specification or other binary files. To be exchanged between different UML tools and other architecture supporting tools, the specification should be represented in XML based on the base of XMI. Since the extended meta-model is an architecture type rather than an instance architecture, we use XML Schema to represent extended meta-model. To effectively represent extended meta-model, the XML schemas must be well structured, we organize XML schema into a hierarchy structure [1].

3.4 Case Study: A Simple Example

In this section, we will illustrate our approach using the calcpi program used in [10]. This program computes an approximation to π by calculating the area under the curve $4/(1+x^2)$ between 0 and 1 using numerical integration. The program is structured as a set of *workers*, each computing the area, and a *supervisor* collecting the results and averaging them.

We will represent *worker* and *supervisor* as *components*, which execute different calculations, while the communication between *worker* and *supervisor* is treated as *connectors*. In the following example, there are two workers. The *logical graph* representing the architecture of this distributed application is described by extended UML meta-model as shown in Figure 4.

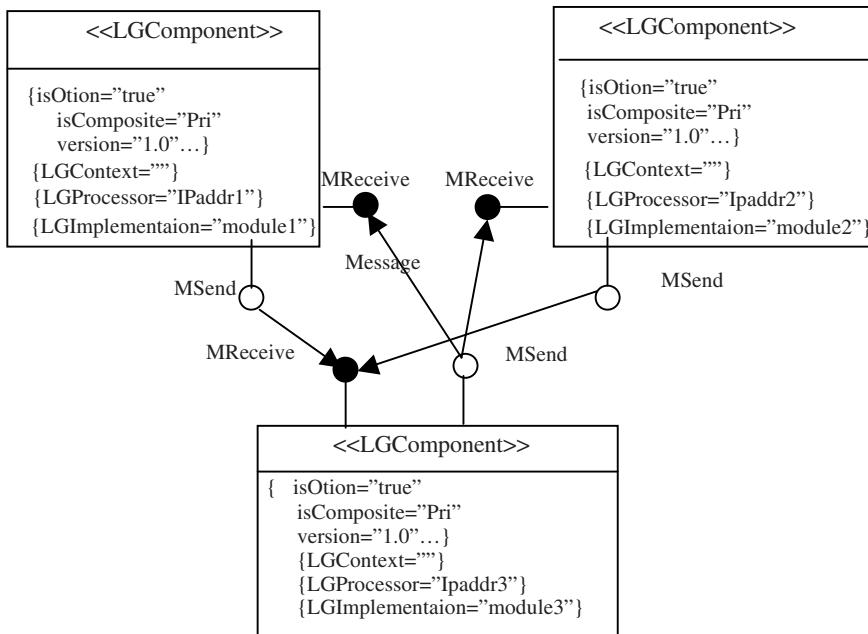


Fig. 4.The Architecture Model Described by Extended UML Class Diagram

The extended UML Sequence Diagram can be used to describe collaborations between *worker* and *supervisor*. In the GOP Model, *nodes-to-processors* mappings are used to deploy the logical graph on the physical networks environments. This can be implemented by specifying the tagged value LGComponent, namely LGProcessor. Similarly, when all components are assembled to run, *local programs* must be bound to the *worker* and *supervisor* of logical graph. This is implemented by *LPs-to-Nodes mappings* in GOP model. In our approach, this can be implemented just by specifying LGImplementation of LGComponent. Moreover, *Version* of LGComponent can be labeled with a different version number to recognize the changes of the implementations of components. The specification for architectural configuration and components of this distributed application described in XML is omitted for the limitation in space.

4 Conclusions

This paper has proposed an extendable and interchangeable architecture description approach based on UML and XML. The idea of our extension strategy can be summarized as follows. 1) UML class is extended as component, 2) UML interface as the interface of component, 3) UML association as connector, 4) UML subsystem as architecture configuration.

Compared with the existing work [11,12], our approach can be characterized by the following features: 1) *Visual modeling* of architecture of distributed system by reusing one standard modeling language UML and its supporting tools. 2) Resulting architecture specification can be *interchangeable* or shared by different supporting tools. 3) *Flexible extension* can be further obtained. Here, we examine some derived features of our approach, and explain why or how they can be obtained.

Integrated Version Management in architecture level can be obtained. In our approach, the component and configuration can be labeled version. If we want to modify a component of the distributed system, the version number of component has been changed from the perspective of version management. This is also easily implemented by modifying the version attribute of node in *Logical Graph*. At any time, the configuration of logical graph maintains the version profile of the current system. When the system evolves, it can be characterized by the evolution of configuration version. If the version information is not required, just set all the version attributes of component and configuration to default or neglect them.

Visual Architectural Description can be supported. UML notations can be reused by our approach, but the semantics for notation of UML is changed, which is entitled with constraints and architectural semantics. For example, LGComponent can be represented as the notation for UML Class, but the semantics must be interpreted as semantics of LGComponent discussed in Section 3.

Existing tools for UML can be reused. Our extension approach is compatible with UML manual, and XMI is accepted as the base of architectural specification, so most of functions provided by UML supporting tools can be reused to describe software architecture.

Further Extension for new architecture properties can be easily obtained. Here, we give an illustration of the further extension. In GOP model, the logical graph can also be deployed on the physical environments while the system is running. This can be

implemented by adding an attribute *LGProcessor* to the construct *LGComponent*, an item responsible for this attribute should at the same time be added into their architecture specification.

Our future work will be conducted to integrate further our approach and the GOP model, including the development of supporting environments, mapping the specification into different platforms, such as Web, Component and Cluster, and representing some large-scale real world applications using our approach in this paper.

Acknowledgement. This research is partially supported by the Hong Kong Polytechnic University under the research grant H-ZJ80, the National High Technology Development 863 program of China (No. 2001AA110244), and RGC Project (No.CUHK4360/02E).

References

- [1] Chang-ai Sun. Contributions to Software Architectural Description and Construction and Reconstruction.[PhD Thesis]. Beijing University of Aeronautics and Astronautics, 2002.12
- [2] David Garlan. Software architecture: a roadmap. In proceedings of the conference on the future of Software Engineering, (2000) 91–101
- [3] Nenad Medvidovic, Richard N.Taylor. A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transaction on Software Engineering, Vol.26 No.1, January 2000,70–93
- [4] D.Garlan, R.T. Monroe, D. Wile. Acme: An architecture description interchange language. In proceedings of CASCON'97 Ontario, Canada, November, (1997) 169–183
- [5] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor A Highly-Extensible, XML-Based Architecture Description Language.In Proceedings of the Working IEEE/IFIP Conference on Software Architectures (WICSA 2001), Amsterdam, Netherlands.
- [6] Mary Shaw, David Garlan. Characteristics of higher-level Languages for Software Architecture, Carnegie Mellon University, Technical Report, CMU-CS-94-210, 1994.
- [7] Jiannong Cao, Xiaoxing Ma, Alvin. T.S. Chan, and Jian Lu, Architecting and Implementing Web-based Distributed Applications Using the Graph-Oriented Approach, to appear in Software: Practice and Experiences (John Wiley & Sons).
- [8] OMG, Unified Modeling Language Specification (Ver 1.5), Mar 2003
- [9] D. Garlan, A. J. Kompanek and P. Pinto. Reconciling the needs of architectural description with object-modeling notations. In proceedings of the Third International Conference on the Unified Modeling Language, York, UK, October 2000.
- [10] N.Rodriguez, R. Ierusalimschy, and R. Cerqueira, Dynamic Configuration with CORBA Components, In proceedings of the Fourth International Conference on Configurable Distributed Systems, IEEE Computer Society Press, May 1998, 27–34
- [11] Nenad Medvidovic, David S.Rosenblum , David F.Redmiles, Jason E. Robbing. Modeling software architectures in the Unified Modeling Language, ACM Transactions on Software engineering and Methodology, Vol.11, No.1, January 2002, 2–57
- [12] C. Hofmeister, R. L. Nord and D. Soni. Describing software architecture with UML. In Proceedings of the First Working IFIP Conference on Software Architecture, San Antonio, TX, February1999, 145–160

Dynamics in Hierarchical CSCW Systems*

Xiaodong Wang, Haiyan Sun, Ming Xu, and Xingming Zhou

Parallel & Distributed Processing Lab.,
National University of Defense Technology,
Changsha, Hunan, P.R.China, 410073
xdwang@nudt.edu.cn

Abstract. Computer supported cooperative work (CSCW) is a popular research topic in recent years. In classical cooperative applications, cooperation cannot be adapted to the dynamics and hierarchies of real world. According to the formal specification of dynamic cooperation in hierarchical systems, a dynamic hierarchical cooperative system model, DHCSCW, is proposed in this paper. In this model, cooperators capable of dynamic cooperation and a new hierarchical structure Group Cluster are constructed. Dynamic cooperation can be implemented through directed information loop among cooperators in adjacent levels. Based on this model, dynamic cooperation can be implemented through the horizontal active cooperation in a group, and hierarchical cooperation can be implemented through vertical cooperation between groups at adjacent levels.

1 Introduction

Over the past decades, the CSCW community has developed a number of experimental collaborative systems and commercial products, such as synchronous groupware [1,2,3] for the collaborative applications, CT[4,5,6] model for database based CAD applications, and Workflow[7,8] model for asynchronous groupware. Cooperation based on negotiation in Multi-Agent system is another branch of research [9,10].

In these systems, cooperation are realized in two modes:

- *Centralized mode.* In this mode, cooperation is conducted by a specific *coordinator*. Group members cooperate based on the schedule passively. These cooperation relationships can only be modified by the *coordinator*.
- *Decentralized mode.* In this mode, cooperation is conducted by all cooperators themselves. They exchange knowledge about workspace environment and negotiate to reach the global agreement about cooperative scheme .

Flexibility is a critical factor for the success of interactive systems. For CSCW systems this property is even more important since there are so many dynamic changes to be adapted with [11,12,13]. These two cooperation modes have their own drawbacks on support flexibility in CSCW systems.

* This paper is founded by project 60273068 supported by National Nature Science Foundation of China.

In *centralized mode*, cooperators cannot alternate their cooperation relationships according to the dynamic workspace environment. The predefined cooperation scheme will degrade the efficiency of cooperation. This mode of cooperation can be classified as *Top-Down cooperation in vertical direction*.

In *decentralized mode*, cooperation can be performed based on negotiation. But a lot of knowledge about environment needs to be exchanged between cooperators. It will take long time for the system to reach agreement in the negotiation process. The efficiency of cooperation is not acceptable especially in a dynamic workspace environment. This mode of cooperation can be classified as *cooperation in horizontal direction*.

To improve the efficiency of cooperation in dynamic environment, a new cooperative system model—DHCSCW, is proposed for **Dynamic Hierarchical CSCW** systems in this paper. In DHCSCW model, participants can cooperate consistently on their global knowledge by themselves in both vertical and horizontal directions along the hierarchies of systems.

2 Dynamics in Hierarchical CSCW

2.1 The Notion of “Dynamical Cooperation” in CSCW

By now it still seems like the research field of CSCW struggles with the issue of what CSCW exactly means. The term CSCW is defined as *the action of working or acting together for a common purpose or benefit*[14]; Gilbert argues that *cooperation means that users work together on shared document, using personal video-conference, or sharing additional resources through the network*[15]; Bond states that *cooperation is a property of interaction among some set of agents performing some collective activity...is a special case of coordination among nonantagonistic actors*[16]; Wooldridge points out that *the cooperation protocol describe how a group of agents work together to achieve some goals*[17]; Wen summarizes that *Cooperation is a process in which a group of agents interact with one another, or reason about each other, to solve a common problem, or share knowledge*[18].

Trying to conceptualize “cooperative work” needs to address the dynamics of this work. Let me illustrate them from three aspects.

- Dynamics of Outer Environment

Group members cooperate in an open, dynamic real world. The state of outer environment will alternate out of the control of all participants’ cooperation.

- Dynamics of Cooperation Scheme

Cooperation scheme consists of partitions of cooperative goal and scene among all group members, and the cooperation rules they must obey. Due to the alterable outer environment, cooperative goal and scene need to be adjusted among all participants, and cooperation rules need to be changed.

- Dynamics of Group Structure

During the cooperation, some participants will leave the group while some others join the group. The structure of group will also change in cooperation.

Hence, cooperative work seems to be dynamic across outer environment of cooperation, group cooperation scheme and the structure of group. So, a framework trying to conceptualize cooperative work activities needs to address such dynamics in work.

2.2 The Notion of Hierarchy in CSCW

According to sociologists' analysis, divide-and-conquer based on the hierarchical structure is the most efficient working way in human society. It decides the hierarchy of massive cooperation in real world, too.

- **Hierarchy of the Partition of Cooperative Goal**

According to the divide-and-conquer mode, a complex cooperative goal can be divided into several relevant subgoals. And these subgoals can be partitioned further. All these goals will form a cooperative goal tree with hierarchical structure. The root node is the global cooperative goal that the group needs to reach as a whole. Those leaf nodes are simple subgoals that can be reached by one member of the whole group.

- **Hierarchy of the Structure of Cooperative Group**

Members belong to a common goal form a group. According to the hierarchical structure of cooperative goal tree, all these groups form a nested structure. A group relating to an upper goal contains several subgroups that concern with those subgoals of the upper one.

- **Hierarchy of the Distribution of Cooperation Info**

Group members need to occupy all related info about cooperation to reach their cooperative goal. Due to the hierarchy of their different cooperative goals, the cooperation info will be distributed hierarchically among cooperators of different groups.

2.3 Summary

Due to the dynamics and hierarchy of general cooperative work, there are several drawbacks when developing such cooperation systems in traditional cooperative modes.

- **Unable to support the dynamics of cooperation**

In *centralized mode*, cooperation can only be arranged by the specific coordinator. Group members can not alternate their cooperation scheme adaptively. The predefined cooperation scheme will not be suitable to those alternated cooperative systems. Although the cooperation scheme can be rearranged by the coordinator, the efficiency of cooperation will be very low in the *centralized mode*.

In *decentralized mode*, cooperation can be realized through negotiation among all the participants. Group members need to re-negotiate to reach the global consistent cooperation scheme in alternated systems. Due to the low efficiency of negotiation, the efficiency of cooperation will be unacceptable.

- Unable to support the hierarchy of cooperation

In *centralized mode*, cooperation can be realized hierarchically. However, a group's cooperation scheme needs to be arranged by the related upper coordinator. That means any alternation of cooperation scheme, no matter how minor it is, can only be decided by the coordinator at the top layer. The efficiency of cooperation will be unacceptable too, especially in a system with several layers.

Thus, the need of building a proper system model for general cooperative work is apparent, especially for its dynamics and hierarchy.

3 Formal Specification of the Cooperative Model DHCSCW

3.1 Formal Specification of the Framework of Cooperative System

As the basis of cooperative system model, a framework of cooperative system is specified formally in this section. In this framework, the basic elements of cooperative system and their relationships are defined.

Definition. A framework of cooperative system is a 5-tuple, $Fr = \langle Op, G, S, \leq, \prec \rangle$ with components as follow.

$Op = \{ i \mid i \text{ denotes a cooperator in the cooperative system} \}$.

$G = \{ g \mid g \text{ indicates a cooperative goal of the system} \}$.

$S = \{ s_g \mid s_g \text{ is the related scene of the cooperative goal } g \}$.

\leq is a *subgoal* relation defined on G , $\leq \subseteq G \times G$. A subgoal is a component goal of another, higher-order, goal. Thus if $g' \leq g$, it means that g' is a subgoal of g . And the logical relationship among the subgoals of one goal determines the cooperative relationships of those cooperators of this goal.

The \leq relation must satisfy the reflexive, transitive, and well-founded properties as follow.

Reflexive property $\forall g \in G, g \leq g$

Transitive property $\forall g, g', g'' \in G, g \leq g' \wedge g' \leq g'' \Rightarrow g \leq g''$

Well-founded property $\forall g \in G, G' = \{ g' \mid g' \leq g \}$ is a finite set.

We can also define the *strict subgoal* relation, $<$, in the similar way.

$\forall g, g' \in G, g < g' \Leftrightarrow g \leq g' \wedge g \neq g'$.

With the exception of reflexivity, $<$ enjoys all the properties of \leq and, in addition, $<$ is asymmetric.

Another useful extension to \leq is the *direct subgoal* relation, \ll . This relation is defined as follows. $\forall g, g', g'' \in G, g \ll g' \Leftrightarrow g < g' \wedge (\neg \exists g'' ((g < g'') \wedge (g'' < g')))$.

Based on the *direct subgoal* relation \ll , the function $dsg : G \rightarrow 2^G$ takes a goal and returns the set of all its *direct subgoals* as $dsg(g) = \{ g' \mid g' \ll g \}$.

The *tasks* of cooperators are represented in a function $ptn : Op \rightarrow 2^G$. It is defined as $ptn(i) = \{ g \mid \text{those goals that are arranged to be achieved by cooperator } i \}$.

The *capabilities* of cooperators are represented in a function $cpb : Op \rightarrow 2^G$. It is defined as $cpb(i) = \{ g \mid \text{those goals that cooperator } i \text{ can achieve in isolation} \}$.

Each goal g in a cooperator's task $ptn(i)$ can be achieved in 2 different ways. If $g \in cpb(i)$, it means the goal g can be achieved by cooperator i alone and $dsg(g) = \emptyset$. If $g \notin cpb(i)$, it means the goal g can not be achieved by cooperator i in isolation and need other cooperators help. So, subgoals of goal g , will be assigned to some cooperators j . That is to say, $dsg(g) \neq \emptyset$. The goal g can only be achieved when all of its subgoals are achieved.

\prec is a *dependency* relation defined on S , in which $s_g \prec s_{g'}$ means the scene of goal g is determined by that one of goal g' . According to the *direct subgoal* relation, it can be concluded that $s_{g'} \prec s_g$ if $g' \ll g$. The \prec relation will also satisfy the reflexive and transitive properties.

Reflexive: $\forall s_g \in S, s_g \prec s_g$

Transitive: $\forall s_g, s_{g'}, s_{g''} \in S, s_g \prec s_{g'} \wedge s_{g'} \prec s_{g''} \Rightarrow s_g \prec s_{g''}$

The outer environment of cooperator i can be represented in a function $scene: Op \rightarrow 2^S$. It is defined as $scene(i) = \{s_g \mid g \in ptn(i)\}$. The $scene$ contains all outer environment related to those goals of one cooperator.

3.2 Formal Specification of the Cooperation Model DHCSCW

To describe the architecture of the cooperation model, the *delegation* relation between cooperators should be defined firstly.

Definition. The relation *delegation* is a 3-tuple $\langle d_{ij}, p_{ij}, q_{ij} \rangle$, in which

$d_{ij} = \{<i, j> \mid \exists g \exists g' (g' \ll g), g \in ptn(i), g' \in ptn(j), i, j \in Op\}$ indicates that cooperator i delegates some subgoals of its goal g to cooperator j ;

$p_{ij} = \{g \mid \exists g (g \ll g'), g \in ptn(i) - cpb(i), g' \in ptn(j)\}$ indicates the set of subgoals that is delegated by cooperator i to cooperator j ;

$q_{ij} = \{s_g \mid \exists s_g (s_g \prec s_{g'}), s_g \in scene(i), s_{g'} \in scene(j)\}$ indicates the set of scenes required by cooperator j to achieve those subgoals delegated by cooperator i .

Based on the *delegation* relation between cooperators, the *group* relation can be defined.

Definition. The relation *group* is a 4-tuple $\langle Coordinator, D_c, P_c, Q_c \rangle$.

$Coordinator \in Op$ is a cooperator who coordinates the cooperation process of the group.

$D_c = \langle \{Coordinator\} \times Op \rangle \cap d_{ij}$ contains all the *delegation* relations involved with *coordinator*.

$P_c = \{p_{cj} \mid Coordinator, j \in D_c\}$ contains all the set of goals that are delegated by *coordinator* to other cooperators.

$Q_c = \{q_{cj} \mid Coordinator, j \in D_c\}$ contains all the set of scenes that are related with those goals delegated to other cooperators.

A *group*, $\langle Coordinator, D_c, P_c, Q_c \rangle$, defines a cooperative group coordinated by the *coordinator*. The group consists of those cooperators who undertake the delegated goals of *coordinator*. The cooperative relationship between them will be decided by the logical relationship of those delegated goals.

According to the *group* relation, the *hierarchical dynamic cooperative system* can be defined as follow.

Definition. A *hierarchical dynamic cooperative system(HDCS)* is a *group* coordinated by $a_0 < a_o, D_{ao}, P_{ao}, Q_{ao} \rangle$. G contains all the goals need to be achieved by the group. The group is structured according to the rules as follow.

1. $G \subseteq cpb(a_o)$. It means a_o can achieve the goals of the group alone. Then cooperation with other cooperators is unnecessary. So, $D_{ao} = P_{ao} = Q_{ao} = \emptyset$;
2. $G \not\subseteq cpb(a_o)$, that is $\exists g \in G' = (G - cpb(a_o))$. Those goals in G' can not be achieved by a_o alone. He needs to cooperate with other cooperators. So, the group consists of those cooperators that help a_o to achieve goals in G' . For each cooperator $b_j < a_o, b_j \in D_{ao}$, P_{aoj} contains all the subgoals that a_o delegates b_j to achieve, and Q_{aoj} contains all the cooperative scenes related with those subgoals in P_{aoj} .
3. For each b_j in the group, if $P_{aoj} \subseteq cpb(b_j)$, b_j can achieve those subgoals in P_{aoj} alone. Otherwise, b_j should coordinate another group to achieve those subgoals in P_{aoj} .

From the definition, it can be concluded that the *hierarchical dynamic cooperative system* consists of cooperative groups nested recursively according to the cooperators' capabilities and their goals. It can also be concluded that the *hierarchical dynamic cooperative system* is a complete cooperative structure.

Theorem. A Dynamic Hierarchical CSCW system (DHCS) is a complete cooperative structure.

Proof: For each cooperator a_i in the *HDCS* system, its goals can be achieved in 2 different ways.

1. $g \in cpb(a_i)$. It means the goal g can be achieved by cooperator a_i in isolation.
2. $g \notin cpb(a_i)$. The goal g should be partitioned as several subgoals $g' \in dsg(g)$ and delegated to other cooperators, b_j , to achieve, which form a group coordinated by a_i . According to the *delegation* relation, $\bigcup_{<a_i, b_j> \in D_{ai}} p_{a_i b_j} = \bigcup_{g \in ptn(a_i)} dsg(g)$. So, for each goal g

which can not be achieved by a_i alone, it can be achieved by a certain group coordinated by a_i . ■

4 The Cooperation Model DHCS

4.1 Architecture of Cooperator in DHCS

The cooperation model consists of cooperators at different layers. In each layer, cooperators participating in a common cooperative goal form a group, and they have a common parent coordinator in the upper layer, which coordinates their cooperation. The coordinator is also a cooperator of some a group in the upper layer. In this way, all cooperators can be arranged to form a hierarchical structure based on *group cluster*, which is specially proposed for DHCS, as depicted in figure 1. The cooperator at top layer coordinates the cooperation of the uppermost group, and the cooperators at bottom layer cooperate under the direction of their common superior.

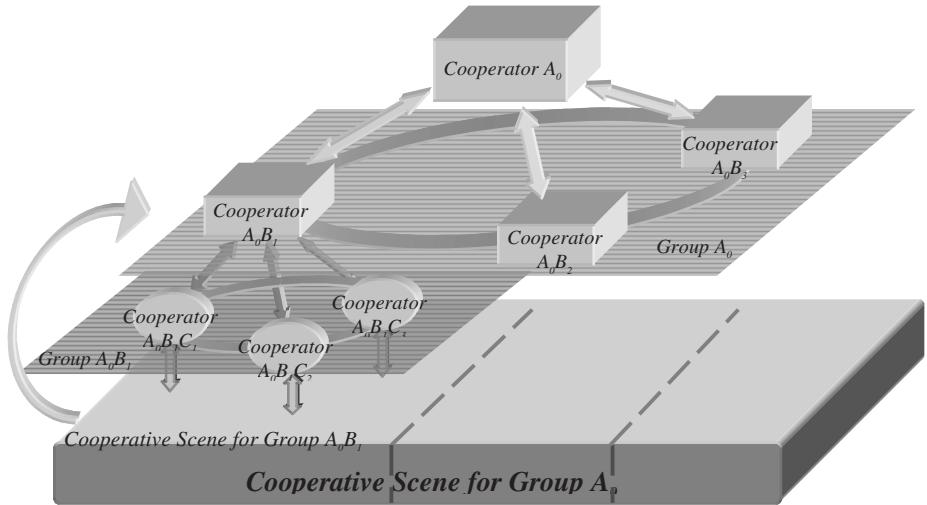


Fig. 1. The hierarchical structure of the DHCSCW model

As the basic elements of cooperation systems, cooperators should contain such info as follow:

Cooperator ::= {ID, GroupID, GroupScene, Scene, GroupGoal, Goal, FriendsID, MembersID}.

A cooperator ID provides the identifier you need to uniquely specify a participant within each group. It is defined as follow.

Definition. Cooperator's ID

Each cooperator's ID contains two parts except for that one at the top layer. One is its unique symbol in its group; the other is its parent coordinator's ID. It can be illustrated as below.

$$ID_i = \begin{cases} Identifier & i=0; \\ ID_{i-1} \cup Identifier & i>0 \text{ and } i \in \mathbb{Z}. \end{cases}$$

ID_i indicates a cooperator at the i th layer of the system. *Identifier* is its unique symbol in its group. ID_{i-1} is the ID of its parent coordinator at the $(i-1)$ th layer. The recursive definition of *ID* will make sure that each cooperator will be assigned with a unique identifier, no matter which layer it lies in or which group it joins.

Definition. Cooperator's GroupID

GroupID indicates which group the cooperator belongs to.

Since all members of a group have a common upper coordinator that can be identified uniquely in the system, the group can be named with its coordinator's ID. It implies that members of a group have the same *GroupID* info.

Definition. Cooperator's GroupScene and Scene

GroupScene is the collection of info about the cooperation condition of the group that the cooperator belongs to, while Scene is the collection of info about its own cooperation condition (or the cooperation condition of lower group that the cooperator instructs).

All cooperators of a group occupy the identical *GroupScene* info. Each cooperator can decide its global cooperative behavior independently based on *GroupScene* without any interaction with other cooperators. Each cooperator resolves the *Scene* from *GroupScene* for its cooperation. The *GroupScene* equals to the union of each cooperator's *Scene*. But it should be pointed out that the *Scene* would alternate due to the dynamic outer environment and be inconsistent with the *GroupScene* during the cooperation.

Definition. Cooperator's GroupGoal and Goal

GroupGoal is the global goal to be reached by the whole group, while the *Goal* is that to be reached by the cooperator itself (or the lower group it coordinates).

The *GroupGoal* is shared by all cooperators of the group. Each cooperator resolves the *Goal* from *GroupGoal* independently. The *GroupGoal* equals to the union of each cooperator's *Goal*. They will keep equivalence during the whole cooperation process. According to the layered relationships between group members and their coordinator, it can be concluded that the *GroupGoal* of each group member equals to the *Goal* of their coordinator.

Definition. Cooperator's FriendsID and MembersID

FriendsID is a collection of ID referring to the friends of the cooperator in the same group. *MembersID* is a collection of ID too, which indicates those cooperative members of the group coordinated by the cooperator.

4.2 Hierarchical Structure of DHCSCW Based on Group Cluster

In traditional *tree-like* hierarchical structure, there is not any direct links between children of a common father. Those children can only directly communicate with their father, and communicate each other indirectly under the father's conduction. The *tree-like* structure cannot be applied in DHCSCW, and some new structures should be proposed.

A new hierarchical structure, which is named *Group Cluster* and shown in figure 2, is formed through adding paths between nodes at the same level of the *tree*.

The *group*, a cell of *Group Cluster*, consists of a common upper node and some children belong to it. Besides of the paths between upper node and its children, those children are directly connected through paths between them. Since each child of a group can also have its own offspring, the *group* can be expanded hierarchically.

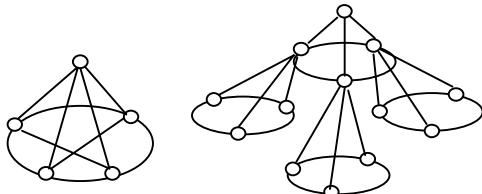


Fig. 2. Hierarchical Structure Based on *Group Cluster*

5 Dynamic Hierarchical Cooperation in DHCSCW

Now we demonstrate the dynamic hierarchical cooperation process in the model. We use $C(ID)$, $G(ID)$, $D(ID)$, and $S(ID)$ denoting cooperator, cooperation group,

cooperation goals that the cooperator undertakes, and cooperation scenes related with its goals.

An experimental system consisting of 7 cooperators is constructed as a 3-level bintree, as shown in figure 1. Cooperator $C(A_o)$ is the root node in the system. Cooperators $C(A_oB_1)$ and $C(A_oB_2)$ form the group $G(A_o)$ coordinated by $C(A_o)$. Cooperators $C(A_oB_1)$ and $C(A_oB_2)$ coordinates group $G(A_oB_1)$ and $G(A_oB_2)$, which consists of cooperators $C(A_oB_1C_1)$, $C(A_oB_1C_2)$ and $C(A_oB_2C_1)$, $C(A_oB_2C_2)$ respectively.

The hierarchical dynamic cooperation in the system can be decomposed as 3 related parts.

5.1 Generation of Cooperation Scheme

Each cooperator (except for $C(A_o)$), receives group info from its superior, which includes the cooperation goals achieved by the group and related cooperation scenes. It is the *global data* that shared by all cooperators of the group and be identical for them. Applying the same cooperation algorithm in the group, each cooperator gets its cooperation scheme from its *group cooperation engine* based on those *global data*. The cooperation scheme contains the cooperation goals that the cooperator undertakes and related cooperation scenes. If the cooperator coordinates another group at lower level, its cooperation scheme should be sent to all of its inferiors as *global data*, too.

5.2 Horizontal Active Intra-group Cooperation

Now let's examine the cooperation process in group $G(A_oB_1)$ to demonstrate the horizontal active intra-group cooperation.

Each cooperator of group $G(A_oB_1)$, that is $C(A_oB_1C_1)$ and $C(A_oB_1C_2)$, cooperates according to its cooperation scheme. During their cooperation, each cooperator will explore the unpredictable alternation of related cooperation scene, $S(A_oB_1C_1)$ and $S(A_oB_1C_2)$, as their *local data*. Furthermore, the *local data* will be merged into *synthesized data*. Each cooperator evaluates its cooperation efficiency on the newly synthesized data. The cooperation efficiency will vary under dynamic circumstances. If the efficiency of one cooperator, suppose $C(A_oB_1C_1)$, lowers than the predefined threshold, it means the cooperation scheme can not fit for the altered environment and need to be adjusted. So, cooperators $C(A_oB_1C_1)$ and $C(A_oB_1C_2)$ exchange their *synthesized data* to form the newest *global data* and reproduce their cooperation scheme respectively. Then the cooperation scheme is fit for the altered cooperation scenes again.

The adjustment of cooperation scheme in group $G(A_oB_1)$ need not be coordinated by its superior $C(A_oB_1)$, and can be achieved through info exchange among the group. So, cooperators can cooperate actively and adaptively in horizontal mode.

5.3 Vertical Hierarchical Inter-group Cooperation

The alternation of cooperation scene $S(A_oB_1C_1)$ and $S(A_oB_1C_2)$ will not only lead to the horizontal cooperation in group $G(A_oB_1)$, but also promote the vertical cooperation between group $G(A_oB_1)$ and $G(A_o)$.

When cooperator $C(A_oB_iC_j)$ and $C(A_oB_jC_i)$ exchange their *synthesized data*, the data will be sent to their superior $C(A_oB_i)$. Due to the *dependency* relation between cooperation scenes, the alternation of scene $S(A_oB_iC_j)$ and $S(A_oB_jC_i)$ will be collected by cooperator $C(A_oB_i)$ and merged into its *local data* and *synthesized data*. As the cooperation scene altered, cooperator $C(A_oB_i)$ needs to evaluate its cooperation efficiency. If the efficiency is lower than the threshold, it means the cooperation of group $G(A_o)$, containing cooperator $C(A_oB_i)$ and $C(A_oB_j)$, is affected by the alternation of cooperation scenes in group $G(A_oB_i)$. The cooperation scheme of $C(A_oB_i)$ and $C(A_oB_j)$ should also be adjusted. Then cooperator $C(A_oB_i)$ and $C(A_oB_j)$ will exchange their *synthesized data* and reproduce their cooperation scheme respectively.

It can be concluded that the alternation of intra-group cooperation scheme will lead to the adjustment of inter-group cooperation scheme when the efficiency of the group is too lower to accept. Cooperators of different groups at adjacent levels can cooperate vertically and automatically.

6 Conclusion

Hierarchical dynamic cooperation cannot be realized in traditional cooperation models. Based on the formal description of hierarchies and dynamics in cooperation systems, a hierarchical dynamic cooperation model DHCSCW is proposed in this paper. Compared with traditional cooperation models, the proposed model can:

1. realize both horizontal active cooperation in a group and vertical hierarchical cooperation between groups at adjacent levels;
2. realize the control of info propagation in hierarchical systems;
3. be applied for applications of any scales because of its hierarchical structure;
4. improve the performance of fault-tolerance because of its redundant shared cooperation info distributed in cooperators at different groups.

References

1. Hai-Bing Zhu, Research on Multimedia Collaboration Techniques Based Object-Oriented Methods Ph'D Dissertation, National University of Defense Technology, 1996.9
2. Mariani J. and Roddon T. Cooperative Information Sharing: Developing a shared object Service. The Computer Journal, Vol.39, No.6, pp. 455–470, Dec, 1996.
3. A. Prakash. *et al*, DistView: Support for Building Efficient Collaborative Applications using Replicated Objects, in Proc. of ACM CSCW'94, pp.153–163. Chapel Hill, NC, USA.
4. Marian H. Nodine, et al A Cooperative Transaction Model for Design Database in Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers, 1996.
5. A. J. Boner, The Power of Cooperating Transactions, <http://www.cs.toronto.edu/~bonner>.
6. Qian Mo, Research and Implement of Cooperative Transaction Agent Ph'D Dissertation. National University of Defense Technology, 1999.5.
7. Peter J. Kammer, et al Techniques for Supporting Dynamic and Adaptive Workflow To be appeared in Journal of CSCW.

8. G. Alonso, *et al*, Functionalities and Limitations of Current Workflow Management Systems. Technical report, IBM Almaden Research Center, 1997.
9. Sarit Kraus, Negotiation and Cooperation in Multi-Agent Environments, a Lecture presented in 14th International Joint Conference on Artificial Intelligence, Montreal, Canada, August, 1995.
10. M. Tambe, Implementing Agent Teams in Dynamic Multi-agent Environments, in Proc. of the International Conference on Multi-agent Systems (ICMAS), Dec. 1996.
11. J. Bardram, Designing for the Dynamics of Cooperative Work Activities, in Proc. of ACM CSCW'98, pp.89–98, Washington, USA, 1998.
12. R. B. Smith, *et al*, Supporting Flexible Roles in a Shared Space, in Proc. of ACM CSCW'98, pp.197–206, Washington, USA, 1998.
13. W. J. Tolone, *et al*, Specifying Dynamic Support for Collaborative Work within WORLDs, in Proc. of 1995 ACM Conf. on Organizational Computing Systems (COOCS'95), Milpitas, CA, Aug. 1995.
14. New Webster's Dictionary & Thesaurus of the English Language, Lexicon Publications, Inc., 1991.
15. Gilbert, D. The Role of Intelligent Agents in the Information Infrastructure. *IBM Intelligent Agent Center*. <http://www.raleigh.ibm.com/iag/iaghome.html>.
16. Bond, A. H., An Analysis of Problems and Research in DAI. *Distributed Artificial Intelligence*, 3–36, Bond, A. H. Eds., Morgan Kaufmann, 1988.
17. Wooldridge, M. J.. Agent Theories, Architectures, and Languages: A Survey. In *Intelligent Agents: ECAI-94 Workshop on Agent Theories, Architectures, and Languages*, 1–25. Lecture Notes in Artificial Intelligence, Vol. 890, Springer-Verlag.
18. Wen Cao, Achieving Efficient Cooperation in a Multi-Agent World: the Twin-Base Agent Modeling Approach. Ph'D Dissertation, Department of Computer Science, University of Tromsø, N-9037 Tromsø, Norway, Aug. 1997.

The Tenure Duty Method (TDM) in the Active Incident Recovery Research*

Zunguo Huang

School of Computer,
National University of Defense Technology,
Changsha, Hunan, P.R. China 410073
zuuguo@yahoo.com

Abstract. When on duty, a server should take charge for everything needed to operate the network information service system, including the switching between service and backup. That is the main idea of tenure duty method (TDM). It is a sharp contrast with the cluster method which has an extra front end load balancer and the domain name resolution method with a dedicated DNS. The idea came from the survivability research, which emphasized the idea of mission critical and sought the goal of recovering the ordinary services rapidly assuming the essential services being secure. So minimize the essential service set is one of the way of trading performance cost for survivability. The implementation of autonomous connection migration and free competition mechanism are the technical support of TDM. The survivability situation of the server is described with the Petri net expression. This is follow by the description of TDM. It featured temporal stochastic tenure and spatial Brown motion model. And finally with the experimental model and the mathematical tool, the paper analyzed the related index showing the survivability of TDM quantitatively.

1 Introduction

Survivability means the ability of information system's continuing to provide service under the catastrophic situation such as being attacked or malfunctioned and so on [1]. The research interesting nowadays is focused on the emergency response and incident recovery problems of information service systems. As one of the component of information assurance technique framework [2] [3], survivability is mainly hammering at the rapid recovering of the ordinary services with the supposition that the essential services are secure. The larger the essential services set which must be protected secure any way, the higher the cost which must be paid for survivability. So minimize the essential services set is one of the ways of trading performance cost for survivability. This paper is mainly for solving the dynamic roaming [4] [5] problem in the survivability research, and proposes a TDM method. It is featured no centralized control avoiding the essential services set expansion and no new security bottleneck added. The randomicity in temporal and spatial realized the target concealment and

* This work is supported by national 863 program under contract No. 2001AA142090.

incident obviation assumption. In the following parts of this paper, the related research in this area is first introduced, and then the method is described in both temporal and spatial viewpoint. Based on this a realized instance and evaluation of its survivability measurement is given at last.

2 The Related Works

Using the research fruits of availability for reference, and taking the special problem raised in the information attack environment into consideration, the connection migrate problem involved the following research contents. The IP redirection [6] fixed the connection migration problems within the same network section. The dynamic DNS resolution [7] can also contribute to the solution of the service roaming problem though its original function is the client IP address migration and agility. Besides these methods, the TCP migration techniques [8], the mobile IP techniques [9] and so on, also made contribution to the solution of the problem. But after all, due to these techniques original goal were not for survivability, they have some shortcomings such as the temporal and spatial restricts, or the addition of some centralized control mechanics. This rose the survive risk and formed new vulnerabilities to some extend. Therefore, the new method must meet the following needs: firstly, add as less as possible the essential service set which needs special protection or is indispensable, and add as more as you can the survival mechanics which can mutual benefit each other. Secondly, make the next alternation of the server unpredictable as where it will be to increase the detect time of hackers. And finally, do away with the life tenure rule of the service nodes, and realize the stochastic interval tenure rule, for the sake that although the hacker may succeed temporarily, they cannot benefit from it once for all.

3 Description of TDM

The main idea of TDM is as following. Information service is provided by a set of servers called backup pool. In the pool only one server can provide service outside and the service period is confined into a stochastic time slice called tenure. When on duty, a server should take charge for everything needed to operate the network information service system, including the switching between service and backup.

