

Ruixuan Li
Jiannong Cao
Julien Bourgeois (Eds.)

LNCS 7296

Advances in Grid and Pervasive Computing

7th International Conference, GPC 2012
Hong Kong, China, May 2012
Proceedings

 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Ruixuan Li Jiannong Cao
Julien Bourgeois (Eds.)

Advances in Grid and Pervasive Computing

7th International Conference, GPC 2012
Hong Kong, China, May 11-13, 2012
Proceedings

 Springer

Volume Editors

Ruixuan Li

Huazhong University of Science and Technology
School of Computer Science and Technology
1037 Luoyu Road, Wuhan 430074, China
E-mail: rxli@hust.edu.cn

Jiannong Cao

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

Julien Bourgeois

University of Franche-Comte, FEMTO-ST
1 cours Leprince-Ringuet, 25200 Montbéliard, France
E-mail: julien.bourgeois@femto-st.fr

ISSN 0302-9743

e-ISSN 1611-3349

ISBN 978-3-642-30766-9

e-ISBN 978-3-642-30767-6

DOI 10.1007/978-3-642-30767-6

Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012938531

CR Subject Classification (1998): F.2, C.2, H.4, D.2, D.4, C.2.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

© Springer-Verlag Berlin Heidelberg 2012

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. Violations are liable to prosecution under the German Copyright Law.

The use of general descriptive names, registered names, trademarks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

Welcome to the 7th International Conference on Grid and Pervasive Computing, GPC 2012. Established in 2006, GPC has been a major international event in the area of grid, cloud and pervasive computing. This conference provides an international forum for scientists, engineers, and users to exchange and share their experiences, new ideas, and latest research results in all aspects of grid, cloud and pervasive computing systems. The previous GPC conferences were held in Taichung, Taiwan (2006), Paris, France (2007), Kunming, China (2008), Geneva, Switzerland (2009), Hualien, Taiwan (2010), and Oulu, Finland (2011).

The 7th GPC conference, held during May 11–13, 2012, was organized by the Hong Kong Polytechnic University, Hong Kong, China. The conference covers current interests in grid, cloud and pervasive computing. We received 55 submissions from 11 countries. Each manuscript was reviewed by at least three reviewers. We are very thankful to the Technical Program Committee members who helped with the review process. The final conference program consists of carefully selected 9 full papers and 19 short papers.

The conference program included two outstanding keynote talks, six technical sessions, and one tutorial. One workshop was held in conjunction with GPC 2012. We would like to thank the Organizing Committee members, Technical Program Committee members, reviewers, Tutorial Chairs, tutorial presenters, Workshop Chairs, workshop organizers, Workshop Program Committee members, Publicity Co-chairs, Publication Chair, Local Arrangements Chair, Financial Chair, Web Chair, Steering Committee Chair and Steering Committee members for their contributions. We are very grateful for the generous support of the Hong Kong Polytechnic University, and their efforts in organizing the conference.

We believe the participants enjoyed the conference and scientific interactions as well as the traditional atmosphere, beautiful sights and delicious local foods of Hong Kong city.

May 2012

Ruixuan Li
Jiannong Cao
Julien Bourgeois

Organization

General Chairs

Jiannong Cao
Mohan Kumar

Hong Kong Polytechnic University, China
University of Texas at Arlington, USA

Program Chairs

Julien Bourgeois
Ruixuan Li

University of Franche-Comté, France
Huazhong University of Science and
Technology, China

Tutorial Chair

Shui Yu

Deakin University, Australia

Workshop Chair

Zili Shao

Hong Kong Polytechnic University, China

Publication Chair

Kunmei Wen

Huazhong University of Science and
Technology, China

Publicity Chairs

Zhiyong Xu
Wenbin Jiang

Suffolk University, USA
Huazhong University of Science and
Technology, China

Financial Chair

Wilfred Lin

Hong Kong Polytechnic University, China

Web Chair

Chenglin Shen

Huazhong University of Science and
Technology, China

Local Arrangements Chair

Wei Lou Hong Kong Polytechnic University, China

Steering Committee

Hai Jin Huazhong University of Science and
Technology, China (Chair)
Nabil Abdennadher University of Applied Sciences, Western
Switzerland
Christophe Cerin University of Paris XIII, France
Sajal K. Das The University of Texas at Arlington, USA
Jean-Luc Gaudiot University of California - Irvine, USA
Kuan-Ching Li Providence University, Taiwan
Cho-Li Wang The University of Hong Kong, China
Chao-Tung Yang Tunghai University, Taiwan

Program Committee

Jemal Abawajy Deakin University, Australia
Nabil Abdennadher University of Applied Sciences, Switzerland
Luciana Arantes LIP6, France
Junwei Cao Tsinghua University, China
Christophe Cerin Université de Paris XIII, France
Ruay-Shiung Chang National Dong Hwa University, Taiwan
Haibo Chen Fudan University, China
Wenguang Chen Tsinghua University, China
Yeh-Ching Chung National Tsing Hua University, Taiwan
Raphael Couturier LIFC, University of Franche-Comté, France
Eugen Dedu University of Franche-Comté, France
Qianni Deng Shanghai Jiao Tong University, China
Xiaoju Dong Shanghai Jiao Tong University, China
David H.C. Du University of Minnesota, USA
Dan Grigoras University College Cork, Ireland
Weili Han Fudan University, China
Xubin He Virginia Commonwealth University, USA
Michael Hobbs Deakin University, Australia
Hung-Chang Hsiao National Cheng Kung University, Taiwan
Chunming Hu Beihang University, China
Kuo-Chan Huang National Taichung University, Taiwan
Mohamed Jemni ESSTT, Tunisia
Hai Jiang Arkansas State University, USA
Wenbin Jiang Huazhong University of Science and
Technology, China
Yong-Kee Jun Gyeongsang National University, Korea

Kuan-Ching Li	Providence University, Taiwan
Ming-Lu Li	Shanghai Jiang Tong University, China
Tao Li	Nankai University, China
Damon Shing-Min Liu	National Chung Cheng University, Taiwan
Pangfeng Liu	National Taiwan University, Taiwan
Pedro Medeiros	New University of Lisbon, Portugal
Henning Mueller	University of Applied Sciences, Western Switzerland
Philippe Navaux	Federal University of Rio Grande do Sul, Brazil
Mohamed Ould-Khaoua	University of Glasgow, UK
Marcin Paprzycki	IBS PAN and WSM, Poland
Jean-Louis Pazat	IRISA, Rennes, France
Ronald H. Perrott	Queen's University Belfast, UK
Dana Petcu	Western University of Timisoara, Romania
Wasim Raad	King Fahd University of Petroleum and Minerals, Saudi Arabia
Omer F. Rana	Cardiff University, UK
Sanjay Ranka	University of Florida, USA
Liria Matsumoto Sato	New University of Lisbon, Portugal
Haiying Shen	Clemson University, USA
Xuanhua Shi	Huazhong University of Science and Technology, China
Pradip K. Srimani	Clemson University, USA
Chien-Min Wang	Academia Sinica, Taiwan
Cho-Li Wang	The University of Hong Kong, China
Jun Wang	University of Central Florida, USA
Lingyu Wang	Concordia University, Canada
Di Wu	Sun Yat-sen University, China
Jan-Jan Wu	Academia Sinica, Taiwan
Song Wu	Huazhong University of Science and Technology, China
Weigang Wu	Sun Yat-sen University, China
Yulei Wu	University of Bradford, UK
Nong Xiao	National University of Defense Technology, China
Weijun Xiao	University of Minnesota, USA
Zhiyong Xu	Suffolk University, USA
Jingling Xue	University of New South Wales, Australia
Chao-Tung Yang	Tunghai University, Taiwan
Shaowen Yao	Yunnan University, China
Baoliu Ye	Nanjing University, China
Zhiwen Yu	Northwestern Polytechnical University, China
Zhifeng Yun	Louisiana State University, USA
Sherali Zeadally	University of the District of Columbia, USA
Zili Zhang	Southwest University, China
Yanmin Zhu	Shanghai Jiao Tong University, China