This method is developed on the base of no centralized control. It is a sharp contrast with the cluster method which has an extra front end load balancer and the domain name resolution method with a dedicated DNS. It is composed of the spatial Brown motion model and the temporal stochastic interval tenure rule. The following definitions are given for the description convenience.

Definition 1. The service node is defined as the node that the client can access. In the service node, the roaming schedule process is activated and the healthy status is under the inspection of the other nodes.

Definition 2. The backup pool is defined as the set of partial mirrored server. In the pool, all nodes can afford the same service without the limitation of geography location, and they synchronize the scene information with the service node by each

connection. The server need not completely mirrored and the other nodes can process a connection with the failed server.

Definition 3. Health detection detects whether the service node is still on duty, and has the authority to decide if the connection should be migrated or multicast the migration message. The migration kernel assures that once the migration trigger is activated, the present service node will no longer on duty and the backup servers in the backup pool take over the client which is still on line via competition.

Definition 4. Petri net [14] of a node is based on the definitions above, and the net is shown in figure 1. The positions and transitions description is given below.

Service: when node in service transition, it activate the ordinary roaming schedule process and refresh the scene message timely.

Migration: when node in migration transition, the node no longer offering service and multicast this information to every nodes in the backup pool to start the free competition in the pool.

Competition: when node in competition transition, node try to take over the connection with the served, and the client will respond the first coming one and refuse the others.

Trigger: Position trigger start the connection migration.

Finished: Position finished complete the migration.

Succeed and failed: Position succeed and failed are the result of competition.

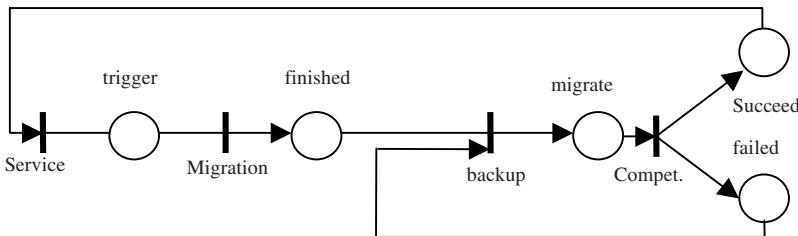


Fig. 1.The Petri net expression of TDM

Definition 5. Continuity of service C can be repressed with the average service time s and the average roaming time r. that is :

$$C = s / (s + r) \quad (1)$$

Notice that s and r is related to the whole backup pool. Due to timely roaming makes the recover process left as background action, recover time hiding is then implemented.

Definition 6. Randomicity R of roaming can be expressed by the entropy of probability of roaming to node i P_i , that is:

$$R = -1 / \ln(n) \sum_{i=1}^n P_i \ln(P_i) \quad (2)$$

Here n is the total number of nodes in the backup pool; $1/\ln(n)$ is used for making the value of R between 0 and 1. R is 1 when the probability of all nodes in the pool is equal.

Definition 7. Fall rate ρ is defined below:

$$\rho = P\{(T_d < T_t | T_u < T_d) \cup (T_u < T_t | T_u > T_d)\} \quad (3)$$

Here T_d , T_t and T_u are stochastic variables and stands for the detect time of a hacker, the tenure of the server and the time interval between adjacent arrived attacks respectively. For instance, when T_d and T_t obey exponential distribution with average value d and t respectively, and suppose $T_u = u$ is constant, according to the renewal theorem [10] and the fact that T_d and T_t are mutual independent, the result is:

$$\rho = (t/(t+d))e^{-u/t} + (d/(t+d))e^{-(u/t+u/d)} \quad (4)$$

It is seen that under certain average detect time, the shorter the average tenure, the lower the fall rate. The detect time is chosen out of the reason that detect is statistically the most time consuming job among the three stages (detect, intrusion and exploring) of attack.

Note that the recovery process of a node is left as a background action due to the timely roaming. So the offline repairing is hiding.

3.1 The Brown Motion Model

The model is on the spatial roaming rule for the service system, two algorithms are given here:

```
transition=service;
if triggered{
    retire;
    transition=backup;
}
if damaged{
    mending in the background;
    if recovered then backup}
```

The method when server is in backup:

```
Transition= backup;
probing status of the server;
if retired{
    According to the synchronized scene message try to
    continue the ongoing service for the client;
    if granted then transition=service else
    transition=backup;
}
if damaged{
    mending in the background;
    if recovered then backup}
```

The method can assure the service node stochastically roaming like the Brown motion in two ways. Firstly, the time the backup server finds the service server no longer on duty is in no order for the query action for every node is completely autonomous. And secondly, the action backup servers send the connection takeover signal is also in no order. As to the client, recently FCFS policy is adopted, and could be another parameter to adjust the stochastic measurement.

3.2 Stochastic Interval Tenure Rule

The temporal roaming rules for an information service system are described here. A 32 bit stochastic sequencer with Feedback polynomial:

$$F(X) = x^{31} + x^6 + x^4 + x^2 + x + 1$$

is used here. This feedback register can be proved as a non strange one. It is known from the sequence cipher generation principle that when the original vector is random, the output 0 and 1 sequence of it is unpredictable. It is shown from several experiments that when the probe period is 1 second, the repeat cycle of this sequencer above 33 years and hence can meet the requirement of the roaming schedule.

Now, we can chose a constant time mark (60s, for example), every time at the mark the shift register make an operation. And once the output of this operation bit0 equals 1, it will trigger the migration mechanism. So generate a stochastic process with the average time equal two time marks Bernoulli distribution. And then implemented the schedule policy featured fair spread time mark and randomly sampling. It is shown from experiment that this schedule method is stable, reliable and has the effect of anticipative desire. Notice that the time mark T has relation to the security threaten grade w , the backup refresh interval τ , sensitivity of job k , network protection level I , etc. they have the following relationship:

$$T \propto \tau / w k_i$$

4 The Implementation and Result Evaluation

For implementation of TDM, a model diagram relied by the method taking the TCP migration for example is shown in fig.2. We have a simple test procedure described below:

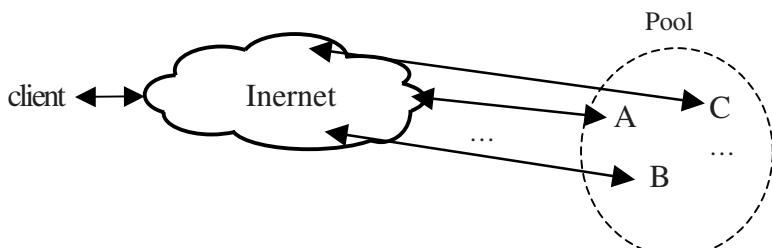


Fig. 2. The experiment model

Start the schedule process on all servers in the backup pool, but only the one on the service node is activated. Once migration happened, the service node multicast the retire message to all the nodes in the pools and then stop the schedule process on this node. Hence the free competition mechanism starts at once. Every node is competing for the next successor and the succeeded one becomes the next service node. It activates its roaming schedule process. The action is going on repeatedly for the next round.

In the test the time mark is 60 second. 62 periods were checked and 31 times roaming were recorded. The server on duty sequence is:

ABBACCBBAABAAACBBABCCAAACCBACBAABBBBBCCCACAABCCCCAAAABB
BCCCCBA

Table 1 and 2 are the test data:

Table 1. The roaming of servers and their roaming time (s)

Roam	A-B	B-A	A-C	C-B	B-A	A-B	B-A	A-C	C-B	B-A	A-B	B-C
Time	3	6.4	5	4.6	6	7	2	3.2	4	7	5.9	6.2
Roam	C-A	A-C	C-B	B-A	A-C	C-B	B-A	A-B	B-C	C-A	A-C	C-A
Time	9	12	11	6	4.5	4	3	8	7.2	6.4	5	5
Roam	A-B	B-C	C-A	A-B	B-C	C-B	B-A					
Time	4	5	3	7	8	13	12					

Table 2. The frequency of a server on duty

node	A	B	C
frequency	12	10	9
Service time(60s)	22	20	20

From the data above, we get the following results:

$$T_s=120s$$

$$T_f=6.24s$$

$$C=95.05\%$$

$$R=99.34\%$$

$$\rho=6.25\% \text{ (Suppose that } e=180s \text{ and } u=360s)$$

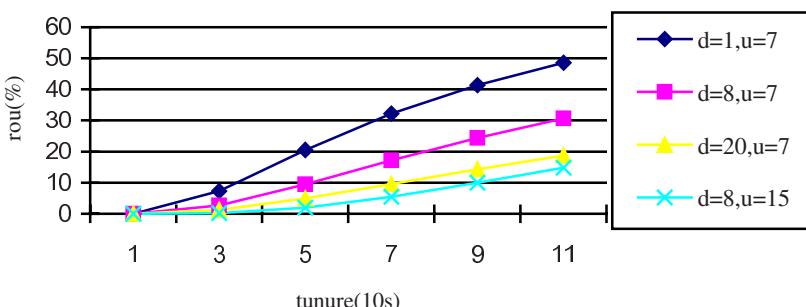


Fig. 3. The t/ρ relation curve

Compared with the domain name resolution or IP redirection, the fall rate and randomicity are improved tremendously and with higher survivability, though the continuity has degraded slightly.

5 Summary

The TDM method proposed here implemented the dynamic roaming policy with stochastic and autonomous property in temporal and spatial aspects. It is an experiment of the application research in emergency response and incident recovery. The difference between the breakpoint reconnection and TDM is that the former related no node roaming and need reconnection. As to cluster technique, the newly added front end administrator becomes new security bottleneck.

As to the stochastic performance, the stochastic sequencer can assure the temporal randomness and free competition mechanism response for spatial roaming. The tenure duty rule and free competition mechanism get rid of the adding of new security bottleneck. So it is a creation with survivability meaning.

Make flexible use of the roaming trigger mechanism can realize the different application of the technique. We will promote the study of real application of TDM and the evaluation model of survivability in the near future.

Reference

1. Robert. J. Ellison, etc: "Survivability: Protecting Your Critical System"
<http://www.sei.cmu.edu/organization/programs/nss/protect-critical-systems.html>, 7,Feb,00
2. The Information Assurance Technical Framework (IATF V3.0); Issued by: National Security Agency, Information Assurance Solutions, Technical Directors IATF Release 3.0. September 2000
3. Hu Huaping, Huang Zunguo and Pang Lihui, etc .. "In-depth Defense and Assurance Framework for Network Security"; Computer Engineering & Science; 2002, Vol.24,No.6; Page7
4. Huang Zunguo, Lu Xicheng and Wang Huaimin: Journal of National University of Defense Technology; 'Survivability and its implication framework'; 2002,Vol.24,No.5; Page29
5. Huang Zunguo, Lu Xicheng and Wang Huaimin: "A Diversified Dynamic Redundancy Method Exploiting the Intrusion Tolerance"; ISW-2000 proceedings;2000.10. Boston, USA , Page 217-221
6. Yang bing, Huang Zunguo and Hu Grangming: "The Study of Dynamic Migration Technique Based on High Availability"; Computer Engineering & Science; adopted
7. Dynamic DNS, <http://www.technopagan.org/dynamic/>
8. Alex C. Snoeren, David G. Andersen, and Hari Balakrishnan : "Fine-Grained Failover Using Connection Migration" <http://nms.lcs.mit.edu/software/migrate/>
9. C.Y. Chen: Mobility with the Internet and IP Networks; <http://www.mobile-ip.com.cn/>
10. Loenard Kleinrock: 'Queuing system'; Aviley-Interscience Publication; 1975.
11. Huang Zunguo, Hu Grangming, Ren Jianrong: "On a Skeleton of Rapid Response and Recovery r-RR) from Info-Sec Incidents"; Computer Engineering & Science; 2001; Vol.23, No.6; Page 43

12. Anotai Srikitja ,etc, "On Providing Survivable QoS Services in the Next Generation Internet"; Supported in part by NSF grant NCR 9506652 and DARPA under agreement No. F30602-97-1-0257;
13. Huang Zunguo, Lu Xicheng and Wang Huaimin, "On diversity Dynamic Backup and Survivability techniques"; Chinese Institute of Electronics Electronic system engineer section 9th C3I theory learning seminar; 2000.11, HeFei, Page 72
14. Lin Chuang: 'The Stochastic Petri net and System Performance Evaluation' Tsinghua University Publication; 2000.1

A Combined Continuous-Time/Discrete-Event Computation Model for Heterogeneous Simulation Systems

Andreas Franck¹ and Volker Zerbe²

¹ Mission Level Design GmbH Ilmenau,
Ehrenbergstrasse 11,
98693 Ilmenau, Germany

² Technische Universität Ilmenau,
Department of Automatic Control and System Engineering,
P.O.Box 100565,
98684 Ilmenau, Germany

Abstract. Complex electronic systems contain components that have to be described by many different model types. An efficient design process requires to validate these models through all phases of development, [7]. It is therefore required to have multi-domain tools that can analyze these complex systems in an integrated way. MLDesigner a design tool of the latest generation is in the process of developing, [9]. In this paper, we describe a model of computation that combines continuous-time and discrete-event elements. We show that the developed formalism is well suited for frameworks like MLDesigner supporting heterogeneous modeling and simulation.

1 Introduction

1.1 Heterogeneous Modeling and Simulation

Heterogeneous modeling and simulation permits analysis and design of complex systems.

Instead of modeling a system as a whole, using a complex description language, in heterogeneous modeling every components is modeled by a suitable representation.

These submodels are combined and executed together to analyze the overall system behaviour.

An important research project in the field of heterogeneous modeling and design is the Ptolemy Project, at the University of California, Berkeley. Research focusses on modeling, simulation, and design of concurrent, real-time, embedded systems. In this project, two application programs have been developed, Ptolemy 0.x [1], now called Ptolemy classic, and Ptolemy II [8], which is the current research environment. Throughout this paper, we refer to the heterogeneous modeling and simulation concepts of the Ptolemy project and furthermore MLDesigner.

Essential to the concept of heterogeneous simulation is the notion of a *model of computation* (*MoC*). A model of computation defines the semantics of the models formulated in this MoC. Many models of computations have been developed and are in use, although in limited fields of application. Examples of computational models are Finite State Machines, Petri Nets or Dataflow graphs.

A model of computation defines the following characteristics of a model:

- The way how behaviour of the model is described.
- How these model descriptions are executed.
- For the case that models are described by sets of components, the model of computation defines how model components interact.

In Ptolemy, a model of computation is encapsulated in a *domain*. A domain is a set of C++ classes in which all aspects of simulation specific to the computational model are implemented.

Heterogeneous simulation frameworks often allow arbitrary couplings of domains. This requires that all domains share a common interface which permits interaction between heterogeneous model parts. This interface has to define:

- How information is exchanged between different components.
- How to coordinate the execution of interacting components.

In Ptolemy classic, this interface is called *EventHorizon* [2].

1.2 Continuous-Time Computation Models

Computation Model The characteristic property of continuous-time formalisms is that the trajectories of input-, output- and state-values are continuous in time.

Mathematically, this class of systems is modeled by systems of ordinary differential equations (*ODE*). The change of a state variable is described by its time derivative $\dot{x}(t)$.

$$\dot{x}(t) = f(x, u, t) \quad (1)$$

$$y(t) = g(x, u, t) \quad (2)$$

A system model is described by its differential equation (1) and an *output equation* (2), where $u(t)$ is the input, $x(t)$ is the state and $y(t)$ is the systems output.

Signal Form. At every instant of the simulation interval, inputs, outputs and states have a distinct value. Therefore, signals in continuous-time models are continuous functions. The resulting signal form is shown in Figure 1.

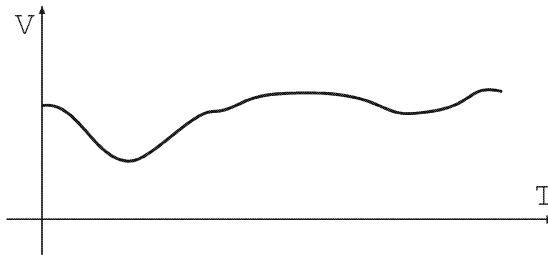


Fig. 1. Signal Form of a Continuous-Time Wave

Modeling. Besides mathematical representation like ODE and simulation languages as CSSL (*Continuous System Simulation Language*), graphical oriented modeling of continuous-time systems is becoming predominant. In applications like SIMULINK from Mathworks, models are created and manipulated as interconnected blocks, defining the dynamics of the system. Besides the easy-to-use interface, the visual representation often reflects the structure of the modeled system.

Furthermore, most visual interfaces are hierarchical, permitting the combination of submodels into modules and using these modules as blocks. This allows hierarchical structuring of complex models and reusability of existing components. These features are essential to modern system development.

Simulation. Simulation of continuous-time models uses numerical methods for solving systems of ordinary equations, more precisely initial value problems. These algorithms are often called numerical integration methods.

Starting with an initial value x_0 , the trajectories of the state and output values are approximated at a finite number of time points in the integration interval.

Numerical methods for solving ODE systems is an area of extensive research. A large number of different algorithms is available, with different accuracy, computational effort and suitability for distinct classes of problems. Therefore, continuous simulation applications must include different integration algorithms and offer capabilities to configure these algorithms according to the system to be simulated.

Limitations. These numerical methods generally require the differential equation $f(x(t), u(t), t)$ and input trajectory $u(t)$ to be smooth. More precisely, these functions must be, depending on the used ODE solver, sufficiently differentiable.

For real systems, these requirements are often not met. One cause is discontinuity in the state transition function. Systems with friction or hysteresis effects show such behaviour.

Often, input functions change their values discontinuously. Especially in systems where continuous components interact with digital devices.

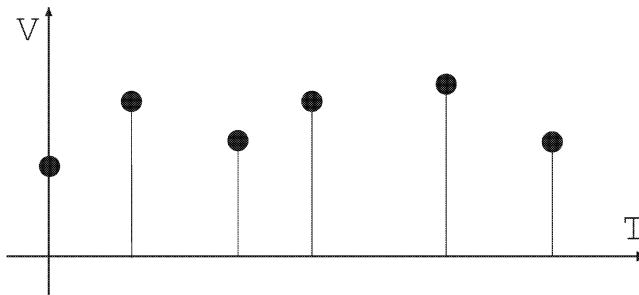


Fig. 2. Signals in Discrete-Event Simulations

Most numerical algorithms fail or decrease or have significantly reduced accuracy when stepping over discontinuity points. Therefore, practical simulation software must offer facilities to manage discontinuity points and handle them appropriately. This feature is usually called *breakpoint handling*.

1.3 The Discrete-Event-Formalism

In Discrete-Event (DE) models, evolution of the system state is modeled as a finite set of discrete state changes. These changes can occur at arbitrary distinct points of the underlying, usually real-valued, time base and are called *events*. A formal definition of this computational model can be found in [3], where it is introduced as DEVS (*Discrete Event Specified System*) formalism.

Modeling. In modern design-tools, Discrete-Event models are usually designed at the coupled model level and represented a graphical form. Examples of such tools are BONeS, a commercial network simulator from Cadence Design Systems, or the DE domain in MLDesigner.

In such coupled models, informations between different blocks are transmitted in form of messages associated with a usually real-valued timestamp. These messages are also called discrete events. Therefore, signals in Discrete-Event-models are sequences of events, as depicted in Figure 2.

Simulation. The heart of a Discrete-Event-simulator, often called *scheduler*, is a priority queue structure usually named *event queue* or *event calendar*. Model components schedule events to be processed by inserting them into this queue. The scheduler executes these events sequentially, according to the order of the time stamps.

Discrete-Event and Heterogeneous Simulation. Among different models of computation, the Discrete-Event-formalism has a special role. It has been stated that DE is suited to represent the input-output-behaviour of models formulated in other models of computation. Zeigler [3]:

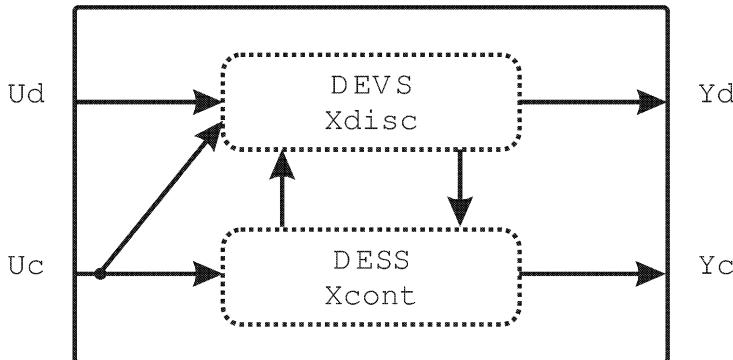


Fig. 3. Basic DEV&DESS-Model (according to [3])

“... DEVS is capable of expressing models formulated in the other major formalisms ..., so that these can be integrated into an all-DEVS simulation environment supporting multiformalism modeling.”

Therefore, the discrete-event-formalism is well suited to define the interface for communication between model components in a heterogeneous simulation environment.

2 Combined Continuous / Discrete-Event Models of Computation

In section (1.2), we described limitations of pure continuous-time formalisms. For real-world applications, this model of computation has to be extended.

Combined continuous/Discrete-Event models of computation [4] form a well-defined and especially expressive extension of pure continuous-time formalisms.

In [5], this formalism is introduced as DEV&DESS (*discrete event and differential equation defined system specification*) and formally defined. Here, we will give only a short, non-formal description.

2.1 The Computational Model

As depicted in Figure (3), a combined model consists of a continuous and a Discrete-Event component.

Both model parts may contain inputs, outputs and states of their respective types. Characteristic of the combined model of computation is the way these components influence each other. These interactions are also called events and can occur as:

Time events, related to events in pure Discrete-Event models. They are initiated in the discrete part, but possibly change the state in the continuous part, too.

External input Events, input messages that occur at the input ports of the Discrete-Event-partition. They are similar to time events and are handled in the same way.

State events, initiated by the continuous model part. A state event is triggered when a condition that depends on continuous state or input values, is satisfied. In general, this condition can be expressed by a *state event equation*:

$$C(u(t), x(t), t) = 0 \quad (3)$$

Only after an integration step algorithm can it be determined if a state event occurred within this time period. This is because the event condition depends on values that change continuously.

2.2 The CTDE-Domain

To investigate this computational model presented here, we developed a new domain called CTDE-domain (*Continuous-Time/Discrete-Event*). We developed representation forms for combined models, and simulation algorithms.

2.3 Model Structure

To represent combined continuous/discrete-event models, we chose a graphical form. Models are represented as graphs of interconnected blocks with input and output ports. We define the base element of the modeling language, the *general hybrid block*. This block is closely related to the atomic DEV&DESS model introduced in section (2.1). This element may contain both continuous and discrete-event elements, and have an arbitrary combination of discrete and continuous input and output ports.

As an example of a general hybrid block, the general integrator block is shown in Figure (4). Beside the continuous input and state output of a pure continuous integrator, it contains a discrete input for resetting the state value and a saturation output, which signals reaching the upper or lower limit as a discrete event.

Pure continuous or discrete blocks are special cases of the general hybrid and can be modeled using this atomic element. This allows a uniform representation of all blocks in a combined model.

Specific to the CTDE-domain is the coexistence of two distinct signal forms. Couplings between continuous ports are continuous waveforms and have value-semantics. Over connections between discrete ports messages are transmitted as discrete events.

Direct connections between discrete and continuous ports are not allowed in CTDE, but must be modeled by the distinct insertion of conversion Blocks. This allows static checks of the model structure before simulation startup and helps in avoiding modeling errors in the design.

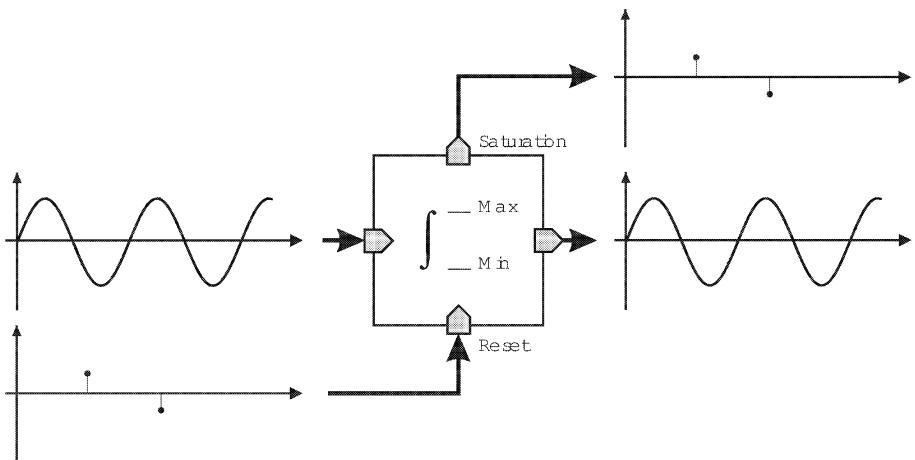


Fig. 4. Hybrid Block Example: General Integrator

2.4 Simulation Algorithm

Much like the model structure, the simulator is divided into a Discrete-Event and a Continuous-Time component. These parts interact as follows:

At a given time in the simulation interval, the Discrete-Event scheduler processes all events with the current timestamp. After that, the simulator changes to the continuous part. In this mode, the trajectory of the continuous states is approximated by a numerical ODE solver. At the time of the next scheduled discrete event, execution changes back to the Discrete-Event scheduler.

Detection and processing of state events requires special treatment. In the graphical model representation used in the CTDE domain, state event conditions are modeled as Blocks, called *state event generators*. Since state event conditions depend on continuous variables, they must be handled in the continuous part of the simulator. After each step of the ODE solver, the state event generator blocks are queried if a state event occurred during the current step. In this case, this integration step has to be rejected. Integration will be repeated with adjusted step sizes until the event condition is met with sufficient accuracy. For this case, the simulator immediately changes into the Discrete-Event mode and executes the action triggered by the state event as an ordinary discrete event.

Simulation algorithms for combined continuous/discrete-event simulators are treated in more detail in [4] and [3].

2.5 Example: *Bouncing Ball-Model*

Here, we show an example commonly used for demonstrating the handling of state-events. A ball bounces repeatedly on a surface. The impact is modeled by a state-event-generator, the block `ZeroCrossingDetector`, which ensures that the time of the hit is determined accurately.

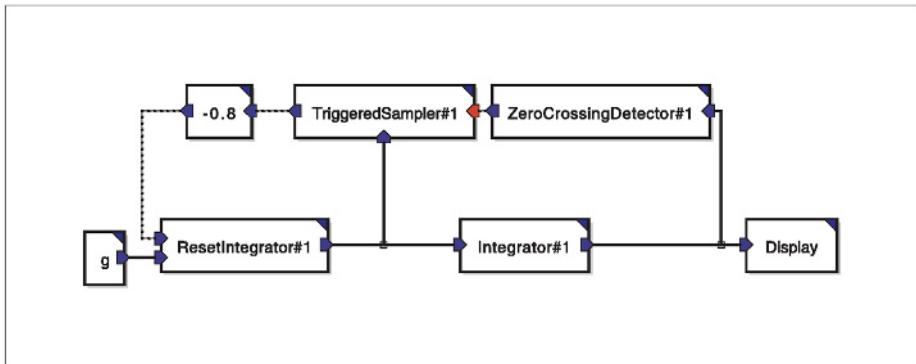


Fig. 5. Model of the bouncing-ball system

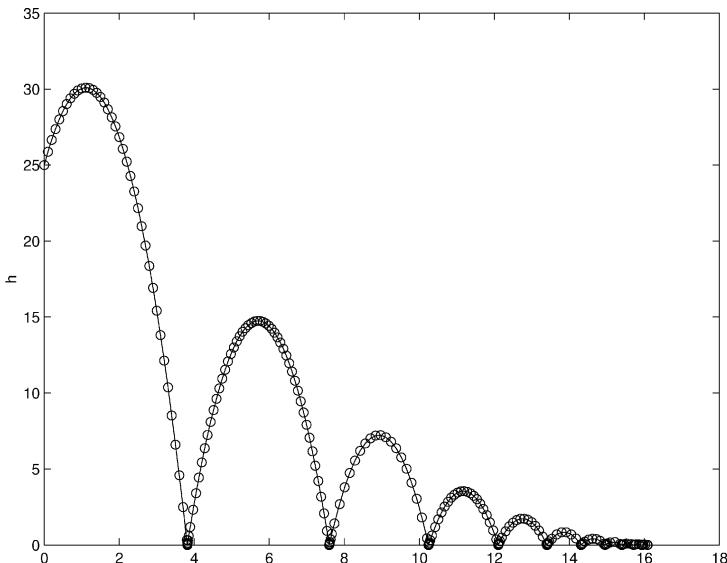


Fig. 6. Bouncing-Ball-Demo: Position of the ball

This model formulated in the CTDE domain shows the combination of continuous and discrete dynamics clearly. Whereas the movement of the ball is modeled with continuous elements and signals (solid lines), the signaling of the impact, calculation of the reflected velocity and inverting the movement is modeled using discrete logic (dotted lines).

3 Combined Formalisms in Heterogeneous Simulations

At the first glance, a combined model of computation contradicts the concept of heterogeneous modeling and simulation. Instead of modeling each subsystem in

an appropriate formalism and to combine these models for simulation, a larger and more complex formalism was created.

In our opinion, the combined continuous/discrete models of computation shows several advantages that make them especially suited for integration into heterogeneous simulation environments.

Therefore, we compare the CTDE domain with an implementation of a pure continuous-time formalism. As an example, we chose the CT-domain (*Continuous Time*) from Ptolemy II [6]. This domain implements a model of computation based on ODE's represented in a graphical form. Additionally, CT offers facilities to interact with Discrete-Event-models. This kind of model interaction is called *Mixed-Signal-Simulation*.

3.1 Signals

Because of their signal form, continuous signals cannot be represented by a finite amount of data. Therefore, continuous signals are not suitable for communication between model components. In section 1.3 it was stated that the Discrete-Event formalism, and the event sequence as the associated signal form, are suitable semantics for interaction between components in a heterogeneous simulation framework. Therefore, a continuous-time domain mixed-signal-simulation, or more generally, heterogeneous simulation, must implement facilities to handle discrete-event signals.

In Ptolemy II, two additional classes of blocks, *EventInterpreters* and *EventGenerators* are introduced to perform the conversion between continuous and discrete waveforms. These elements have to be inserted at the boundaries of continuous models.

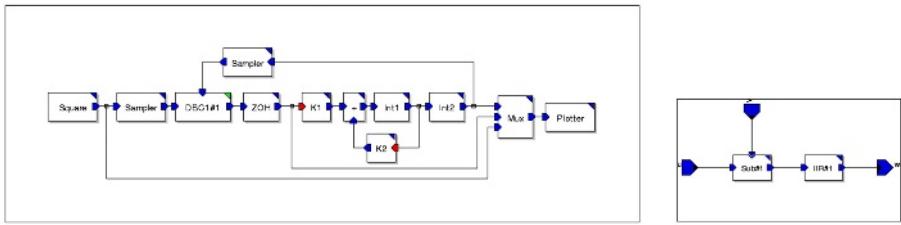
In CTDE, discrete-event signals are an integral part of the domain. Neither the computational model nor the hierarchy of the modeling elements has to be extended to realize interaction with heterogeneous components.

3.2 Modeling

The choice of the general hybrid block as the base element of modeling increases the expressiveness of the modeling language. There are elements that cannot, or not effectively, be expressed as combinations of discrete, continuous and signal-conversion blocks. The **ResetIntegrator** used in the Bouncing-Ball demo is an example for such an element.

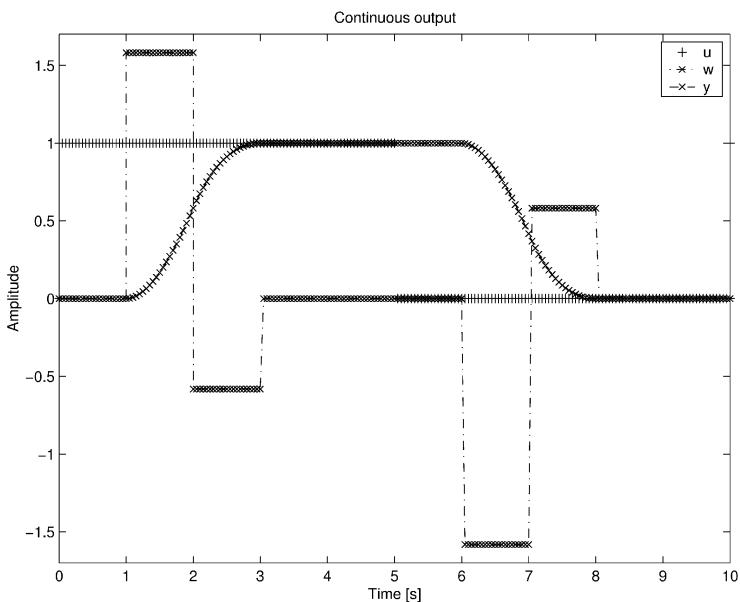
Another difference in the modeling methodology is the structure of a mixed-signal model. In a pure continuous-time computation model such as the Ptolemy II CT-Domain, every model part with discrete dynamics has to be encapsulated in a separate submodel. In CTDE of MLDesigner, the model structure can be chosen according to the logical partitioning of the system. Again, the Bouncing-Ball demo serves as an example where integration of continuous and discrete dynamics into a single model makes sense and is more concise.

In modeling environments like Ptolemy, composition of heterogeneous components is done by means of hierarchical embedding. A submodel formulated in



(a) Top-Level-Model with continuous plant

(b) Controller model as dataflow graph

Fig. 7. Example (3.4): Continuous-time plant with Discrete-Time Controller**Fig. 8.** Example (3.4): Continuous values

a different formalism is inserted as a block into the embedding model. Since heterogeneous components communicate with the CTDE domain through Discrete-Event semantics, these submodels are handled as pure discrete blocks. Since discrete elements form a special case of the general hybrid block, no special treatment is necessary. In pure Continuous-Time models of computation, these submodels form a separate class of modeling elements.

3.3 Simulation

As shown above, in the CT domain in Ptolemy II, the model representation was extended by three classes of blocks (EventInterpreters, EventGenerators and

bloccks representing submodels) to permit mixed-signal simulation. Since these element classes need to be handled by the simulator separately, this makes the simulation algorithms more complicate and cumbersome.

In contrast, the CTDE has need to extend the model of computation to allow interaction with heterogeneous model components. This results in a better structured simulator design and (presumably) better efficiency.

3.4 Example: Discrete Controller

As an example of a Mixed-Signal-model formulated in the CTDE domain, we present closed-loop control consisting of a continuous plant and a discrete-time controller (Figure 7). The discrete controller, more precisely a deadbeat-algorithm, is modeled as a dataflow graph in the SDF domain (Figure 7(b)).

The coupling between the model components is modeled by dedicated elements for signal conversion. These elements enable the designer to accurately model the behaviour of systems combining digital and continuous components.

The results of a simulation run are depicted in Figure (8).

4 Conclusion

In this article, we present a model of computation that combines discrete-event and continuous dynamics in a single model description.

This formalism forms a distinct model of computation and is especially suited for integration in heterogeneous modeling and simulation environments, implemented in MLDesigner. This extended computational model permits to simulate complex continuous and mixed-signal models accurately.

The use of well-defined and better structured simulation algorithms simplifies simulation software design.

References

1. Buck, J.; Ha, S.; Lee, E.; Messerschmitt, D.; Ptolemy: A framework for simulating and prototyping heterogeneous systems. *Int. Journal of Computer Simulation, special issue on Simulation Software Development* 4/1994, p. 155–182
2. Chang, W. T.; Ha, S.; Lee, E. A.; Heterogeneous simulation - mixing discrete event models with dataflow. *Journal of VLSI Signal Processing* 15/1997, p. 127–144
3. Zeigler, B. P.; Praehofer, H.; Kim, T. G.; *Theory of Modeling and Simulation*. second. ed. Academic Press, 2000
4. Cellier, F. E.; *Combined Continuous/Discrete System Simulation by Use of Digital Computers: Techniques and Tools*. PhD thesis, ETH Zurich, 1979
5. Praehofer, H.; Auring, F.; Reisinger, G.; An environment for DEVS-based multi-formalism simulation in common Lisp/CLOS. *Discrete Event Dynamic Systems: Theory and Applications*, 3/1993, p. 119–149
6. Liu, J. Continuous time and mixed-signal simulation in ptolemy ii. Ms report, University of California, Berkeley, 12/1998

7. Zerbe, V.; Mission Level Design of Control Systems. In: Proceedings of SCI/ISAS 5th International Conference on Information Systems, 1999, p. 237–243
8. II., J. D.; Goel, M.; Hylands, C.; Kienhus, B.; Lee, E. A.; Ptolemy II: Heterogeneous Concurrent Modelling and Design in Java. 1.0 ed. Department of Electrical Engineering and Computer Science, University of California, Berkeley, 2001
9. MLDesigner: <http://www.mldesigner.com>

Information Geometry on Modular and Hierarchical Neural Network

Yunhui Liu, Siwei Luo, Aijun Li, Hua Huang, and Jinwei Wen

Department of Computer Science,
Northern Jiaotong University,
100044 Beijing, China
{mount.tai,Jw_wen}@263.net
Swluo@center.njtu.edu.cn
laj1964@163.com
idopro@sina.com.cn

Abstract. Although theoretic research and applications have proved that modular neural network (NN) and hierarchical NN have notable effect to solve the complex problem, there is still little work on a mathematical framework about their structure mechanism. This paper gives a theoretic analysis based on information geometry. By studying the dual manifold architecture for modular NN and hierarchical NN and analyzing the probability of knowledge increasable based on information geometry, the paper also proposes a new modular and hierarchical model: multi-HME that has knowledge-increasable and structure-extendible ability.

1 Introduction

Information geometry [1][2] has been successfully applied to many fields such as information theory, neural network (NN) and others. It studies the intrinsic geometrical structure to be introduced in the manifold of a family of probability distributions. By using information geometry, the intrinsic geometrical structure properties of modular and hierarchical structures can be elucidated.

The idea of modularity and hierarchy in NN is driven by the inspiration from the neurobiological basis of the human brain. Modular NN is an embodiment of divide-and-conquer principle in which a complex problem is decomposed into simpler tasks which are settled by modules or sub-networks and solution is then reassembled from the results of the subtasks. Hierarchical design emphasizes on connecting smaller networks into a big one. We can construct a ‘network of networks’ and it can fulfill task better than any sub network, or, in some cases, can do a complex work that any of its sub-networks can’t do [3].

Although there is much research on modular and hierarchical NNs, little work has been done on their mathematical framework about their structure mechanism. In this paper we give an analysis based on information geometry and propose a knowledge-increasable model based on the idea of modular and hierarchical NN.

The rest of paper is organized as follows. Section 2 gives an information geometric analysis about modular and hierarchical structure of NN. Section 3 proposes a multi-

HME model having knowledge increasable learning ability and the final section draws some conclusions.

2 Modular and Hierarchical Analysis Based on Information Geometry

Consider an ANN with modifiable parameters $\theta = (\theta_1, \dots, \theta_n)$. All the possible NNs realized by changing θ forms an n-dimensional dual manifold S, where θ plays the role of a coordinate system [1][2].

2.1 Information Geometric Analysis about Modularity

A modular NN is a group of loosely connected NNs in which each divided subtasks is settled by one module. All modules specified by submanifolds are integrated and cooperated to realize the original complex problem.

1. Suppose the submanifold corresponding to the modular network (sub-network) is denoted as $M_i : i = 1 \dots m$, the points in it have the vector parameter: $M_1 : \Theta(\theta_1) = (\theta_1, \dots, \theta_j), \dots M_m : \Theta(\theta_m) = (\theta_l, \dots, \theta_n)$. M denotes the manifold corresponding to the monolithic structure NN to solve the original complex problem. It has an n-dimensional parameter vector: $\Theta = (\theta_1, \dots, \theta_n)$. We can demonstrate that M can be expressed by $M : M = M_1 * M_2 * \dots * M_m$, and M is diffeomorphic to $M_1 * M_2 * \dots * M_m$ which means that they have the same geometric structure.

We can use mathematic induction to demonstrate it but to simplify the proof we only consider the case of $m=2$.

Proof: Let (M, ψ) is an n-dimensional manifold where ψ is called differentiable structure of M, $\Theta = (\theta_1, \dots, \theta_n)$ is the parameter vector of M, (M_1, ψ_1) is a k-dimensional manifold where $\psi_1 = \{(V_\alpha, \Theta_\alpha)\}, \Theta_\alpha = (\theta_1, \dots, \theta_k)$ and (M_2, ψ_2) is an $n-k$ dimensional manifold where $\psi_2 = \{(W_\beta, \Theta_\beta)\}, \Theta_\beta = (\theta_{k+1}, \dots, \theta_n)$.

We construct a differentiable structure in

$$M_1 * M_2 : \psi' = \{(V_\alpha * W_\beta, \Theta_\alpha * \Theta_\beta) | (V_\alpha, \Theta_\alpha) \in \psi_1, (W_\beta, \Theta_\beta) \in \psi_2\},$$

$$\text{and } \forall (v, w) \in V_\alpha * W_\beta, ((\Theta_\alpha * \Theta_\beta)(v, w)) = (\Theta_\alpha(v), \Theta_\beta(w)).$$

(1) We first prove that $\Theta_\alpha * \Theta_\beta$ is a homeomorphism.

$$\text{Because } (\Theta_\alpha * \Theta_\beta)(v * w) = (\Theta_\alpha(v), \Theta_\beta(w)) = (x, y) \in \Theta_\alpha(V_\alpha) * \Theta_\beta(W_\beta)$$

$$\text{and } (\Theta_\alpha^{-1} * \Theta_\beta^{-1})(x, y) = (\Theta_\alpha^{-1}(x), \Theta_\beta^{-1}(y)) = (\Theta_\alpha^{-1} \circ \Theta_\alpha(v), \Theta_\beta^{-1} \circ \Theta_\beta(w)) = (v, w)$$

$$\text{so } (\Theta_\alpha * \Theta_\beta)^{-1} = \Theta_\alpha^{-1} * \Theta_\beta^{-1} \text{ ie. } (\Theta_\alpha * \Theta_\beta)^{-1}(x, y) = (\Theta_\alpha^{-1}(x), \Theta_\beta^{-1}(y))$$

hence $\Theta_\alpha * \Theta_\beta$ is a one-one correspondence mapping, as $\Theta_\alpha, \Theta_\beta, \Theta_\alpha^{-1}, \Theta_\beta^{-1}$ are continuous, so $\Theta_\alpha * \Theta_\beta, (\Theta_\alpha * \Theta_\beta)^{-1}$ are both continuous. Therefore

$$(\Theta_\alpha * \Theta_\beta) : V_\alpha * W_\beta \rightarrow (\Theta_\alpha * \Theta_\beta)(V_\alpha * W_\beta) \subset R^n \text{ is a homeomorphism.}$$

(2) Property of overlay

Because $M_1 = \bigcup_{\alpha} V_{\alpha}$, $M_2 = \bigcup_{\beta} W_{\beta}$ so $M_1 * M_2 = \bigcup_{\alpha} V_{\alpha} * \bigcup_{\beta} W_{\beta} = \bigcup_{\alpha, \beta} V_{\alpha} * W_{\beta}$

(3) Property of consistency

Suppose $(V_{\alpha} \cap V_{\alpha'}) \neq \emptyset$ and $(W_{\beta} \cap W_{\beta'}) \neq \emptyset$. So $(V_{\alpha} * W_{\beta}) \cap (V_{\alpha'} * W_{\beta'}) \neq \emptyset$,

hence $\Theta_{\alpha'} \circ \Theta_{\alpha}^{-1}, \Theta_{\alpha} \circ \Theta_{\alpha'}^{-1}, \Theta_{\beta'} \circ \Theta_{\beta}^{-1}, \Theta_{\beta} \circ \Theta_{\beta'}^{-1}$ are all smooth mappings,

thus $(\Theta_{\alpha'} * \Theta_{\beta'}) \circ (\Theta_{\alpha} * \Theta_{\beta})^{-1}$ is smooth, and so is $(\Theta_{\alpha} * \Theta_{\beta}) \circ (\Theta_{\alpha'} * \Theta_{\beta'})^{-1}$.

Therefore $(M_1 * M_2, \psi')$ is an n-dimensional differentiable manifold.

From the proof we know that $M_1 * M_2$ is homeomorphic to R^n and so is M, thus $M_1 * M_2$ is homeomorphic to M, meaning they have the same geometric properties and $\psi = \psi'$.

So far we have demonstrated that several manifolds (submanifolds) can be integrated as one whose parameter vector consists all the parameters in the submanifolds. This proof can give an interpretation mathematically that several modules network can be integrated to realize the function of the monolithic network.

2.2 Information Geometric Analysis about Hierarchy

Suppose a flat manifold S with modifiable parameter vector $\theta = (\theta_1, \dots, \theta_n)$, is equipped with the Riemannian metric $g_{ij}(\theta)$ given by the fisher information matrix and moreover is equipped with dually flat affine connections [1]. Two fundamental and dual coordinate systems θ and η in it.

Let consider a nested series of e-flat submanifolds[4]

$$E_1 \subset E_2 \subset \dots \subset E_n . \quad (1)$$

where every E_k is an e-flat submanifold of E_{k+1} . Each E_k is automatically dually flat. We call such a nested series an e-flat hierarchical structure or, e-structure. A typical example of the e-structure is the exponential-type distributions [1]. Let

$$\eta_k = E_{\theta}[r_k(x)], \quad k = 1, \dots, n . \quad (2)$$

where E_{θ} is expectation with respect to $p(x, \theta)$, the two fundamental coordinate systems θ and η are mutually orthogonal.

If consider the k -cut by which

$$\theta_{k^+} = (\theta_{k+1}, \theta_{k+2}, \dots, \theta_n), \quad \eta_{k^-} = (\eta_1, \eta_2, \dots, \eta_k) \quad (3)$$

when θ_{k^+} is fixed to be equal to $c_{k^+} = (c_{k+1}, \dots, c_n)$ but other θ coordinates are free, we have an e-flat submanifold for each k

$$E_k(c_{k^+}) = \{p(x, \theta) \mid \theta_{k^+} = c_{k^+}\}. \quad (4)$$

They give a foliation of the entire manifold

$$\bigcup_{c_k^+} E_k(c_k^+) = S . \quad (5)$$

For $c_k^+ = 0$, a hierarchical structure is introduced in S because of

$$E_1(0) \subset E_2(0) \subset \cdots \subset E_n(0) = S . \quad (6)$$

Dually to the above, let fix $\eta_{k^-} = m_{k^-}$, we have an m-flat submanifold for each k

$$M_k(m_{k^-}) = \{ p(x, \theta) | \eta_{k^-} = m_{k^-} \} . \quad (7)$$

They define m-flat foliations. The submanifolds M_k and E_k are complementary and orthogonal at any points.

So far we find that combining multi hierarchical structures into one is possibble [5]. A single structure can be divided into many hierarchical structures and a hierarchical structure can even be divided into many small hierarchical structures. This idea is true in NN because of the correspondence between the NN and manifold.

Every such hierarchical structure forms a dually flat submanifold M, embedded in the flat envelope manifold S. The dimensionality of M is smaller than that of S.

Then we can continuously construct submanifold M that can be embedded into manifold S by the above analysis. It is possible to achieve system dynamic extension without destroying the most original submanifolds. In NN learning that means we can expand learning machine without destroying acquired knowledge [6]. Thus we can propose our knowledge increasable model.

3 A New Knowledge-Increasable Model

We propose a new knowledge-increasable model multi-HME based on the typical modular and hierarchical NN: HME [5]. We present the idea of ‘multi’ in consideration that it is hard to accomplish the training and recognition tasks by only one HME in case of large data sample or many pattern classes. We use our model in classification problems. The model and its extension is depicted in Fig1.

In Fig1, the multi-HME NN knowledge-increasable model is combined by k HME and a dynamic classifier. HME_1 to HME_k are constructed according to different training sample sets. When extension a new HME_{k+1} is added, others keep unchanged, which equals to a new submainfold M embedded into the manifold S. It extended the system and the original submanifolds are unchanged.

The model is constructed as follows:

1. Divide the training samples into k training subsets: $T\{T_1, T_2, \dots, T_k\}$ without intersection. Each may have different sample numbers.
2. Construct corresponding HME_i according to training subsets T_i , $i = 1, 2, \dots, k$.
3. Input the test sample X into all the k HME systems in Fig1. Partition the data according to the confidence of each HME system on pattern X. Then we got a new training set with very less train samples that are very similar.
4. Train dynamic classifier on the new training set, input test sample X into the dynamic classifier and get the last result.

5. If there is a new data class needs to be classified, we extend model by adding new HME_{k+1} which is trained on the new class dataset.

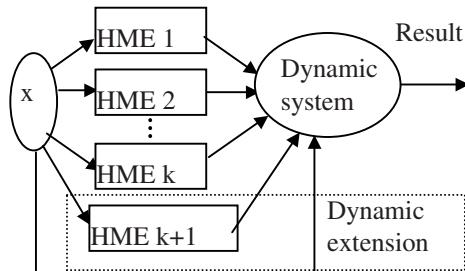


Fig.1. Knowledge-increasable model and its dynamic extension

We use divide-and-conquer principle to divide training samples into multi subsets without intersection, and then construct HME classifiers corresponding to each set. The results of all the HME classifiers are used to reduce the target scope, and then a dynamic classifier is used to improve the classifier rate because of limited data classes after selecting by HME systems.

4 Conclusions

The present paper used information geometry to elucidate the modular and hierarchical structure mechanism of NN mathmatically. The relation among modules or hierarchies can be explained using geometric properties of cooresponding submanifolds. This paper also proposed a new model: multi-HME that has knowledge-increasable and structure-extendible ability based on the information geometric analysis.

References

1. Amari S., Differential-geometrical Method in Statistics: Lectures Notes in Statistics, Springer-Verlag, Berlin Heidelberg New York Tokyo (1985)
2. Amari S., Information Geometry of the EM and em algorithms for neural networks, Neural Networks, Vol.8, no.9 (1995) 1379–1408
3. Jie Bao, Hierarchical Learning in Neural Network, A research note (2002), <http://www.cs.iastate.edu/~baojie/acad/current/hnn/hnn.htm>
4. Amari S., Information Geometry on Hierarchy of Probability Distributions, IEEE Trans. Inform. Theory, Vol. 47,No.5 (2001)
5. Jinwei Wen, Siwei Luo, Information Geometry on Ensemble HME Model, Tencon'02, Beijing (2002)
6. Siwei Luo, Zhen Han, Large Scale Neural Networks And its Application on Recognition of Chinese Character, the Fourth International Conference on Signal Processing Proceedings (ICSP'98) (1998) 1273–1277.

A Novel Speed-Up Algorithm of Fractal Image Compression

Yusong Tan and Xingming Zhou

PDL, School of Computer,
National Univ. of Defense Tech.
Changsha, Hunan, P. R. China
`{ystan, xmzhou}@nudt.edu.cn`

Abstract. The paper analyzes the weakness of traditional speed-up techniques of fractal image compression. Moreover, it presents a novel idea that entropy can be used to improve fractal image compression's performances. It has proofed a theorem that the IFS cannot change the entropy values of image blocks. Therefore, it gives two novel fractal compression algorithms based on entropy. The simulation results illuminate that new methods can improve the compression's performances.

1 Introduction

At 1988 M. Barnsley presented the idea of *iterated function system*([1]). He thought that IFS should be applied in fractal image compression. His student, A. Jacquin, had developed more detailed technologies for image encoding ([2], [3], [4]) later.

Subsequently, these results inspired an upsurge in fractal image compression at 1990s. But those compression algorithms all had a fatal weakness, i.e. compression speed was so slow that could not be applied practicality. As a result, most of researches focused on speed-up technologies of compression. But it still was not solved completely.

This article also focuses on speed-up method. It will present a basic speed-up algorithm and its extension based on entropy. The novel method can improve the compression performances greatly. Section 2 will give some basic concepts and related works briefly. Our novel algorithms are presented in section 3. Finally the paper will give some experimental results and their analysis.

2 Related Works

The main idea of Fractal image compression is that the image can be represented by IFS, i.e. some *contractive map* ([1], [5]). With Fix-Point Theorem we know that the IFS have and only have one attractor. For every image we can find a proper IFS and its attractor is the image.

The most important technology is how to find the best match between *domain blocks* with the special *range block* as fast as possible. It need decrease the number of

domain blocks needed to match and does not descend performances at same time. Essentially, for the special range block we should find the smallest domain blocks set which contains the best matching domain block.

Clearly, the simplest method is to calculate all domain blocks to get the best mapping for every range block. This method is the best algorithm that can get the best performance. But it cannot be applied practicality because every image will have a lot of domain blocks. For example, a 256×256 image will have $(256-32+1) \times (256-32+1) = 50625$ domain blocks that is 32×32 .

As a result, there are so many speed-up technologies, in which the most common method is classified technique. It partitions the image into several types of domain block, and only match the same type domain block with range block

Y. Fisher ([5]) developed the *quadtree algorithm*. Partition the whole image into four subblocks, i.e. left-up, right-up, left-down and right-down. Then calculate the average intensity and variance of each block and sort them. If we ignore the block's rotation and symmetry, we will classify all blocks into three main categories. Moreover, using the variances of blocks we can partition every main category into 24 subcategories and get 72 subcategories totally.

B. Hurtgen's method ([6]) is similar as Fisher's. It also quadtrees the image and calculates the average intensity. But this method will compare every block's intensity with average intensity. Bigger is 1 and else is 0. So we get 16 main categories. In fact there are just 15 main categories because all 0 cannot be true. Similarly, using the variances of blocks we can partition every main category into 24 subcategories and get 360 subcategories totally.

D. Saupe([7]) had presented a new speed-up algorithm, i.e. *Nearest Neighbor Search(NNS)*. It defines the distance of image blocks. Moreover, it finds the nearest domain blocks for each range block. The algorithm has pretty performances.

Tan([8]) develops a algorithm, *FASON algorithm*, which need not storage the domain blocks and match the range blocks directly. FASON speed-up the compression speed greatly and without big performance decreasing at same time.

3 Entropy-Based Speed-Up Algorithm

3.1 Weakness of Traditional Methods

In essential, most of traditional classified methods are based on average intensity and contrast. They have pretty performance in practice. But they have a serious weakness at same time, i.e. they maybe lost more precise match. As a result, the match error produced by old methods should be big; the compress ratio may be small and compress quality may be worse.

Follows is a simple definition of IFS ([5]). Assume gray image's pixel is a vector (x, y, g) , in which x, y is pixel's coordinate, and g is its intensity.

Definition 1. *IFS of fractal image compression is a set of affined maps.*

$$IFS = \bigcup_i W_i \quad (1)$$

in which, W_i as follows,

$$W_i \begin{pmatrix} x \\ y \\ g \end{pmatrix} = \begin{pmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{pmatrix} \times \begin{pmatrix} x \\ y \\ g \end{pmatrix} + \begin{pmatrix} e_i \\ f_i \\ o_i \end{pmatrix} \quad (2)$$

Follows is a counterexample of a traditional method – B. Hurtgen method. It explains the weakness of the traditional ones.

Counterexample: Assume 4×4 image block $I=\{(1,1,3), (1,2,2), (2,1,1), (2,2,0)\}$, Clearly it is $\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$ type image block. We define a affined map as follow,

$$W \begin{pmatrix} x \\ y \\ g \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ g \end{pmatrix} \quad (3)$$

Get new image block $I'=\{(1,1,3), (1,2,1), (2,1,2), (2,2,0)\}$. But the new block is $\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$ type block.

With old methods, the source block I cannot get a precise match with I' since they are different types, and just can get worse match with other blocks of same type. Match error is great and the compress quality is worse. Moreover, it may induce the more partition of the range block, longer compression cost time and lower compression ratio. \square

There are similar counterexamples for other traditional algorithms.

3.2 Improved Algorithm Based on Entropy

What is the solution of the problem? Entropy should be a good answer. The entropy of all pixels' intensities can describe the their statistical characters. More information about the entropy is in [9]. If we just consider the entropy of pixels' intensity, any image's topology transform cannot change the entropy, and we think all these blocks are same. Operator $E(I)$ is calculating the entropy for image block.

Now we will give a theorem that is the core idea of the novel speed-up method.

Lemma 1. Give a linear bijective $f : D \rightarrow R$, then $E(D)=E(I)$. In which $D=\text{dom}(f)$ and $R=\text{ran}(f)$.

Proof: In generally, assume $f(x) = ax + b$, and $a \neq 0$.

Clearly, $P\{f(x)=r_i\} = P\{f(x)=r_i | x \in \{d_i\}\} \times P\{x \in \{d_i\}\}$, in which r_i is the image of $\{d_i\}$.

Since the mapping is injective, for $\forall r_i, \#\{d_i\} = 1$, and

$$P\{f(x) = r_i \mid x \in \{d_i\}\} = 1.$$

$$\text{Get } P\{f(x) = r_i\} = P\{x = d_i\},$$

$$\text{Entropy defined as } E = \sum p_i \log_2 p_i,$$

So the lemma is true. \square

Based on the lemma, we can proof the follow theorem.

Theorem 2. Affined IFS transform cannot change the entropy of image block.

Proof. With the lemma, we know a single affined mapping cannot change the entropy of the image block, i.e. $E(I) = E(f(I))$

And the IFS just only have finite affined mappings, assume as $\{f_1, f_2, \dots, f_n\}$

Using the lemma 1 repeatedly, get $E(I) = E(f_{i_1} \circ f_{i_2} \circ \dots \circ f_{i_n}(I))$, in which

$$f_{i_k} \in \{f_1, f_2, \dots, f_n\}$$

So the theorem is true. \square

Remark 1. the converse theorem is not true. That is, even some image blocks have same entropy; they maybe not have any affined mapping between them. For example, $\{1, 0, 1\} \rightarrow \{0, 1, 1\}$ does not have any affined mapping, even they have same entropy. But if two image blocks have affined relation, their entropy must be same.

From the theorem, we know that we should select some domain blocks those entropy equal with the range block approximately to get a pretty match.

With this idea we present a novel speed-up algorithm as follows, in which δ is the threshold to judge whether the entropy values of two blocks are equal and it is adjustable.

Step1: Partition source image into overlapped $2N \times 2N$ domain blocks.

Step2: Calculate the entropy of all domain blocks, and sort them ascending.

//Match procedure; assume the current range block is R_i .

Step3: Find in all domain blocks, get a set of domain blocks that satisfy $\{D_i \mid |E(R_i) - E(D_i)| \leq \delta\}$

Step4: For all blocks in the set, matching with the range block and get the domain block that has minima matching error.

Step5: If the minima error less than the threshold, stop the range block's matching; else, quadtree the range block for next matching.

Alg. 1. Entropy-based Speed-up Algorithm

Remark 2, the entropy-based algorithm is just a speed-up method for fractal image compression. It cannot be applied in analysis of texture. For the entropy is just a

statistical parameter, two blocks with equal or approximately equal entropy value maybe have different texture entirely. This is a weakness of the algorithm.

3.3 More Speed-Up

From the Algorithm (1), we can develop a more accelerated method easily. Combined with the classification idea, we classify the domain blocks with the entropy to accelerate the compression. In general, assume the entropy value of all domain blocks belong to $(E_{\min}, E_{\max}]$. Firstly, sort the values ascendingly and partition into N intervals to get $\bigcup_{i=0}^{N-1} (E_i, E_{i+1}]$, in which $E_0 = E_{\min}$, $E_N = E_{\max}$. With the partition results, classify every domain block into one interval. Therefore, when match a range block we should judge the range block's interval firstly and just match those domain blocks within same interval.

Clearly, this algorithm can decrease the number of domain blocks needed to match, so the compression will be accelerated.

Step1: Partition the source image into overlapped $2N \times 2N$ domain blocks

Step2: Calculate all domain blocks' entropy value

Step3: Sort all domain blocks ascending, partition into N intervals and get $\bigcup_{i=0}^{N-1} (E_i, E_{i+1}]$. Classify all domain blocks based on these intervals.

//Match procedure; assume the current range block is R_i

Step4: Calculate the entropy value of R_i , and judge the interval for its

Step5: Match all domain blocks of same interval, and find the block with minimum match error

Step6: If the minimum less than the threshold, stop the range block's match procedure; else quadtree the range block for next match.

Alg. 2. Speed-up Method Based on entropy classify

There are two strategies to partition the entropy values.

1. Partition evenly, i.e. $E_{i+1} - E_i = \epsilon$, in which ϵ is constant. The strategy is very simple. But the number of blocks contained in one interval maybe various extremely. Therefore some intervals would be bottleneck.
2. Using values' histogram partition entropy values into irregular interval to assume every interval contains same number of domain blocks. This strategy ensures that the number of domain blocks that every range block needed to match is equal approximately. And the weakness is that some range blocks maybe have coarse match precise because the strategy must use fine granularity around the dense value. Therefore these blocks are just have small match error threshold.

4 Experimental Results and Analysis

The experimental platform is FracCompress ([10]) developed by Jude Sylvestre. It is a Windows program and writes with MFC. It implements quadtree fractal image compression presented by Y. Fisher. We have extended the module of experimental result data collection, and implement the module of entropy-based speed-up algorithm and the NNS method. The experimental computer is AMD Duron 550 with 128 MB RAM. We use four standard images, two are Lena and Barbara, the other two are selected from natural image library(van Hateren image library, [11]) randomly, i.e. imk00032.imc and imk000170.imc. In fact, these two images can be classified into two categories, normal image (Lena, Barbara) and natural image (imk00032.imc, imk00170.imc).

Table 1 give the performance compare between the traditional quadtree method, NNS method and the entropy-based method, in which the latter has four implements with different δ . The considered performances are compression cost time (CCT), PSNR, compression ratio (CR).

Table 1. Lena (256×256, 8b gray)

	QuadTree	NNS	Algo. 1 $\delta =0$	Algo. 1 $\delta =0.05$	Algo. 1 $\delta =0.1$	Algo. 1 $\delta =0.2$
CCT	7	4	3	4	6	9
PSNR	29.16	32.61	25.41	30.13	32.30	32.31
CR	9.11	6.05	6.78	9.22	10.64	9.01

We will analysis the results in table 1. And all performance comparisons are based on the performance values of Quad-Tree method.

First we consider the compression cost time. Clearly, with small δ value the number of domain blocks selected is small too, and the matching speed will be faster. For example, when $\delta=0$ the CCT is faster double; while when $\delta=0.2$ the CCT is longer.

Now we will consider both PSNR and compression ratio of six methods together. Why the image quality is worse when $\delta=0$? Since the method need the entropy values of two blocks must be equal strictly (though the experiment has relax 0 to a small value, 0.0005), none domain blocks maybe be selected for a given range block. Even there are some domain blocks, with the theorem's note we know that two blocks with same entropy maybe not have any affined mapping. So we cannot assure the every range block will find a precise match with one domain block. Moreover, the given range block will be quadtreeed with more iteration and the compression ratio will be worse too.

At the same time, when $\delta=0.05$ the performance of algorithm 1 is as same as quadtree method approximately. The main reason is that the number of domain blocks selected is big enough to find a pretty precise matching for every range block.

When $\delta=0.1$ the performances of algorithm 1 are better. Because the number of selected domain blocks is bigger and we can find more precise matching now.

But after $\delta>0.2$ the performances should not improved greatly. The main reason is that the number of selected domain blocks is big enough to find enough precise

matching. Moreover, the number of domain blocks needed to match is bigger and the cost time is longer now. It is not worth to set $\delta > 0.2$ for the Lena image.

Compared with NSS, we can find that the compression speed of NNS is pretty fast and equal to $\delta = 0.05$. At same time, its PSNR is better than the latter. But when δ is bigger the entropy-based algorithm will get better performance than NNS.

Follows are the experimental results of other images. We would not give any analysis.

Table 2. Barbara (256×256, 8b gray)

	QuadTree	NNS	Algo. 1 $\delta = 0$	Algo. 1 $\delta = 0.05$	Algo. 1 $\delta = 0.1$	Algo. 1 $\delta = 0.2$
CCT	10	6	4	7	10	12
PSNR	28.41	31.21	25.11	29.87	31.25	31.31
CR	8.99	8.21	6.42	9.00	10.01	10.10

Table 3. imk00032.imc (1536×1024, 12b gray)

	QuadTree	NNS	Algo. 1 $\delta = 0$	Algo. 1 $\delta = 0.05$	Algo. 1 $\delta = 0.1$	Algo. 1 $\delta = 0.2$
CCT	257	209	194	217	241	229
PSNR	32.25	34.21	28.54	33.10	36.78	37.01
CR	12.34	11.13	10.21	13.57	15.12	15.17

Table 4. imk00170.imc (1536×1024, 12b gray)

	QuadTree	NNS	Algo. 1 $\delta = 0$	Algo. 1 $\delta = 0.05$	Algo. 1 $\delta = 0.1$	Algo. 1 $\delta = 0.2$
CCT	314	240	231	250	291	341
PSNR	31.24	34.01	27.11	33.03	35.49	35.70
CR	11.99	10.13	10.14	12.18	13.44	13.44

From the experiments we find that the performances of normal images' fractal compression are worse than wavelet-based ones. We guess that these normal images are small and the textures are simple. As a result, the "most" precise matching founded is not enough precise.

But the performances of the natural images are satisfying. The reason is that these images are big enough, the textures are abundant and the similarities between blocks are distinct. The compression method can find more precise matching.

5 Conclusion

The paper presents two entropy-based speed-up algorithms of fractal image compression. These algorithms can find more precise match easily than traditional

methods. Their compression performances, such as cost time, compression ratio and PSNR, are better than the traditional ones distinctly. Moreover, they have lower calculate complexity.

The entropy-based speed-up methods are just the improvement of fractal image compression. If we want those fractal methods can be applied practically, we must make a break though at match strategies between range blocks with domain block. The simple affined mapping cannot satisfy the requirements. As a result, all previous works are valueless.

References

1. M. Barnsley. *Fractal everywhere*, Academic Press, San Diego, CA, 1988
2. A. Jacquin,. *A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding*. PhD thesis, Georgia Institute of Technology, August 1989
3. A. Jacquin, *Image coding based on a fractal theory of iterated contractive Markov operators, Part I: Theoretical Foundation*. Technical Report 91389-016, Georgia Institute of Technology, 1989
4. A. Jacquin, *Image coding based on a fractal theory of iterated contractive Markov operators, Part II: Construction of fractal codes for digital images*. Technical Report 91389-017, Georgia Institute of Technology, 1989
5. Y. Fisher, *Fractal Image Compression : Theory and Application*, Springer-Verlag Press, New York, 1995
6. B. Hurtgen, Fast Hierachical codebook search for fractal coding of still images, Proc. EOS/SPIE Visual Comm. PACS Medical App.'93, Berlin, Germany, 1993
7. Y. Fisher, *Fractal Image Encoding and Analysis*[M], Springer-Verlag, 1998
8. Tan Yu-song, Fason: a Novel fast improvement of fractal image compression (Chinese), Computer Eng. and Tech. Accepted
9. Zhang Zhao-Zhi, *Information Theory and Optimal Coding* (Chinese), Shanghai Sci. & Tech. Press,1993
10. Jude Sylvestre, <http://inls.ucsd.edu/y/Fractals/Other/FracComp.zip>
11. van Hateren, Independent component filters of natural images compared with simple cells in primary visual cortex, Proc. Royal Stat. Soc. of London (B)265,359–366

Implementation and Evaluation of a Novel Parallel SAR Imaging Method on Clustering Systems*

Jizhong Han and Shichao Ma

Institute of Computing Technology,
Chinese Academy of Sciences
P.O.Box 2704-25#,
Beijing 100080 P.R.China
{hjz, scma}@ict.ac.cn

Abstract. Tremendous computational throughput and real time constraints make sequential computing unable to satisfy requirements of applications. Therefore, attentions are paid to parallel SAR imaging, which can efficiently decrease the time of imaging. Combining the best features of two existing approaches: data parallel method and program parallel method, we propose a novel parallel SAR imaging method on clustering systems. Detailed implementation and evaluation are also presented. The results show that the novel method is more preferable than traditional method.

1 Introduction

This work focuses on improving performance of Synthetic Aperture Radar (SAR) imaging on clustering systems.

SAR is an important tool for the collection of all-weather image data on earth. Furthermore, these data can be made at extremely high resolution by signal processing. It is widely used in the fields of disaster forecasting, military inspecting, terrain mapping and resources investigating, etc[1]. SAR imaging always spends a lot of hours because of complicated algorithms and a large volume of raw data. However, in many cases, it is critical to extract information from raw data and send to decision-maker in time. Thus, many attentions have been paid to manage SAR imaging by parallel method[2-6].

Some efforts have been made on developing real time parallel processor systems to process SAR data so that these systems can be used in space-borne or airborne environments. On the other hand, some efforts have been made on satisfying the strict demands of satellite ground station where a large volume of SAR data need to be accurately processed to form high resolution images. There are many same techniques used in both of the two researches. Nevertheless, the former, also called embedded system, has been always complex and expensive and has often relied on custom hardware designs; the latter is oriented to computing of huge amount data in short

* This work was supported in part by National Natural Science Foundation of China under Grant No. 69896250 and by Grant-in-Aid for Encouragement of Young Scientists of the Institute of Computing Technology under Contract No. 20026180-9.

time, which need large-scale high performance parallel computers. With the development of architecture of parallel computers, there is a trend to utilize clustering systems on SAR imaging because its high-performance, scalability and availability.

Until now, three main parallel SAR imaging methods on parallel computers had been proposed, including pipeline parallel method, data parallel method and program parallel method. In [4], they were called respectively vertical partitioning, horizontal partitioning, and vertical-horizontal partitioning. However, pipeline parallel and data parallel methods could not make good use of computing resources. Pipeline parallel and program parallel method are limited by high latency and low bandwidth of their communication networks. Moreover, all of them ignored the influences of continuous properties of SAR dataflow. Thus they were not suitable to form a large SAR image.

This work proposes a novel parallel SAR imaging method on clustering systems, which is the combination of data parallel method and program parallel method. Therefore, this method can not only overcome the limitation of network throughput, but also reach the maximum of parallelism. Furthermore, section convolution and raw data division are considered in this method so that long and seamless SAR images can be formed.

The rest of the paper is organized as follows: Fundamentals of SAR Imaging are presented in section 2. Section 3 gives a overview of parallel SAR imaging methods. The novel method is presented in section 4. Section 5 implements this method on a clustering system – Dawning 4000L and gives the performance evaluation about influences of network throughput and number of nodes. Section 6 concludes the paper.

2 Fundamentals of SAR Imaging

Synthetic Aperture Radar is an all-weather imaging tool able to penetrate through clouds and collect data at night. It also can distinguish unique responses of terrain and cultural targets to radar frequencies. Therefore, it is widely used in the fields of disaster forecasting, military inspecting, terrain mapping and resources investigating, etc.

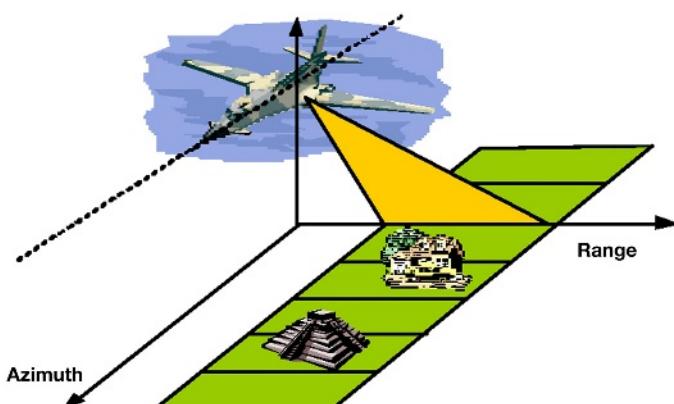


Fig. 1. Synthetic Aperture Radar Imaging Concept

Consider an airborne SAR imaging perpendicular to the aircraft velocity as shown in the figure above. Typically, SAR produces a two-dimensional (2-D) image. Successive radar pulses are transmitted and returned in the range dimension, which is orthogonal to the line of flight. The other dimension is called azimuth which is parallel to the line of flight. Thus, the entire range dimension is processed at once, but processing of the azimuth vector need to be involved sectioned convolution.

SAR imaging algorithms achieve high range resolution by the technology of pulse-compression and high azimuth resolution by the principle of synthetic aperture. Mature SAR imaging algorithms include Range-Doppler, Step-transform, Rectangular format, Polar format, Wavenumber interpolation/range migration/w-k, and Chirp Scaling. This research mainly discusses the parallelization of Chirp Scaling (CS) algorithm that is widely used because it can successfully avoid interpolations, which were necessary when we had to deal with strong range-cell migration data.

As the figure 2 shown, processing of the CS algorithm includes range compression and azimuth compression[1].

SAR imaging is a kind of two-dimension computing. Except these compression operations, corner turn should also be considered which can transpose matrix from range dimension to azimuth dimension and vice versa. As described above, processing of the entire azimuth vector cannot be created once, thus, sectioned convolution is necessary.

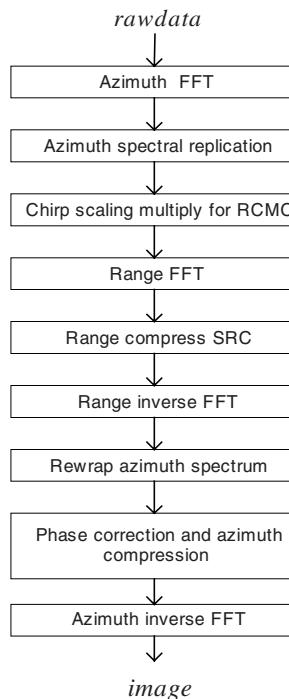


Fig. 2. Chirp Scaling Algorithm

As figure 3 shown, although data is arranged by lines in range dimension, they are sectioned to data blocks along azimuth dimension. Data block is always called frame.

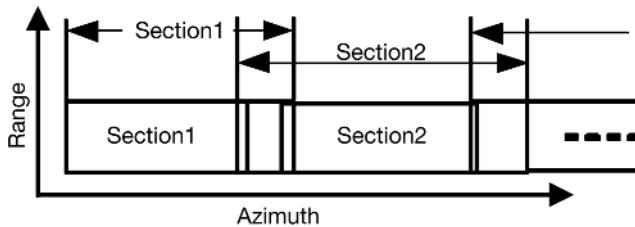


Fig. 3. Data Sections Convolution

Moreover, overlap between frames is necessary since convolution is involved in SAR imaging. The data redundancy can guarantee the result images are continuous. In figure 3, shadow parts present overlaps between two neighboring frames. The length of overlap is decided by algorithms and the properties of data.

3 Parallel SAR Imaging Method Overview

The tremendous computational throughput of SAR imaging results in a need to partition the SAR algorithm to run on multiple processors. According to partitioning methods, parallel SAR imaging methods can be classified into three classifications. The three methods include vertical partitioning, horizontal partitioning and vertical-horizontal partitioning[4].

Vertical partitioning was also called pipeline parallel method, which divides the process of SAR imaging process into several parts according to algorithms. The computing task of every part can be allocated on single or several nodes[2,4]. Multiple computing nodes take part in processing by Ping-Pong mode. In [2], this method had been implemented on Dawning-1000 according to RD algorithms. However, because computing must be finished on single nodes, matrix transposition and tremendous data throughput became the performance bottleneck.

Horizontal partitioning was also called data parallel method, which allocates entire of imaging process of raw data on single computing node. The algorithm of all nodes is same. This method has high scalability, and sequential program can easily be realized in this environment. In [3], the method had been implemented on Dawning-2000 according to CS algorithms. However, it has obvious disadvantages. Firstly, nodes must have high capability and huge memory space. Secondly, it could not make good use of the computing resources of advanced architecture of multiply computing units CPUs, like Intel Xeon and Power3-II.

Vertical-horizontal partitioning was also called program parallel method. It is a medium grained parallel processing algorithm proposed for SAR imaging. In this parallel processing algorithm, every processing stage is done in parallel, and the degree of parallelism is task-level. The raw data block is divided into several data strips along range direction, so every computing node can get one data strip. However, in corner turn operations, data exchanges among computing nodes must be considered since data matrix is distributed among several nodes. As to every computing node, local data are transposed. Then, data exchange makes local node get transposition results of other nodes. In [5], this algorithm had been implemented on

DAWNING3000. However, experiments showed that performance was limited by collective communication.

All methods above did not consider section convolution. So they are not competent for those SAR imaging applications which input dataflow is continuous.

4 Novel Parallel Processing Method

This research presents a novel parallel SAR imaging method which is suitable for a bath of continuous raw data. This method is the combination of data parallel method and program parallel method.

As figure 4 shown, this method divides raw data into several sections which are distributed on different computing nodes. Every computing node may have several CPU, and every CPU can run several processes. Generally, these processes include one I/O process and several computing processes. Computing processes can exchange data each other. One section data can be dispatched to one or several computing processes. Different section data are processed by different computing nodes. For instance, data of section 1 are dispatched to processes 1~4, data of section 2 are dispatched to processes 5~8.