Table of Contents

Cloud Computing

From Web Cache to Cloud Cache	1
<i>Thepparit Banditwattanawong</i>	
pCloud: An Adaptive I/O Resource Allocation Algorithm with Revenue Consideration over Public Clouds	16
<i>Jianzong Wang, Yanjun Chen, Daniel Gmach, Changsheng Xie, Jiguang Wan, and Rui Hua</i>	
A Gossip-Based Mutual Exclusion Algorithm for Cloud Environments	31
<i>JongBeom Lim, Kwang-Sik Chung, Sung-Ho Chin, and Heon-Chang Yu</i>	
An Effective Partition Approach for Elastic Application Development on Mobile Cloud Computing	46
<i>Zhuoran Qin, Jixian Zhang, and Xuejie Zhang</i>	
Memory Virtualization for MIPS Processor Based Cloud Server	54
<i>Li Ruan, Huixiang Wang, Limin Xiao, Mingfa Zhu, and Feibo Li</i>	
Implementation of a Distributed Data Storage System with Resource Monitoring on Cloud Computing	64
<i>Chao-Tung Yang, Wen-Chung Shih, and Chih-Lin Huang</i>	

Grid and Service Computing

Design, Verification and Prototyping the Next Generation of Desktop Grid Middleware	74
<i>Leila Abidi, Christophe Cérin, and Kais Klai</i>	
A Request Multiplexing Method Based on Multiple Tenants in SaaS	89
<i>Pingli Gu, Yanlei Shang, Junliang Chen, Bo Cheng, and Yan Jiang</i>	
An Adaptive Design Pattern for Genetic Algorithm-Based Composition of Web Services in Autonomic Computing Systems Using SOA	98
<i>Vishnuvardhan Mannava and T. Ramesh</i>	
Service-Oriented Ontology and Its Evolution	109
<i>Weisen Pan, Shizhan Chen, and Zhiyong Feng</i>	

Green Computing

Energy Efficient Activity Recognition Based on Low Resolution Accelerometer in Smart Phones 122
Yunji Liang, Xingshe Zhou, Zhiwen Yu, Bin Guo, and Yue Yang

Energy Efficient Allocation of Virtual Machines in Cloud Computing Environments Based on Demand Forecast 137
Jian Cao, Yihua Wu, and Minglu Li

Energy Conservative Mobile Cloud Infrastructure 152
Ashok Chandrasekar, Karthik Chandrasekar, Harini Ramasatagopan, and Rafica Abdul Rahim

Power-Constrained Actuator Coordination for Agricultural Sensor Networks 162
Junghoon Lee, Gyung-Leen Park, Ho-Young Kwak, and Jikwang Han

Mobile and Pervasive Computing

Design and Evaluation of Mobile Applications with Full and Partial Offloadings 172
Jennifer Kim

A Cross-Layer Scheme to Improve TCP Performance in Wireless Multi-hop Networks 183
Fu-Quan Zhang and Inwhee Joe

A Fully Abstract View for Local Cause Semantics 198
Jianxin Xue and Xiaoju Dong

Efficiency Considerations in Policy Based Management in Resource Constrained Devices 210
Jignesh Kakkad and Nandan Parameswaran

Agent Based Quality Management Middleware for Context-Aware Pervasive Applications 221
Di Zheng, Jun Wang, and Ke-rong Ben

Scheduling and Performance

A Virtual File System for Streaming Loading of Virtual Software on Windows NT 231
Yabing Cui, Chunming Hu, Tianyu Wo, and Hanwen Wang

TBF: A High-Efficient Query Mechanism in De-duplication Backup System 244
Bin Zhou, Hai Jin, Xia Xie, and PingPeng Yuan

Estimating Deadline-Miss Probabilities of Tasks in Large Distributed Systems	254
<i>Dongping Wang, Bin Gong, and Guoling Zhao</i>	

Global Pricing in Large Scale Computational Markets	264
<i>Lilia Chourou, Ahmed Elleuch, and Mohamed Jemni</i>	

Trust and Security

A New RBAC Based Access Control Model for Cloud Computing	279
<i>Zhuo Tang, Juan Wei, Ahmed Sallam, Kenli Li, and Ruixuan Li</i>	

QoS Monitoring and Dynamic Trust Establishment in the Cloud	289
<i>Ashok Chandrasekar, Karthik Chandrasekar, Malairaja Mahadevan, and P. Varalakshmi</i>	

Multihop-Based Key Management in Hierarchical Wireless Sensor Network	302
<i>Yiyi Zhang, Xiangzhen Li, Yan Zhen, and Ling kang Zeng</i>	

A Bullet-Proof Verification Using Distributed Watchdogs (BPV-DW) to Detect Black Hole Attack in Mobile Ad Hoc Networks	312
<i>Firoz Ahmed, Seok Hoon Yoon, and Hoon Oh</i>	

Performance Analysis for Workflow Management Systems under Role-Based Authorization Control	323
<i>Limin Liu, Ligang He, and Stephen A. Jarvis</i>	

The 2012 International Workshop on Mobile Cloud and Ubiquitous Computing (Mobi-Cloud 2012)

A Medical Image File Accessing System with Virtualization Fault Tolerance on Cloud	338
<i>Chao-Tung Yang, Cheng-Ta Kuo, Wen-Hung Hsu, and Wen-Chung Shih</i>	

Enhanced Password-Based User Authentication Using Smart Phone	350
<i>Inkyung Jeun, Mijin Kim, and Dongho Won</i>	

Development of <i>m</i> -TMS for Trusted Computing in Mobile Cloud	361
<i>Hyun-Woo Kim, Eun-Ha Song, Jun-Ho Kim, Sang Oh Park, and Young-Sik Jeong</i>	

An Efficient Cloud Storage Model for Cloud Computing Environment	370
<i>Hwa Young Jeong and JongHyuk Park</i>	

Author Index	377
-------------------------------	-----

From Web Cache to Cloud Cache

Thepparit Banditwattanawong

Information Science Institute of Sripatum University
Bangkok, Thailand
thepparit.ba@spu.ac.th

Abstract. To run off-premise private cloud, consumer needs budget for public cloud data-out charge. This amount of expenditure can be considerable for data-intensive organization. Deploying web cache can prevent consumer from duplicated data loading out of their private cloud up to some extent. In present existence, however, there is no cache replacement strategy designed specifically for cloud computing. Devising a cache replacement strategy to truly suit cloud computing paradigm requires ground-breaking design perspective. This paper presents a novel cloud cache replacement policy that optimizes cloud data-out charge, the overall responsiveness of data loadings and the scalability of cloud infrastructure. The measurements demonstrate that the proposed policy achieves superior cost-saving, delay-saving and byte-hit ratios against the other well-known web cache replacement policies.

Keywords: Cloud computing, cache replacement policy, contemporaneous proximity, cost-saving ratio, window size.

1 Introduction

More organizations are adopting cloud computing paradigm due to several benefits such as low up-front costs, better ubiquity, increased utilization of computing resources and reduced power consumption. These are enabled by statistical multiplexing and risk transferences of over- and under-provisionings through elasticity [1]. Public cloud providers like Amazon Web Services [2], Google AppEngine [3] and Windows Azure [4] currently offer several pricing criteria for building off-premise private clouds [5]. Those similarly include the volume charges of data loaded outgoing of private clouds down into consumer sites. These charges can be tremendous expenditures to the running costs of private clouds of data-intensive organizations. The significance of this problem can be realized through a realistic scenario where the data is transferred through 1 Gbps Metro Ethernet with 50% bandwidth utilization for 8 work hours a day, and 260 workdays per annum, which is 39 TB per month, would cost \$44,280 per annum based on the Amazon's data-transfer-out pricing data. This is a representative scenario used throughout this paper.

Due to the fact that most of cloud services especially those of SaaS [5] are accessible via HTTP-supported applications such as web browsers and Web OS

[6], cloud data-out charge can be reduced by deploying a caching proxy on a consumer premise. This avoids as many repeated data loadings as possible by letting caching proxy reply succeeding requests for previously loaded data with the data fetched from local cache, unless spoiled, rather than reloaded from cloud.

Nevertheless, acquiring a caching proxy with sufficient space to cache entire data objects from private cloud might be infeasible for consumer organizations because, on one hand, the data-intensive consumers are supposed to export their huge amounts of business data onto their private clouds to truly benefit from cloud computing notion. On the other hand, the overall business data continues to grow with orders of magnitude as modern enterprises increasingly present their business contents in forms of videos, sounds, pictures and other forms of digital contents like electronic publications. Therefore, caching proxy must be equipped with a cache replacement strategy, such as LRU [7], GDSF [8] and LFU-DA [8] that are all supported by the most widely-used web caching software Squid [9]. Cache replacement policies control the provision of enough room inside limited cache spaces on the fly for caching missed objects.

However, there is no cache replacement policies in present existence that has been designed specifically to minimize cloud data-out charge. Additionally, all of the existing policies aim to maximize hit ratio, which means the frequency of serving small data in no time [7, 10, 11] as the first priority. This notion has been evolving with the advancement of broadband communication technologies, which have obviously made the delays (and stability) of the loadings of the small data objects from remote servers no longer distinguishable from those from local caching proxies. In contrast, fetching big objects such as those of multimedia has still kept users experiencing long delays although caching proxy has been in place since the objects have to potentially be retrieved across the network. The latter situation impedes SaaS's content evolution.

The core contributions of this work include: (1) opening up a new design perspective of cache replacement strategy that breaks new ground to suit cloud computing environment, (2) a new performance benchmark, cost-saving ratio, which can be used to capture the economical efficiency of cache replacement policy, (3) a novel replacement policy based on the principle of contemporaneous proximity and optimized for cost-saving, delay-saving and byte-hit ratios to be particularly of use in the era of cloud computing, and (4) a set of comparative measures of strategies in use worldwide including the proposed strategy that gives useful hints for developing more sophisticated cache replacement policies in cloud computing era.

The merits of the proposed policy to the communities of private cloud consumers include the reduction of cloud data-out expenditure and overall speeding up of cloud data loading as well as serving faster large data objects. To both private cloud consumers and public cloud providers, the policy is so network bandwidth friendly that it enables more scalable cloud infrastructure.

2 Proposed Strategy

This section describes the design rationales and practicality analysis of the proposed cache replacement policy.

2.1 Design Rationales

The proposed policy lies itself in two principles, temporal affinity and spatial locality, which are referred to together as contemporaneous proximity [12] for the sake of conciseness. Contemporaneous proximity refers to a time and space property indicating that a particular reference to a certain data object is likely to be repeated in short time (i.e. temporal affinity) and that a set of multiple references to a certain data object tentatively leads to another reference to the same object (i.e. spatial locality). The policy captures the manifest degrees of contemporaneous proximity of data objects by factorizing their recencies and frequencies of accesses.

Considering merely access recency and popularity, however, is unable to satisfy optimal data-out charge reduction in a consistent manner. The policy therefore mandates controlling data-out charge factor by explicitly embracing object sizes and data-out charge rate on a per-object-size basis that altogether accumulates data loading expenditure. It is intuitive that object whose monetary cost of transfer is high, if still usable, should be retained longer in cache to minimize its cost-benefit ratio than inexpensive object.

As another important design facet, shifting into cloud computing paradigm requires that desktop applications be transformed into SaaS model in which requests to the applications are dispatched across the network. This paradigm requirement causes SaaS less responsive as compared to the desktop applications whose requests are received, processed and returned locally. As a result, using SaaS applications encounters network delays that in overall affect organization productivity. To relieve the effect, cache replacement policy for cloud computing ought to parameterize data loading latency in such a way that data object with short loading latency should be replaced before longer one. This allows higher utilizations of slowly loaded objects to improve overall responsiveness.

Next design consideration is time remaining before the expiration of each object. This characteristic is referred to herein as Time-To-Live (TTL). Data objects whose ages have gone nearly or beyond their expirations should be evicted from cache to give space to newly arriving object as the almost stale ones remain lower chances to get referenced than fresher ones.

Based on the above design rationales, the proposed policy works as follows. Whenever available cache space becomes inadequate to store a newly loaded object, the policy formulates a cluster of least recently referenced objects. The number of objects in the cluster is specified by a preset ‘window size’ value. Given the formulated cluster, the policy subsequently seeks out an object with the lowest current profit to give preference for eviction. The profit value associated with each object i is defined as:

algorithm Caching**description** Manipulates hits & misses and calls Cloud**input** $rURL$: requested URL**output** requested object**declare** cd : cache database (hash table with URL keys) ro : requested object, fs : free cache space**begin****if** $((rURL \in cd) \wedge (cd.getObject(rURL)$ not expired)) //if cache hit occurs $ro \leftarrow cd.getObject(rURL)$ $ro.updateFrequency()$ $ro.setProfit(ro.getObjectSize() \times ro.getChargeRate()$ $\times ro.getLoadingLatency() \times ro.getFrequency() \times ro.getTTL())$ $cd.updateObject(ro)$ **else** //if cache miss occursUse $rURL$ to load ro from cloud and initialize its properties $ro.setProfit(ro.getObjectSize() \times ro.getChargeRate()$ $\times ro.getLoadingLatency() \times ro.getFrequency() \times ro.getTTL())$ $fs \leftarrow cd.getFreeSpace()$ **if** $(fs < ro.getObjectSize())$ $Cloud(ro, cd)$ //invoking Cloud policy here $cd.putObject(ro)$ **return** ro **end****algorithm Cloud****description** Implements Cloud replacement policy**input** ro : requested object, cd : cache database (hash table with URL keys)**output** -**declare** rs : required cache space, ws : window size cdq : cache database (recency-keyed min-priority queue) coq : profit-keyed min-priority queue of evictable objects eo : evicted object, fs : free cache space**begin** $rs \leftarrow ro.getObjectSize(), \quad ws \leftarrow$ predetermined value $cdq \leftarrow cd$ //building cdq from cd **if** $(cd.getTotalNumberOfObjects() < ws)$ $ws \leftarrow cd.getTotalNumberOfObjects()$ **for** 1 to ws **do** $coq.addObject(cdq.retrieveLeastRecentlyObject())$ **do** $eo \leftarrow coq.removeMinProfitObject()$ $cd.evict(eo)$ **while** $(eo.getSize() + cd.getFreeSpace()) < rs$ $fs \leftarrow (cd.getFreeSpace() + eo.getSize()) - rs$ $cd.setFreeSpace(fs)$ **end****Fig. 1.** Cloud (below) and related (top) algorithms

$$s_i \times c_i \times l_i \times f_i \times TTL_i$$

where s_i is the size of i , c_i is data-out charge rate in loading i , l_i is latency in loading i , f_i is i 's access frequency, and TTL_i is the TTL of i . If revoked cache space is still not sufficient for the new object, additional objects with least profits are evicted in order. Note that cache miss on a highly profitable object imposes more penalty in terms of technical and/or economical efficiencies than a low profitable one.