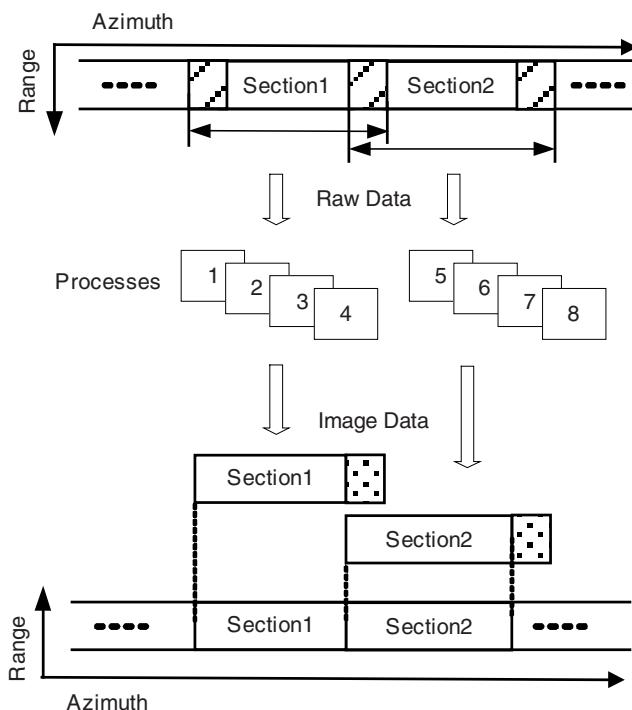


Fig. 4. Section Parallel Method

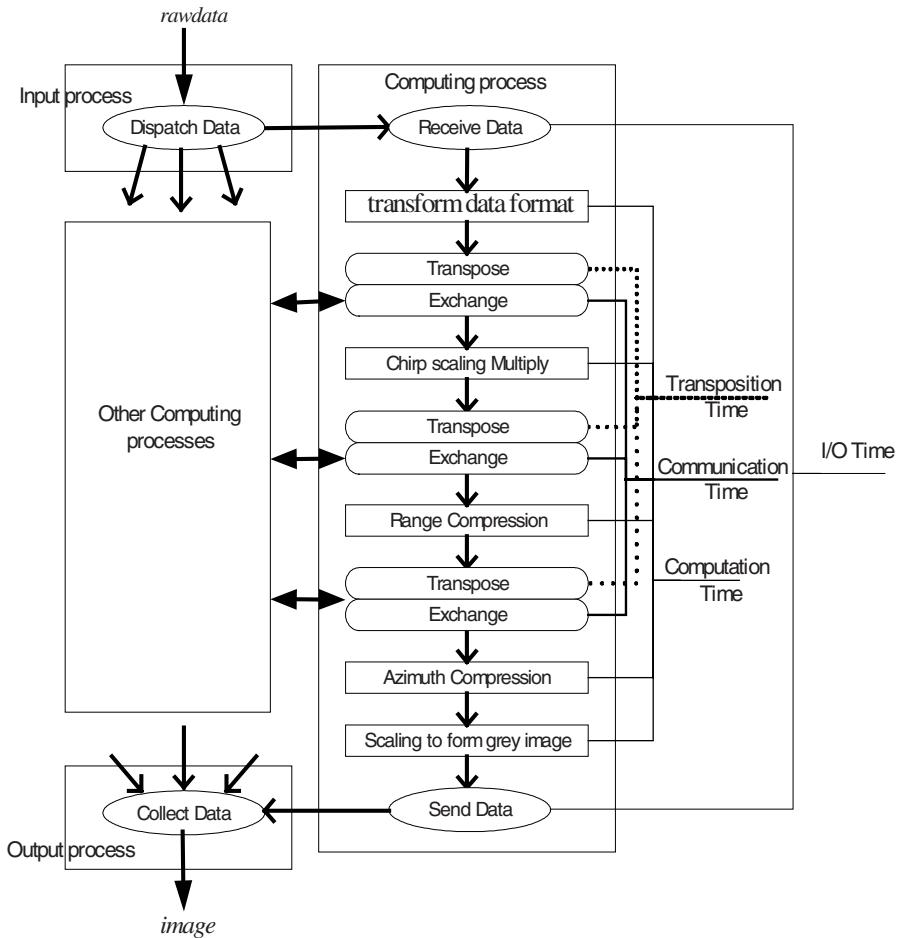


Fig. 5. Parallel Imaging Flow Chart and Time Classification

According to the CS algorithms, the process of SAR imaging can be separated several parts as figure 4 shown. There are three kinds of processes. Input processes fetch SAR raw data and divide them into several stripes and dispatch these stripes to computing processes. When computing processes run, exchange is necessary for corner turn. Output processes collect all of results from computing processes and form the image.

Matrix transpose is an important part of SAR imaging, so data exchanges among processes are inevitable. As figure 6 shown, the stripe of data allocated to one process are transposed, and then send this stripe of data to other processes and receive rest of data stripes from other processes. After transposition and exchange, corner turn operation has been finished.

Therefore, this method can not only overcome the limitation of network throughput, but also reach the maximum of parallelism. Furthermore, section convolution and raw data division are considered in this method so that long and seamless SAR images can be formed.

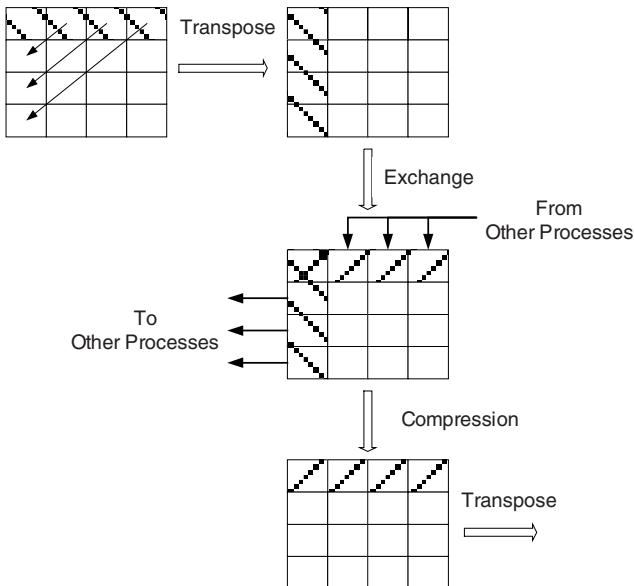


Fig. 6. Transposition and Exchange

5 Performance Evaluation

This section presents the performance of section parallel SAR imaging method on Dawning 4000L, which belongs to cluster of workstations (COW). Each computing node of this system is composed of two Intel Xeon chips. High-speed networks connected nodes include Myrinet, Gigabit Ethernet and 100Base Fast Ethernet.

5.1 Influence of Network Performance on Performance of Parallel Imaging

As figure 5 shown, the time spent in SAR imaging can be classified into the four parts, including I/O time, communication time, transposition time and computing time.

We have processed parallel 8192*4096 data block employing CS algorithm on DAWN4000L and tested the program's performance under different network circumstance. Two nodes of Dawning 4000 were used, and each node runs two computing processes.

It can be concluded from the figure 7 that the performance of Myrinet network is best. The communication throughput of Myrinet network roughly is two times that of Gigabit Ethernet, and is eight times that of 100Base Fast Ethernet. Network performance influence the Communication time and I/O time directly, but is irrelevant to computation time and transposition time. The better the performance of network, the better performance of parallel imaging can be reached.

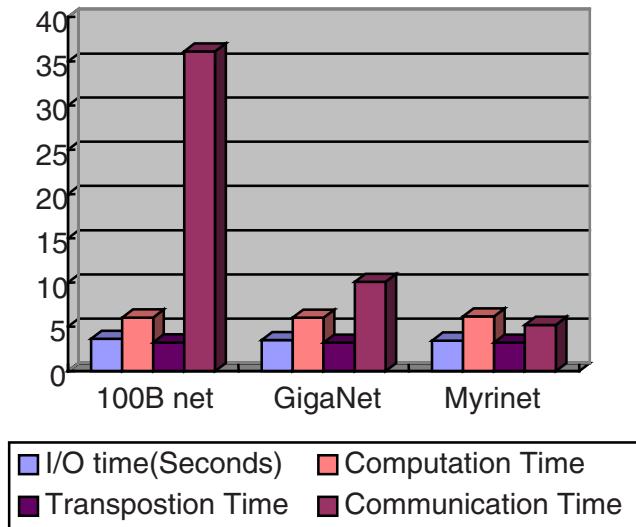


Fig. 7. Influence of Net Performance on the Runtime of Parallel Program

5.2 Influence of Node Number and Process Number on Performance of Parallel Imaging

In order to accurately value the influence of node number and process number, it is supposed that there are only one section to be processed. Under this condition, the section parallel imaging method is same as program parallel imaging method.

It can be concluded from the two tables above that: 1) Input and output of data is fulfilled by a single process so that I/O time is irrelevant to the number of nodes and computing processes. 2) Communication and transposition time are relevant to the number of nodes, but irrelevant to the number of computing processes. 3) Computation time is relevant to the size of data, that is to say it is relevant to the number of computing processes (The more the computing process, the shorter the computation time). In addition, Computation time is directly ratio to the number of nodes.

5.3 Comparison of Parallel Imaging Methods

To verify the effectiveness of section parallel imaging method, we use four nodes of Dawning 4000L to process a 16384*16384 size data block. In section parallel imaging method, the number of section can be configured. If there are only one section to be processed by all of computing nodes, section parallel method is same as Program parallel method.

As shown in figure 8, using parallel program, we can cut down the memory requirement of single node. The least memory requirement is inverse ratio to the node

number. Section parallel method provides flexibility for SAR imaging process, hence not only can improve the performance of system but also reduce the memory requirement.

Table 1. Test of 8192*4096 Data Imaging on Dawning 4000L Interlinked by Myrinet

Operation Time	Serial Program	Parallel Program		
		Node Number=1	Node Number=2	
		Computing Process=2	Computing Process=2	Computing Process=4
Transposition	6.39	6.37	3.21	3.23
Communication	0	8.39	3.77	3.71
Computation	22.25	11.36	12.80	7.36
I/O	3.44	3.44	3.49	3.44
Total Time	32.08	29.56	23.28	17.72

Table 2. Test of 8192*8192 Data Imaging on Dawning 4000L Interlinked by Myrinet

Operation Time	Node Number=2		Node Number=4	
	Compute Process=2	Compute Process=4	Compute Process=4	Compute Process=8
Transposition	6.47	6.47	3.33	3.38
Communication	10.27	10.68	7.12	6.38
Computation	20.24	11.66	11.42	6.96
I/O	6.89	6.93	6.89	6.90
Total Time	43.87	35.74	28.76	23.62

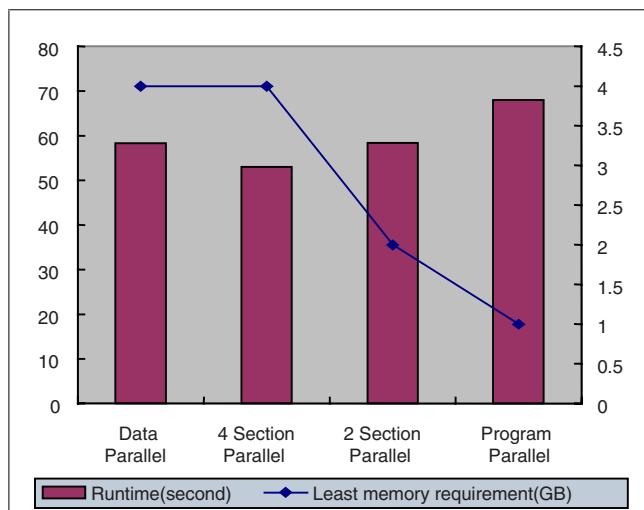


Fig. 8. Performance and Memory Requirement when Process 16384*16384 Data Using Four Nodes

6 Conclusions

In this paper, we proposed a novel parallel SAR imaging method on clustering systems. This method combine the best features of two existing approaches: data parallel method and program parallel method. It separates raw data into several sections and distributes them on different computing sets which process imaging by program parallel method. This method provides flexibility for parallel SAR imaging, and reduces the bottleneck of low bandwidth of networks. And it reduces the least memory requirement of computing nodes. By detailed analysis and test on Dawning 4000L, the results show that the section parallel SAR imaging method is more preferable than traditional pipeline parallel, data parallel and program parallel methods.

References

1. J. C. Culander, and R.N. McDonough, "Synthetic Aperture Radar", John Wiley and Sons, Inc., New York, 1991
2. Z. Y. Tang, "The system Design of Synthetic Aperture Radar & the New Operating Mode of Linear Frequency Modulated Burst", Master Thesis of IECAS,1999.
3. Y. J. Xu, "Research on SAR Real-Time Parallel Imagery System", PH.D. Thesis of IECAS, 2001.
4. G. M. Peter, R. I. Mabo, G. C. Ian, "Parallel Synthetic Aperture Radar Processing on Workstation Networks" 10th International Parallel Processing Symposium (IPPS '96), April 15–19, 1996, p. 716
5. Y. M. Pi, H. Long, S. J. Huang, "A SAR Parallel Processing Algorithm and its Implementation", Pecora 15/Land Satellite Information IV/ISPRS Commission I/FIEOS 2002 Conference Proceedings.
6. P.B. Curtis, A.G. Richard, J.V. John, "Implementation of the RASSP SAR Benchmark on the Intel Paragon", Proceedings of the Second Annual RASSP Conference, ARPA, 24–27 July 1995, pp. 191–195

Shape Registration Based on Modified Chain Codes

Muhammad Bilal Ahmad¹, Jong-An Park², Min Hyuk Chang²,
Young-Suk Shim², and Tae-Sun Choi¹

¹ Signal and Image Processing Lab,

Dept. of Mechatronics,

Kwangju Institute of Science and Technology,

Kwangju, Korea

{bilal, tschoi}@kjist.ac.kr

² Electronics Information & Communications Engineering,

Chosun University,

Kwangju, Korea.

{japark, millcre}@chosun.ac.kr

Abstract. Shape descriptions based on the traditional chain codes are very susceptible to small perturbations in the contours of the objects. Therefore, direct matching of traditional chain codes could not be used for image retrieval based on the shape boundaries from the large databases. In this paper, a histogram based chain codes are proposed which could be used for image retrieval. The modified chain codes matching are invariant to translation, rotation and scaling transformations, and have high immunity to noise and small perturbations.

1 Introduction

Pattern recognition is the area that studies the operation and design of systems that recognize patterns in data [1]. Important application areas include image analysis, character recognition, speech analysis, man and machine diagnostics, person identification and industrial inspection. In recent years shape recognition has received a lot of interest. It plays an important role in a wide range of applications. For example, imagine that you have a large number of images and wish to select some of them, which are similar to a certain image and by using some image properties such as color, texture and shape structure, then we can retrieve all images that are alike or so.

Shape is one of the salient features of visual content and can be used in visual information retrieval [2]. Shape analysis is useful in a number of applications of machine vision, including medical image analysis, aerial image analysis, and manufacturing. In order to retrieve an image from a large database the descriptor should have enough discriminating power and immunity to noise. In addition, the descriptor should be invariant to scale, translation and rotation.

In this paper, a new method is proposed for shape similarity of 2D objects. The proposed method uses contour-based shape descriptors for shape registration. The objects are recognized using histogram based chain codes matching for contour-based descriptors.

The histogram-based chain-codes are found for the objects, and are used for contour descriptors. The traditional chain codes [3,4,5,6] have many problems for matching shapes. A small change in the object's shape due to noise creates a very large difference in chain codes. Also, the standard chain codes are not invariant to scaling and rotation transformations. In the modified version proposed in this paper, chain codes are found first as usual. 8-connectivity chain codes are used. A normalized histogram is found for the detected chain codes, and histogram is then rearranged in ascending order. The Euclidean distance is then found for the two chain codes, and the minimum distance is the candidate for the similar objects. The histogram-based chain codes are invariant to translation, rotation and scaling transformations. The histogram-based chain codes also have noise immunity.

This paper is organized as follows. Section 2 describes briefly about the traditional chain codes for shape representations. Section 3 explains the proposed algorithm for histogram-based chain codes. Section 4 explains the flow of the proposed algorithm. The simulation results are shown in section 5. At the end, we will conclude our paper with few final remarks

2 Traditional Chain Codes

Chain coding has been widely used for non-predictive loss-less content-based shape coding because of its simplicity. Chain codes describe an object by a connected sequence of straight-line segments of length and direction. In the orthogonal coordinate system, 8-connected objects can be compactly encoded by indicating the directions of transitions along their contours [5]. A code is assigned to each of the possible directions of moving along a contour, where 1 to 8 is assigned to the movements going around clockwise with 45 degrees apart (See figure 1). A large number of descriptive features can be derived from chain codes [6,7]: the area enclosed in a closed chain code, first and second moments about the x- and y-axes, and the distances between two points on the contour.

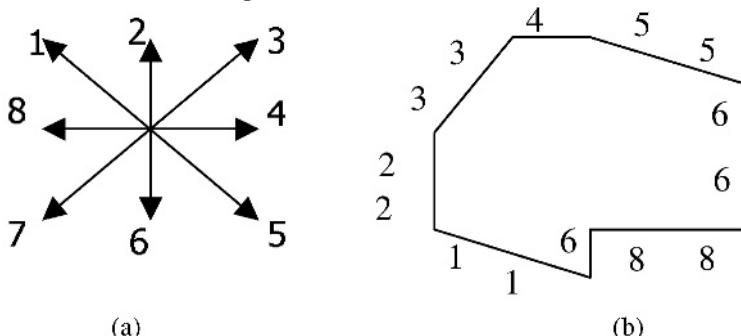


Fig. 1. (a) 8-directional chain code (b) Chain code for an arbitrary object

We easily see that the chain code for a closed contour is not unique; it depends on the starting point. To obtain unique code, one arbitrary encoding is selected and all cyclic permutations are generated from it. Interpreting the codes as ternary numbers, the code that is the smallest is taken as the unique representation of the region.

3 Modified Chain Codes

The standard chain codes have problems in the presence of a small noise. A small change in the chain code gives mismatch results for two images of the same object. Small changes in orientation, occlusion and scaling cause problems in matching the chain codes. For image retrieval, it is very common that the query image has different view of the same object and it would be much difficult to get the similar images using standard chain codes. We, therefore, suggested a new chain code matching based on the histogram, which will overcome the above-mentioned problems.

Let $I_1(x,y)$, and $I_2(x,y)$ are the two different images (possibly two views of a same object). Find the boundaries of the object.

Find the traditional 8-connectivity chain codes of the segmented objects in I_1 and I_2 images, as:

$$cc1 = \text{chaincodes}(I_1); \text{ and} \quad (1)$$

$$cc2 = \text{chaincodes}(I_2); \quad (2)$$

Find the normalized histograms of the resulted chain codes as:

$$NH1 = \text{histogram}(cc1)/\text{length}(cc1); \text{ and} \quad (3)$$

$$NH2 = \text{histogram}(cc2)/\text{length}(cc2). \quad (4)$$

$NH1$ and $NH2$ have only 8 bins corresponding to 1 to 8 directions of the traditional chain codes (see Fig. 1). The $\text{length}(\cdot)$ means the number of chain codes (Length of 45566886112233 is 14). Since the length of each segment is fixed in 8-connectivity chain codes, the normalized histogram of chain codes is scale-invariant. If the object is magnified, its length of chain codes will increase by the same scaling. If we normalize the chain codes by the length of the chain codes, we will get the scale-invariant chain codes.

To get rotation invariant, the normalized histograms of chain codes are sorted in the ascending (or descending) order as:

$$SNH1 = \text{sort}(NH1) \text{ and} \quad (5)$$

$$SNH2 = \text{sort}(NH2) \quad (6)$$

and are placed together with highest number at the right side. This will disturb the original sequence of histogram. This disturbance will give the rotation invariant property. Two images of a same object with different rotations will have different histogram chain codes. For example, one image might have the highest number of '0's in chain codes while in other view of the same object it will might have the highest number as '5's in the chain codes. With sorting, both images will have the highest as '7's in sorted normalized histograms (SNHs). If both the highest numbers are same, the difference will be zero. Similarly, all the 8 bins will be checked. If the two images are of the same object, we will get zero difference of SNHs. Thus, we also

get rotation invariant along with scaling invariance by using sorted normalized histogram chain codes.

To find the difference (or distance) of two SNHs, we used the Euclidean distance measure, so that to account for the small noise or occlusion presents in the images. The direct difference of SNHs will give almost similar results as the traditional chain codes.

4 Shape Recognition

In this paper, after the objects with closed contours have been detected, the shape of the objects is matched using the boundary representation classes. Histogram-based chain codes are used for boundary representations. The objects are well segmented from the input image. We find the histogram-based chain codes for the segmented object. For finding chain code, we first find the vertices of the object, called “Nodes”. Chain codes are then found. Figure 2 shows the flowchart of the proposed recognition method.

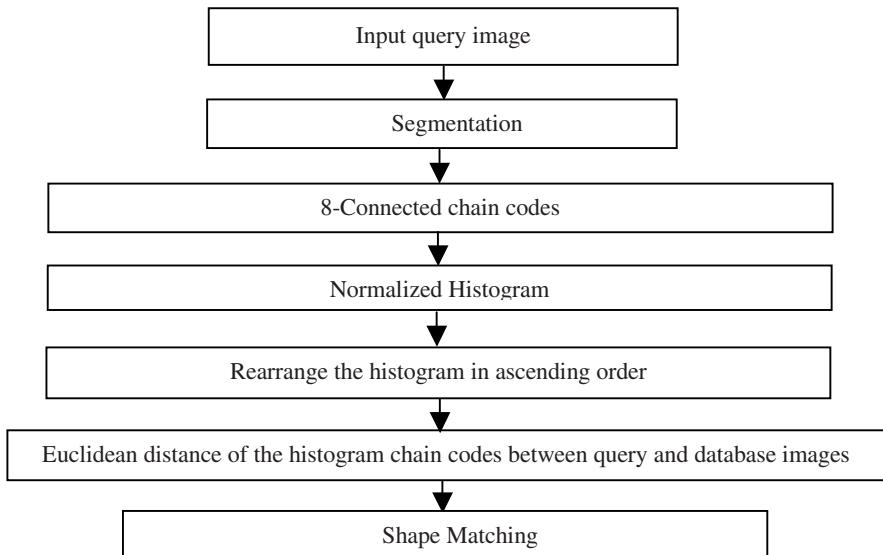


Fig. 2. Flow chart for the proposed method

The noise in the images highly deteriorates the recognition rate, as the boundary of the objects becomes blurred, and creates problems for traditional chain code comparison. To avoid this problem, we use the histogram chain codes, which has immunity against the presence of noise, occlusion, or in the case of inaccurate contour determination. The final decision of object recognition is based on the minimum Euclidean distances found for the chain code features.

5 Simulations

The proposed method was tested on different test images. For matching criterion, we used Euclidean Distance criterion.

Euclidean Distance Matching Criterion

The Euclidean distance is one of the most immediate, simplest and most frequently used similarity measures between two vectors. This measure is defined as the square root of the sum of the squared differences between two feature vectors, one belonging to the query's shape, Q, and the other belonging to a shape in the contents' database, C, for which a description is available.

$$D_E = \sqrt{\sum_{k=0}^{N-1} (Q[k] - C[k])^2}, \quad (7)$$

Figure 3(a), (b) and (c) show three images of an object. In the figure 3(b) image is rotated and one edge is cut to show the noise and occlusion problem. In figure 3(c), image has two cut edges. The histograms of the three chain codes are shown in figure 3(d), (e) and (f). The histograms are normalized to account for the scaling of the objects. The three histograms look very different, but actually they are of the same object. The histograms are then rearranged in ascending order as shown in figure 3(g), (h) and (i). The rearrangement of histogram accounts for rotation invariant. Now the two histograms look very similar. The Euclidean distance is then found for the three chain codes, and the minimum distance is the candidate for the similar objects. The Euclidean distance measure accounts for overcoming the noise and occlusion problem. We used the normalized distances to find the chain codes. The histogram-based chain code is invariant to rotation, scaling, and has noise immunity.

Similarly, we get the results on different image as shown in fig. 4 and fig. 5. The shape matching results are combined and shown in table 1. We used 9 objects as shown in fig 3, 4 and 5. Each image is taken as query and compare with all other images. The Euclidean distances are measured and are shown in table 1. The minimum distances are called as matched objects. If the query object exactly matches with the database object, its Euclidean distance is equal to zero. The very closed matches give very small value of Euclidean distances, as can be seen from the table 1. Suppose, figure 5 (a) is a query object. The Euclidean distances of the rearranged normalized histograms are found for each of the other objects in the database. From table 1, we can see that the Euclidean distances are higher for figure 5 (a) with respect to figure 3 and figure 4 objects. However, the Euclidean distances are very small for the objects in figure 5. The least four distances from the column (or row) of figure 5 (a) are figure 5 (a) itself, figure 5 (b), (c) and (d). It means we get the closely matched objects for the query object.

The drawbacks of traditional chain code are: susceptible to noise. Chain code gives better results on simple closed objects. The histogram based chain codes give a lot of flexibility to chain code based image retrieval system. The histogram based chain codes are invariant to scale, rotation and have immunity against noise and occlusion.

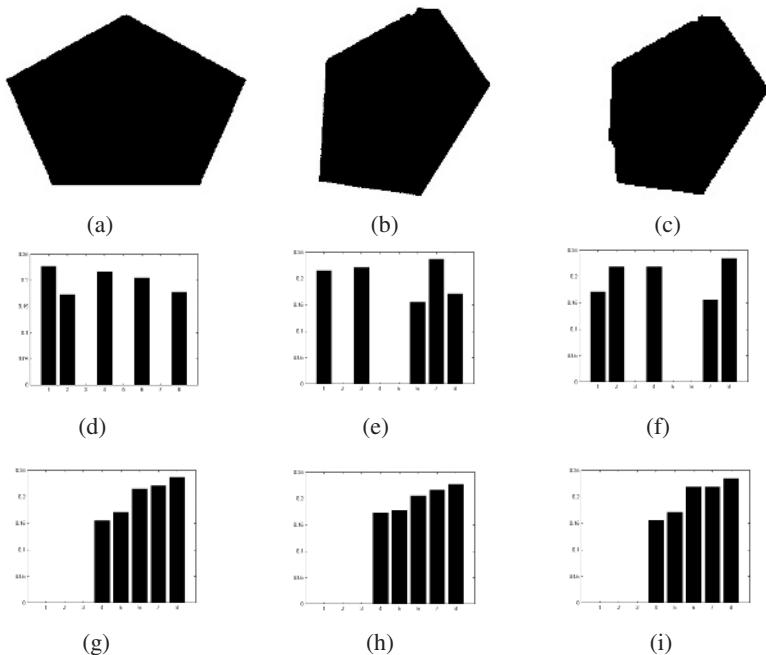


Fig. 3. (a)Original object (b) Scaled, rotated and one cut edge (c) with two cut edges (d) (f) (h) Normalized histograms (e) (g) (i) Rearranged histograms

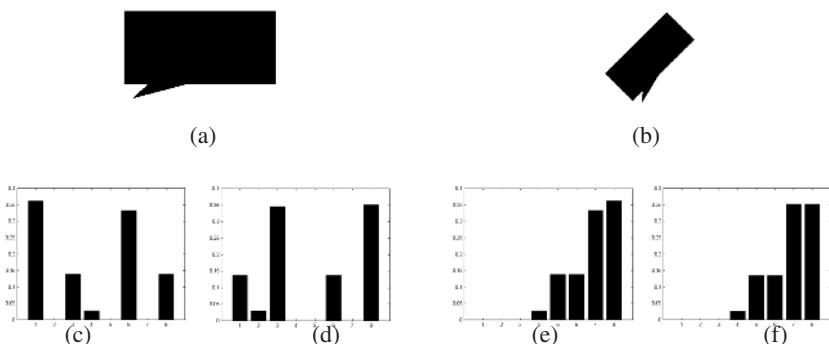


Fig. 4. (a)Original object (b) Scaled and rotated object (c) and (d) Normalized histograms (e) and (f) Rearranged histograms

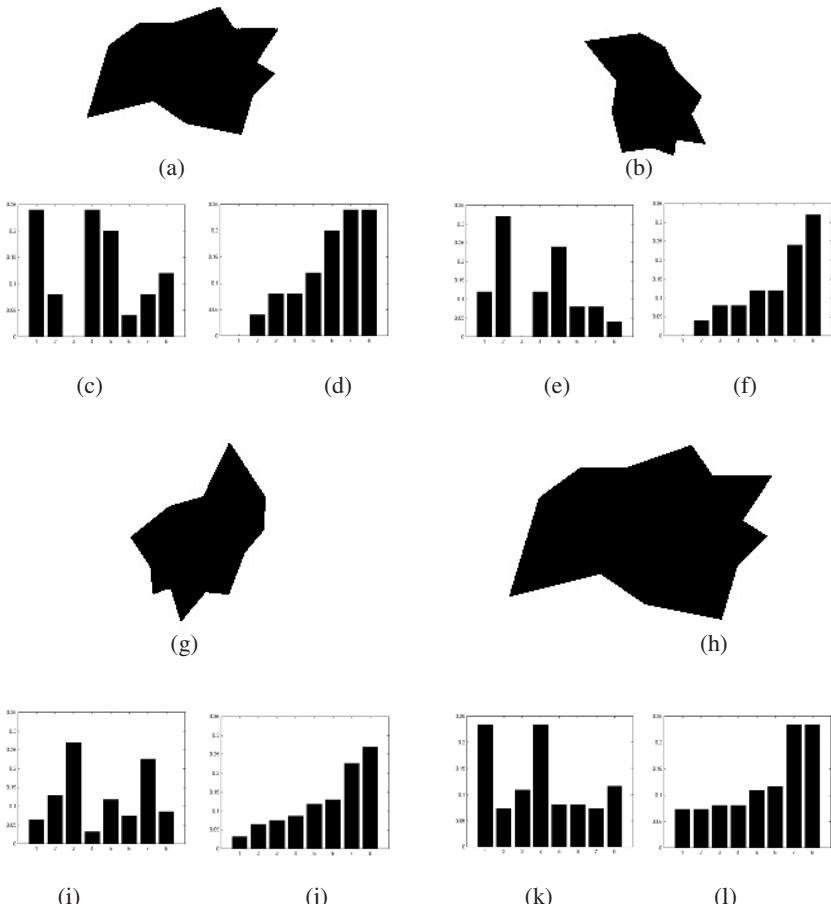


Fig. 5. (a) Original object (b) (g) (h) Scaled and rotated images (c) (e) (i) (k) Normalized histograms (d) (f) (j) (l) Rearranged histograms

Table 1. Euclidean distance between query object and the database objects

	Fig.3 (a)	Fig.3 (b)	Fig.3 (c)	Fig.4 (a)	Fig.4 (b)	Fig.5 (a)	Fig.5 (b)	Fig.5 (g)	Fig.5 (h)
Fig.3(a)	0.0	0.024	0.023	0.242	0.242	0.196	0.191	0.172	0.144
Fig.3(b)	0.024	0.0	0.005	0.225	0.226	0.190	0.180	0.163	0.129
Fig.3(c)	0.023	0.005	0.0	0.230	0.231	0.193	0.184	0.167	0.131
Fig.4(a)	0.242	0.225	0.230	0.0	0.015	0.217	0.147	0.186	0.195
Fig.4(b)	0.242	0.226	0.231	0.015	0.0	0.217	0.152	0.188	0.195
Fig.5(a)	0.196	0.190	0.193	0.217	0.217	0.0	0.118	0.057	0.116
Fig.5(b)	0.191	0.180	0.184	0.147	0.152	0.118	0.0	0.067	0.113
Fig.5(g)	0.172	0.163	0.167	0.186	0.188	0.057	0.067	0.0	0.088
Fig.5(h)	0.144	0.129	0.131	0.195	0.195	0.116	0.113	0.088	0.0

6 Conclusions

In this paper, we have proposed an algorithm for shape matching. The shape recognition using histogram based chain code works much efficiently as compared to traditional chain codes for image retrieval from a large database. Most of the images are highly deteriorated with noise, occluded by other objects, scaled and rotated. The standard chain codes have matching problems in the presence of noise, and are not invariant to rotation and scaling transformation. The histogram based chain codes are invariant to rotation, scaling and reflection transformations. The histogram based chain codes also have noise immunity.

Acknowledgements. This study was supported by research fund from the Chosun University, Kwangju, Korea (2001).

References

1. Morton Nadler, and Eric P. Smith, “Pattern Recognition Engineering” A Wiley Interscience Publication.
2. A Goshidasby, “Registration of images with geometric distortions”, IEEE Trans. Geosci. Remote Sensing, vol. 26, pp. 60–64, Jan. 1988.
3. Berthold Klaus Paul Horn “Robot Vision” The MIT Electrical Engineering and Computer Science Series.
4. R.C. Gonzalez and R. E. Woods, Digital Image Processing, Addison-Wesley, 1993Allen.
5. H. Freeman and J. M. Glass, “Boundary Encoding and Processing”, in Picture Processing and Psychopictorics, B. S. Lipkin and A. rosenfeld, eds., Academic Press, new York, pp. 241–263, 1970.
6. H. Freeman and J. M. Glass, “Computer Processing of Line Drawing Images”, Computing Surveys, 6, No. 1, 57–97, 1974
7. A Kadyrov, M Petrou, “Object descriptors invariant to affine distortions”, The British Machine Vision Conference, pp. 391–400, 2001.
8. E. Bibiesca and A. Guzman, “Shape Description and Shape Similarity for Two-Dimensional Regions”, in: Proceedings of the Fourth International Join Conference on Pattern Recognition, Kyoto, November 7-10, pp. 608–612, IEEE, 1978.
9. E. Bibiesca and A. Guzman, “How to Describe Pure Form and How to Measure Differences in Shapes Using Shape Numbers”, Pattern Recognition, 12, No. 2, 101–112, 1980.

A Distributed Parallel Resampling Algorithm for Large Images

Yanhuan Jiang, Xuejun Yang, Huadong Dai, and Huizhan Yi

Institute of Computer Science,
National University of Defence Technology,
Changsha, 410073, Hunan Province, P.R. China
{jiang-yh,yihuihan}@163.com
xjyang@nudt.edu.cn
hddai@163.net

Abstract. Image resampling is an important and computation-intensive task in many fields. In order to improve its efficiency, a distributed parallel resampling algorithm with good data locality is provided, in which each processor entirely localizes its computation through getting and resampling the corresponding area in output image for local sub input image. A data structure is put forward to save the information of irregular sub output image, and a method is presented to compute local output area. At last, all of the sub output images are gathered and stitched into the integrated target image. By implementing the algorithm on a cluster system, the results show that, for large images, this parallel algorithm improves the efficiency of resampling greatly.

1 Introduction

Image resampling is an important issue in many field, such as texture mapping in computer graphics field [1], image warping in image processing field [2], and geometry correction in remote sensing field [3].

There are two resampling methods: forward mapping and inverse mapping [4]. In forward mappings, input pixels arrive in scanline order, but can be projected into arbitrary positions in the output space. Therefore, a full two-dimensional accumulator array must be retained throughout the duration of the mapping in order to get accuracy intensity for target (output) image. In inverse mappings, operations are performed in scanline order at output space, each output pixel is projected into an arbitrary input position through some inverse mapping function, and a interpolation of the neighbored input pixels is performed in order to get intensity value for the output pixel.

Resampling is a computation-intensive task, and always needs lots of time when process large images. Fast resampling methods in literature are reducing computational complexity and hardware implementation. Catmull and Smith [5] first presented the idea of separable resampling, which decomposes the forward mapping function into a series of simple 1-D transforms. Their method saves I/O time substantially with efficient data access. Wolberg and Massalin [6] used lookup tables to fasten resampling process. Meinds and Barenbrug [7] provided a hardware structure to implement resampling more efficiently.

Parallel processing is an efficient way to fasten computing in many applications. Cluster system [8] has high performance/price ratio and good scalability, which is preferred by many researchers and users. This paper explores a distributed parallel algorithm of inverse mapping resampling with good data locality. In the algorithm, each processor gets the corresponding area in output image for local sub input image, and does resampling for the area. This makes all of data needed be in local memory and no communication happens during parallel computing. We present an algorithm to get local output area, and also design a data structure to save the irregular sub output image, which are two key problems in the distributed parallel resampling. Section 2 sketches the serial algorithm of inverse mapping resampling. Section 3 describes our distributed parallel resampling algorithm in detail. Experimental results and their analysis are reported in section 4, show that our algorithm fastens resampling process greatly for large images. Section 5 gives a summarization of our work.

2 Serial Inverse Mapping Resampling

Image resampling is the process of transforming a sampled image from input coordinate system $[x, y]$ to the output one $[u, v]$. The transformation is expressed as a mapping function that defines the geometric relationship between each point in the input and output images. The forward mapping is $[u, v] = [U(x, y), V(x, y)]$, and the inverse one is $[x, y] = [X(u, v), Y(u, v)]$, where X, Y, U, V are arbitrary mapping functions that uniquely specify the spatial transformation.

Inverse mapping method is more commonly used in image resampling. In this paper, we are given a rectangular input image and forward mapping functions U, V to get the warping result by using inverse mapping resampling. The following is the description of the main steps in the resampling process:

1. Getting inverse mapping functions: it is always too difficult to compute their expressions. In applications, we calculate some pairs of homonymic points $([x, y], [u, v])$ according to the given forward mapping functions, where all $[x, y]$ are on some coarse grid of the input image; then use least squares method to regress these coordinate pairs and get approximate inverse mapping functions, always in polynomial form.
2. Geometric transformation: for each output pixel $[u_i, v_i]$, get its conjugate point $[x_i, y_i]$ in input space according to the inverse mapping functions.
3. Intensity interpolation: interpolate the intensity values of the neighbor pixels of $[x_i, y_i]$ in input space to get the intensity value of $[u_i, v_i]$.
4. Repeat steps 2 and 3, until we get intensity values for all of output pixels.

For above steps, computing inverse mapping functions takes little computing time; while step 2 and step 3 will be executed repeatedly until all of pixels in output image get their intensity value, they are computation-intensive steps. Supposed that inverse mapping functions are complete cubic polynomials, and intensity interpolation uses cubic convolution, all of polynomial computations are simplified by QinJiushao method, then one output pixel needs 20 floating multiply-add operators for geometric transformation and 36 floating multiply-add operators for intensity interpolation.

3 Distributed Parallel Resampling Algorithm

Parallel processing is the most efficient way to fasten computing in many applications. Resampling large images adapt to data parallelism [9]. In this section, we introduce direct parallel algorithm and analyze its limitations at first, then explore a distributed parallel algorithm which solves data locality, two key problems of distributed parallel resampling are described consecutively.

3.1 Direct Parallel Resampling

A direct parallel algorithm for image resampling is: regular 1-D or 2-D data partition for the storage of input image and also the resampling computation of output image straightforwardly. Fig. 1 describes the result of 1-D partition along the height direction, where each node saves a regular sub input image, and computes the intensity for a regular sub output image.

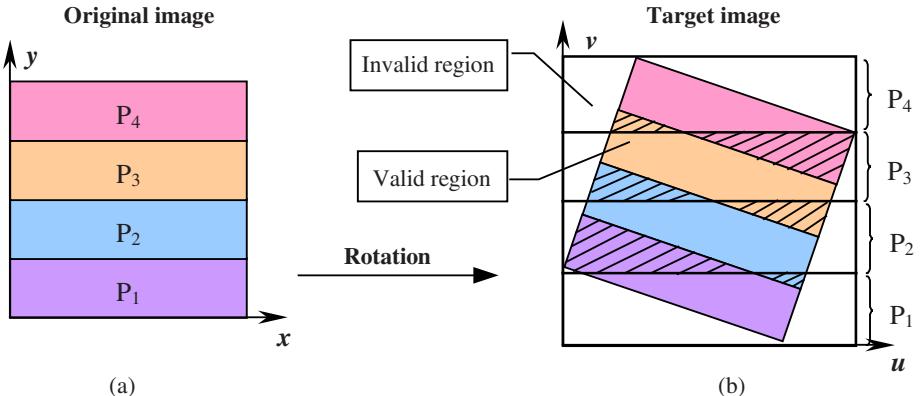


Fig. 1. Straightforward partition for direct parallel algorithm: (a) regular 1-D partition for data storage; (b) Regular 1-D partition for resampling computation

If we use direct parallel algorithm under distributed environment, a lot of data needed in computing process were not stored locally. For example, the conjugated points of the output pixels in the hatched area in Fig. 1(b) were not stored locally, and can only be gained from other processor through communication. This leads to frequent small data transfer among processors, which needs a lot of time. On the other hand, the background pixels in invalid area (see Fig. 1(b)) also need geometric transformation to decide whether their conjugate points are in or out of input image. In fact, if we know the position of invalid pixels in output space, we need not waste computing time on them, just evaluate their intensity values to the background color (in default is black). For these two reasons, we know that direct parallel resampling is not suitable to distributed environment.

3.2 Distributed Parallel Resampling

Under the distributed environment where multiple computers work cooperately, we explore a parallel resampling algorithm for large images, which has good data locality through computing output area for the locally saved sub input image. Fig. 2 gives the flow chart of our distributed parallel resampling algorithm.