The proposed policy is entitled 'Cloud' to imply its intended application domain. One possible algorithm solving the problem in choosing object(s) for eviction according to Cloud policy is shown in Fig. 1 together with a caller algorithm. It should be realized that in practice data-out charge rates for all objects to be loaded from clouds are preconfigured values provided by cloud providers from which the objects are loaded [2-4], while TTL values can be calculated from the values of 'Expires' or 'max-age' fields available inside HTTP message headers [13].

2.2 Practicality

With respect to the time complexity analysis of the algorithm of Cloud illustrated in Fig. 1, the statements that take significant part in processing time are: building the priority queue cdq from cd is traditionally $O(N \log N)$ where N is the number of data objects in a cache; the for loop takes $O(N \log N)$ as the window size can be set to as many as N while adding each object into coq is $O(\log N)$; the do loop has the worst-case running time of $O(N \log N)$ because the number of evicted objects is bounded by N , while removing each object from coq takes $O(\log N)$; deleting an object from the hash table cd is less significant and thus neglected. The other statements are all identically $O(1)$. Therefore, the algorithm is $O(N \log N)$. In other words, Cloud strategy can be implemented with an algorithm whose worst-case running time is guaranteed to be practical.

3 Performance Evaluation

This section describes the simulation configuration followed by comparative performance results and discussion of Cloud policy as well as another three popular policies: LRU, GDSF and LFU-DA, which have been supported by Squid caching proxy.

3.1 Input Data Sets

HTTP trace-driven simulation technique has been used for performance measurements. Provided by IRCache project [14], raw trace files are various in sizes and have been collected from three caching proxy servers located in Boulder (BO), Silicon Valley (SV) and New York (NY).

Each of the raw traces contains the stream of requests to large numbers of various HTTP domains. In order to emulate realistic HTTP accesses occurring

on private cloud(s) of a single midsize organization where totally 50 domains are running on the cloud(s), the traces have been preprocessed by counting up top 50 popular domains, and only the requests to these domains have been extracted into three new trace files. As a remark, the number 50 is the approximation of the number of domains administrated by the author’s university.

As the other part of preprocessing, unused fields have been removed and an expiration field has been added to every record of every trace to be used to compute TTL values. Expiration field values have been figured out based on three following assumptions. First, an object expired right before its size changed as appeared in a trace. Second, as long as its size was constant, an object’s lifespan was extended to its last request appearing in a request stream. Finally, an object appearing only once throughout a trace expired right after the only its use seen in a trace.

Table 1. Characteristics of each of the simulated traces

Traces	BO	SV	NY
Total requests	205,226	441,084	599,097
Requested bytes	2,401,517,003	7,113,486,583	4,712,041,132
Unique objects	70,944	248,508	158,552
Max. total bytes of unique objects	694,759,006	1,065,863,067	1,323,954,264

Table 1 summarizes the basic characteristics of the preprocessed traces. The ‘Total requests’ designates the total number of records contained in each trace as the results of top 50 domain filterings. The ‘Requested bytes’ is the total size of requested objects appearing in each trace. The ‘Unique objects’ represents the number of unique URLs appearing in each trace. As some unique objects had their sizes changed from time to time, by considering only their largest sizes, the ‘Max. total bytes of unique objects’ indicates minimum cache sizes without cache replacement at all (equivalent to infinite cache sizes).

3.2 Performance Metrics

The three traditional performance metrics, hit rate, byte-hit rate, delay-saving ratio, and the newly proposed economical performance metric ‘cost-saving ratio’ have been used. For an object i ,

$$\text{cost-saving ratio} = \sum_{i=1}^n c_i s_i h_i / \sum_{i=1}^n c_i s_i r_i$$

where c_i is the data-out charge rate of i , s_i is the size of i , h_i is how many times a valid copy of i is found in a cache, and r_i is the total number of requests to i .

This study has aimed for the best cost-saving ratio, delay-saving ratio and byte-hit rate, respectively, except hit ratio as justified in Sect. 1. Whilst it is clear why using cost-saving ratio, delay-saving ratio captures how responsive

SaaS would be in overall as the result of a certain cache replacement policy; byte-hit rate captures how good each particular policy foster cloud infrastructure's scalability by reducing as many total bytes transmitted across the network as possible.

3.3 Cost Models

For critical business such as hospital and stock trading, it is not acceptable to experience cloud downtimes and bottlenecks. Consumer organization of this kind must establish continuity plan by implementing private cloud running on more than one independent public cloud to achieve fault tolerance and load balancing. As a consequence, if public cloud providers offer different data transfer prices, objects of the same size loaded from different providers will have different monetary costs.

To realize this practice, the simulations have been conducted based on two cost models. One is uniform cost model where a single data-out charge rate is applied to organization who rents its private cloud from single public cloud provider. The rate of Amazon S3's, which is \$0.117997 per GB by average (for the total amount of data transfer out between 11 to 51 TB per month in the US region as of August 2011), has been used in this model. (The range of 11 to 51 TB per month can cover the realistic scenario demonstrated in Sect. 11) The other is nonuniform cost model, which employs dual charge rates to emulate situation where organization implements its private cloud(s) rented from a pair of independent public cloud providers. The rates used in the latter model are those of Amazon S3's \$0.117997 per GB and Windows Azure's \$0.15 per GB (for data transfers from North American locations as of August 2011). The simulator has associated the dual charge rates with unique objects found throughout the traces in an interleaving manner.

3.4 Window Sizes

Since data objects in different communities of interests manifest different degrees of contemporaneous proximity, it is not sensible to assume that any recency- and/or frequency-based policy performing perfectly in one environment will perform well against any other environments or even the same environment in different time periods. The control parameter window size is thus engaged to allow the fine tuning of Cloud policy to be adaptive and perform fairly well in any real working environments. In addition to the levels of contemporaneous proximity exhibiting in each workload, the superior value of window size is affected by cache size: a series of pre-experiments have shown that the larger the absolute cache size, the larger the optimal window size. Table 2 presents a set of fine-tuned window sizes (and relative ones inside the parentheses) used in the simulations against each workload and cache size regardless of the cost models. The simulated cache sizes are presented in percents of the maximum total bytes of unique objects belonging to each workload. At 100% cache size, there is no replacement at all, thus all the

policies yield the same upper-bound performance results in all metrics. The right-most column provides absolute cache sizes in relation to those of BO workload that can be all used in conjunction with the percent cache sizes as a guideline to tune up optimal window sizes in other target environments.

Table 2. Optimal window sizes used in simulations of Cloud policy

Workloads	Simulated cache sizes (% of Max. total bytes of unique objects)			Relative absolute cache sizes
	10%	20%	30%	
	BO	215(1.00,1.00)	625(1.00,2.91)	
SV	425(1.98,1.00)	1175(1.88,2.76)	1200(1.78,2.82)	1.53
NY	700(3.26,1.00)	1550(2.48,2.21)	1575(2.33,2.25)	1.91

3.5 Empirical Results

The simulation results of Cloud and the other three policies (LRU, GDSF and LFU-DA) are compared in this section. As for GDSF, its particular version called GDSF-Hits (whose cost parameter is equal to 1 for all objects) has been employed. It should also be noted that, unlike some previous works, uncacheable requests have not been excluded from the simulated traces to reflect actual performance ones can really gain from utilizing those certain replacement policies; the caching efficiencies of all the simulated policies would otherwise be improved in all the performance metrics but spurious.

Fig. 2 shows the economical performances rendered by using cost-saving ratio metric. The following findings can be drawn.

- As the main achievement of this study, it can be seen that Cloud has most economized among the other examined policies at all the investigated cache sizes, cost models and workloads (exceptions have lain in 10% cache size of BO workload where LFU-DA has outperformed Cloud slightly by about 0.14% of the Cloud’s for both cost models). To realize the merit of Cloud policy implied by its superior performance, the cost-saving ratio of Cloud at 30% cache size of NY workload in the uniform cost model, when applied to the representative scenario in Sect. 4 can significantly save up to \$10,569 per annum. Cloud could even save up to \$427.26 annually, more than GDSF when using 10% cache size based on SV workload and the uniform cost.
- The cost-saving performances of LRU and LFU-DA have been closely alike, whereas GDSF has performed worst. This is because only GDSF chooses big objects to be replaced at first. To facilitate economical comparisons, 0.001 margin of the cost-saving ratios can be translated as \$44.28 per annum as of the representative scenario.

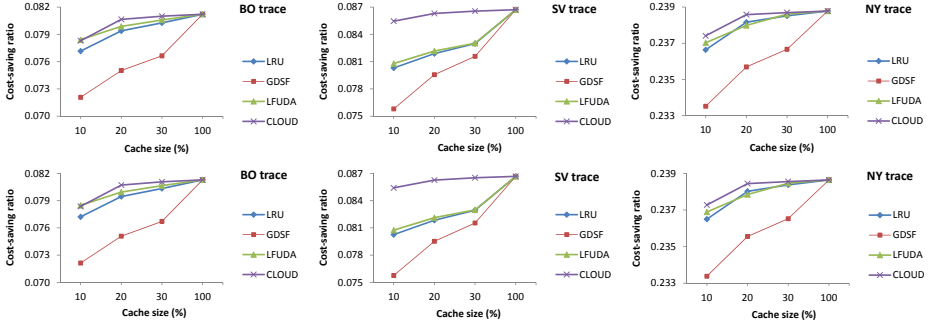


Fig. 2. Comparisons of cost-saving ratios using uniform cost (top row) and nonuniform cost (bottom row)

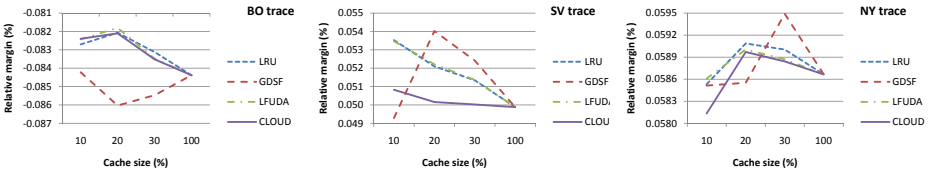


Fig. 3. Comparisons of relative margins of the cost-saving ratios of the uniform cost above the nonuniform one

- For both cost models and all the examined workloads, Cloud has produced the outstanding steady states of cost-saving ratios across the broad range of cache sizes when allocated beyond 20%.
- With the simulated values of data-out charge rates, the cost-saving performance gaps between the uniform and nonuniform cost models have come out subtle and thus magnified in Fig. 3. The relative margins are the cost-saving ratios of uniform cost deducted by those of nonuniform ones in percents of those of the nonuniform costs. The figure has demonstrated that the cost-saving performances of all the policies using the uniform cost model can be slightly better in SV and NY workloads and worse in BO workload than those in the nonuniform cost model. Further observation on these margins will be presented numerically at the end of this section.

With respect to delay saving, the simulation results are portrayed in Fig. 4. The findings from the results are as follows.

- For 20% or larger cache size, Cloud has achieved the best overall responsiveness of data loadings among the others since Cloud has considered retaining slowly loaded objects. This is the minor accomplishment of this work. To translate a merit implied by Cloud policy’s superior performance, the delay-saving ratio of Cloud at 30% cache size using BO workload with the uniform

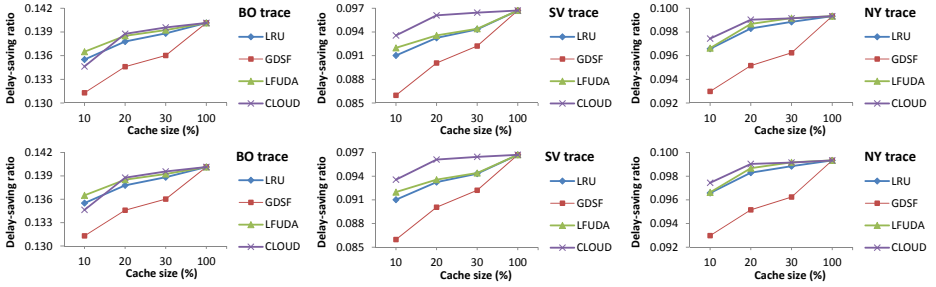


Fig. 4. Comparisons of delay-saving ratios using uniform cost (top row) and nonuniform cost (bottom row)

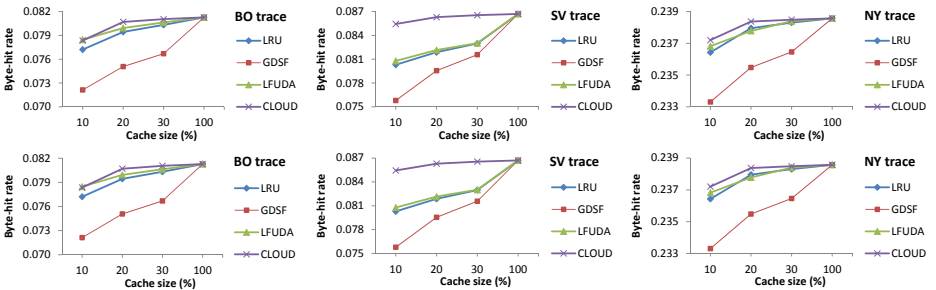


Fig. 5. Comparisons of byte-hit rates using uniform cost (top row) and nonuniform cost (bottom row)

cost, when applied to the representative scenario can significantly save up to around 290 work hours per annum.

- LRU and LFU-DA have delivered similar delay-saving performances, whereas GDSF has saved least total delays. This is because GDSF evicts bigger objects, which generally impose longer loading latencies.
- When cache sizes have been beyond 20% for both cost models and all the examined workloads, Cloud has delivered the most steady delay-saving ratios.
- The differences of delay-saving performances under the same workload between the different cost models have not been recognizable through the ranges of studied cache sizes. Further numerical observation on these differences will be presented at the end of this section.

Fig. 5 demonstrates the byte-hit performances with the following findings.