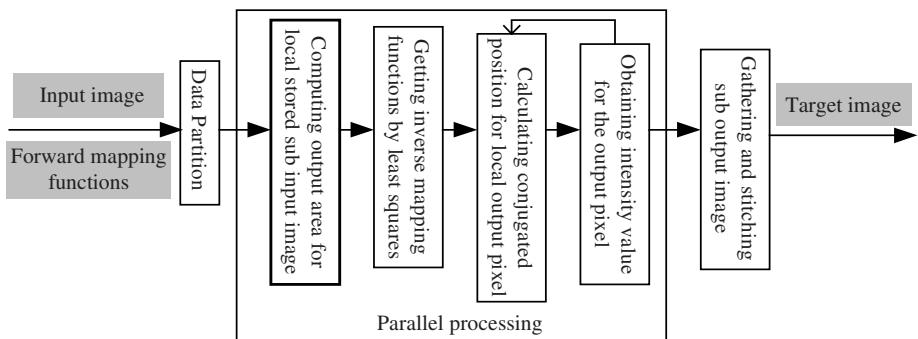


Fig. 2. Flow chart of distributed parallel resampling algorithm

We let computing node denote the node doing parallel computing, and managing node be the node taking charge of global operators. Different from direct parallel algorithm, the output image is not parted regularly in distributed parallel algorithm, where each computing node does resampling computation for the pixels in local output area. The steps of the algorithm can be described as follows in detail:

1. Data partition: regular 1-D or 2-D data partition for the input image, each computing node stores one sub image. In order to avoid communications during parallel computing, some extra data should be stored redundantly according to the size of resampling template. For example, a 4*4 template for cubic convolution needs two columns of extra data at the margin of local sub input image.
2. Computing local output area: by using the given forward mapping functions, each computing node gets local output area, and uses a new data structure to save the information of the irregular sub output image. This is the key step of our algorithm for it guarantees data locality.
3. Getting local inverse mapping functions: obtains some pairs of homonymic points $([x,y],[u,v])$ at first, where each point $[x, y]$ is the coordinate of a input pixel on some coarse grid of local sub input image, then uses least squares method to get local inverse mapping functions.
4. Local resampling: does geometric transformation and intensity interpolation for each pixel in local output area.
5. Gathering and stitching sub output images: all of irregular sub output images are gathered and stitched into the integrated target image with normal format at managing node.

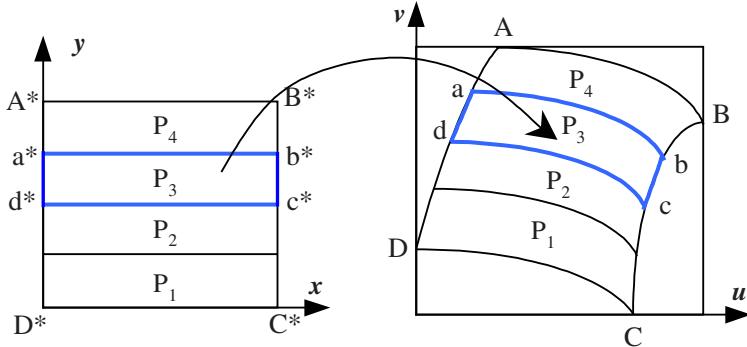


Fig. 3. Data locality of computing node P_3 , when calculating intensity for any pixel in local sub output area $abcd$, all required input pixels are in $a^*b^*c^*d^*$ which stored locally

Steps 2,3 and 4 are implemented on computing nodes according to data parallelism, while steps 1 and 5 are global operators and be realized on managing node. Fig. 3 describes data locality of our algorithm, where input image is divided into 4 sub images with equal size regularly along the height direction. For computing node P_3 , its local sub input image $a^*b^*c^*d^*$ corresponds to the output area $abcd$ in output image, when computing intensity for any pixel in $abcd$, all required input pixels are in $a^*b^*c^*d^*$ which stored locally (including redundantly saved data). To guarantee the data locality, getting and saving the information of irregular sub output image are two key problems.

3.3 Saving Structure for Local Sub Output Image

We provide a new data structure to save the edge and intensity information for each local sub output image. Supposed that there are p computing nodes, the edge information of the i^{th} ($1 \leq i \leq p$) sub output image includes following items:

v_{i1} : the minimum v coordinate value of the pixels in local sub output image;

L_i : number of scanlines on which there are local output pixels;

s_{ij} : number of pixel segments on the j^{th} ($1 \leq j \leq L_i$) scanline;

u_{ijk} : the u coordinate value of the first pixel on the k^{th} ($1 \leq k \leq s_{ij}$) pixel segment;

r_{ijk} : number of pixels on the k^{th} ($1 \leq k \leq s_{ij}$) pixel segment;

In Fig. 4(a), the j^{th} scanline in the sub output image on computing node P_i has two pixel segments, the starting u coordinate value and number of pixels on the first pixel segment are u_{ij1}, r_{ij1} , and those of the second pixel segment are u_{ij2}, r_{ij2} .

Fig. 4(b) gives our saving structure for sub output images. The content of the structure consists of edge information and intensity values. There are two sub data structures in the saving structure, BlockHead and SegHead:

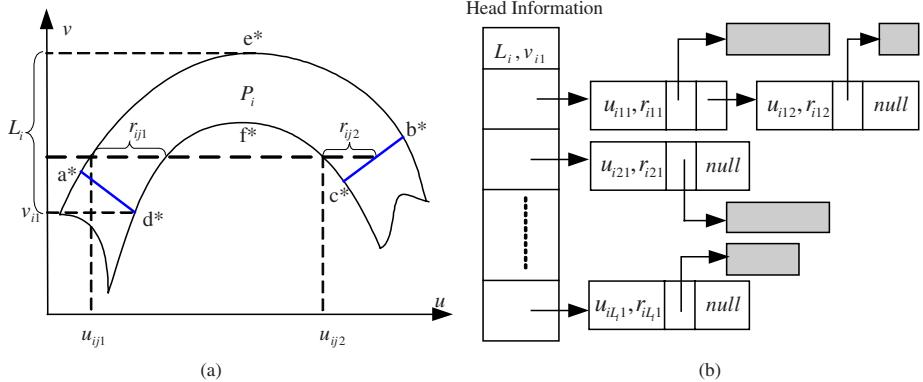


Fig. 4. Design of saving structure: (a) edge information of local sub output image on P_i ; (b) saving structure of sub output image on P_i

```

struct BlockHead
{
    int StartVertical;
    int BlockHeight;
    struct SegHead** SegInfo;
}

struct SegHead
{
    int StartHorizontal;
    int SegWidth;
    int* Intensity;
    struct SegHead* NextSeg;
}

```

BlockHead saves v_{ij1} and L_i , and memory addresses pointing to the information of L_i scanlines. SegHead saves the information of pixel segment, including u_{ijk}, r_{ijk} , memory address of to intensity value for these r_{ijk} pixels, and the address of next segment on the same scanline.

3.4 Computing Local Output Area

For our resampling problem, valid output pixels form a closed area with no background in it. We use closed line segments connected end to end with each other to represent the ideal edge of sub output area approximately, two end points of each segment are the homonymous points of two neighbored margin pixels of the input image. This approximation simplifies the local output area computing. For the convenience of description, we give two definitions at first.

Definition 1. (Local extremum point): Supposed that sub input image has N margin pixels on the edge $a^*b^*c^*d^*a^*$, among their homonymous points, the minimum point is the one which has the least u coordinate value in those pixels whose v coordinate values are the least. Starting at the minimum point, we access these homonymous points in anti-clock direction along the edge of local output area, and let the accessing sequence be $[u_1, v_1], [u_2, v_2], \dots, [u_i, v_i], \dots, [u_N, v_N], [u_{N+1}, v_{N+1}]$, where both $[u_1, v_1]$ and $[u_{N+1}, v_{N+1}]$ are the same minimum point, then:

- (1). The minimum point is a local minimum point;
- (2). If we have $v_i \geq v_{i-1}$ and $v_i > v_{i+1}$, then $[v_i, v_i]$ is a local maximum point;
- (3). If we have $v_i \leq v_{i-1}$ and $v_i < v_{i+1}$, then $[v_i, v_i]$ is a local minimum point.

From above definition, we can know that local minimum point and local maximum point appears alternately. Using local extremum points, we give following definition:

Definition 2. (Left/right edge segment): Beginning from the minimum point, we access local extremum points along the closed edge of sub output image in anti-clock direction, the edge segment from a local minimum point to the next local maximum point is a left edge segment where neighbored valid pixels are on its right side; the edge segment from a local maximum point to the next local minimum point is a right edge segment where neighbored valid pixels are on its left side.

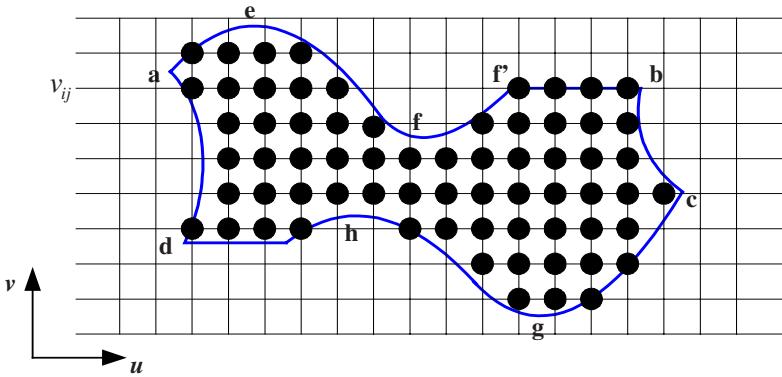


Fig. 5. Output area of one sub input image with some complex transformation

Supposed that the sub input image on the i^{th} computing node is $a*b*c*d*$, its corresponding output area is $abcd$. Our method of computing local output area for the i^{th} ($1 \leq i \leq p$) computing node can be described as follows:

1. Obtaining coordinate values of the homonymous points for each marginal pixel of input image according to forward mapping function.
2. Getting all of local extremum points. Fig. 5 is a complex warping result of local sub input image, g, d, f are the local minimum points, the set of them is called *MinSet*, g is the minimum point for $v_{\min} = v_g$, and h, e, b are the local maximum points, the set of them is called *MaxSet*, e is the maximum point for $v_{\max} = v_e$.
3. Acquiring all left/right edge segments for the local sub output image. In Fig. 5, gh, de, fb are left edge segments and hd, ef, bg are right edge segments. We save these edge segments by using data structure as Fig. 6(a). For left edge segment, the bool variance $sign = 0$, for the right one, $sign = 1$. Variance v_{lower} (v_{upper}) is the minimum (maximum) v coordinate value among the homonymous points on this edge segment. The coordinates of these homonymous points are all saved in the sequence of corresponding marginal input pixels in anti-clock direction. Fig. 6(b) is the saved information of the edge segments of Fig. 5.

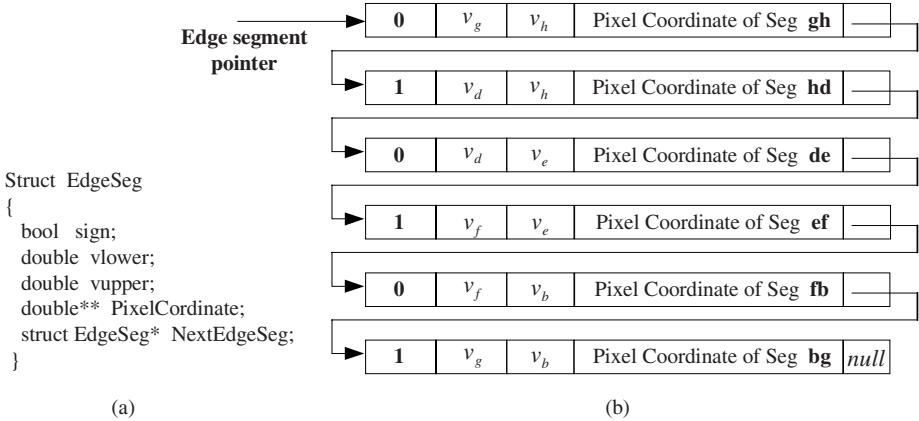


Fig. 6. Sketch of edge segment structure: (a) the structure of edge segment; (b) the saved edge segment information of Fig.5

4. Getting head information for the local output image. We have $v_{il} = \lceil v_{\min} \rceil$, $L_i = \lfloor v_{\max} \rfloor - \lceil v_{\min} \rceil + 1$, and let $v_{i0} = \lceil v_{\min} \rceil - 1$, $j = 0$.
 5. Computing the information of pixel segments on the $(j+1)^{th}$ scanline:
 - (1). If $j \geq L_i$, local output area computing is finished.
 - (2). $j = j + 1$, $v_{ij} = v_{i,j-1} + 1$.
 - (3). Computing the intersections of the j^{th} scanline with each edge segments: for the k^{th} edge segment, if $v_{ij} \notin [v_{lower_k}, v_{upper_k}]$, then the scanline does not intersect with it, check the next edge segment in turn; else we connect the homonymous points in the saved sequence to represent the ideal edge segment, and get the intersection of the approximate edge segment with scanline $v = v_{ij}$. For the left (right) edge segment with one part of its just on the scanline, we just let the left (right) most point of this part to be the intersection.
 - (4). Getting the information of pixel segments: after step (3), we will get even number of intersections on the scanline. From the left to the right of the scanline, the intersections belong to left and right edge segments alternately. Let the sequence of their u coordinates is $u_{l1}, u_{r1}, u_{l2}, u_{r2}, \dots, u_{lm}, u_{rm}$, then there are m pixel segments on this scanline. The start u coordinate of the k^{th} ($1 \leq k \leq m$) one is $u_{ijk} = \lceil u_{lk} \rceil$, and its length is $r_{ijk} = \lfloor u_{rk} \rfloor - \lceil u_{lk} \rceil + 1$. In Fig. 5, four edge segments *de*, *ef*, *fb*, *bg* intersect with scanline $v = v_{ij}$, and form two pixel segments with 5 and 4 pixels on them accordingly.
 6. Turn to step 5, calculate information for the next scanline.
- After getting local output area, we can acquire the position of each pixel in this area. In Fig. 5, all of black dots are valid pixel positions in the area *abcd*. Each computing node resamples the intensity for its local valid output pixels. For any valid pixel of the output image must be in one of local output area, this guarantees the

integrality of the target image. At last, all of local sub output images are gathered and stitched into the integrated target image with normal format, such as *.bmp, on the managing node.

4 Experimental Results and Their Analysis

We implement our parallel algorithm on a cluster system, and use several images with different sizes to test the performance.

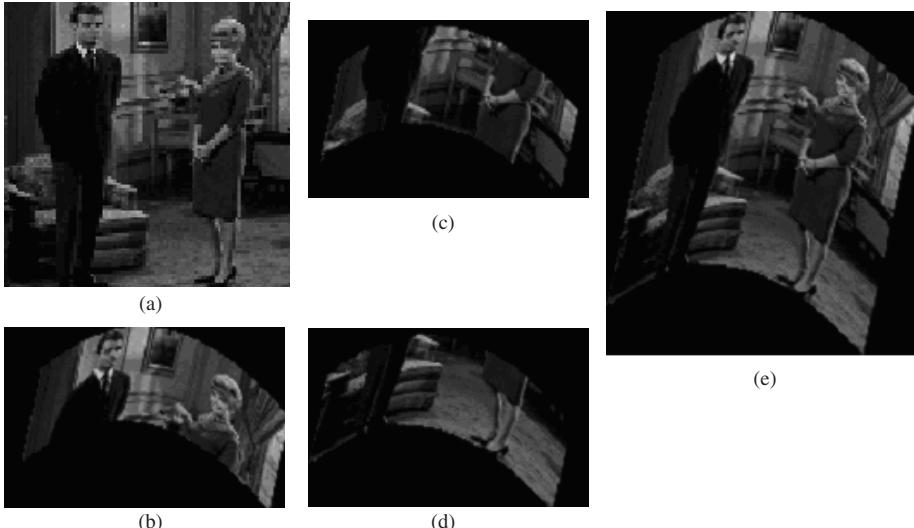


Fig. 7. Result of distributed parallel resampling: (a) input image; (b) sub output image of P_3 ; (c) sub output image of P_2 ; (d) sub output image of P_1 ; (e) target image

The configuration of our platform is: eight computers with Pentium4-2G CPU connected by 100Mbps fast ethernet switch, seven of them are computing nodes, each has 512MB local memory, the remaining one is managing node with 1GB local memory. The algorithm works on the Windows2000 OS with support of the NT-MPICH. Both forward and inverse mapping functions are complete cubic polynomials. Intensity interpolation is implemented by cubic convolution. There are 100 coarse grid points when using least squares to calculate inverse mapping functions. Input image is divided in regular 1-D partition way by managing node and allocated to each computing node, after parallel computing, all of output data are gathered and integrated on managing node. Fig. 7 gives the result of one test image by using three computing nodes.

When original images come continuously, parallel computing and data transfer can be overlapped with each other. Operators performed on managing node are not intensive and can be overlapped completely with data transfer, so the whole processing time of one image resampling is the larger of data transfer time and parallel computing time. In order to test the overlapping performance, we let the same image be the original image repeatedly. Table 1 gives our test results:

Table 1. Time results of distributed parallel algorithm with different image size

Size of original image	Serial algorithm (s)	Distributed parallel algorithm (s)		
		Parallel computing	Data transfer	Total
3000^2	16.68	2.49	1.62	2.49
6000^2	66.12	9.85	6.48	9.85
10000^2	178.31	26.34	17.99	26.34
20000^2	710.43	104.56	71.98	104.56

From table 1, we can see that, under our test platform, input and output data transfer can be completely overlapped with parallel computing, the total time for the one image resampling lies on its parallel computing time.

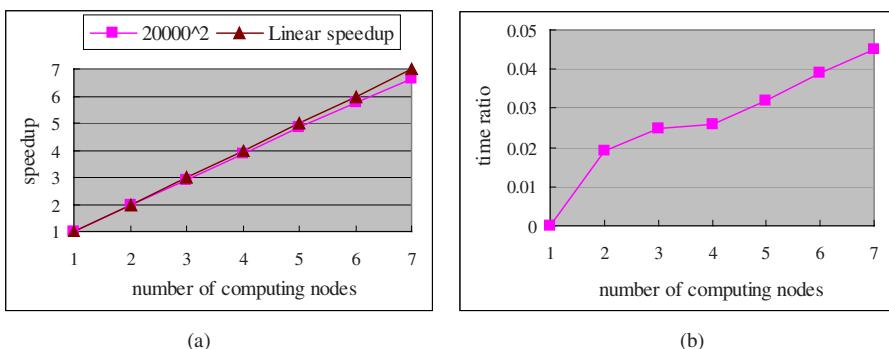
**Fig. 8.** Result of distributed parallel resampling: (a) speed-up with different computing nodes; (b) time ratio of the local output area computing to parallel computing

Fig. 8(a) shows that our parallel algorithm has good performance for it can get linear speedup approximately. From Fig. 8(b) we know that the ratio of the time needed for local output area computing to the whole parallel computing increases with computing nodes. This means that our algorithm can't adapt to fine-grained parallel computing, because for large distributed system with many computing nodes, each node gets small part of the input data, and the number ratio of local marginal pixels to the local internal pixels is high, this reduces the efficiency of parallel processing.

5 Conclusion

Parallel processing is the most efficient way to fasten computing in many applications. This paper explores a distributed parallel resampling method for large images with good data locality, where each computing node computes the output area for local stored sub input image and resamples the intensity for local valid output pixels. The important problem in our algorithm is how to get and save the edge of local output area. This paper proposes a method of computing local output area, and also put forward a saving structure for sub output images. All sub output images are

gathered and stitched into the integrated target image on managing node at last. We implement our distributed parallel resampling algorithm on a cluster system, the experimental results show that our algorithm can get linear speedup approximately for large images.

Image resampling is an important and computation-intensive task in many fields, and always needs much process time; while distributed parallel environment such as cluster system has good scalability and high performance/price ratio, so our distributed parallel resampling algorithm has usefulness in many applications.

References

1. Heckbert, Paul S.: Survey of texture mapping. *IEEE Computer Graphics and Application* (1986) 207–212
2. Chen, B.Q., Dachille, F., Kaufman, A.: Forward perspective image warping. *IEEE Visualization* (1999) 89–96
3. Zhang, Y.S., Wang R.L.: Dynamic Inspection in the Remote Sense. Peking: Publishing House of Liberation Army (1999) 65–80
4. Wolberg, G., Sueylam. H.M., Ismail, M.A., Ahmed, K.M.: One-dimensional resampling with inverse and forward mapping functions. *Journal of Graphics Tools*, Vol. 5. 3 (2001) 11–33
5. Catmull, E., Smith, A.R.: 3-D transformations of images in scanline order. *Computer Graphics, (SIGGRAPH'80 Proceedings)*, Vol.14. 3(1980) 279–285
6. Wolberg, G., Massalin, H.: Fast convolution with packed lookup tables. *Graphics Gems IV*, Ed. By P. Heckbert, Academic Press (1994) 447–464
7. Meinds, K., Barenbrug, B.: Resample hardware for 3D graphics. *SIGGRAPH/Eurographics Graphics Hardware workshop* (2002)
8. Buyya R.: *High Performance Cluster Computing: Architectures and Systems*, Vol. 1. Prentice Hall PTR, Upper Saddle River, NJ (1999)
9. Daniel Hillis, W., Guy L., Steele, Jr.: Data parallel algorithms. *Communcations of ACM* (Special issue on parallelism), Vol. 29. 12 (1986) 1170–1183

Collaborative Supervision of Machine Vision Systems: Breaking a Sequential Bottleneck in the Supervised Learning Process

Steve Drew, Sven Venema, Phil Sheridan, and Chengzheng Sun

School of Computing and Information Technology,
Logan Campus, Griffith University,
Brisbane, Qld 4131, Australia

{S.Drew, S.Venema, P.Sheridan, C.Sun}@griffith.edu.au

Abstract. This paper describes a computer vision system in the context of exploiting parallelism. The key contribution is a description of a network design that breaks a long-standing bottleneck in the supervision phase of the vision process. The proposed solution draws from and contributes to the disciplines of machine learning, computer vision and collaborative editing. The significance of the solution is that it provides the means by which complex visual tasks such as mammography can be learned by an artificial vision system.

Keywords: Collaborative editing, computer vision, machine learning, supervised learning

1 Introduction

Recent developments in the fields of computer vision and collaborative editing have created new synergies amongst the hitherto independent communities engaged in research in the disciplines of machine learning, computer vision and collaborative editing. Machine learning algorithms (eg. artificial neural networks [7], Bayesian learners [1], etc.) have for some time played an important role in computer vision systems. However, one of the impediments to a robust employment of such algorithms is obtaining a timely quantity and quality of training examples required for the training algorithms.

Recently, Sheridan demonstrated a method for creating high quality training examples of a visual task [12]. Developments in collaborative editing [14] have provided the potential to impact on the ‘quantity issue’ side of the problem. In this paper, we will overview the current state of a computer vision system that delivers high quality training examples. Then a major weakness in the system, a bottleneck, will be described. A solution to the problem emerges out of considerations of a new paradigm to both computer vision and machine learning. This paradigm, collaborative supervision, was presented in [16].

The main thrust of this paper is to describe a parallel implementation of the collaborative supervision paradigm. In the following sections we present an overview of the Akamai machine vision/learning system and identify inherent parallelism and the sequential bottleneck that exists within it. We then consider some requirements for

a concurrent, distributed solution to the bottleneck problem. Processes for collaborative supervision in machine learning are defined and some models that take advantage of inherent social and architectural parallelism are developed.

2 Akamai: A Parallel Computer Vision System

2.1 Overview of Akamai

Akamai is an artificial vision system, which mimics many aspects of sophisticated biological vision systems. The key features that distinguish Akamai from conventional computer vision systems [6] are:

- Space-variant sensing: Sensing is performed with varying resolution across the sensor. This is the dominant sensor architecture in biological vision. Mathematical analysis shows that it is 10,000 times more efficient than space-invariant sensing [2], which is employed by conventional artificial vision systems.
- Hexagonal lattice: The fundamental units of vision, pixels, are situated and processed in clusters of seven equally spaced units. All primates, cats and birds employ this feature. It derives its superiority over the rectangular lattices employed by conventional vision systems from a mathematically provable property, known as the Honeycomb Conjecture [11]. It simply produces less error for the system and thus makes the system more robust.
- Saccadic eye movement: This is the process of rapid and continuous eye movements. It facilitates visual recognition with the visual target either moving or still, with the system immobile. In biological vision, this feature has manifested itself at the top of the evolutionary tree, i.e., the human vision system [8].
- Local/global processing in the visual cortex: This is a process where by the input image is processed as a multitude of sub-images concurrently, while simultaneously processing the entire image. This feature is neurologically observable in the human cortex [9].
- Supervised learner: Anything that Akamai recognises, it must first learn from a human expert. Its unique architecture allows it to capture the visual expertise of a human without the human supervisor even knowing how she/he makes a visual recognition of some local artefact of interest. This is the single greatest distinguishing feature that separates Akamai from all other artificial vision systems [15]. Akamai interrogates the human supervisor during the process of visual recognition. On completion of the interrogation phase, Akamai learns the concepts that it has obtained from the supervisor. However, there is at least one significant weakness in the system. The process of capturing the visual expertise of human supervisors is a sequential bottleneck in the current version.

2.2 Parallelism in Akamai

The concepts of space-variant sensing and the hexagonal lattice were combined to form the underlying architecture for a novel approach to artificial vision [12]. This approach, named Spiral Architecture, attempts to extract computational principles inherent in biological vision systems and implement them in digital technology. The mathematical structure of the Spiral Architecture is Lie Algebra and is described in [12]. This approach is potentially productive because optical flow in a biological vision system is inherently parallel. From the eye to the optic nerve, and the optic nerve to the visual cortex, vision systems tend to exploit parallelism in a multitude of ways [8]. For example, a principle observable in the visual behaviour of humans has been described as the search/learn principle [15]. This principle states:

When executing a visual search, operate at the lowest possible resolution and widest possible angle of view, going to higher resolution viewing only when more information about candidate objects across a localised area is required. When executing visual learning, foveate the object and learn its features at highest possible resolution within the smallest possible angle of view. Then view the object at lower resolution and wider angle of view attempting to learn the object's abstracted features in a wider context.

At first glance, a casual reading of the search/learn principle may suggest that the process is purely sequential. However, the process is actually a combination of sequential and parallel components. The process of moving from lower resolution to higher resolution in the case of search and moving from higher resolution to lower resolution for learning are sequential processes. However, the resolution at which to commence either the search or the learn process is not known a priori. Moreover, the process of finding candidates of the target objects at a given resolution is an inherently sequential one. Consequently, when performing the search operation, it is efficient to initiate independent search processes at all possible resolutions in parallel. The lowest level of resolution at which the target appears will be the first process to find an occurrence of the search target. [3]

The locality information of this low-resolution representation of the candidate target is employed to commence the search at the next higher resolution. Once again, computational considerations predict that the search target will be found at this resolution through the lower resolution sidestep before the process that was initially commenced to find the target at this resolution. On the other hand, if the target had not appeared at the lower resolution, then the process at the higher resolution would have been the first to find the occurrence of the target. The point is that the computational demands of a search grow exponentially with increased resolution. The use of the locality information is the key to reducing the size of the search space in the higher resolutions.

A similar argument applies to the learn process. Locality information provided by the supervisor is employed to commence a search for the target at all resolutions independently. Computational considerations of Akamai's space-variant sensor suggest that search complexity is equivalent for all resolutions [10]. Therefore, the highest resolution at which the target is located is the optimal one in which to begin the process of learning the features of the target using progressively lower resolutions.

Computationally, the point in this case is that the highest resolution at which the target will appear in the internal representation of the vision system is not known a priori. Consequently, the search for the highest resolution at which the target appears should proceed in parallel at all resolutions.

2.3 The Bottleneck in the Vision Process

In Akamai's single supervisor system, the bottleneck in the machine-learning process is the collection of enough supervision input (training examples) such that the machine has a high degree of accuracy in its artefact recognition. In single-supervisor operation the scanning and analysis of a single image file might take up to twenty minutes and several images may have to be analysed in order to successfully teach the machine how to discriminate accurately. The conventional approach to producing a text document through the use of a word processor describes a similar bottleneck. Research to date has produced a number of approaches to the text-editing bottleneck [14]. It is from considerations of these approaches, including distributed and real-time collaborative solutions that a design for concurrent collaborative supervision for machine learning has emerged.

3 A Concurrent Distributed Solution to the Bottleneck Problem

3.1 Collaboration in the Supervision of Machine Learning

Collaborative supervision in machine learning is effected when human experts can work together effectively to produce training examples for a machine-learning algorithm. Advantages of such an arrangement include accelerated generation of training examples, improved accuracy of analyses through real-time supervisor awareness, and interactive task completion. In this instance, distribution of analysis tasks over a number of discrete sites requires significant levels of social and architectural design for parallelisation to be successful.

In a distributed machine-learning environment where individual supervisors are connected via the network, there are different modes of operation that the system might employ based upon different levels of mutual awareness, interaction and collaboration between supervisors and with the system. Modes range from "disjoint" or asynchronous to real-time collaboration and describe different levels of social parallelism and concurrency that can be applied to the supervision task. Factors such as level of need for conflict resolution and acquisition of multiple analyses affect the levels and types of concurrent interaction that might be needed. Network service quality affects the overall type of system architecture that is most appropriate for maintenance of system responsiveness. This also has implications for the types of operation that are available to the supervisor.

Machine learning, as implemented in the Akamai system, may be broken down into a number of discrete steps. Step one is the collection of training examples. This is the sequential bottleneck in the stand-alone Akamai model. In a distributed and collaborative model the operation of this step is based around facilitated human-

human and human-machine interaction. Each supervisor can work on separate analyses concurrently, thereby improving the rate of training example creation. Supervisors can also interact in a real-time collaboration on each other's analyses to provide feedback and aid correctness of submitted training examples.

In the process of merging examples from various supervisors analyses, coincidences and conflicts are used to start building a set of generalised rules. Examples that do not fit the generalised rules may be resubmitted to one or more supervisors for conflict resolution. Once the rule-base is at a testable stage, rules may be distributed to each discrete learning site to start the third step, rule verification. As required, each peer site may be elevated to a more aggressive learning mode. Human supervisors may then supervise a machine analysis, or compare analyses with a machine's attempts, to verify correctness and completeness of the learning.

The set of tasks presented in the steps above have a significant impact on interface design. A user interface becomes much like a virtual whiteboard with a work area and communication area that all collaborators can interact with and contribute to. The communication area might include a consensus or voting section where training examples can initially be submitted for conflict resolution, and then to the rule engine by mutual agreement.

For real-time analysis to be effective, it is beneficial to have a reasonably strict process and procedure to guide the collaborative exercise [4,5]. A set of mutual synchronised steps, defined roles and objectives for all collaborators will help keep process efficiency high. In the rest of this section we explore models that address the "bottleneck" of collecting training examples in a distributed, collaborative working environment. We will look at two distinct architectures that can be combined with one of two supervision arrangements. Each combination has implications for communications infrastructure and overall efficiency.

3.2 Supervisor Social Structure

With a highly interactive real-time arrangement as suggested below, much of the interpretation and analysis can be done collaboratively amongst the supervisors before submitting a training example or committing a rule to the rule-base. In this instance, social arrangements have consequences for network traffic, individual supervisor workload and general process efficiency. Where each of the supervisors is considered a peer with comparable knowledge to other supervisors, initial analyses, conflict resolution and verification tasks can be allocated using a system algorithm. Decisions can be made collaboratively by consensus or consultation where there are distinct areas of expertise.

Where one supervisor is considered a leader and has superior knowledge or experience, tasks and traffic to one node may increase. Social interaction in analysis may be governed by how supervision tasks are allocated. Distribution of analysis tasks amongst supervisors has implications for rate of production and accuracy of training examples. Allocation of separate analysis tasks, one to each supervisor, may lead to a high production rate for training examples, but may produce a high error rate due to lack of collaboration between supervisors. Conversely, having several supervisors collaborating on the analysis of one image can produce a high level of accuracy, but may lead to a very low production rate for training examples.

3.3 Models for Collaborative Akamai Architecture

Two potential models for arranging processing and communication are client-server and peer-to-peer. In a client-server architecture, tasks such as image storage and distribution, machine-based resolution of training example “conflicts”, generation of rules and storage of analysis related data structures might all be done centrally. Akamai clients would communicate with servers to accomplish processing tasks, generating significant network traffic in the process. In a high-bandwidth networking environment this might be a reasonable arrangement but it does not take full advantage of the processing power sitting on each supervisor’s desk. Nor does it provide maximum system responsiveness should network communications unexpectedly degrade [13].

The level of supervisor interaction and collaboration in analysis also has an impact on network traffic. Where supervisors analyse separate images, each Akamai client might communicate separately with the example collation and conflict resolution service. In the collaborative supervision scenario only one set of data is transmitted to the server, so one client might take the lead role, in terms of example collation and storage, and other clients take a mutually aware and interactive role only. Transmission of multiple sets of identical training examples to the server would only add unnecessarily to network traffic.

Consequences of uncertain lag time in Internet-based communications [14] leads to the need to adopt a more responsive peer-to-peer model for collaborative image editing, conflict resolution and rule development. Sun and Chen [13] agree that for collaborative editing of graphical images and text documents, a network of fully functional peers provides a more responsive system than relying upon individual servers to centrally complete key processing. With each peer fully capable of conducting graphical editing, learning data structure maintenance, conflict resolution, rule determination and related functionality; updates to information can be broadcast from one peer to each of the others as a minimal communication strategy. Of significant importance in [13] and [14] is algorithmic methodology for maintaining data-structure consistency in a collaborative editing environment.

Collaborating Akamai peer layers may need to communicate relevant information depending upon the level of collaboration. It is assumed that all peers share all information at each processing stage. If collaboration is done at only the human-computer interface (HCI) level where image and mark-up information is shared, then for each image, each peer will complete the conflict resolution stage independently. In addition, each peer will use the HCI layer to share data for human facilitated analysis, mark-up and conflict resolution as needed. Machine level conflict resolution can proceed in isolation and updated rules might then be shared by broadcast. Given sufficient preliminary information shared at the HCI stage, and data-structure consistency, each peer will formulate identical rule sets.

As each image is analysed and marked up, a related training example data structure is developed in tandem. How this data structure might be used is an important issue. At present the data structure holds enough information to completely mark up a “base” image and identify each contiguous collection of pixels (blob) as a particular artefact. For future reference it may be practical to store base images and related training example data structures in some repository. Once a peer machine has taken part in an analysis session and rules have been developed and stored for use, there

may be no need to keep images and data structures locally. It might be more efficient to store data structures at a central data server along with images to save memory space and optimise performance on each peer machine.

4 Human Computer Interface

From the previous discussion, it is clear that there are two aspects of the computer vision system where parallelism may be exploited. The first of these is the user-driven provision of training examples and the verification of rules based on these training examples. These correspond to steps 1 and 3 of the machine learning process, discussed in Section 3.1. The second aspect of the system where parallelism may be exploited is in the machine driven processes of visual search and learning. Whilst a user may initiate these processes, they are performed by the vision system without human interaction. Therefore, the focus of the interface for the vision system is to enable concurrency in the provision of training examples and rule verification.

The goal of the interface for the computer vision system is to enable these concurrent processes to take place. To achieve this, the interface must allow a supervisor to interact with the machine learning system, as well as enable interaction with other supervisors. To enable collaboration, supervisors must share both an awareness of each other, and an awareness of any operations performed by other collaborators. In a collaborative endeavour, conflict may occur, and therefore, the system must also provide processes for conflict resolution.

The interface addresses a bottleneck in the initial step of the machine learning process and affords concurrent processing when creating training examples. Specifically, the interface enables a user to analyse an image and, using the image mark-up tools, create both positive and negative training examples for the Akamai vision system. The addition of collaborators, communication channels, and additional image mark-up tools enables concurrent and collaborative processing of images. The result of this concurrency is a potential increase in both the rate of production of training examples, and in their quality.

The interface also addresses a bottleneck in the third stage of the machine learning process by allowing concurrent processing when human supervisors verify the system's rule base. Specifically, the interface enables either a single human supervisor, or a group of collaborating human supervisors, to supervise or evaluate a machine analysis. This supervision or evaluation can be used to verify correctness and completeness of the learning. The result of this concurrency is a potential increase in both the number of evaluations and analyses, and in the quality of the system's knowledge.

The interface consists of the following components:

- *Vision System Control Panel* provides the interface to the learning component of the system. It allows the user to specify the concept for Akamai to learn, and provides the tools to inform Akamai about the user's interpretation of the artefact under examination. It also displays what Akamai 'sees', and its predicted classification.

- *Current Image List* displays the images that the user is currently working on, grouped by the target concept for which Akamai is to be trained. This window displays information pertinent to the user, such as the key processor for this image, etc.
- *Image Mark-up Tools* provide a set of tools for the user to apply on the image that is displayed in the current image window. It also provides a history of mark-up and manipulation operations that have been performed. Some of these tools may trigger collaborative processes.
- *Current Image* displays the image that this user is processing. The user may be the key processor for this image, or they may be collaborating with another supervisor. Key processor status may influence the operations and mark-up seen by the user. The main features of this window will be the current artefact that is under examination by Akamai (cross hair), and any mark-ups that have been made by either this user or the collaborators.
- *List of Collaborators for the Current Image* displays the set of users collaborating on the current image.
- *Concept and Collaborator Pool* displays all potential collaborators.
- *Concept/Image pool* displays all images potentially available for processing.
- *Video, Audio and Text* control panels provide standard video, audio and text-based communication features. These tools will provide communication channels necessary for collaboration.

5 Discussion – Implications for Achieving Real-Time Interaction

Several factors affect the ability to achieve real-time interaction between collaborating machine-learning supervisors. Socially there needs to be a common goal, well-defined roles and recognised need for collaboration to take place. Potential collaborators need to be given a mechanism that eases the interaction process and manages the social aspects of any session to maintain progress. Importantly, there needs to be genuine support for the process from all collaborators so that there is a commitment for results to be complete and accurate.

Distributed systems that are networked via the Internet must be designed to maintain responsiveness and consistency in the event of degraded communications. In most cases this means decentralising as much of the computational load as possible so that individual progress can be made while waiting for communication. Architecture must be such that consistency of data structures, interface and system state can be assured at all times. Integration of suitable exception handling and error recovery mechanisms is essential for both distributed and stand-alone parts of the system to maintain integrity.

With potential for working across time zones and at inconvenient hours, good user interface design is of paramount importance to maintain user confidence and enthusiasm for the task. Design should invite collaborators to achieve their intended goals, facilitate operations and not intrude upon the collaboration process unduly.

Real-time interaction may require that a process change or be suspended in reaction to some circumstance. Systems need to be designed for flexibility, with a full appreciation of the roles and needs of users, and with potential contingencies for alternative operation and interaction.

Real-time collaboration requires effective interaction between users and with any innate intelligence in the system. In most cases this requires multiple channels of communication so that collaborators are aware of others' activities and the wider system state. Collaborators have to be receptive and open to suggestion, and be able to solicit assistance where required. It is expected that conflicts in assessments or analyses will most often be addressed and resolved at the human-to-human level. Delicate operations such as this require open, rich communication, and mutual awareness to maintain a cooperative atmosphere. Infrastructure governs the level and appropriate use of communication channels including text, audio, video and appropriate collaborative tools.

Systems such as the one described in this paper have greater purpose than simply proving that they can be built. Potential exists in many application areas where expertise may be pooled for the purposes of teaching both students and machine learners. In areas where expertise is lacking, there exists a potential to interact, collaborate and share awareness with experts from across the globe in real-time. Knowledge based upon the experience of many experts can be pooled and represented to help make the most informed decisions for anyone with access to the Internet. By parallelising people power both the speed and accuracy of graphical information analysis can be improved. Based on this design it is possible to rapidly build multi-expert systems that can be deployed in a range of life and mission critical environments. In addition, these systems facilitate the timely sharing of analyses, and provide expert knowledge where there would normally be none.