- Cloud has saved the largest volume of data transfers among the other policies across all the simulated cache sizes, cost models and workloads (exceptions have lain in 10% cache size with the BO workload where LFU-DA has outperformed Cloud slightly by about 0.14% of the Cloud’s for both cost

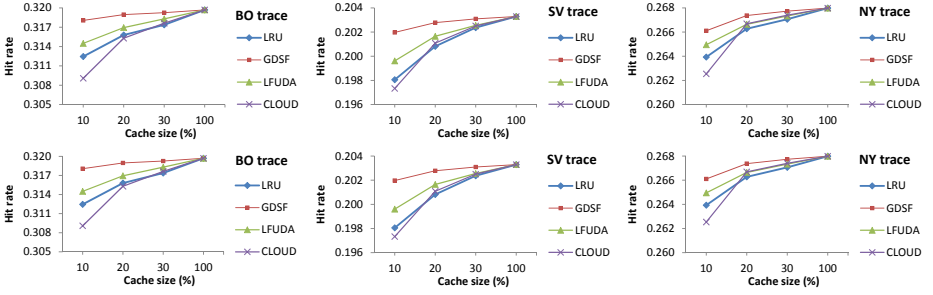


Fig. 6. Comparisons of hit rates using uniform cost (top row) and nonuniform cost (bottom row)

models). This achievement arises because Cloud policy favors large objects to be retained in cache.

- LRU and LFU-DA have delivered similar byte-hit performances, whereas GDSF has performed worst. This is partly because only GDSF evicts bigger objects at first.
- For both cost models and all the examined traces, when cache sizes have grown beyond 20%, Cloud has produced more stable byte-hit rates than the other policies.
- The differences of byte-hit performances of the same workload between the different cost models have not been noticeable through the ranges of investigated cache sizes. Further observation on these differences in terms of numerical data will be presented at the end of this section.

In terms of hit rates, the performances are illustrated in Fig. 6. The below findings have been reached.

- Cloud’s performance has been reasonably worst at 10% cache sizes but quickly increased and better than LRU in most other cases and even better than LFU-DA in some cases. This phenomenon can be generally clarified by the fact that a strategy evicting larger objects is optimized for hit rate [7, 10, 11], the opposite applies to Cloud strategy as it tends to retain larger objects in cache.
- Though worst in all previous metrics, GDSF has outperformed all the other policies in hit rate metric. This finding can be explained by the same reason as in the above finding.
- The differences of hit rates of the same workload between the different cost models have not been discernible via the ranges of simulated cache sizes. Further observation on these differences will be presented in the next paragraph.

In a big picture, the following facts have been inferred.

- Since data transfer costs are proportional to object sizes, the cost-saving performances have shown the same growth rates as those of the byte-hit performances meaning that ones can save both data-out costs and network bandwidths simultaneously of the same order of magnitude regardless of utilized policy.
- By looking at Fig. 2, Fig. 5 and Fig. 6 together, the policies that have given higher ratios of cost-saving or byte-hit have tended towards lower hit rates. This behavioral trade-off reinforces the finding that strategy revoking cache space from bigger objects for smaller ones is good at hit rate but poor at byte-hit ratio [7, 10, 11] (and cost-saving ratio).
- Further experiment has revealed that the performance gaps of all kinds of metrics between the uniform and nonuniform cost models will become more noticeable over the wider range of charge rates: using the nonuniform costs of \$0.117997 and \$1.17997 instead of \$0.117997 and \$0.15 at 20% cache size under the NY workload, Cloud has delivered the cost-saving, delay-saving, byte-hit and hit rates of 0.00026097, 0.00000348, -0.00000057 and 0.00000334, respectively, lower than those of the uniform cost.
- In terms of cost savings, delay savings and byte hits, Cloud with optimal window sizes running on 20% or more cache size has delivered almost steady-state performance for both cost models as if it was running with an infinite cache size. Therefore, Cloud policy can be characterized by graceful degradation as it has continued to deliver the best performances over the differently constrained cache sizes.

4 Related Work

4.1 Object Sizes, Loading Costs and Access Frequencies

A number of policies surveyed in [7]: LRU, LFU-DA, EXP1, Value-Aging, HLRU, LFU, LFU-Aging, α -Aging, swLFU, SLFU, Generational Replacement, LRU*, LRU-Hot, Server-assisted cache replacement, LR, RAND, LRU-C, Randomized replacement with general value functions, including policies ARC [15], CSOPT [16], LA2U [17], LAUD [17], SEMALRU [18] and LRU-SLFR [19] have not parameterized object sizes. If big objects were requested frequently but often evicted by these policies (as blind to object sizes), caching proxy would have to frequently reload the big objects from their original servers. Therefore, object-size uncontrollable scheme permits unnecessarily poor cost-saving ratios.

Another group of policies surveyed in [7]: GDSF, LRU-Threshold, LRU-Min, SIZE, LOG2-SIZE, PSS, LRU-LSC, Partitioned Caching, HYPER-G, CSS, LRU-SP, GD-Size, GD*, TSP, MIX, HYBRID, LNC-R-W3, LRV, LUV, HARMONIC, LAT, GDSP, LRU-S, including LNC-R-W3-U [20], SE [21], R-LPV [22], MinSAUD [23], OPT [24], LPPB-R [25], OA [26], CSP [27] and GA-Based Cache Replacement Policy [28] have considered object sizes in such a way that replacing bigger objects first, thus not aiming for cost-saving performance. The other policies M-Metric [7], NNPCR-2 [29] and Bolot and Hoschka's [30] have favored bigger objects like Cloud. In particular, M-Metric allows bigger objects to stay

longer in cache but does not support loading cost parameter; NNPCR-2 applied neural network to decide the evictions of small or big objects but does not embed cost parameter; Bolot and Hoschka's policy replaces bigger objects first but ignores spatial locality by not considering access frequencies and does not support nonuniform costs.

4.2 Access Recencies

All known policies have prioritized the recencies of object references either implicitly or explicitly. By implicitly, every policy always accepts a newly loaded missing object (i.e., the most recently used object) into cache rather than rejects it. By explicitly, several policies such as LRU, LRU-Threshold, SIZE, LRU-Min, EXP1, Value-Aging, HLRU, PSS, LRU-LSC and Partitioned Caching have parameterized elapsed times since the last requests to objects. Cloud policy has explicitly regarded the recency property of objects in its model.

4.3 Object Loading Latencies

Several policies: GD-Size, GDSF, GD*, GDSP, HYBRID, LAT, LUV, MIX, LNC-R-W3, LNC-R-W3-U, LRU-SLFR and GDSP have taken object loading latencies into account. All of them have replaced objects with shorter latencies first. Cloud policy also follows such a design approach.

4.4 Object Expirations

Very rare policy considers object expiration. LA2U, LAUD and LNC-R-W3-U have replaced frequently updated objects first. The former two have not described how update frequencies are derived. The latter has estimated update frequencies from changes detected in HTTP's 'Last-Modified' header fields; however, if frequently updated objects are seldom requested, updated 'Last-Modified' values will be rarely perceived by policies and update frequencies will be then underestimated. This problem can be solved by using explicit expiration times or TTL as in Bolot and Hoschka's policy even though this parameter has not yet been implemented in their empirical

5 Conclusion

This paper addresses an economical and technical perspective from which a new cache replacement policy must be devised specifically for cloud computing era. A simple and efficient policy, Cloud, is proposed. The Cloud's efficiencies in terms of cost-saving, delay-saving and byte-hit ratios except hit ratios (which are justifiable) are found fairly outperforming all the other investigated policies at most cache sizes.