6 Conclusion

Advances in the field of collaborative editing present the means to relieve a sequential bottleneck in the artificial vision/machine learning process. Our design applies parallelism to the supervised creation of training examples, providing potential increases in both machine-learning speed and quality. The architecture of the Akamai artificial vision/machine learning system is described, and the inherently sequential and parallel parts of the system's algorithmic structure are discussed. Human supervision of training example creation and learning validation is shown to be a significant bottleneck of the process. This bottleneck can be addressed by application of collaborative editing techniques to increase the input data acquisition speed and accuracy.

Designs of potential models for a Collaborative Akamai are compared and discussed. Issues of system responsiveness regardless of network quality and consistency in the group editing process indicate the suitability of a peer-to-peer architecture. Importance of facilitating human-human interaction and mutual awareness as well as human-machine interactivity with careful user interface design is discussed. A range of potential interaction modes are explored based upon social and software architectures. An initial human-computer interface design is presented and the requisite functionality highlighted.

A Griffith University Research and Development grant supports the development and testing of a prototype collaborative Akamai system. The intention is to demonstrate the application of this design to other areas of artificial vision and machine learning.

References

- [1] Duda, R. and Hart, P. 1973, *Pattern classification and scene analysis*, John Wiley & Sons., New York.
- [2] Schwartz, E. 1980, 'Computational Anatomy and Functional Architecture of Striate Cortex: A Spatial Mapping Approach to Perceptual Coding', *Vision Research*, vol. 20, no., pp.
- [3] Marr, D. 1982, *Vision*, Freeman, San Francisco.
- [4] Norman, D. A. 1986, 'Cognitive engineering' in *User centered system design*, (Eds, Norman, D. A. and Draper, S. W.) Lawrence Erlbaum Associates.
- [5] Norman, D. A. 1988, *The psychology of everyday things*, Basic Books, New York.
- [6] Gonzalez, R. C. and Woods, R. E. 1992, *Digital Imaging Processing*, Addison-Wesley, New York.
- [7] Bishop, C. M. 1996, *Neural networks for pattern recognition*, Oxford University Press.
- [8] Tovee, M. J. 1996, *An Introduction to the Visual System*, Cambridge University Press, Great Britain.
- [9] Tal, D. and Schwartz, E. L. 1997, 'Topological singularities in cortical orientation maps: the sign theorem correctly predicts orientation column patterns in parimate striate cortex', *Network: Computational Neural Systems*, vol. 8, no., pp. 229–238.
- [10] Sheridan, P., Hintz, T. and Alexander, D. 1998, 'Space-variant sensing for robotic vision', *In 5th International Conference on Mechatronics and Machine Vision in Practice*, Nanning, pp. 185–190.
- [11] Peterson, I. 1999, 'The Honeycomb Conjecture', *Science News*, vol. 156, no. 4, pp. 60–61.
- [12] Sheridan, P., Hintz, T. and Alexander, D. 2000, 'Pseudo invariant transformations on a hexagonal lattice', *Image and Vision Computing*, vol. 18, no. 11, pp. 907–917.
- [13] Sun, C. and Chen, D. 2002, 'Consistency Maintenance in Real-Time Collaborative Graphics Editing Systems', *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 1, pp. 1–41.
- [14] Sun, C. 2002, 'Undo as concurrent inverse in group editors', *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 4, pp. 1–51.
- [15] Sheridan, P. 2002, 'Supervised learning of complex visual tasks: Implementation of a principle that connects visual search with visual learning', *International Conference on Parallel Distributed processing Techniques and Applications*, Las Vegas, pp. 7–13.
- [16] Sheridan, P., Drew, S., Venema, S. and Sun, C. 2003, 'Supervised Learning Issues for Machine Vision: Parallelising People Power', *International Conference Imaging Science, Systems and Technology*, Las Vegas, pp. In Press.

Parallel Storing and Querying XML Documents Using Relational DBMS*

Jie Qin, Shuqiang Yang, and Wenhua Dou

School of Computer,
National University of Defense Technology,
410073 Changsha, China
qinjie0160@163.com

Abstract. The widespread adoption of XML is creating a new set of data management requirements, such as the need to store and query XML documents. Traditional methods of storing and querying XML documents are in single processor environment. This paper proposes a parallel processing technology based on multi-processor. The key issues such as XML data model, storage model, data placement strategy, paralleling query, indexing etc. are also discussed.

1 Introduction

The eXtensible Markup Language (XML) is an emerging standard for data representation and exchange over the Internet. In recent years, increasing attention has been focused on the problem of storing and querying XML documents [1,2,3]. Most of the current studies on XML documents storing and querying are based on single processor environment. In some occasions, such as XML-based bioinformatics research, digital library, parallel processing technology can get better query performance. In order to study the efficiency of parallel querying XML data, we are building a parallel querying XML documents system (PQXDS) over a relational DBMS based on multi-processor. This paper proposes our research result on the key issues of parallel querying XML documents using relational DBMS, such as XML data model, XML data storage strategy, parallel querying XML, and indexing XML data. The main contributions of this paper are the following: putting forward a new model-mapping-based storage model, Xpath-edge, which is tailored to satisfy the need of parallel querying XML data, and proposing an XML data placement strategy based on this model which can random evenly place a large number of XML documents to parallel processors, which is fit for parallel querying.

The rest of this paper is organized as follows. Section 2 briefly discusses XML data model. Section 3 introduces Xpath-edge data storage model and data placement strategy. Section 4 discusses parallel querying XML, and Section 5 concludes this paper.

* This research is supported by the National Natural Science Foundation of China under grant No. 60003001.

2 Data Model

W3C proposed four XML data model: the Infoset model, the XPath data model, the DOM model ,and the XQuery 1.0 and XPath 2.0 data model [1]. Among the four models, only the XQuery 1.0 and XPath 2.0 data model [4,5] acknowledges that the data universe includes more than a single document, and it is also the only model that includes inter-document and intra-document links in a distinct node type (i.e., reference). It models XML documents as an ordered graph using 7 types of nodes: root, element, text, attribute, namespace, processing instruction and comment.

We adopt the XQuery 1.0 and XPath 2.0 data model to represent XML documents. Fig.1. is a simple XML document example. Fig.2. is its data graph illustrating the data model. In brief, here we emphasize on four types: root, element, text and attribute. All the round nodes are element nodes. Each element node has an element-type name, which is represented as an edge label of the incoming edge pointing to the element node itself. The text nodes are marked with underline. The IDREF attribute node (octagon) is used for intra-document reference.

```
<Proceedings>
  <Proceeding>
    <conference> SIGMOD </conference>
    <Year> 2002 </Year>
    <Article Id=A1 >
      <Title> Storing and querying XML </Title>
      <Author> Igor </Author>
      <Author> Stratis </Author>
      <Reference>Id =A2 </Reference>
    </Article>
  </Proceeding>
  <Proceeding>
    <conference> SIGMOD </conference>
    <Year> 2001 </Year>
    <Article Id=A2 >
      <Title> Updating XML </Title>
      <Author> Igor </Author>
    </Article>
  </Proceeding>
</Proceedings>
```

Fig. 1. A small XML document

3 Data Storage Model and Storage Strategy

Effective storage for XML documents must consider both the dynamic nature of XML data and the stringent requirements of XQuery [5].

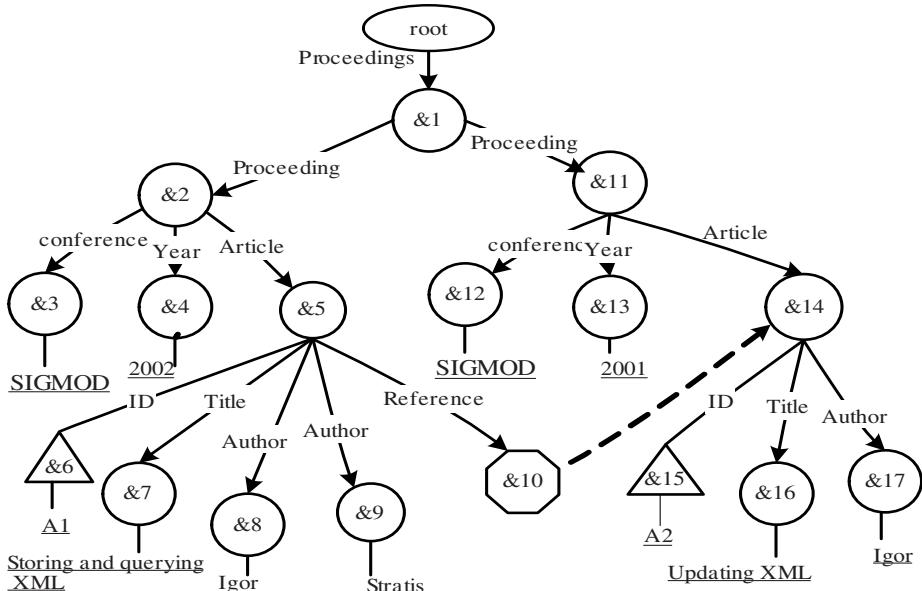


Fig. 2. A data graph for a small XML document

3.1 Storage model

There exist numerous different approaches to store XML document, such as, file systems, object-oriented database systems, native XML database systems, relational (object-relational) DBMS. For easy management and realization, we choose relational database to store XML data. When XML documents are stored in off-the-shelf DBMS, the problem of storage model design for storing XML data becomes a database schema design problem. In [2], the authors categorize such database schemas into two categories: structure-mapping approach and model-mapping approach. In the former, the design of database schema is based on the understanding of DTDs(document type declarations) that describes the structure of XML documents. In the latter, a fixed database schema is used to store any XML documents without assistance of DTDs. In many applications, the schema of XML documents is unknown. In order to facilitate the task of parallel querying XML documents, we present a new model-mapping-based approach, namely, Xpath-edge. The unique feature of Xpath-edge is: it is a two tables schema, Path table, and Edge table. The Path table keeps all label-paths of XML documents. It allows navigation through the data with regular path expressions [4], which can speed up querying. The Edge table keeps parent-child relationships, and describes all edges of XML data graphs. This model can be efficiently supported using the conventional index mechanisms.

Table 1 is the Path table of Figure 2. Table 2 shows the Edge table of Figure 2. Each edge is specified by two node identifiers, Source and Target. The Path_id indicates which path the edge belongs to. The Doc_id indicates which document the edge belongs to. The Label attribute keeps the edge-label of an edge. The Ordinal attribute records the ordinal of the edge among its siblings. For example, the triple (8,

"X1", &5, &9, "Author", 2, "Stratis"), describes the edge from the element node (&5) to the element node (&9). The edge has an edge-label "Author", and a value "Stratis". Its ordinal is 2 (the second author of the article "Storing and querying XML"), and its path is //Proceedings/Proceeding/Article/Author, which belongs to XML document "X1".

Table 1. The Path table of Fig. 2

Path_id	Pathexpress
1	//Proceedings
2	//Proceedings/Proceeding
3	//Proceedings/Proceeding/conference
4	//Proceedings/Proceeding/Year
5	//Proceedings/Proceeding/Article
6	//Proceedings/Proceeding/Article/ID
7	//Proceedings/Proceeding/Article/Title
8	//Proceedings/Proceeding/Article/Author
9	//Proceedings/Proceeding/Article/Reference

Table 2. The Edge table of Fig. 2

Path_id	Doc_id	Source	Target	Lable	Orderal	Value
2	X1	&1	&2	Proceeding	1	
3	X1	&2	&3	Conference	1	SIGMOD
4	X1	&2	&4	Year	1	2002
5	X1	&2	&5	Article	1	
6	X1	&5	&6	Id	-	A1
7	X1	&5	&7	Title	1	Storing and...
8	X1	&5	&8	Author	1	Igor
8	X1	&5	&9	Author	2	Stratis
9	X1	&5	&10	Reference	-	A2
2	X1	&1	&11	Proceeding	2	
3	X1	&11	&12	Conference	1	SIGMOD
4	X1	&11	&13	Year	1	2001
5	X1	&11	&14	Article	1	
6	X1	&14	&15	Id	-	A2
7	X1	&14	&16	Title	1	Updating XML
8	X1	&14	&17	Author	1	Igor

3.2 XML Data Placement Strategy

The objective of XML Data placement strategy is trying to find a nearly optimal data distribution to maximize the system throughput and resource utilization, and reduce I/O operations. Basing on Xpath-edge storage model, a new XML data placement strategy is put forward:(1) First evenly place those independent small documents to all parallel processors.(2)Then place inter-connected small documents to the same processor, and ensure that all the different inter-connected XML documents are evenly distributed to all parallel processors.(3)If some documents are too big, use graph partition algorithm [6] to divide these documents to some smaller parts, then place each part to different processors.

4 Parallel Querying Methods

Traversing tree for answering query is the inevitable method to be used. There are three basic methods to traverse tree for answering query, Top-down, Bottom-up, and Hybrid. All these methods can be used in parallel querying XML data [3]. RDBMS provided much strong index mechanism. A number of indexes have been proposed to speed up XML querying [1,2]. [7] gives three parallel indexing schemes in parallel RDBMS: (1) Non-Replicated Index (NRI),(2) Partially-Replicated Index (PRI), and(3) Fully-Replicated Index (FRI). The decision to choose another index for other attributes is determined by the balance between storage versus performance.

5 Conclusions

In this paper, a multi-XML documents storage model, Xpath-edge is proposed, which is designed for parallel storing and querying XML documents. This storage model does not require any special indexing structures, and can utilize conventional indexing structures. Basing on this storage model, the proposed XML Data placement strategy can evenly distribute XML documents to all parallel processors. For future work, we will study parallel querying optimization of XML data.

References

1. A. Salminen, and F. Wm. Requirements for XML Document Database Systems. ACM Symposium on Document Engineering, p85–94, November 2001
2. M. YoshiKawa, T. Amagasa, and T. Shimura. XRel: A path-based approach to storage and retrieval of XML documents using relational databases. ACM Transactions on Internet Technology, 2001.1(1)
3. K. Lü, Y. Zhu, W. Sun, S. Lin, and J. Fan. Parallel Processing XML Documents. IDEAS 2002
4. A. Berglund, et al. XML Path Language (XPath) 2.0. W3C Working Draft. 16 August 2002. <http://www.w3.org/TR/xpath20>
5. W3C. XQuery 1.0: An XML Query Language. W3C working draft 16 Aug. 2002. <http://www.w3.org/TR/xquery>
6. G. Karypis, and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular graphs. SIAM Journal on Scientific Computing, Vol. 20 No.1, p359–392. 1998.
7. J. W. Rahayu, D. Taniar. Parallel Selection Query Processing Involving Index in Parallel Database Systems. ISPLAN 2002
8. H. Jiang, H. Lu, W. Wang, and J.X. Yu. Path Materialization Revisited: An Efficient Storage Model for XML Data. Australasian Database Conference 2002

A Variable Consistent Server Replication Model for Mobile Database*

Jinhui Xu and Ming Xu

National Laboratory for Parallel & Distributed Processing,
National University of Defense Technology,
Changsha, P. R. China, 410073
xusamuel@hotmail.com
xuming64@cs.hn.cn

Abstract. This paper presents a variable consistent replication model, called Variable Consistent Server Replication (VCSR), for mobile database. Based on the three-tier data replication architecture, VCSR uses the decentralized transaction processing approach to support both weakly consistent transaction and strict consistent transaction. It provides Write-Any-Read-Any replication mechanism for database, and keeps the replica consistency and transaction serializability by periodical pair-wise synchronization. We compare VCSR model with related models and systems. Reasonable simulations and promising results are illustrated.

1 Introduction

Database replication is quickly becoming a critical technology for providing high availability, survivability and high performance for database applications, especially for mobile applications. However, providing useful data replication means one has to solve the non-trivial problem of maintaining data consistency between all the replicas.

For the restrictions imposed by the nature of mobile environments (disconnection, low bandwidth, high variability, etc.)[1] the servers can't maintain stable link between them. Hence strict consistent replication protocol wouldn't work well. A number of researches are currently being carried on in providing high available replication with weakly consistent replication protocol, which allows presenting the users stale data in a controlled way. This kind of replications also allow the users to read or update the data while they are disconnected, merging the modifications with other nodes when they are re-connected[2]. The applications with similar demands include mobile file systems (e.g. Coda[3] and Roam[4]) and mobile e-mail systems (e.g. Bayou[5] and Lotus Notes[6]).

However, even in the weakly consistent application, there are the needs of strict consistency supported by the database system. It means that some read transaction can get the most recent committed value in the weakly consistent replicated system, or a update transaction request commit as soon as it arrives the server, while its results are made permanent.

* The research is supported by National Natural Science Foundation of China, No.60073002.

For example, a salesman always uses laptop computers or PDA to monitor the data of order forms and clients. When he contacts some client for sale promotions, he can link to the headquarter for the related information. In most cases, for the salesman, stale data is enough, thereafter weakly consistent system can acquire higher efficiency. But he must have the ability to deal with an important form and give the client the formal response as soon as possible. Thus the system must have the ability to commit the transaction and return the result immediately.

In this paper, we propose a variable consistent server replicated model for mobile database, so called VCSR. The goals of VCSR include: providing read-any-write-any data replication, maintaining consistency of whole system, supporting variable consistency transaction processing, and balancing the network overloads.

The remainder of this paper is organized as follows. Section 2 presents an overview of the model. Section 3 gives the description of the transaction processing under VCSR in detail. Section 4 discusses the performance evaluation of our experimental system. In section 5, we compare our work with other related researches, and section 6 summarizes the paper.

2 Overview

We assume VCSR model follows the general replication principles. All replica servers construct a Replica Set, called RS. Each server S in RS is assigned a global unique identifier, S.id. Every replica server can communicate with other servers in RS. Also, the network communication is connection-oriented, that means the messages run on the network would keep exactly in the original order. For the instability of mobile environments, some servers may disconnect from the network temporarily. At the meantime, we assume the network failure that probably occurs can be checked out and revived.

The transaction is the basic unit of user's access to the replica server in our VCSR model. Every transaction composes a series of write/read operations. Query transaction contains only read operations while update transaction contains both. Each server logs all update transactions so that the system can roll back or redo the failed transactions correctly.

In mobile environments, connections between sites are unstable and frequently disconnected. To fit to such a situation, disconnected operation, which we call weak operation, is essential to mobile clients. On the other hand, the standard operation with usual semantics is called strict operation.

Definition 1. Weak operation includes weak read and weak write, which means it accesses replica that resided in the local site. The transaction that includes only weak operations is called weak transaction.

Definition 2. Strict transaction includes only strict operations, which maintain the ACID property during the transaction processing phase.

VCSR model adopts Read-Any-Write-Any replica mechanism, it means each replica server can support both query transaction and update transaction. And VCSR allows the inconsistency existing between servers temporarily. Hence the user only needs to access one replica server when he operates the database system. Obviously, the commitment of transaction consists of two phases, local commitment and global

commitment. The transactions are locally committed at where they are executed. All operations of transaction are recorded in the local temporary transaction logs.

For a weak transaction, its execution results are returned to the user just after the local commitment. So the user needn't wait until the updates are propagated to all the replica servers, and the response time decrease obviously. At the same time, the updates are visible only to transactions in the same site, and other transactions are not aware of these updates. They might become permanent only after global commitment that guarantees ACID property of transaction.

3 How to Maintain VCSR Consistency?

As stated above, the transaction is the basic unit of user's access to the replica server in the VCSR model. Hence, how to maintain the global consistency is equal to how to maintain the consistent parallel transactions at different sites.

3.1 Transaction Processing

For the query operation, transaction wouldn't affect the consistency of the replicated database, thus we ignore the existence of query transaction. Without special indication, the term 'transaction' refers to the update transaction thereafter.

VCSR model allows any replica server accept and execute update transaction and then propagates the updates to the other servers. This approach can improve data usability of mobile clients and therefore improve system efficiency and reduce average transaction response time. But it also introduces potential data conflict between updates at different sites inevitably. Some conflict transactions may be canceled. Hence the execution results of update transaction are temporary, and the transaction must be globally committed so that the results become permanent at the pending time. It means that differing from other strict consistency database systems, VCSR have two copies of every record, the committed version and the temporary version.

Obviously, there must be a reconciliation process to detect data conflicts and to restore data consistency. Approaches of reconciliation can be classified into syntactic ones and semantic ones. We adopt a pure syntactic approach and it thus is application-independent. Consistency is achieved by maintaining the serializability of transactions.

Every replica server maintains a local logical timer (The algorithm in [7] is used to make all servers synchronous). When a server takes over a weak transaction it assigns a timestamp and updates the local logical timer. Then the weak transaction, called temporary transaction, is recorded in the local Temporary Transaction Log (TTL). And temporary version of the local database replica is updated with the results of transaction.

Supposing the server that firstly accepts the weak transaction T has the identifier $T.sid$, we assign this transaction a global unique identifier, $T.id = \langle T.ts, T.sid \rangle$. $T.ts$ is the transaction's timestamp. Then every replica server can re-order all the transactions in TTL, including the transactions accepted from users and any other servers. The order relation of any two temporary transactions, T_1 and T_2 , is defined as:

$$T_1.id < T_2.id \stackrel{\text{def}}{=} \begin{cases} & T_1.ts < T_2.ts, \text{ or} \\ & T_1.ts < T_2.ts, \text{ and } T_1.sid < T_2.sid \end{cases}$$

For strict transaction, the most common approach is to select a primary replica server. However Yasushi pointed out: such kinds of system, i.e. single-master system, increase the update propagation delay and sometimes create a load imbalance among replica servers [8]. To solve this problem, we take a decentralized mechanism to handle the strict transaction problem. We treat the strict transaction commitments as a series of elections. In our model, a server is analogous to a voter, creating an update is analogous to a voter deciding to run for office, and a committed update is analogous to a candidate winning the election. Candidates win elections by cornering a plurality of the votes. Each election begins with an underlying agreement of the winners of all previous elections. Once an election is over, a new election commences. Following sections will describe propagation of the voting information and transaction global-commitment in detail.

The processing of strict transaction is the same as weak transaction. And the strict transactions are assigned another field, eid, which means the election number. When the transactions are inserted to TTL, they will always be in the ascending order by eid. In order to implement the processing mechanism, each voter maintains the following three states:

1. $S_i.\text{completed}$ — the number of elections completed locally;
2. $S_i^k.[j]$ — either the index of the candidate voted for by S_j in S_i 's in election k, or which means that S_i has not yet seen a vote from S_j . The election is understood to be S_i 's current election if the superscript k is omitted. The size of the array is bounded by the total number of voters;
3. $S_i.\text{curr}[j]$ — The amount of currency voted by S_j in S_i 's current election. Currency allocation may change with each election.

The total amount of the currency in any election is 1.

3.2 Transaction Propagation

VCSR model adopts the periodical pair-wise synchronization process to synchronize replica servers. That means each replica server chooses another replica server randomly to exchange temporary transactions unknown to each other, hence synchronization of the two servers is completed.

During the synchronization process, a server always sends its TTL to another one in the order of transaction id ($\langle T.ts, T.sid \rangle$). The accepting server combines the TTL accepted from another server with the local TTL, and all transactions in new TTL are arranged in the ascending order by transaction id. But all the weak transactions conflict with any strict transaction must be aborted so as to ensure the later ones' priority. This approach can guarantee that all other transactions started from the same server have reached S already when a transaction T reaches server S. Due to the length of the paper, we omit its proof.

According to this feature, every server maintains a vector NTSV(Node Time Stamp Vector) to record the latest temporary transaction's timestamp starting from server S_i . The vector's structure is as follows:

Table 1. The structure of the vector NTSV

Replication sever	Timestamp
S ₁	ts ₁
S ₂	ts ₂
...	...

During the synchronization process, the two servers compare their NTSV and then exchange the part of TTL unknown to each other.

There are more works to be done for supporting the strict transaction. We give three definitions as follows:

$$\text{Definition 3: } \text{uncommitted}(S_i) = \sum_{j=1}^n S_i.\text{curr}[j] \quad \text{s.t. } S_i[j] = \perp$$

$$\text{Definition 4: } \text{votes}(S_i, k) = \sum_{j=1}^n S_i.\text{curr}[j] \quad \text{s.t. } S_i[j] = k$$

Definition 5: A candidate S_j wins S_i's current election when:

- 1) $\text{votes}(S_i, k) > 0.5$, or
- 2) $\forall k \neq j :$

$$\begin{aligned} &\text{votes}(S_i, k) + \text{uncommitted}(S_i) < \text{votes}(S_i, j), \text{ or} \\ &\text{votes}(S_i, k) + \text{uncommitted}(S_i) = \text{votes}(S_i, j) \text{ and } j < k \end{aligned}$$

Definition 3 essentially means a candidate wins with a voter if it has a majority or plurality of the vote. Ties are broken with a simple deterministic comparison between the indexes of the servers that created the competing updates. The winner of the jth vote at S_i is denoted S_i.commit(j). When an election is won at S_i, all votes S_i[j] are reset to \perp .

Election information flows from voter to voter with the synchronization process. The information is transferred from S_i to S_j according to the following three rules:

Rule 1: if S_j.completetd > S_j.completed, then

$$S_j.\text{completed} \leftarrow S_i.\text{completed}, \text{ and}$$

$$\forall k, S_j[k] \leftarrow \perp, \text{ and}$$

$$S_i.\text{commit}$$

$$\forall k = S_j.\text{commit} + 1 \quad S_j.\text{commit}(k) \leftarrow S_i.\text{commit}(k)$$

Rule 2: if S_j.completed = S_i.completed, then

$$\forall k, \text{s.t. } S_j[k] = \perp, S_j[k] \leftarrow S_i[k]$$

Rule 3: if S_j[j] = \perp , $S_j[j] \leftarrow S_i[j]$

Based on the above definitions, after all elections have finished by all voters, distinct voters arrive at the same election results.

The propagation mechanism in our model combines the pull transfer and the push transfer. One of its advantage is that tend to commit updates more quickly. Let p_i be the probability that a server has not seen a new update after the ith interval after the creation. Then the probability that a server has not seen the update after the i+1th intereration is just

$$p_{i+1} = p_i \cdot \left[p_i \left(1 - \frac{1}{n} \right)^{n(1-p_i)} \right] = p_i^2 \left(1 - \frac{1}{n} \right)^{n(1-p_i)}$$

which converges rapidly. The corresponding recurrences for pull and push are respectively

$$p_{i+1} = p_i^2$$

and

$$p_{i+1} = p_i \left(1 - \frac{1}{n}\right)^{n(1-p_i)}$$

These two recurrence converges more slowly than the first.

3.3 Transaction Commitment

For the method of transaction propagation, the temporary transaction must be propagated from the server, at which it started, to all other servers. Also, different servers get the same election result. At this point, all transactions, whose id less than this transaction's id, have reached local server and wouldn't be roll back or aborted. Hence, to some server, if it can make assure that all other servers have acquired some weak transaction T or the result of one election, it can make the updates of the weak transaction T or corresponding (according) strict transaction permanent. This is the main idea of our transaction commitment algorithm, called natural converge commitment algorithm.

To judge whether a transaction T has been known by all replica servers or not, a vector table, NTSV_Table, is introduced into every server. Every member of the table has the form such as $\langle S_k, S_k.NTSV \rangle$, which has the most recent value of $S_k.NTSV$ known by the table's owner.

During the synchronization process, the two servers must exchange their NTSV_Table and generate a new table (NTSV_Table). The later presents the most recent status of all replica servers known by these two servers. For every member of NTSV_Table, we can define the magnitude relation of two vector tables NT_1 and NT_2 as follows:

$$NT_1[S] \leq NT_2[S] \stackrel{\text{def}}{=} \forall k \in RS, NT_1[S][k] \leq NT_2[S][k]$$

(RS is the replica server set)

With above definition, a server can judge whether a transaction T, which started from server T.sid, has been known by all replica servers with the following formula:

$$IsAllKnown(T) = \begin{cases} TRUE, & \forall S_k, NTSV_Table[S_k][T.sid] \geq T.ts; \\ FALSE, & \text{otherwise} \end{cases}$$

The server executes this algorithm at the end of the synchronization cycle and commits the transactions that satisfying the condition.

4 Performance Evaluation

To validate the correctness of VCSR model, we construct a prototype system in a local area network. The results of experiment indicate that, after all servers that execute the prototype stop generating new transaction, all replicas become consistent finally. The result shows VCSR model can reach strict convergence.

Table 2. Experimental parameters

Param	Description	Values
PERIOD	Time of every synchronization cycle	100s
MEAN	Mean numbers of update transactions incepted from users by a server in one synchronization cycle	10
NODE	The number of replica server	4,8,...,32

Also, we construct a simulation program to evaluate the performance of VCSR. Table 1 presents some real values for the input parameters used in the experiments.

The first simulation experiment tests the commitment time of our replicated database system. WCSR (Weakly Consistent Server Replication) is a pure weak consistent part of VCSR. It just supports weak transaction. Figure 1 shows that our system has good access performance.

We select the primary server algorithm as the counterpart of VCSR's decentralized algorithm to support strict transaction in the second experiments. Figure 2 shows that two algorithms have similar response time. But decentralized algorithm has advantages in balancing the network overload and scalability.

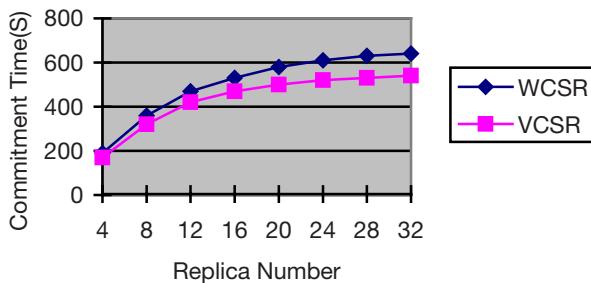


Fig. 1. Commitment Time. In WCSR, the frequent actions of redo, which ensure some transactions have the same result to strict transaction in VCSR, weaken the system's performance.

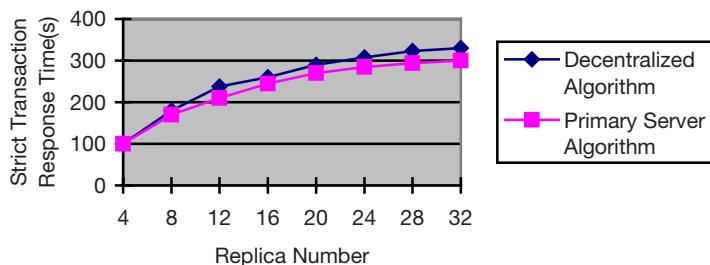


Fig.2. Strict Transaction Response Time. The performance of decentralized algorithm is closed to the primary server algorithm.

5 Related Works

Replicas are useful for many reasons, such as efficiency, availability, and fault tolerance etc. But traditional replication technologies aren't so easy to be adapted to weak connection systems, especially in mobile environments[9].

Aiming at the features of mobile computing, J. Gray and J. J. Kistler proposed the concept of two-tier replication[10,11]. Li designed a three-tier replication architecture, combining data broadcast, replication and cache in one architecture[9]. Our research is based on Li's three-tier architecture.

Recent work has explored lazy strategies that still provide consistency, but as a major drawback, these approaches restrict the placement of primary and secondary copies in the system, which is likely to be a significant liability in real applications[12,13,14].

Coda and Ficus share many of the goals of our work in the more limited domain of distributed file systems[3,18]. This choice in domain allows the use of strong assumptions on the relative scarcity of contention. Additionally, reconciliation automated for many types of files. Hence, these systems both use replication that is optimistic in the sense of allowing conflicting transactions to commit.

6 Conclusion

In this paper, we introduce our research on consistency problem in mobile database. With the periodical pair-wise synchronization process, variable consistent server replication model can synchronize all replica servers and maintain their consistency. The transaction processing mechanism of our model guarantees the transaction serializability. And the transaction abort rate is notably reduced. Results of the reasonable simulations show the improvement of efficiency for mobile database system. This approach is well-suited to weakly-connected environments specifically because it is highly decentralized.

In mobile computing environments, dramatic variations of wireless bandwidth and system configuration determine that adaptation is a key to mobile applications. The adaptation to environments is best achieved by collaboration between applications and systems[15]. Thus, how to express service request from users and select the best transaction processing manner is the next step of our research.

References

1. Forman. G. H, Zahorjan J.: The Challenges of Mobile Computing, IEEE Computer, Apr. 1994, 17(4): 38–47.
2. Petersen K., Spreitzer M.J., Terry D.B., Marvin M. T., Alan J.D.: Flexible Update Propagation for Weakly Consistent Replication, Proc. of 16th Symp. on Operating Systems Principles(SOSP), St. Malo, France, Oct 1997:288–301.
3. Lee Y. W., Leung K. S., Satyanarayanan M.: Operation-based Update Propagation in a Mobile File System. Proceedings of the USENIX Annual Technical Conf., Monterey, CA, Jun. 1999

4. Ratner D. H.: Roam, A Scalable Replication System for Mobile and Distributed Computing, PhD Dissertation, UCLA, 1998.
5. Terry D. B., Petersen K., Spreitzer M. J., Theimer M. M.: The Case for Non-transparent Replication: Examples from Bayou. IEEE Data Engineering, Dec. 1998:12–20.
6. LS 5.0 Technology Whitepaper, IBM corp. Aug. 2001
7. Lamport L.: Times, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, 1978, 21(7):558–565.
8. http://www.hpl.hp.com/personal/Yasushi_Saito/
9. Ling L.: Data Broadcast and Replication/Cache Technology Research in Mobile Database, PhD Dissertation, NUDT, 1998.
10. Gray J., Helland P., O’Neil P., and Shasha D.: The dangers of replication and a solution, Proc. of ACM SIGMOD Int’l Conf. on Management of Data, Montreal, Canada: ACM Press, 1996:173–182.
11. Kistler J., Satyanarayanan M.: Disconnected operation in the Coda system. ACM TOCS, Feb. 1992, 10(1):3–25.
12. P. Liu, Ammann P., Jajodia S.: Incorporating transaction semantics to reduce reprocessing overhead in replicated mobile data applications, Proc. of 19th Int’l Conf. on Distributed Computing Systems. Austin, Texas: IEEE Computer Society Press, 1999:414–423.
13. Breithart Y., Komondoor R., Rastog R., seshadri S., Silberschatz A.: Update propagation protocols for replicated databases. Proc of ACM SIGMOD Int’l Conf. on Management of Data, Philadelphian, Pennsylvania: ACM Press, 1999: 97–108.
14. Pacitti E., Minet P., Simon E.: Fast algorithms for maintaining replica consistency in lazy master replicated databases, Proc. Of 25th Int’l Conf. on Very Large Database. Edinburgh, Scotland: Morgan Kaufmann Publisher, 1999:126–137
15. Noble B.: System support for mobile, adaptive applications. IEEE Personal Computing Systems, 2000, 7(1):44–49.

Multi-scheduler Concurrency Control for Parallel Database Systems

Sushant Goel¹, Hema Sharda¹, and David Taniar²

¹ School of Electrical and Computer systems Engineering,
Royal Melbourne Institute of Technology, Australia
s2013070@student.rmit.edu.au

hema.sharda@rmit.edu.au

² School of Business Systems,
Monash University, Australia
David.Taniar@infotech.monash.edu.au

Abstract. Increase in amount of data stored and requirement of fast response time has motivated the research in *Parallel Database Systems* (PDS). Requirement for correctness of data still remains one of the major issues. Concurrency control algorithms used by PDS uses *single scheduler* approach. Single scheduler approach has some inherent weaknesses such as – very big lock tables, overloaded centralized scheduler and more number of messages in the system. In this paper we investigate the possibility of *multiple schedulers* and conclude that single scheduler algorithms cannot be migrated in the present form to multi-scheduler environment. Next, we propose a *Multi-Scheduler Concurrency Control* algorithm for PDS that distributes the scheduling responsibilities to the respective *Processing Elements*. Correctness of the proposed algorithm is then discussed using a different serializability criterion – *Parallel Database Quasi-Serializability*.

1 Introduction

Research and development efforts have made Parallel Database Systems (PDS) a reality during the last decade. PDS are specifically used to meet requirements of Very Large Databases (VLDB) – volume of data spanning to terabyte (10^{12}) size. Any database system must guarantee the consistency of the database.

Researchers in the recent past have concentrated on query optimization [12,13] in parallel database environment. Very little work has been done in the area of transaction processing and concurrency control to meet the specific needs of PDS. Single scheduler approach, and thus serializability [9], has been used as the correctness criterion for concurrency control. High performance and multiprocessor database systems such as *Parallel Database Systems* (PDS) [2,15], Distributed Database Systems (DDS) [15,16], Multidatabase Systems (MDS) [14] have made the notion of *conflict serializability* [9] insufficient as a correctness criterion.

This paper proposes a new serializability criterion, *Parallel Database Quasi-serializability (PDQ-serializability)*, and then formulates a Multi-scheduler Concurrency Control algorithm to meet the requirements of PDS. We focus on the

problem of concurrent transaction execution in parallel database systems. Serializability as a correctness criterion assumes the existence of single scheduler to ensure the database consistency. Single scheduler strategy may not meet the requirements of PDS. Also, Single scheduler strategy may not be directly migrated to multi-scheduler environment, the underlying problem is discussed in detail in the next section. The single scheduler scheme has some inherent weaknesses, such as: 1) Due to centralized scheduling, the lock table grows very big at the scheduler. 2) Abundant computing power of multiple processors is not utilized to the fullest. 3) All sites have to communicate to a central scheduling node thus increasing the number of messages in the system. 4) Sometimes it is difficult to decide the coordinator-node.

The proposed multi-scheduler concurrency control approach minimizes the above-mentioned weaknesses for obvious reasons; we would not discuss them in detail as they are out of the scope of the paper. The purpose of this research is to explore the issues in concurrency control of parallel database systems that uses multiple schedulers. The paper covers two major aspects: 1) We show that the concurrency control algorithms implemented in single scheduler environment may produce incorrect results in multi-scheduler environment. We use multiple schedulers to distribute the load, contrary to single scheduler environment. 2) Next, we propose a Multi-scheduler Concurrency Control algorithm that distributes the scheduling responsibilities to the respective *Processing Elements (PE)* and guarantees a PDQ-serializable schedule. We must emphasize that this concept is similar yet different than multidatabase serializability (MDBS) or heterogeneous distributed database serializability (HDDBS). MDBS and HDDBS consider two levels of transactions – local and global, and treat them differently, but the proposed algorithm treats all the transactions equally, for detailed description and comparison refer [2, 14, 15, 16].

Rest of the paper is organized as follows: Section 2 gives the fundamental definitions of database terms and PDS, Section 3 discusses the motivation of this work, Section 4 presents the formal model of PDS that we consider for the algorithm. Section 5 elaborates the proposed algorithm and proves the correctness of the algorithm. Finally, Section 6 concludes the paper and discusses the future extension of the work.

2 Background

In this section we first give the fundamental definitions of database terminologies that we use through out the paper. Next we briefly discuss most common architecture of the PDS.