A concrete finding from this study is that if most recently used objects with large sizes, costly charge rates, long loading latencies, high access frequencies, and long lifespans last longer in cache in a profit-inside-recency-window manner, cost-saving, delay-saving and byte-hit performances will be greatly improved.

Left as future work, to compare Cloud policy with others by using both technical and economical performance metrics based on longer traces is challenging and requires considerable effort. Also, we have been planning to conduct a future research to analyze static and dynamic factors as well as their interrelationship to help determine the optimal values of window sizes for a given environment that are dynamically and timely adjusted according to workload evolution.

Acknowledgment. This research is financially supported by Sripatum University. The author would like to thank Duane Wessels, National Science Foundation (grants NCR-9616602 and NCR-9521745) and the National Laboratory for Applied Network Research for the trace data used in this study.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A Berkeley view of cloud computing. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28 (February 2009), <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html>
2. Amazon.com, Inc., Amazon web services (2011), <http://aws.amazon.com/s3/>
3. Google Inc., Google app engine (2011), <http://code.google.com/intl/en/appengine/>
4. Microsoft, Windows azure (2011), <http://www.microsoft.com/windowsazure/>
5. Mell, P., Grance, T.: The NIST definition of cloud computing (draft): Recommendations of the national institute of standards and technology. NIST Special Publication 800-145 (Draft) (2011), http://csrc.nist.gov/publications/drafts/800-145/Draft-SP-800-145_cloud-definition.pdf
6. Wright, A.: Ready for a web os? Commun. ACM 52, 16–17 (2009)
7. Podlipnig, S., Böszörmenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. 35, 374–398 (2003)
8. Arlitt, M., Cherkasova, L., Dille, J., Friedrich, R., Jin, T.: Evaluating content management techniques for web proxy caches. SIGMETRICS Perform. Eval. Rev. 27, 3–11 (2000)
9. Wessels, D.: Squid: The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol (2004)
10. Abrams, M., Standridge, C.R., Abdulla, G., Fox, E.A., Williams, S.: Removal policies in network caches for world-wide web documents. SIGCOMM Comput. Commun. Rev. 26, 293–305 (1996)
11. Balamash, A., Krunz, M.: An overview of web caching replacement algorithms. IEEE Communications Surveys and Tutorials 6(1-4), 44–56 (2004)
12. Banditwattanawong, T., Hidaka, S., Washizaki, H., Maruyama, K.: Optimization of program loading by object class clustering. IEEJ Transactions on Electrical and Electronic Engineering 1 (2006)

13. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Rfc 2616, hypertext transfer protocol – http/1.1, United States (1999)
14. National Laboratory for Applied Network Research, Weekly squid http access logs, <http://www.ircache.net/>
15. Megiddo, N., Modha, D.S.: Outperforming lru with an adaptive replacement cache algorithm. *Computer* 37, 58–65 (2004)
16. Jeong, J., Dubois, M.: Cache replacement algorithms with nonuniform miss costs. *IEEE Transactions on Computers* 55, 353–365 (2006)
17. Chen, H., Xiao, Y., Shen, X.S.: Update-based cache access and replacement in wireless data access. *IEEE Transactions on Mobile Computing* 5, 1734–1748 (2006)
18. Geetha, K., Gounden, N.A., Monikandan, S.: Semalru: An implementation of modified web cache replacement algorithm. In: NaBIC, pp. 1406–1410. IEEE (2009)
19. Shin, S.-W., Kim, K.-Y., Jang, J.-S.: Lru based small latency first replacement (slfr) algorithm for the proxy cache. In: Proceedings of the 2003 IEEE/WIC International Conference on Web Intelligence, WI 2003, pp. 499–502. IEEE Computer Society, Washington, DC (2003)
20. Shim, J., Scheuermann, P., Vingralek, R.: Proxy cache algorithms: Design, implementation, and performance. *IEEE Transactions on Knowledge and Data Engineering* 11, 549–562 (1999)
21. Sarma, A.R., Govindarajan, R.: An Efficient Web Cache Replacement Policy. In: Pinkston, T.M., Prasanna, V.K. (eds.) HiPC 2003. LNCS (LNAI), vol. 2913, pp. 12–22. Springer, Heidelberg (2003)
22. Chand, N., Joshi, R., Misra, M.: Data profit based cache replacement in mobile environment. In: 2006 IFIP International Conference on Wireless and Optical Communications Networks, p. 5 (2006)
23. Xu, J., Hu, Q., Lee, W.-C., Lee, D.L.: Performance evaluation of an optimal cache replacement policy for wireless data dissemination. *IEEE Transactions on Knowledge and Data Engineering* 16, 125–139 (2004)
24. Yin, L., Cao, G., Cai, Y.: A generalized target-driven cache replacement policy for mobile environments. In: IEEE/IPSJ International Symposium on Applications and the Internet, p. 14 (2003)
25. Kim, K., Park, D.: Least popularity-per-byte replacement algorithm for a proxy cache. In: Intl. Conf. on Parallel and Distributed Systems, p. 0780 (2001)
26. Li, K., Nanya, T., Qu, W.: A minimal access cost-based multimedia object replacement algorithm. In: International Symposium on Parallel and Distributed Processing, p. 275 (2007)
27. Triantafillou, P., Aekaterinidis, I.: Web proxy cache replacement: Do’s, don’ts, and expectations. In: IEEE International Symposium on Network Computing and Applications, p. 59 (2003)
28. Chen, Y., Li, Z.-Z., Wang, Z.-W.: A ga-based cache replacement policy. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 1, pp. 263–266 (August 2004)
29. El Aarag, H., Romano, S.: Improvement of the neural network proxy cache replacement strategy. In: Proceedings of the 2009 Spring Simulation Multiconference, SpringSim 2009, Society for Computer Simulation International, San Diego (2009)
30. Bolot, J.-C., Hoschka, P.: Performance engineering of the world wide web: application to dimensioning and cache design. *Computer Networks and ISDN Systems* 28, 1397–1405 (1996)

pCloud: An Adaptive I/O Resource Allocation Algorithm with Revenue Consideration over Public Clouds

Jianzong Wang^{1,2}, Yanjun Chen¹, Daniel Gmach³, Changsheng Xie^{1,2,*}
Jiguang Wan^{1,2}, and Rui Hua¹

¹ School of Computer Science, Huazhong University of Science and Technology

² Wuhan National Laboratory for Optoelectronics, Wuhan, China

³ HP Labs, Palo Alto, CA, USA

cs_xie@mail.hust.edu.cn

Abstract. Cloud-based services are emerging as an economical and convenient alternative for clients who don't want to acquire, maintain and operate their own IT equipment. Instead, customers purchase virtual machines (VMs) with certain Service Level Objectives (SLOs) to obtain computational resources. Existing algorithms for memory and CPU allocation are inadequate for I/O allocation, especially in clustered storage infrastructures where storage is distributed across multiple storage nodes. This paper focuses on: (1) dynamic SLO decomposition so VMs receive proper I/O service in each distributed storage node, and (2) efficient and robust local I/O scheduling strategy. To address these issues, we present pCloud, an adaptive I/O resource allocation algorithm that at runtime adjusts local SLOs. The local SLOs are generated for each VM at each storage node based on access patterns. We also adopt dual clocks in pCloud to allow automatic switching between two scheduling strategies. When system capacity is sufficient, pCloud interweaves requests in an earliest deadline first (EDF) manner. Otherwise resources are allocated proportionate to their normalized revenues. The results of our experiments suggest that pCloud is adaptive to various access patterns without significant manual pre-settings while maximizing profits.