2.1 Fundamental Definitions

We would like to briefly define the *transaction* and properties of transactions before we proceed with the proposed algorithm. From the user's viewpoint, a transaction is the execution of operations that accesses shared data in the database, formally [9]:

Definition 1: A transaction T_i is a set of read (r_i), write (w_i), abort (a_i) and commit (c_i). T_i is a partial order with ordering relation \prec_i where:

- 1) $T_i \subseteq \{r_i[x], w_i[x] \mid x \text{ is a data item}\} \cup \{a_i, c_i\}$
- 2) $a_i \in T_i$ iff $c_i \notin T_i$
- 3) If t is a_i or c_i , for any other operation $p \in T_i$, $p \prec_i t$
- 4) If $r_i[x], w_i[x] \in T_i$, then either $r_i[x] \prec_i w_i[x]$ or $w_i[x] \prec_i r_i[x]$. \square

Definition 2: A complete history (or schedule) H over T (let, T be set of transactions, $T = \{T_1, T_2, \dots, T_n\}$) is a partial order with ordering relation \prec_H where [9]:

- 1) $H = \bigcup_{i=1}^n T_i$;
- 2) $\prec_H \supseteq \bigcup_{i=1}^n \prec_i$; and
- 3) For any two conflicting operations $p, q \in H$, either $p \prec_H q$ or $q \prec_H p$. \square

Definition 3: A history H is *Serializable (SR)* if its committed projection, $C(H)$, is equivalent to a serial execution H_s [9]. \square

Definition 4: A database history H_s is serial iff [9]

$$(\exists p \in T_i, \exists q \in T_j \text{ such that } p \prec_{H_s} q) \text{ then } (\forall r \in T_i, \forall s \in T_j, r \prec_{H_s} s).$$

\square

A history (H) is serializable iff serialization graph is acyclic [3, 9].

2.2 Parallel Database Systems

The enormous amount and complexity of data and knowledge to be processed by the systems imposes the need for increased performance from the database system. The problems associated with the large volumes of data are mainly due to: 1) Sequential data processing and 2) The inevitable input/output bottleneck.

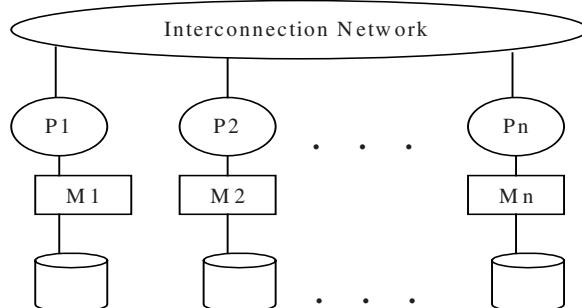


Fig. 1. Shared Nothing PDS

Parallel database systems have emerged in order to avoid these bottlenecks. Different strategies are used for data and memory management in multiprocessor environment to meet specific requirements. In *shared memory architecture* processors have direct access to the disks and have a global memory. This architecture is good for load balancing. In *shared disk architecture* processors have direct access to all disks but have private memory. *Shared-nothing architecture* has individual memory and disk for each processor, called *processing element (PE)*. The main advantage of shared-nothing multi-processors is that they can be scaled up to hundreds and probably thousands of processors, for detailed discussion and comparison see [1, 2, 5, 7, 8]. We assume shared nothing architecture for this study.

3 Motivating Example

The following example shows that single scheduler algorithms may not be directly migrated to meet the requirements of multi-scheduler environment. We consider two PE and four data objects (O_1, O_2, O_3, O_4) to demonstrate the motivation of our work. The two PE's are denoted as P_1 and P_2 . Say, the data objects are located as follows:

$$\begin{aligned} P_1 &= O_1, O_2 \\ P_2 &= O_3, O_4 \end{aligned}$$

Now, consider two transactions are submitted to the database as shown below.

$$\begin{aligned} T_1 &= r_1(O_1) r_1(O_2) w_1(O_3) w_1(O_4) C_1 \\ T2 &= r_2(O_1) r_2(O_3) w_2(O_4) w_2(O_1) C_2 \end{aligned}$$

Each transaction is divided into subtransactions according to the location of data objects. From T_1 and T_2 following subtransactions are obtained:

$$\begin{aligned} T_{11} &= r_{11}(O_1) r_{11}(O_2) w_{11}(O_1) C_{11} \\ T_{12} &= w_{12}(O_3) C_{12} \\ T_{21} &= r_{21}(O_1) w_{21}(O_1) C_{21} \\ T_{22} &= r_{22}(O_3) w_{22}(O_4) C_{22} \end{aligned}$$

T_{11} and T_{12} are read as subtransaction of transaction 1 at PE_1 and subtransaction of transaction 1 at PE_2 respectively. Assume, the following histories are produced by the local scheduler at the processing elements:

$$\begin{aligned} H_1 &= r_{11}(O_1) r_{11}(O_2) w_{11}(O_1) C_{11} r_{21}(O_1) w_{21}(O_1) C_{21} \\ H_2 &= r_{22}(O_3) w_{22}(O_4) C_{22} w_{12}(O_3) C_{12} \end{aligned}$$

H_1 and H_2 are history at PE_1 and PE_2 respectively. Both the local scheduler produces serializable history with following serialization order:

Scheduler 1: schedules transaction 1 \prec transaction 2

Scheduler 2: schedules transaction 2 \prec transaction 1

The two serialization orders are contradictory. Although individual schedulers generate serial history the combined effect produces a cycle $T_1 \rightarrow T_2 \rightarrow T_1$. Thus, the responsibility of scheduler at each PE is much more than that of a single scheduler. Hence, we conclude that the concurrency control strategies applicable to single scheduler may not be migrated in its present form to multi-scheduler environment.

In the literature, we could not find any algorithm that considers multiple schedulers for PDS. Weaknesses of single scheduler algorithms motivated us to distribute the scheduling responsibilities to the respective PE and consider multiple schedulers. We propose a new serializability, *Parallel Database Quasi-serializability*, criterion to

meet requirements of multi-scheduler concurrency control algorithms in the following sections.

4 Model of Parallel Database System

Shared-nothing parallel database system is considered in this study. For the sake of simplicity we consider only two processing elements to demonstrate our algorithm.

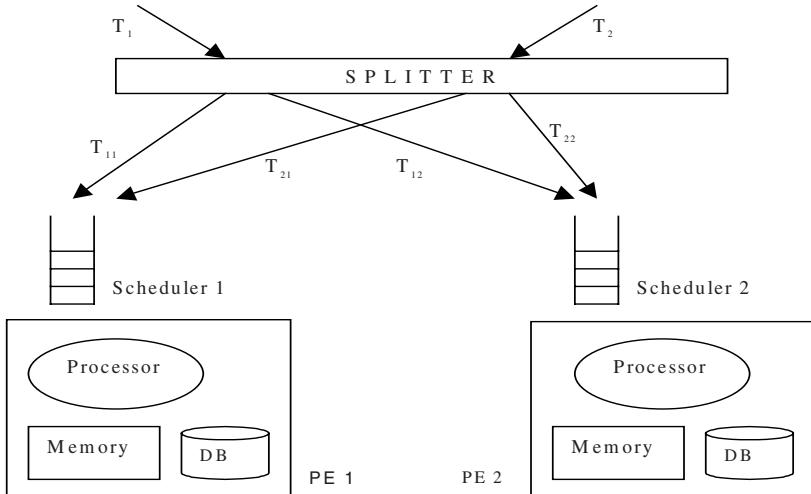


Fig. 2. Conceptual model of multi-scheduler concurrency control

We assume that local schedulers are capable of producing serializable schedule. Different parts of the model is described as follows:

- Splitter:* Splitter contains the information regarding the data allocated to the processing elements and splits the transaction into multiple subtransactions depending on data location. In the literature *splitter* had the responsibility of central scheduler. We will refer to scheduler as shown in figure-2.
- Processing Elements:* PE contains a processor, a memory and a fragment of database as discussed in the shared nothing architecture (section 2).
- Schedulers at each PE:* Schedulers are responsible for generating a serializable schedule. The responsibility of each scheduler is much more than in a single scheduler environment as discussed in the previous section.
- Transactions:* Transactions are exactly same as that of single processor environment and follow the properties of *Definition 1*.

5 Proposed Serializability Theorem and Algorithm

In this section we first discuss the correctness criterion for the Multi-scheduler Concurrency Control algorithm namely *Parallel Database Quasi-serializability*

(PDQ-serializability) subsection 5.2 proposes a PDQ-serializability theorem, subsection 5.3 explains the proposed *Timestamp based Multi-scheduler Concurrency Control* algorithm and finally subsection 5.4 proves the correctness of the algorithm.

5.1 PDQ-Serializability

Scheduling responsibilities in PDS is distributed to respective PE according to the partitioned data. The motivating example shows that although individual schedulers produce serial histories the database is not in a consistent state. Some additional criterion has to be enforced to ensure multi-scheduler concurrency control in addition to subtransaction level granularity. We propose a new serializability criterion that separates two types of transactions – transactions having only one subtransaction and transactions having more than one subtransaction. The following definition states the correctness criterion for PDS.

Definition 5: A *Multi-Scheduler Serial (MS-Serial)* history is considered correct in parallel database system. A history is *MS-Serial* iff:

1. Every PE produces serializable history and
2. Any transaction having more than one subtransaction i.e. accessing more than one PE, executes the transaction according to *total order*. Total order for two transactions T_i and T_j can be defined as – if T_i precedes T_j at any PE then all of T_i 's operations precede T_j 's operations in all PE in which they both appear. \square

The second condition is similar to *quasi-serializability*. Hence, we name the correctness criterion as *Parallel Database Quasi-Serializability*.

Definition 6: A history in multi-scheduler environment is PDQ-serializable iff it is equivalent to a *MS-Serial* history (definition of equivalence (\equiv) from [9], page 30). \square

5.2 Proposed PDQ-Serializability Theorem

The PDQ-serializability of the history is determined by analysing *Parallel Database Quasi-Serializability* (PDQ-serializability) graphs. Only the committed projection [9] of the history $C(H)$ is considered in the definition. The following definition describes the PDQ-serializability graph.

Definition 7: At any given instance, histories of all the schedulers at each PE can be represented using a directed graph defined with the ordered three: (T^1, T^n, A) . The graph would be referred as *Multi-scheduler Serializability Graph (MSG)*.

1. T^1 : set of labelled vertices representing transactions with one *subtransaction*.
2. T^n : set of labelled vertices representing transactions with more than one *subtransaction*.
3. A is the set of arcs representing the ordering of transaction in each PE. \square

In rest of the paper we would only consider transaction having more than one subtransaction and denote that as T . Transactions with single subtransaction are taken care by the individual PE and do not pose any threat to the concerned problem. Based on the definition of MSG we next formalize the following theorem.

Theorem 8: A history in multi-scheduler environment is PDQ-Serializable iff MSG is acyclic.

Proof: (if) Let us assume without loss of generality that the committed history $C(H)$ consists of the set $\{T_1, T_2, \dots, T_n\}$. T_1, T_2, \dots, T_n are transactions with more than one subtransaction. We assume that all processing elements always schedule transactions in serializable order. The n vertices of the MSG ($\{T_1, T_2, \dots, T_n\}$) are acyclic and thus it may be topologically sorted (definition of topologically sorted from [9]). Let $T_{i1}, T_{i2}, \dots, T_{in}$ be a topological sort of the multi-scheduler history for permutation of 1, 2, ..., n. Let the MS-serial history be $T_{i1}, T_{i2}, \dots, T_{in}$. We show that: $C(H) \equiv \text{MS-serial}$. Let $p \in T_i$ and $q \in T_j$ and p and q conflict such that $p \prec_H q$. This means an arc exists in the MSG from T_i to T_j . Therefore in any topological sort of multi-scheduler history T_i precedes T_j . Thus, all operations of T_i precede all operations of T_j in any topological sort. Thus $C(H) \equiv \text{MS-serial}$ and from Definition-6 the history H is PDQ-serializable.

(only if): Suppose a history H is PDQ-serializable and let H_s be a MS-serial history equivalent to H . Consider an arc exists in the MSG, T_i to T_j . This implies there exists two conflicting operations $p \in T_i$ and $q \in T_j$ such that $p \prec_H q$ at some PE_i . This condition is valid as both operation conflicts at the same PE_i . Since, $H_s \equiv H$, all operations of T_i occur before T_j at that specific PE. Suppose there is a cycle in MSG. This implies at any other PE_j , $T_j \prec T_i$ in MS-serial history. But T_i is known to precede T_j at PE_i , which is contradictory to the earlier assumption. \square

5.3 Timestamp Based Multi-scheduler Concurrency Control Algorithm

In this section we propose a *Timestamp based Multi-scheduler Concurrency Control* (TMCC) algorithm that enforce *total order* in the schedule to ensure PDQ-serializability. *Total order* is required only for those conflicting transactions that accesses more than one PE being accessed by other active transactions.

Following functions are used to demonstrate the algorithm:

1. *Split_trans(T_i):* This function takes the transaction submitted to the *splitter* as parameter and returns a set of subtransactions that access different PEs.
2. *PE_accessed(T_i):* This function also takes the transaction as parameter and returns the set of PEs where the subtransactions for T_i are executed.
3. *Active_trans(PE):* This function takes the processing element as parameter and returns the set of transactions that have an active subtransaction executing at that PE.
4. *Cardinality():* This function can take any set as parameter and returns the number of elements in the set.
5. *Append_TS(Subtransaction):* This function takes a *subtransaction* of a transaction T_i and appends a *timestamp* to the subtransaction. All the subtransaction of a transaction will have the same timestamp value.

Algorithm 1: TMCC Algorithm (Transaction submission)

```

begin
  input  $T_i$  : Transaction
  var Active_trans : set of active transactions
  generate timestamp  $ts$  : unique timestamp in increasing order is generated

  Split_trans( $T_i$ )
   $PE_{accessed}(T_i) \leftarrow$  set of Processing Elements accessed
  (1) if  $Cardinality(PE_{accessed}(T_i)) = 1$  then
    begin
    (2)  $Active\_Trans(PE_k) \leftarrow Active\_Trans(PE_k) \cup T_i$ 
    (3) submit subtransaction to PE
    end
  else begin
    (4)  $Active\_trans \leftarrow \bigcup Active\_Trans(PE_k)$ 
    (5)  $Active\_trans \leftarrow \{T \mid T \in Active\_trans \wedge Cardinality(PE_{accessed}(T)) > 1\}$ 
    (6) for each subtransaction of  $T_i$ 
    (7) Append_TS(Subtransaction)
    end for
    (8) if  $Cardinality(PE_{accessed}(T_i) \cap (\bigcup_{T \in Active\_trans} PE_{accessed}(T_j))) \leq 1$ 
      then begin
      (9) for each  $PE_k \in PE_{accessed}(T_i)$ 
        begin
        (10)  $Active\_Trans(PE_k) \leftarrow Active\_Trans(PE_k) \cup T_i$ 
        (11) submit subtransaction to  $PE_k$  ::Subtransaction executes immediately.
        end
      end for
      end
      else
    (12) for each  $PE_k \in PE_{accessed}(T_i)$ 
      begin
      (13)  $Active\_Trans(PE_k) \leftarrow Active\_Trans(PE_k) \cup T_i$ 
      (14) submit subtransaction to PE's Queue ::Subtransaction submitted to queue
      end
    end for
    end if
  end
  end if
end

```

Working of the algorithm is explained below:

1. As soon as the transaction arrives at the splitter, $split_trans(T_i)$ splits the transaction into multiple subtransactions according to the allocation of data.

2. If there is only one subtransaction required by the transaction, the transaction can be submitted to the PE immediately without any delay. (**Line 1**).
3. All the transactions having more than one subtransaction are added to the **Active_Trans** set (**Line 4** and **Line 5**).
4. *Splitter* appends a timestamp with every subtransaction for those transactions that require multiple subtransactions, before submitting to the PE (**Line 6**).
5. If there are active transactions that access one or less than one PEs accessed by the transaction being scheduled then the subtransactions can be scheduled immediately (**Line 8**).
6. If there are active transactions that access more than one of the PEs accessed by the transaction being scheduled then the subtransactions are submitted to the PE's *wait_queue*. The subtransactions from the queue are executed strictly according to the timestamps (**Line 12**).
7. Subtransactions from *step-2* can be assumed to have lowest timestamp value e.g. 0 and can be scheduled immediately.
8. When all subtransactions of any transaction complete the execution at all the sites, the transaction commits and is removed from *Active_trans(PE)* list.

Algorithm 2 explains the steps when any subtransaction terminates at any particular processing element.

Algorithm 2: TMCC Algorithm (Transaction termination)

```

begin
  input subtransaction of  $T_i$  completes execution
   $Active\_Trans(PE_j) \leftarrow Active\_Trans(PE_j) - T_i$  :: removes transaction from the PE
   $PE\_accessed(T_i) \leftarrow PE\_accessed(T_i) - PE_k$ 
    :: removes the PE being accessed from the PE_accessed set
end

```

5.4 Correctness of TMCC Algorithm

The model assumes that each PE is capable of producing serializable schedule (Proposition-1). At the same time motivating example demonstrates the necessity of additional strictness criterion to guarantee the serialization of parallel database systems in multi-scheduler environment.

Proposition 9: All processing elements always schedule all transactions in serializable order. \square

To prove the correctness of the proposed algorithm we show that the additional criterion enforced by the *splitter* will guarantee serializable schedule in parallel database systems. The *splitter* does not control the execution of schedules but only determines the way subtransactions are submitted to the PE. If any transaction

accesses more than one database being accessed by other active transactions, the subtransactions cannot be scheduled immediately and thus have to be submitted to *wait_queue* (Proposition-2). Each processing element's *queue* schedules the subtransactions according to their timestamp.

Proposition 10: *Splitter* submits the subtransactions to the *wait_queue* of PE if active transactions access more than one common database. \square

This ensures that the conflicting transactions are executed according to the timestamp value to ensure the PDQ-serializable execution of the multi-scheduler system. The following Lemma proves this.

Lemma 11: For any two transactions T_i, T_j scheduled by TMCC algorithm, either all of T_i 's subtransactions are executed before T_j at every PE or vice versa.

Proof: There are three cases to be considered.

Case 1) *A transaction T_i requires only single subtransaction (ST_i):* This situation is shown in **line (1)** of the algorithm. The subtransaction is submitted immediately as shown in the algorithm flow chart. From Poposition-1 it follows that any other subtransaction $ST_j \in T_j$ either precedes or follows ST_i .

Case 2) *A transaction T_i splits into multiple subtransactions but accesses only one PE accessed by other active transaction:* This situation is shown in **line (8)** of the algorithm. **Line (8)** checks for the above mentioned condition and the subtransaction is submitted to the PE immediately with a timestamp. The timestamp will be used for case-3. Consider two active transactions that overlap only at one PE. Since they overlap at only one PE, Proposition-1 ensures that the transactions would be ordered in a serializable way.

Case 3) *A transaction T_i splits into multiple subtransactions and accesses more than one PE accessed by other active transaction:* This situation is shown in **line (12)** of the algorithm. Under this condition schedulers at PEs may schedule the transactions in conflicting mode. To avoid this conflict we submit the transactions in the PEs *wait_queue* instead of scheduling it immediately (Proposition-2). The transactions in the *wait_queue* are executed strictly according to the timestamp order.

Say, transaction T_i has two subtransactions T_{i1} and T_{i2} already executing at PE₁ and PE₂. When T_j arrives and it also has T_{j1} and T_{j2} . Then **if** condition at **line (8)** fails and the subtransactions are submitted to the *wait_queue* at each PE. Assume that the timestamp of T_i , $TS(T_i) \prec TS(T_j)$. Timestamp is appended to the subtransactions of T_i and T_j during the execution of **line (6)**. Then T_i will precede T_j at both the sites because the transactions are scheduled strictly according to the timestamp value, thus avoids execution of incorrect schedules. \square

Theorem 12: The Timestamp based Multi-scheduler Concurrency Control algorithm presented produces PDQ-serializable histories.

Proof: All PDQ-serializable histories can be shown by acyclicity of multi-scheduler serializability graph, as demonstrated by Theorem 1. Thus, we will show that the proposed TMCC algorithm avoids cycle in the MSG.

Without loss of generality, consider that there is a sequence of arcs from $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n$. This implies that there exists an operation of subtransaction

$T_{ik} \in T_I$ that precedes and conflicts with an operation of $T_{nk} \in T_n$ at any particular, say PE_k . For a cycle, another sequence of arcs must exist from $T_n \rightarrow T_{n-1} \rightarrow \dots \rightarrow T_I$. Two possibilities can exist due to this sequence. (1) An operation of $T_{nk} \in T_n$ precedes and conflicts with an operation of $T_{ik} \in T_I$ at the same PE_k . This contradicts Proposition 1. (2) An operation of some other subtransaction $T_{nl} \in T_n$ at another PE_l precedes and conflicts with an operation of $T_{ii} \in T_I$. This contradicts Lemma 1. \square

6 Conclusion

Due to the inherent weaknesses of single schedulers (discussed in the paper) multi-scheduler approach has to be investigated. Our investigation shows that algorithms developed for single scheduler cannot guarantee correct schedules in multi-scheduler environment. The notion of conflict serializability is insufficient of producing correct schedule and a new serializability criterion is required for multi-scheduler approach. We discuss the PDQ-serializability and propose an algorithm that guarantees PDQ-serializable schedules. The proposed algorithm distributes the scheduling responsibilities to the PEs, and reduces the overheads of single scheduler strategy.

References

- [1] A. Bhide, "An Analysis of Three Transaction Processing Architectures", *Proceedings of 14th VLDB Conference*, pp. 339–350, 1988.
- [2] D.J. DeWitt, J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems", *Communication of the ACM*, vol. 35, no. 6, pp. 85–98, 1992.
- [3] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
- [4] L. Sha, R. Rajkumar, J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-time Synchronization", *IEEE Transactions*, vol-39, No.9, pp. 1175– 1185, 1990.
- [5] M. Stonebraker, "The Case for Shared-Nothing", *IEEE Data Engineering*, vol. 9, no. 1, pp. 4–9, 1986.
- [6] M.-A. Neimat, D.A.Schneider, "Achieving Transactional Scaleup on Unix", *Parallel and Distributed Information Systems, Proceedings of the 3rd Intl. Conf. on*, pp. 249–252, 1994.
- [7] P. Valduriez, "Parallel Database Systems: The Case For Shared Something", *Proceedings of the International Conference on Data Engineering*, pp. 460–465, 1993.
- [8] P. Valduriez, "Parallel Database Systems: Open Problems and New Issues", *Distributed and Parallel Databases*, volume 1, pp. 137–165, 1993
- [9] P. A. Bernstein, V. Hadzilacos, N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, 1987.
- [10] S. S. Thakkar, M. Sweiger, "Performance of an OLTP Application on Symmetry Multiprocessor System", *Proceeding, 17th Intl. Conf. on Comp. Arch.*, pp. 228–238, 1990.
- [11] T.-W. kuo, J. Wu, H.-C. Hsieh, "Real-time concurrency control in multiprocessor environment", *IEEE Transaction, Parallel and Dist. Sys.*, vol.-13, no.-6, pp. 659–671, '02.
- [12] S.D.Chen; H. Shen; R. Topor, "Permutation-based range-join algorithms on N-dimensional meshes", *Parallel and Dist. Sys., IEEE Trans,Vol-13, No-4, p: 413–431, '02.*

- [13] C.-M. Chen, R. K Sinha, "Analysis and comparison of declustering schemes for interactive navigation queries", *Knowledge and Data Engineering, IEEE Transactions on*, Vol-12 Issue: 5 ,pp: 763–778, 2000.
- [14] K. Barker, "Transaction Management on Multidatabase Systems", PhD thesis, Department of Computer Science, The university of Alberta, Canada, 1990.
- [15] T.Ozsu, P.Valduriez, "*Distributed and Parallel Database Systems*", *ACM Computing Surveys*, vol.28, no.1, pp 125–128, March 1996.
- [16] M.T. Ozsu and P. Valduriez, editors. *Principles of Distributed Database Systems* (Second Edition). Prentice-Hall, 1999.

Optimal Broadcast Channel for Data Dissemination in Mobile Database Environment

Agustinus Borgy Waluyo¹, Bala Srinivasan¹, and David Taniar²

¹ School of Computer Science and Software Engineering,
Monash University, Australia

{Agustinus.Borgy.Waluyo,Bala.Srinivasan}@infotech.monash.edu.au

² School of Business Systems,
Monash University, Australia
David.Taniar@infotech.monash.edu.au

Abstract. The increase number of mobile users in wireless environment affects query access time substantially. To minimise the query access time, one possible way is to employ data broadcasting strategy. In this paper, we propose cost models for both query access time over broadcast channel and on-demand channel. We examine the cost models to find optimum number of broadcast items in a channel while utilising query access time over on-demand channel as a threshold point. The optimum number indicates a point to split the broadcast cycle and allocate the data items in the new channel or else the on-demand channel outperforms the broadcast channel. The cost model involves several factors that dynamically change the optimum number of broadcast items like request arrival rate, service rate, size of data item, size of request, and bandwidth. Simulation model is developed to verify the performance of the cost model. This paper focuses on request that returns a single data item.

1 Introduction

The invention of wireless technology has created a new era of computing application. Nowadays, people are no longer attached to a stationary machine to do their work, with wireless application they are enabled to conduct their business anywhere and anytime using portable size wireless computer powered by battery. These portable computers communicate with central stationary server via wireless channel. This technology is known as *mobile computing* [2,3,6]. Mobile computing has its own database management system (DBMS) that provides the same function as in traditional databases, which is called *mobile databases*. Mobile databases face a number of limitations particularly power, storage and bandwidth capacity. In regard to power capacity, it has been investigated that the life expectancy of a battery is anticipated to increase only 20% for every 10 years [11]. Consequently, the need to use power efficiently and effectively is a crucial issue. In a mobile environment, most applications involve read operations rather than write operations [5]. This paper concerns with broadcast strategy to optimize the read operation or referred as query operations in mobile databases.

Broadcast strategy refers to periodically broadcast database items to clients through one or more broadcast channels. Mobile clients filter their desired data on the fly. This strategy is known as an effective way to disseminate database information to a large set of mobile clients. The query performance using this strategy is independent from the number and frequency of the query, or known as scalable paradigm. However, the challenge in broadcast strategy is to maintain the query performance of the client to obtain information from the channel.

It is the *aim* of this paper to optimize the query access time of accessing data from broadcast channel. The query access time over on-demand channel is used as a threshold point to determine the optimum number of database items to be broadcast in a channel. Once, the optimum number is located, the number of broadcast channel should be increased. Subsequently, the length of broadcast items is split, and broadcast over multiple channels. With this strategy, the access time of mobile users for accessing data items over broadcast channel is kept considerably low.

To achieve our objective, we propose cost models for both on-demand and broadcast channel. Subsequently, we use simulation model to verify the results of our cost models in finding the optimum number of broadcast items. In this paper, we focus on request that returns single data items.

The rests of the section in this paper are organised as follows. Section 2 describes the background of mobile databases, and section 3 contains the related work of the proposed technique. It is then followed by the description of the optimal broadcast channel, cost model for both broadcast and on-demand channel, and its application in section 4. Section 5 introduces parameters of concern of the optimal broadcast channel, which includes performance results of cost models and simulation models to determine optimal broadcast channel. Finally, section 6 concludes the paper.

2 Background

In general, each mobile user communicates with a Mobile Base Station (MBS) to carry out any activities such as transaction and information retrieval. MBS has a wireless interface to establish communication with mobile client and it serves a large number of mobile users in a specific region called *cell*. Mobile units or mobile clients in each cell can either connect to the network via wireless radio or wireless Local Area Network (LAN). Wireless radio bandwidth has asymmetric communication behavior in which the bandwidth for uplink communication is smaller than downlink communication [7]. Uplink bandwidth is applicable when mobile clients send a request or query to the server, while downlink bandwidth is from the server back to the clients.

With the increase number of mobile users in a cell, the required number of data items to be broadcast also increases accordingly. This situation may cause some mobile clients to wait for a substantial amount of time before receiving desired data item. Consequently, the advantages of broadcast strategy will be eliminated. On the other hand, when the channel contains too few data items, a high percentage of mobile clients waste the time by listening to the channel until they find out that the desired data is not broadcast. Alternatively, they send a request via point-to-point channel or on-demand channel to the server. The server processes the query and sends

the result back to the client. The last scenario severely affects the power consumption of mobile clients, as it involves queuing in the server.

Figure 1 illustrates the two mechanisms of mobile clients to obtain the desired data, first mechanism is via on-demand channel, and the other is via broadcast channel. It is sometimes called pull-based, and push-based approach respectively. In this paper, the term mobile client, mobile computer, mobile device, mobile user and client are used interchangeably and the server refers to MBS.

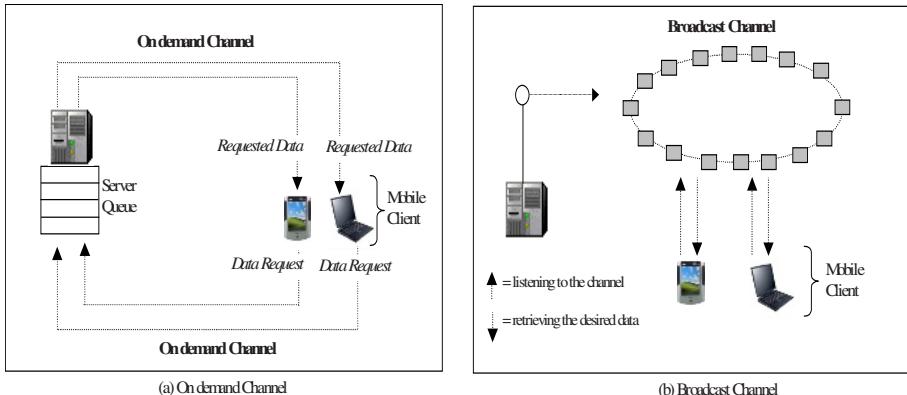


Fig. 1. The architecture of on-demand channel and broadcast channel

3 Related Work

Strategies to broadcast data through wireless channel have been introduced in the past few years [1, 8, 9, 12]. Broadcast strategy is known to be effective when involving a large number of users in the cell, which corresponds to a high frequency of requests.

[4] has introduced the hybrid and dynamic channel allocation method, which manipulates both the on-demand channels and broadcast channels. The hybrid allocation method provides a solution by determining the number of data to be served from on-demand channel and broadcast channel to enhance the query performance. Dynamic hybrid method is the improvement of the hybrid method, which changes the number of channel allocated for on-demand mode and broadcast mode dynamically by considering the load of the server and request access pattern. However, the access overhead of on-demand channels only considers the waiting time of connection to the server. Several factors that should be taken into account include transmission cost of the query from the mobile client to the server, processing cost in the server and the cost to send the query response back to the mobile client. The cost to transmit the query to server is even worse when the query requires related data items that belong to a number of entity types due to the uplink bandwidth limitation. Furthermore, the frequency or the rate of the query initiated affects the performance of the on-demand channel.

Our optimal broadcast channel tries to eliminate the consequence of having a massive set of data to be broadcast. The strategy is used to determine the optimum number of broadcast items in a channel so that the average access time is kept minimum [13].

4 Optimal Broadcast Channel

Optimal broadcast channel is designed to find out optimum number of broadcast items in a broadcast channel. Having known the optimum number of broadcast items in a channel enables us to determine the optimum number of channel required to broadcast a certain number of data items. In this paper, the query performance is measured by the average access time of broadcast channel. Access time is defined as the amount of time or waiting time needed from the time a request is initiated until data item of interest is received. In general, the data items are broadcast over a single channel with underlying reason that it provides the same capacity (bit rate/bandwidth) as multiple channels, and it is used to avoid problems such data broadcast organization and allocation while having more than one channel [7]. Nevertheless, the use of single channel will show its limitation when there is a vast number of a data item to be broadcast.

The proposed strategy is used to split the length of the broadcast cycle when the number of broadcast items reaches an optimum point. Figure 2 illustrates a situation when the optimum number of items in a channel is known.

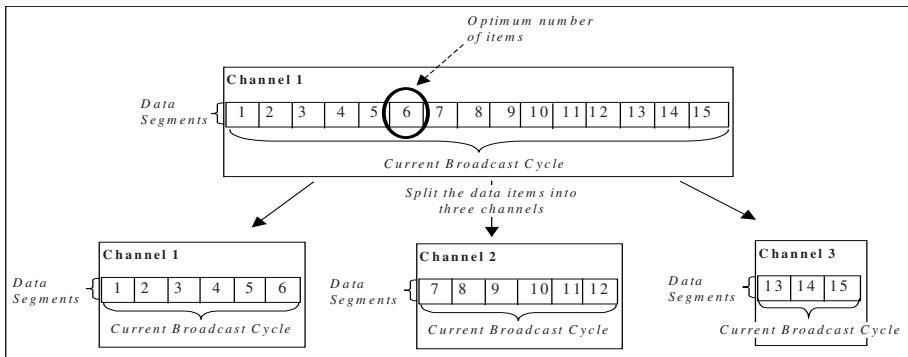


Fig. 2. Optimal Broadcast Channel

The allocation of broadcast cycle into an optimal number of broadcast channels will eliminate the chance of long delay before obtaining the desired data items. As mobile clients are able to listen to different channels simultaneously, they can determine which channel contains the required data items. Once it has been found, they may wait to retrieve the data items in the corresponding channel. Since the length of the broadcast cycle is optimal, the waiting time will be considerably short.

To find the optimum number of broadcast items, we develop cost models to calculate average access time of broadcast and on-demand channel. In this context, mobile client can only initiate a single request at a time, the next request has to wait until the first request has been completely responded.

4.1 Cost Model for Broadcast Channel

To calculate the average access time for retrieving data (T_B) using broadcast channel, we consider the following scenario:

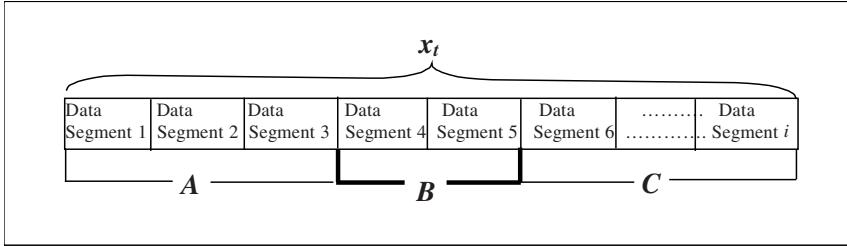


Fig. 3. Broadcast Cycle Partitioned Area

In figure 3, the broadcast cycle is partitioned into three areas: area A contains data segments preceding data segments in area B , area B contains the number of desired data segments, and area C includes the rest of data segments. x_t is the total number of broadcast data that makes up of $A+B+C$. There are three different scenarios when mobile client probe into one of these area.

Probe A: When mobile client probes into area A , the total access time is given:

$$\frac{\sum_{i=0}^{A-1} (A - i + B) \times s}{b_s} \text{ or similarly } \frac{[A \times (A + 2B + 1)] \times s}{2 \times b_s} \quad (1)$$

Probe B: When mobile client probes into area B , then the total access time is equal to the total length of broadcast cycle ($A+B+C$):

$$\frac{(A + B + C) \times B \times s}{b_s} \quad (2)$$

Probe C: When mobile client probes into area C , the total access time can be calculated from:

$$\frac{\sum_{i=0}^{C-1} (C - i + A + B) \times s}{b_s} \text{ or equally } \frac{C \times (2A + 2B + C + 1) \times s}{2 \times b_s} \quad (3)$$

We need to calculate the average access time from equation (1), (2), and (3) as follows:

$$\frac{[(A \times (A + 2B + 1)) + (2B \times (A + B + C)) + (C \times (2A + 2B + C + 1))] \times s}{2 \times x_t \times b_s} \quad (4)$$

Equation (4) can be rewritten as:

$$T_B \approx \frac{[(x_t - B)^2 + (2 \times B \times x_t) + (2 \times B \times (x_t - B)) + (x_t - B)] \times s}{2 \times x_t \times b_s} \quad (5)$$

s corresponds to the size of data item, we consider the size is uniform. The downlink bandwidth is denoted by b_s .

4.2 Cost Model for On-Demand Channel

As mentioned earlier, mobile clients can send their request to be processed via point-to-point or on-demand channel. Thus, the average access time of on-demand channel (T_D) is made for comparison. It is used as a threshold point to determine the optimum number of database items to be broadcast in a channel. To calculate the average access time of on-demand channel, we classify the cost model into the following scenarios:

a. Pre-determined Number of Request

In this scenario, the total number of request has been known. The average access time can be calculated using the following cost model:

$$T_D \approx \frac{r}{b_r} + \frac{\sum_{i=1}^n i \times \frac{1}{\mu}}{n} + \frac{s}{b_s} \quad (6)$$

The cost model to calculate the access time of on-demand channel in equation (6) is based on the architecture that is depicted in figure 1a. The access time of on-demand channel comprises of the time needed to transmit a request to the server, process the request that involves the time spent in the server queue and retrieve the relevant data, and send the result back to the client. The transmission of request to the server and send the requested data item back to mobile client are affected by uplink bandwidth, b_r and downlink bandwidth, b_s . Size of a request and the size of requested data item are indicated by r and s respectively, n denotes the total number of request, i reads the request number and the server service rate is given by μ . It is assumed the request size is uniform, and each request returns a same size of data item. However, in real situation the number of requests is hardly known. Thus, we consider arrival rate of request, which is described in the subsequent strategy.

b. Request Arrival Rate

We replace determined number of request with the arrival rate of request (λ). Furthermore, we specify the cost model based on the load of the server (ρ). Thus, the average access time of on-demand channel (T_D) is now calculated using the following cost model:

If the server is at light load $\rho = \left(\frac{\lambda}{\mu} \leq 1 \right)$, then:

$$T_D = \left(\frac{r}{b_r} + \frac{1}{\mu} + \frac{s}{b_s} \right) \quad (7)$$

Else if the server at heavy load $\rho = \left(\frac{\lambda}{\mu} > 1 \right)$, then:

$$T_D = \left(\frac{r}{b_r} \right) + \frac{\sum_{i=1}^{q_{\max}} \left(i \times \frac{1}{\mu} \right) - \left[(i-1) \times \frac{1}{\lambda} \right]}{q_{\max}} + \left(\frac{s}{b_s} \right) \quad (8)$$

The average time in server queue and average processing time by the server are dynamically changed depending on the arrival rate of request (λ) and service rate (μ). The maximum number of request in the server queue is defined by q_{\max} .

Table 1. Parameters of Concern

Parameter	Description	Initial Value
<i>Broadcast channel</i>		
x_l	Total number of Broadcast Items	250
B	Number of relevant data item	1
<i>On-demand channel</i>		
n	Number of request	100
b_r	Uplink Bandwidth	192 bytes
r	Size of request	50 bytes
μ	Server Service Rate	2 request per second
<i>Broadcast and On-demand channel</i>		
b_s	Downlink Bandwidth	2000 bytes
s	Size of each data item	500 bytes

5 Performance Evaluation

In this section, we introduce two cases, and employ the cost model to determine the optimum number of broadcast data items. We also develop some simulation models to compare with our analytical results. The simulation is carried out using a simulation package *Planimate*, animated planning platforms [10].

The simulation environment is set to apply random delays for both arrival rate and service rate with given average value. We run the simulation process for three times, and derive the average result accordingly.

Case 1: To find the optimum number of broadcast items based on set of parameters in table 1.

As shown in figure 4, we try to find the cross line of on-demand channel and broadcast channel. The average access time of on-demand channel is the threshold point, which appears in a straight line since the number of broadcast items does not affect its value. Our analytical results stay closely with the simulation results. The intersection point indicates the optimum number of broadcast items.

Case 2: Introducing a new parameter, arrival rate of request (λ). Given the set of parameters in case 1, we specify one request arrival rate per second, $\lambda = 1$, to obtain server utilization $\rho \leq 1$, and three request per second, $\lambda = 3$, to get $\rho > 1$.

As shown in figure 5 (a), the on-demand channel performs well when the traffic request is low. The broadcast channel seems to have to allocate its data items in every 5-10 data items to a new channel to keep up with the performance of on-demand channel. However, this is not a good way since the number of channels may be excessive and may cause some overhead by listening to too many channels. Thus, on-demand channel in this case is a better way. Figure 5(b) illustrates the optimum number of broadcast items when the on-demand channel is at heavy load.

From table 3, we can see the optimum number of broadcast items derived from our analytical and simulation results on the two cases. Our analytical calculation performs very well as compared to the simulation, especially for case 1, and case 2 (server utilization > 1) with 9% and 4% error rate respectively. As for case 2 (server

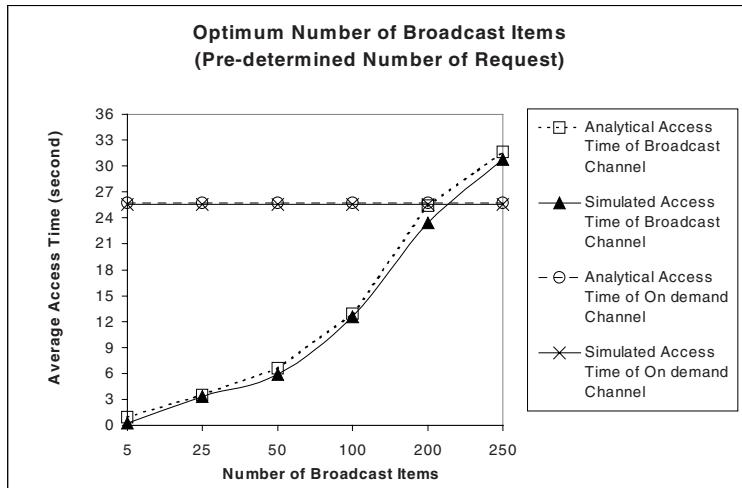


Fig. 4. Optimal Number of Broadcast Items with pre-determined number of request

Table 2. Average Access Time of On-demand and Broadcast channel

On-Demand Channel		Broadcast Channel		
Analytical Average Access Time (sec)	Simulated Average Access Time (sec)	Number of Broadcast Items	Analytical Average Access Time (sec)	Simulated Average Access Time (sec)
25.760	25.55	5	0.95	0.25
		25	3.49	3.375
		50	6.62	5.93
		100	12.8725	12.57
		150	25.37375	23.41
		200	31.624	30.75

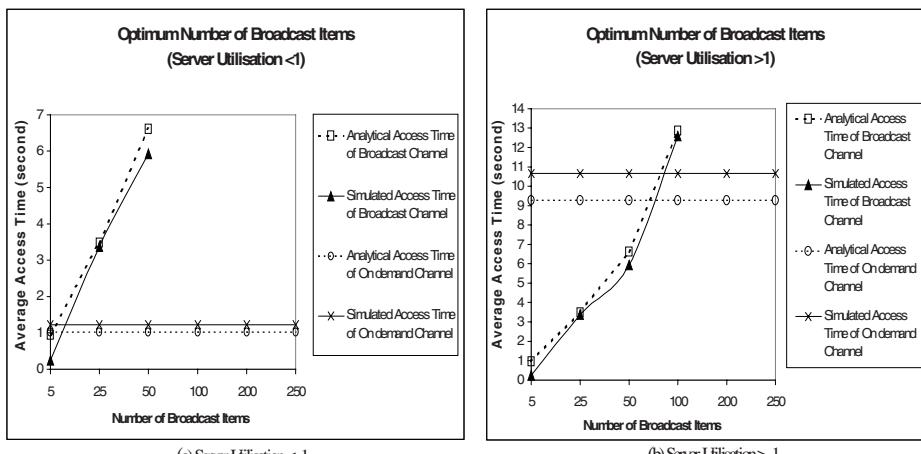


Fig. 5. Optimal Number of Broadcast Items with $\rho < 1$ and $\rho > 1$

Table 3. Simulation and Analytical Performance in Determining Optimum Broadcast Items

Optimum Number of Broadcast Items in a Channel					
		Analytical	Simulated	Difference	Error Rate
Case 1	Pre-determine Number of Requests	204	225	21	0.09
Case 2	Server Utilisation <1	5	9	4	0.44
	Server Utilisation >1	71	74	3	0.04

utilization < 1), we presume that since the value is too small, the random delays in simulation parameters cause a greater affect towards optimal number of broadcast items.

In general our analytical calculation presents a less value than the simulation. Our cost models to find an optimum number of broadcast items are considered close to accurate. Subsequently, having known the optimum number of items in a broadcast channel, we can decide how many channels are required to broadcast a certain amount of data items. Thus, the average access time of broadcast channel is kept minimum.

6 Conclusions and Future Work

Data dissemination or known as data broadcasting strategy is an effective way to keep up with number of clients in a cell and their frequency of requests. Mobile clients are enabled to retrieve the required data with considerably low access time. To maintain the performance of broadcast strategy, we introduce a technique to find the optimal broadcast channel by determining the optimum number of database items to be broadcast in a channel. The optimum number is used as an indication of when the broadcast cycle needs to be split into different channel, forming multiple channels.

We develop cost models to find the average access time of on-demand channel and broadcast channel. The cost models are applied with focusing on request that returns a single data item. To locate the optimum value of broadcast items, we compare the access time of broadcast channel and on-demand channel. The access time of broadcast channel is calculated based on certain factors, particularly number of database items to be broadcast, size of data item, and downlink bandwidth. As for on-demand channel, it takes into account the server service rate, request arrival rate, number of requests, size of request, uplink and downlink bandwidth. The optimum number of broadcast items is changed dynamically depending on the value of these factors. We also conduct a simulation to see the performance of our cost model in determining the optimum broadcast items. The simulation results show our cost models are considerably close to accurate.

For future work, we will consider multi data items retrieval in a single request, which includes a non-uniform size of request and data items. We will also measure the tuning time or the amount of time a client must listen to the channel, which is used to indicate its energy consumption.

References

1. Acharya S., Alonso R., Franklin M. and Zdonik S., "Broadcast Disks: Data Management for Asymmetric Communication Environments", *Proceedings of ACM Sigmod International Conference on Management of Data*, pp.199–210, May, 1995.
2. Barbara D., "Mobile Computing and Databases – A Survey", *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, No. 1, pp.108–117, January/February, 1999.
3. Badrinath B. R. and Phatak S. H., "An Architecture for Mobile Databases", *Technical Report DCS-TR-351*, Department of Computer Science, Rutgers University, New Jersey
4. Hu Q., Lee D. L. and Lee W.C., "Optimal Channel Allocation for Data Dissemination in Mobile Computing Environments", *Proceedings of 18th International Conference on Distributed Computing Systems*, pp.480–487, May, 1998.
5. Huang Y., Sistla P. and Wolfson O., "Data Replication for Mobile Computers", *Proceedings of the ACM SIGMOD*, pp.13–24, June, 1994.
6. Imielinski T. and Viswanathan S., "Adaptive Wireless Information Systems", *Proceedings of SIGDBS (Special Interest Group in Database Systems) Conference*, October, 1994.
7. Imielinski T., Viswanathan S. and Badrinath B. R., "Data on Air: Organisation and Access", *IEEE Transactions on Knowledge and Data Engineering*, Vol.9, No. 3, pp.353–371, May/June, 1997.
8. Imielinski T., Viswanathan S. and Badrinath B. R., "Energy Efficient Indexing on Air", *Proceedings of the ACM Sigmod Conference*, pp.25–36, May, 1994.
9. Leong H. V. and Si A., "Database Caching Over the Air-Storage", *The Computer Journal*, Vol.40, No. 7, pp.401–415, 1997.
10. Seeley D. et al, *Planimate™-Animated Planning Platforms*, InterDynamics Pty Ltd, 1997.
11. Sheng S., Chandrasekaran A. and Broderson R. W., "A Portable Multimedia Terminal for Personal Communication", *IEEE Communications*, pp.64–75, December, 1992.
12. Si A. and Leong H. V., "Query Optimization for Broadcast Database", *Data and Knowledge Engineering*, Vol.29, No. 3, pp.351–380, March, 1999.
13. Waluyo, A.B., Srinivasan B., Taniar D., "Current Trend in Mobile Database Query Optimization", *submitted to ACM Transactions on Database Systems*, 2003.

Global System Image Architecture for Cluster Computing*

Hai Jin, Li Guo, and Zongfen Han

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

Abstract. This paper presents a novel single system image architecture for cluster system, called Glosim. It is implemented in the kernel layer of operating system, and modifies system invokes relative to IPC objects and process signals. This system provides global system image support. It not only support global IPC objects including message queue, semaphore and shared memory, but also a new concept of global working process and it SSI support smoothly and transparently. Combined with Linux Virtual Server, single IO space, it completely constructs a high performance cluster server with SSI.

1 Introduction

Single system image (SSI) [1][2][3][4] is the property of a system that hides the heterogeneous and distributed nature of the available resources and presents them to users and applications as a single unified computing resource. SSI can be enabled in numerous ways, ranging from those provided by extended hardware to various software mechanisms. SSI means that users have a global view of the resources available to them irrespective of the node to which they are physically associated.

The design goals of SSI for cluster systems are mainly focused on complete transparency of resource management, scalable performance, and system availability in supporting user applications. In this paper, we present a novel single system image architecture for cluster system, called *Global System Image* (Glosim). Our purpose is to provide system level solution needed for a single system image on high performance cluster with high scalability, high efficiency and high usability. Glosim aims to solve these issues by providing a mechanism to support global *Inter-Process Communication* (IPC) objects including message queue, semaphore and shared memory and by making all the working process visible in the global process table on each node of a cluster. It completely meets the need of people of visiting a cluster as a single server, smoothly migrating Unix programs from traditional single machine environment to cluster environment almost without modification.

The paper is organized as follows: Section 2 provides the background and related works on single system image in cluster. Section 3 describes a generic and scalable global system image project overview. Section 4 describes Glosim software

* This paper is supported by National High-Tech 863 Project under grant No. 2002AA1Z2102.

architecture and introduces the global IPC and global working process. In Section 5, we discuss the implementation of Glosim including software infrastructure and error detection. Section 6 presents analysis of performance effects on the developed prototype of the Glosim. Section 7 describes the Glosim enabled applications. Finally, a conclusion is made in Section 8 with more future work.

2 Related Works

Research of OS kernel-supporting SSI in cluster system has been pursued in a number of other projects, including SCO UnixWare [5], Mosix [6][7], Beowulf Project [8][9], Sun Solaris-MC [10][11] and GLUnix [12].

UnixWare NonStop cluster is a high availability software. It is an extension to the UnixWare operating system in which all applications run better and more reliably inside a SSI environment. The UnixWare kernel has been modified via a series of modular extensions and hooks to provide single cluster-wide file system view, transparent cluster-wide device access, transparent swap-space sharing, transparent cluster-wide IPC, high performance internode communications, transparent cluster-wide process migration, node down cleanup and resource fail-over, transparent cluster-wide parallel TCP/IP networking, application availability, cluster-wide membership and cluster time sync, cluster system administration, and load leveling.

Mosix provides the transparent migration of processes between nodes in the cluster to achieve a balanced load across the cluster. The system is implemented as a set of adaptive resources sharing algorithms, which can be loaded into the Linux kernel using kernel modules. The algorithms attempt to improve the overall performance of the cluster by dynamically distributing and redistributing the workload and resources among the nodes of a cluster of any size.

Beowulf cluster refers to a general class of clusters built for speed, not reliability, built from commodity off the shelf hardware. Each node runs a free operating system like Linux or Free-BSD. The cluster may run a modified kernel allowing channel bonding, global PID space or DIPC [13][14][15][16]. A global PID space lets users see all the processes running on the cluster with *ps* command. DIPC make it possible to use shared memory, semaphores and wait-queues across the cluster.

Solaris MC is a distributed operating system prototype, extending the Solaris UNIX operating system using object-oriented techniques. To achieve these design goals each kernel subsystem was modified to be aware of the same subsystems on other nodes and to provide its services in a locally transparent manner. For example, a global file system called the proxy file system, or PXFS, a distributed pseudo */proc* file-system and distributed process management were all implemented.

GLUnix is an OS layer designed to provide support for transparent remote execution, interactive parallel and sequential jobs, load balancing, and backward compatibility for existing application binaries. GLUnix is a multi-user system implementation built as a protected user-level library using the native system services as a building block. GLUnix aims to provide cluster-wide name space and uses network PIDs (NPIDs) and virtual node numbers (VNNs). NPIDs are globally unique process identifiers for both sequential and parallel programs throughout the system. VNNs are used to facilitate communications among processes of a parallel program. A suite of user tools for interacting and manipulating NPIDs and VNNs is supported.

3 System Overview

Glosim is a single system image implementation prototype based on Linux cluster. It is mainly used to construct high performance cluster server, such as distributed web server, distributed BBS server, distributed MUD server. The design goal of Glosim is to conveniently migrated UNIX programs from former single machine environment to distributed cluster environment without too much modification. For example, former FireBird BBS can be easily migrated to Glosim distributed cluster environment just by putting global variables into shared memory structure.

Figure 1 is the complete infrastructure of Glosim. Combined with Linux Virtual Server [17], SIOS [18][19], it completely constructs a high performance cluster network server with SSI. Cluster network server based on Glosim consists of three major parts with three logical layers to provide high load network services.

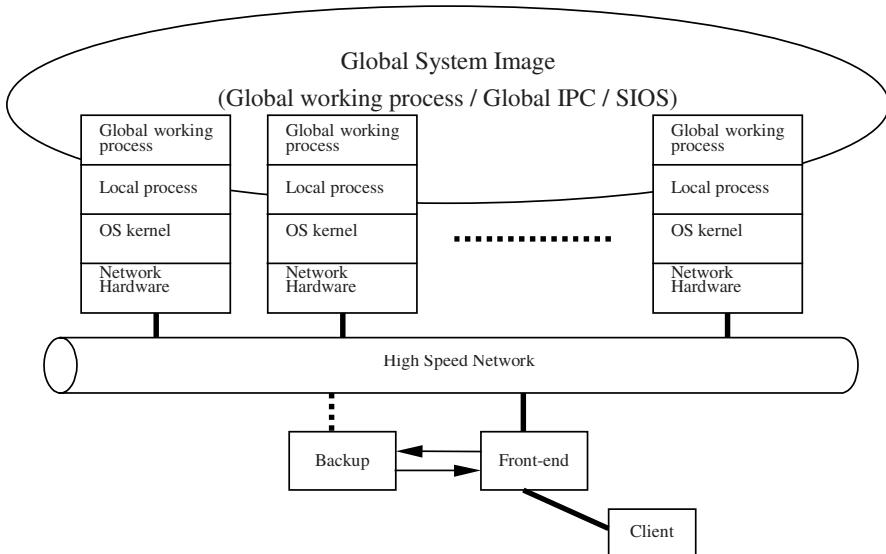


Fig. 1. Glosim Software Architecture

The first layer includes front-end and backup running Linux Virtual Server to schedule and backup each other to improve system reliability. They are the message entrance to the whole server in charge of receiving client requests and distributing these requests to the processing nodes of the cluster according to certain strategy (such as WRR [17] algorithm). Because Front-end and Backup are only in charge of redirections requested by clients and do not involve data processing, they are quite slightly loaded. Besides, in front-end and backup, hot standby daemon is running for fault tolerance. In case of failure in front-end, backup can take charge of its functions by robbing IP and recover Front-end kernel data to achieve high reliability of the whole system network connection.

The second layer are local OS kernel and local processes of the processing nodes inside cluster, which are in charge of system booting, providing basic system call and application software support. Logically, they are outside the boundary of SSI.

The third layer is Glosim layer, as middleware of cluster system, which includes global IPC, global working process, SIOS. Logically, this layer is inside the boundary of SSI. Glosim is implemented in OS kernel and transparently provides SSI service to user applications by modifying system invokes relative to IPC objects and process signals and by mutual cooperation and scheduling between nodes inside the cluster.

4 Glosim Software Architecture

4.1 Global IPC

Linux supports three types of inter-process communication mechanism, which are respectively message queue, semaphore and shared memory. These three communication mechanisms first appear in Unix System V using the same authorization method. Processes can only access these IPC objects by transferring a unique reference designator to the kernels through system invokes. This unique reference designator is the key point to implement the single system image of IPC objects within the cluster.

Global inter-process communication IPC mechanism is the globalization of IPC objects, mainly based on DIPC. Through DIPC, it provides the whole cluster with global IPC, including consistency of shared memory, message queue and semaphore.

What should be specially pointed out here is that distributed shared memory [1][2] in global IPC uses a multiple-read/single-write protocol. Global IPC replicates the contents of the shared memory in each node with reader processes, so they can work in parallel, but there can be only one node with processes that write to a shared memory. The strict consistency model is used here, meaning that a read will return the most recently written value. It also means that there is no need for the programmer to do any special synchronization activity when accessing a distributed shared memory segment. The most important disadvantage with this scheme is possible loss of performance in comparison to other DSM consistency models.

Global IPC can be configured to provide a segment-based or a page-based DSM. In the first case, DIPC transfers the whole contents of the shared memory from node to node, with no regard to whether all that data are to be used or not. This could reduce the data transfer administration time. In the page-based mode, 4KB pages are transferred as needed. This makes multiple parallel writes to different pages possible. In global IPC, each node is allowed to access the shared memory for at least a configurable time quantum. This lessens the chances of the shared memory being transferred frequently over the network, which could result in very bad performance.

The establishment of global inter-process communication inside cluster system provides the basis for data transfer between processes in each node of the cluster. The programmer can use standard inter-process communication functions and interface of System V to conveniently write application programs suitable to cluster system. What's more, former programs can be migrated to Glosim without too much payout. Actually, Glosim enabled applications only require the programmer to do some special transactions to global variables. For example, global variables can be stored as structures in shared memory, and accessed normally.

4.2 Global Working Process

Global Working Process includes Global Working Process Space, Global Signal Mechanism and Global Proc File System.

Considering a practically running cluster system, it is necessarily a server providing single or multiple services and we just call these serving processes working process. What we should do is to include working processes to the boundary of SSI and provide them SSI service from system level. In this way, working processes in different nodes can communicate transparently and become global working processes.

However, not all processes in each node are necessarily to be actualized with SSI in the whole cluster system. That is to say, only some of the processes reside within the boundary of SSI. By introducing the concept of working process, we may solve the inconsistency of OS system processes between each node. OS system process of each node is relatively independent, outside SSI boundary while global working process is provided by the system with single system image service, inside SSI boundary. The independence of working process is convenient for the cluster system to extend to OS heterogeneous environment.

As for practical programming implementation, it is quite easy to express working process. For example, we can tag working process with a special uid for the convenience of kernels identification and independent transaction. This enactment is closely related to the practical system, such as httpd process in Apache is often running as an apache user and bbsd process in BBS is executed as a bbs user.

(a) Global Working Process Space

Single process space has the following basic features:

- Each process holds one unique pid inside the cluster.
- A process in any node can communicate with other processes in any remote node by signals or IPC objects.
- The cluster supports global process management and allows the management of all working processes just like in local host.

In order to uniquely tag the working process in the whole cluster, we must introduce the concept of global working process space. There are mainly two solutions as follows:

- Method 1: modulus the process number to tag the node number in which the process runs. For example, in a cluster system with 16 nodes, the way of calculating the node number of process 10001 is $10001\%16=1$. So process 10001 is running in node 1.
- Method 2: distribute different working process pid space to each node. For instance, the working process space of node 1 is between 10000 and 20000, the working process space of node 2 is between 20000 and 30000, and so on.

No matter which algorithm we adopt, the purpose is to make working process pid unique within the whole cluster. As for each node, local process pid is decided by last_pid of OS kernel itself. Each node is independent on each other but every local pid is smaller than the lower limit of global process space fixed in advance. In short, local process pid is independent on each other while global working process pid is unique in the whole cluster.

(b) Global Signal Mechanism

Signal is the oldest inter-process communication mechanism in Unix system, used to transfer asynchronous signals. In global signal mechanism, all working processes can send each other signals using the functions like signal(), kill() and so on just like in the same computer. That is to say, processes can transparently conduct mutual communication within the cluster by using functions like signal(), kill() and so on.

The transaction flow is as follows:

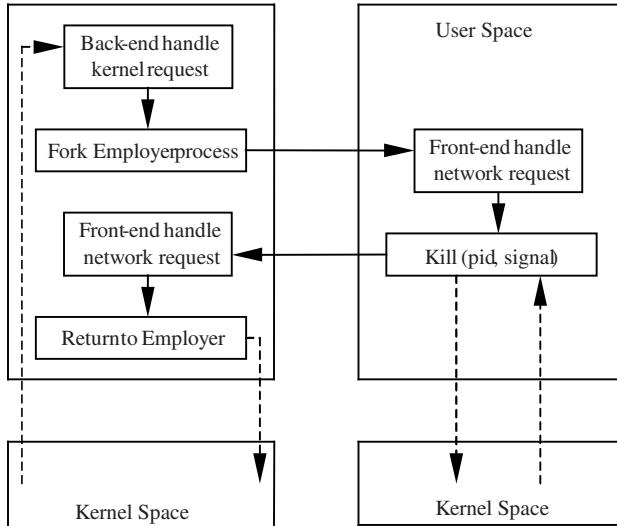


Fig. 2. Processing Flow of Glosim

(c) Global Proc File System

Global proc file system is to map status messages of all processes in every node to the proc file system of each node within the cluster for the convenience of united control and management. Global proc file system includes local processes with the process number smaller than max_local_pid and global processes with the process number bigger than max_local_pid. This is because Glosim system aims to actualize the single system image of working processes and one basic characteristic of working processes is the process number bigger than max_local_pid. So we can only consider working processes.

With the actualizing of global proc file system, we can use `ps -ax` command to examine local processes running in current node and all global working processes in the whole Glosim system. We can employ `kill` command to send signals to any process and even use `kill -9` command to end any process we want with no regard to which node it actually runs. In this way, complete working process single image is implemented in cluster system.

5 Implementation of Glosim

Glosim has two components: The major one is a program called glosimd, which runs in the user space with root privileges. The other component is hidden inside the linux kernel, which allows glosimd to access and manipulate kernel data. This design is the result of the desire to keep the needed changes in the operating system kernel to a minimum.

Glosimd creates some processes to manage all the necessary decision makings and network operations. They use predetermined data packets to communicate with glosimd processes on other nodes. All the necessary information for a requested action are included in these packets, including the system call's parameters obtained from inside the kernel. The whole system infrastructure is shown in Figure 3.

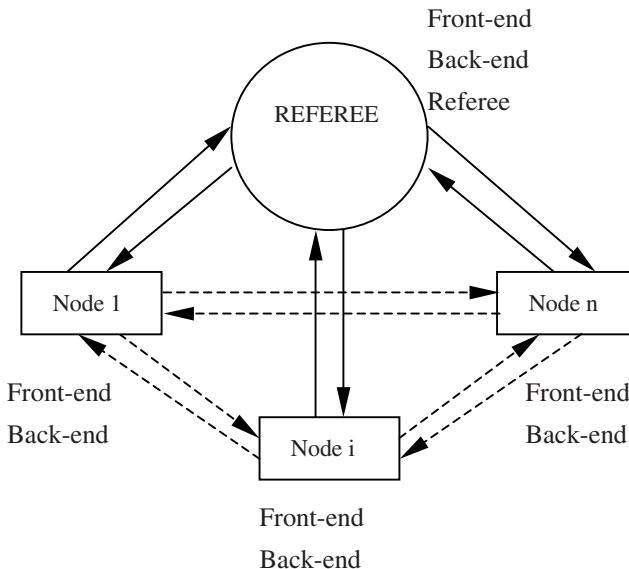


Fig. 3. System Implementation Infrastructure

Glosim includes three function modules: Front-end, Back-end and Referee. Among those, Front-end runs in every node, in charge of transacting data requests from network and forking worker process. Back-end also runs in every node, in charge of transacting data request from kernel and forking employer process. Referee, as the central arbitrating node, only runs in the central node, maintaining a chain table of all IPC structures and pid. Employer, forked by Back-end, executes remote system invokes. Worker, forked by Front-end, transacts remote requests from other nodes.

Glosim adopts a fail-stop distributed environment. So it uses time-outs to find out any problem. Here the at-most-once semantics is used, meaning that Glosim tries everything only once. In case of an error, it just informs it of the relevant processes; either by a system call return value, or, for shared memory read/writes, via a signal. Glosim itself does not do anything to overcome the problem. It is the user processes that should decide how to handle it.

6 Performance Analysis

This section presents some basic analytical results on the performance of the Glosim Cluster. The benchmark program sets up a shared memory, a message queue, and a semaphore set. The tested system calls are: semctl(), msgctl(), shmctl() with the IPC_SET command, semop() and kill() operations. Besides, msgsnd() and msgrcv() with 128, 256, 512, 1024, 2048 and 4096 bytes messages are also tested.

Figure 4 is an experiment conclusion to measure the speed of executing some of the Glosim system calls in a system. It shows the executing times of semctl(), msgctl(), shmctl() with the IPC_SET command, semop(), and kill() in practical system benchmark with the unit ms.

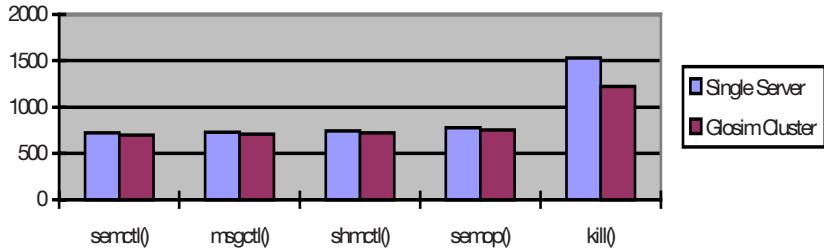


Fig. 4. Speed of executing some Glosim system calls in a system

Figure 5 shows some experiment conclusions to measure the speed of send and receive message in the system. Figure 5(a) shows the system message transfer bandwidth with the message size of 128, 256 up to 4096 bytes in single server. Figure 5(b) shows the system message transfer bandwidth with the message size of 128, 256 up to 4096 bytes in Glosim server with 8 nodes.

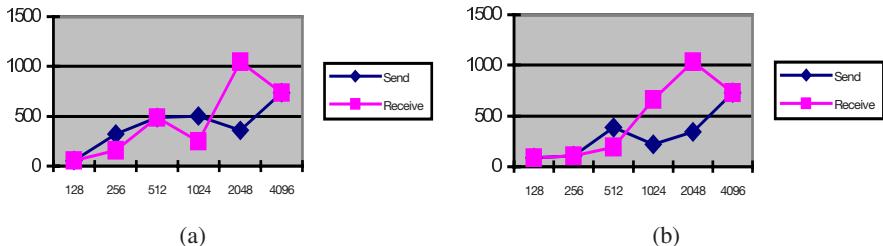


Fig. 5. Speed of send and receive messages for (a) single server (b) Glosim server with 8 nodes

As the data shown in figure 4, Glosim has a normal and acceptable response speed to atom operation of IPC objects and global signal processing mechanism. According to the data shown in the figure 5, in case of low data load (4KB below), msg send/receive bandwidth in Glosim server is a little smaller than that in single server. The results appear to be quite normal and acceptable because Glosim server involves network transfer and the overload of data copy of kernel and user space. Thus, Glosim also has excellent performance support for practical system.

7 Glosim Enabled Applications

In fact, programs based on Glosim can exchange IPC objects just by numerical keys and they can tag each global working process using global process pid. This indicates that there is no need for them to make sure where the corresponding IPC structure lies physically and in which node the practical global working process runs. Glosim system ensures that working processes can find out the needed resources and other working processes only by fixed key values and global pid. These resources and working processes lie in different nodes when running. However, through the logical addressing and global pid addressing of resources, programs can be independent on any physical network characteristics and thus transparently actualize single system image for upper application programs.

Glosim just aims to smoothly migrate Unix programs from traditional single machine environment to cluster environment almost without modification. So, as a traditional Unix program, BBS is quite a typical example which involves different aspects of Unix including network socket, IPC object, signal operation, file I/O operation and so on. What's more, it is very popular and much requires serving efficiency after login. If supported by Glosim, BBS can be smoothly migrated to cluster environment with considerable performance improvement compared with that in single environment, it fully proves that Glosim is of high usability, performance and stability.

Scalable BBS cluster server is a practical application based on Glosim. In order to efficiently extend the traditional BBS with single server to the BBS with cluster server and provide services of high performance, file consistency, balanced load and transparency, we make the design requirement as follows:

- (1) BBS original program based on IPC mechanism in former single environment must be extended to the distributed one.
- (2) Materials in shared memory and file system of each node in cluster must keep consistent.
- (3) Servers can be increased and deleted dynamically or statically.
- (4) Numbers of the registration and the login must increase in proportion to the numbers of servers.
- (5) Transparently support upper applications from system layer almost without modifying user program codes.

Figure 6 shows the max login numbers of BBS in Glosim cluster with different memory settings and different nodes. X-axis indicates the number of nodes, while Y-axis indicates the numbers of people.

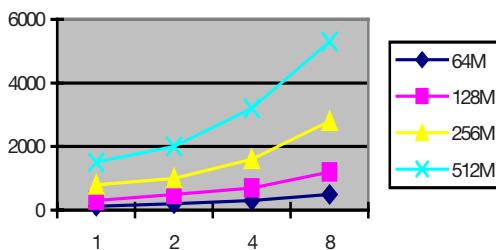


Fig. 6. Max login numbers of BBS in Glosim cluster with different node numbers

8 Conclusions and Future Works

In this paper, we present a novel architecture of a single system image built on cluster system, named Glosim. This system not only supports global IPC objects including message queue, semaphore and shared memory, but also presents a new concept of global working process and provides it SSI support smoothly and transparently. Combined with Linux Virtual Server and SIOS, it completely constructs a high performance SSI cluster network server.

In this case, a great number of applications in single Linux environment such as BBS, Apache and MUD server can be migrated into cluster environment almost unmodified. In this way, single system image service is provided to upper application programs transparently.

Based on Glosim, processes can exchange IPC data through numerical key value. Global process pid is used to identify each global working process, which means it is unnecessary to know where the responding IPC structure physically exists and in which node a global working process physically runs. Logical addressing of resources and global pid addressing make processes independent of physical network and transparently provide single system image to upper application programs.

With all its advantages mentioned, however, Glosim is utilized currently with the disadvantages of relatively high system overhead, fault tolerance to be improved and only indirect support to application program global variables. These are the issues we need to solve in our future plan.

References

- [1] K. Hwang and Z. Xu, *Scalable Parallel Computing: Technology, Architecture, and Programming*, McGraw-Hill, New York, 1998.
- [2] R. Buyya (ed), *High Performance Cluster Computing: Architectures and Systems*, Vol. 1., Prentice Hall, 1999.
- [3] R. Buyya, "Single System Image: Need, Approaches, and Supporting HPC Systems", *Proceedings of Fourth International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'97)*, 1997.
- [4] R. Buyya, T. Cortes, and H. Jin, "Single System Image", *The International Journal of High Performance Computing Applications*, Vol.15, No.2, Summer 2001, pp.124–135.
- [5] B. Walker and D. Steel, "Implementing a full single system image UnixWare cluster: Middleware vs. underware", *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*.
- [6] A. Barak and O. La'adan, "The MOSIX multicomputer operating system for high performance cluster computing", *Future Generation Computer Systems*, 1998.
- [7] MOSIX – Scalable Cluster Computing for Linux, <http://www.cs.huji.ac.il/mosix/>
- [8] The Beowulf Project, <http://www.beowulf.org/>
- [9] BPROC: Beowulf Distributed Process Space, <http://www.beowulf.org/software/bproc.html>
- [10] Y. A Khalidi, J. M Bernabeu, V. Matena, K. Shirriff, and M. Thadani. "Solaris MC: A Multi-Computer OS", *Proceedings of 1996 USENIX Conference*, January 1996.
- [11] Sun Solaris-MC, <http://www.cs.umd.edu/~keleher/dsm.html>
- [12] D. Ghormley, D. Petrou, S. Rodrigues, A. Vahdat, and T. Anderson, "GLUnix: A global layer Unix for a network of workstations", *Software Practice and Experience*, 1998.

- [13] K. Karimi and M. Sharifi, "DIPC: A System Software Solution for Distributed Programming", *Proceedings of Parallel and Distributed Processing Techniques and Applications Conference (PDPTA'97)*, Las Vegas, USA, 1997.
- [14] K. Karimi and M. Sharifi, "DIPC: The Linux Way of Distributed Programming", *Proceedings of the 4th International Linux Conference*, Germany, 1997.
- [15] K. Karimi, M. Schmitz, and M. Sharifi, "DIPC: A Heterogeneous Distributed Programming System", *Proceedings of the Third International Annual Computer Society of Iran Computer Conference (CSICC'97)*, Tehran, Iran, 1997.
- [16] DIPC, <http://www.gpg.com/DIPC/>
- [17] W. Zhang, "Linux virtual servers for scalable network services", *Proceedings of Ottawa Linux Symposium 2000*, Canada.
- [18] K. Hwang, H. Jin, E. Chow, C. L. Wang, and Z. Xu, "Designing SSI clusters with hierarchical checkpointing and single I/O space", *IEEE Concurrency*, 7 (1): 60–69, 1999.
- [19] K. Hwang and H. Jin, "Single I/O space for scalable cluster computing", *Proceedings of 1st International Workshop on Cluster Computing*, 1999, pp.158–166.

Author Index

- Ahmad, Muhammad Bilal 600
Alt, Martin 363
Baek, Jang-Woon 483
Bin, Xuelian 194
Bo, Xiaoming 467
Butler, Anthony 281
Byun, Tae-Young 435, 496
- Cao, Jian 382
Cao, Jiannong 254, 352, 404, 536
Cao, Qiang 167
Cao, Zhiyang 104
Chan, Alvin T.S. 352
Chan, Fan 254
Chan, Y.K. 57
Chang, Min Hyuk 600
Che, Yonggang 226
Chen, Heng 398
Chen, Hongsong 172
Chen, Yu 125
Chen, Zujue 467
Cheng, Hui 404
Choi, Tae-Sun 600
- Dai, Huadong 215, 311, 608
Ding, Kai 249
Dong, Huanqing 140
Dou, Lei 520
Dou, Wenhua 387, 629
Dou, Yong 12
Drew, Steve 619
Du, Bin 373
Du, Xutao 204
Du, Zhihui 125
- Fang, Xinxin 414
Felser, Meik 85
Franck, Andreas 565
Fu, Bo 152
Fu, Changdong 31
- Ge, Jianfang 236
Gerlach, Jens 301
Goel, Sushant 643
- Golm, Michael 85
Gorlatch, Sergei 363
Grosspietsch, K.-E. 57
- Gu, Jian 162
Guo, Li 665
Guo, Weibin 322
Guo, Yufeng 41
Guo, Zhaoli 322
- Han, Jizhong 590
Han, Ki-Jun 435, 496
Han, Weihong 520
Han, Zongfen 665
He, Chuan 373
He, Nangzhong 322
He, Zhenxing 249
Hu, Mingzeng 96, 104, 172
Hu, Xiaofeng 3
Huang, Hua 577
Huang, Min 404
Huang, Zunguo 557
- Jeong, Chang-Won 525
Ji, Zhenzhou 96, 104, 172
Jia, Yan 520
Jiang, Changan 236
Jiang, Hongshan 50
Jiang, Jingfei 509
Jiang, Shengyi 241
Jiang, Yanhuang 608
Jiao, Zhenqiang 125
Jin, Hai 665
Jin, Maozhong 536
Jin, Shiyao 194
Joo, Su-Chong 525
- Kaiser, Jörg 488
Kang, Myoung-Suk 525
Kim, Hyongsuk 265
Kim, Jai-Hoon 478
Kim, Min-Su 435, 496
- Kleinöder, Jürgen 85
Kneis, Joachim 301
Ko, Young-Bae 478

- Lee, Kenneth K.C. 57
 Li, Aijun 577
 Li, Dong 152
 Li, Dongsheng 414
 Li, Jing 352
 Li, Minglu 382
 Li, Qiong 41
 Li, Sanli 373
 Li, Wenlong 109
 Li, Xiaomei 226, 292
 Li, Xiaoming 157
 Li, Xing 430
 Li, Zhanhuai 140
 Li, Zhoujun 204
 Lim, Seungbum 478
 Lin, Haibo 109
 Liu, Changling 488
 Liu, Chao 536
 Liu, Dan 75
 Liu, Fuyan 114
 Liu, Guangming 41
 Liu, Hengzhu 41
 Liu, Jian 136
 Liu, Peng 125
 Liu, Tao 398
 Liu, Xinsong 75
 Liu, Yunhui 577
 Lu, Jianzhuang 147
 Lu, Kai 414
 Lu, Xiaolin 425
 Lu, Xicheng 3, 12, 414
 Lu, Xinda 343
 Luan, Zhongzhi 398
 Luo, Siwei 335, 577
 Luo, Zhigang 292
 Lyu, Michael R. 536

 Ma, Shichao 590
 Mao, Xinjun 189
 Ming, Mei 152

 Na, Sangik 265
 Ni, Xiaoqiang 509

 Oh, Taewan 265

 Park, Jong-An 600
 Park, Sung-Hoon 177

 Qi, Yan 189
 Qi, Zhichang 189
 Qian, Depei 398
 Qian, Xi 136
 Qin, Jie 629
 Qiu, Zhijie 75

 Ren, Zhihong 352

 Seo, Dae-Wha 483
 Sharda, Hema 281, 643
 Shen, Li 147
 Sheridan, Phil 619
 Shi, Baochang 322
 Shi, Shuming 446
 Shim, Young-Suk 600
 Shin, Chang-Sun 525
 Shu, Jiwu 31
 Srinivasan, Bala 655
 Su, Jinshu 3
 Sun, Changai 536
 Sun, Chengzheng 619
 Sun, Haiyan 457, 546
 Sun, Yudong 254
 Sun, Zhigang 3

 Tan, Yusong 582
 Tang, Feilong 382
 Tang, Yuhua 311
 Tang, Zhimin 23
 Tang, Zhizhong 109
 Taniar, David 281, 643, 655
 Tian, Jing 387
 Tian, Jinlan 50

 Venema, Sven 619

 Waluyo, Agustinus Borgy 655
 Wang, Dingxing 446
 Wang, Hui 241
 Wang, Jilong 430
 Wang, K.F. 96
 Wang, Nengchao 322
 Wang, Tao 157
 Wang, Xiaodong 457, 546
 Wang, Xin 189
 Wang, Xingwei 404
 Wang, Yijie 414
 Wang, Zhenghua 226

- Wang, Zhijun 404
Wang, Zhiying 147
Wawersich, Christian 85
Wen, Jinwei 577
Weng, Chuliang 343
Wu, Jianping 430
Wu, Quanyuan 520
Wu, Yongwei 446

Xia, Jun 215
Xiang, Dong 162
Xiao, Mingyan 387
Xiao, Nong 414
Xiao, Zhiting 241
Xie, Changsheng 167
Xie, Jun 125
Xu, Jinhui 634
Xu, Ming 546, 634

Yan, Gongjun 75
Yan, Wei 189
Yang, Guangwen 446
Yang, Shuqiang 629
Yang, Xuejun 215, 311, 608
Yang, Yuhai 194
Yao, Jun 31
Yi, Huizhan 608

Yi, Xiaodong 311
Yoon, Changbae 265
You, Jinyuan 114
Yu, Jin 446
Yu, Shengsheng 162
Yu, Wen 274

Zerbe, Volker 565
Zhang, Gongxuan 236
Zhang, Hongjun 241
Zhang, Jinxiang 430
Zhang, Minxuan 509
Zhang, Suqin 50
Zhang, Xingjun 398
Zhang, Xuebo 292
Zhang, Zhirou 335
Zheng, Gang 249
Zheng, Weimin 31, 136, 274
Zheng, Yanxing 387
Zhong, Lin 162
Zhou, Bin 457
Zhou, Jingli 162
Zhou, Xingming 546, 582
Zhou, Xu 23
Zhu, Ziyu 125
Zou, Peng 457