Keywords: Cloud Computing, Cloud Storage, I/O Scheduling, Service-Level Objectives, Revenue Maximization.

1 Introduction

1.1 Background

The trend towards cloud services backed up by server virtualization has granted greater significance to workload consolidation. Generally virtualized hosts run multiple virtual machines (VMs) that share the resources of the underlying

* Corresponding author.

physical hosts. Further, servers are grouped in pools sharing a centralized storage system to ease data exchange. The complexity of I/O resource allocation and I/O scheduling in virtualized cloud environments presents a new set of challenges.

Basically, although each VM has the illusion that it possesses dedicated physical resources, its performance is subject to I/O competition since the aggregate throughput of a storage node is limited. Unlike CPU and memory allocation, I/O allocation is vulnerable to the bursty nature of workloads and fluctuations of available capacity. Such unpredictability requires I/O allocation algorithms to accomplish two tasks: (1) providing robust isolation and (2) achieving efficient I/O scheduling.

To address these issues, SLOs are selected to measure the provided service quality. Existing algorithms [4, 6–8] can be divided into algorithms that provide proportional allocation of I/O resources and algorithms that focus on providing latency controls based on arrival curve. The general idea of proportional allocation of I/O resources is to divide the resources at a fine granularity in proportion to client weight, and the SLOs are assigned in terms of weight. Some variants, for example, [6], adopt more complicated tagging mechanisms to achieve other control such as resource reservation and usage limits. The second group of algorithms focuses on providing latency control based on arrival curve [4, 7, 8]. These regulate the arrival rates of workloads with a leaky bucket (σ, ρ), where σ stands for maximum burst size and ρ for average throughput. In addition, individual requests are guaranteed a maximum response time provided the workload satisfies stipulated constraints on burst size and arrival rate.

1.2 Motivation Example

In cloud-based distributed storage systems, however, the per-VM requirements should be further segmented to indicate the local demands in the participating nodes. Take the two VMs sketched in Table 1 for example. Assuming the first four requests of each VM arrive at $t=0$ ms and the remaining arrive at $t=500$ ms. All requests should be completed within 500ms. Then the global throughput requirements of the two VMs are 8 IOPS (I/O per second) each. VM2 sends alternating requests to the two nodes and consequently requires 4 IOPS in either node, which sums up to its global requirements. However, since VM1 first requests node 1 and subsequently node 2, either node needs to complete four requests within 500ms. That corresponds to 8 IOPS for both nodes for VM1. Consequently, the sum of local reserved throughput is 16 IOPS, suggesting that more capacity should be provisioned to meet the requirements.

Table 1. Example: Decomposed SLO Demands Cautious Provision

VM	Requests Flow							
VM_1	s_1	s_1	s_1	s_1	s_2	s_2	s_2	s_2
VM_2	s_1	s_2	s_1	s_2	s_1	s_2	s_1	s_2

Generally, a global SLO is accomplished in shares by all contributing storage nodes rather than completed as a whole. How to configure the sub-SLOs, namely the SLOs provided for a VM from a per-node point of view, is quite challenging. Some existing algorithms assign fix sub-SLOs to VMs to guarantee the performance. However, fixed decomposition of global SLOs may lead to undesirable outcomes: (1) a time-consuming analysis of the characteristics of workloads beforehand to make proper sub-SLOs, (2) low utilization since VMs are not granted priority based on their demand, and (3) difficulties in online management such as data migration and strategy making due to inaccurate provisioning. Hence a more flexible approach towards SLO decomposition which allocates resources on demand is an urgent need.

Also, as computational resources are traded like utilities such as electricity [3, 5, 10], market-driven characteristics need to be considered in for resource scheduling especially when I/O contention is fierce and service of the most value shall be considered first. Existing algorithms such as [4, 6-8] use either weight or latency alone when evaluating overall priority. However, they fail to balance revenue against provided service quality and decide for the service provider which service could bring in more profits.

1.3 Contributions and Paper Organization

In this paper, we propose a novel algorithm for I/O allocation, pCloud, and make the following contributions:

- (1) pCloud on-line models the access patterns of VMs and dynamically generates local SLO specifications in distributed nodes with little communication cost.
- (2) It achieves high utilization by scheduling in an earliest deadline first manner and VMs can use spare capacity without being penalized.
- (3) The local scheduler employs an alternative strategy to maximize revenue when the system is under provisioned where requests are scheduled in descending order of their normalized revenues.

The remainder of the paper is organized as follows. In Section 2 we introduce the trade-offs in I/O scheduling and propose the scheduling goal of pCloud. Section 3 presents the pCloud algorithm in detail. A detailed performance evaluation using diverse configurations is presented in Section 4. Section 5 presents related work and compares pCloud with existing approaches. Finally, Section 6 concludes the work and shows some directions for future work

2 Trade-offs and Scheduling Goals

To provide robust isolation, proportional allocation algorithms arrange the requests in such a cautious way that all the VMs are serviced at a certain ratio

within an arbitrary interval. Such fairness, however, fails to recognize the urgency of requests, and lacks flexibility and may not achieve ideal capacity utilization. On the other hand, latency sensitive algorithms work well by scheduling urgent requests first when the system is well provisioned while insufficient provisioning may lead to unfairness.

Service providers intend to make the utmost utilization of their infrastructures to maximize profits. Besides services with fixed prices, public Infrastructure-as-a-Service (IaaS) providers like Amazon EC2 [3] have implemented new types of market-driven services. Spare capacity is gathered and provided for Amazon EC2 Spot Instances. The price for a spot instance depends on the availability of unused resources. Clients bid for spot instances and higher bids get favored. Some recent work [5, 10] proposed the dynamic resource allocation for spot markets in clouds. A placement strategy is made based on historical records such as market price and VM lifecycle. Revenues can be maximized by solving certain optimization problems. Balancing revenue against cost, we can more accurately evaluate the priority of a request from a service provider’s aspect.

Our algorithm, pCloud, aims at utilizing the full capacity while avoiding the undesirable outcomes of priority based scheduling on latency alone. First, it employs a leaky bucket model to shape request flows. The arrival rates of each flow are constrained by two leaky bucket parameters: the maximum burst size and the average throughput. Additionally, it records the maximum allowable latency and employs EDF scheduling when there are sufficient I/O resources.

Meanwhile, given the price paid for a VM, we balance the expected revenue against the required service. We then schedule requests according to their normalized revenue, i.e., the expected revenue per unit service, to maximize revenue when demand exceeds supply.

Further, requests from one VM can be mapped to different storage nodes. In this case, sub-SLOs, or local SLO assignment must be dynamically configured in order to deal with the variability of workloads. pCloud explores a generic and adaptive approach towards SLO decomposition that helps reduce redundant provisioning.

3 Adaptive SLO Decomposition and Scheduling Algorithm

Figure 1 shows the proposed architecture of pCloud. The coordinator is implemented in hosts where multiple VMs may work simultaneously. The I/O requests from VMs are analyzed by the coordinator and then mapped to different storage nodes. At the coordinator, statistics are appended to each I/O request to help individual storage nodes learn the access patterns. The arrival sequence and the statistics are extracted and processed in individual nodes for Access Pattern Modeling (APM). During this phase, the APM module analyzes the statistics and decides (1) the portion of service the local node serves for that VM historically, and (2) the request rate from that VM to the node recently. Combining the two factors, the APM module divides the global SLO into smaller pieces

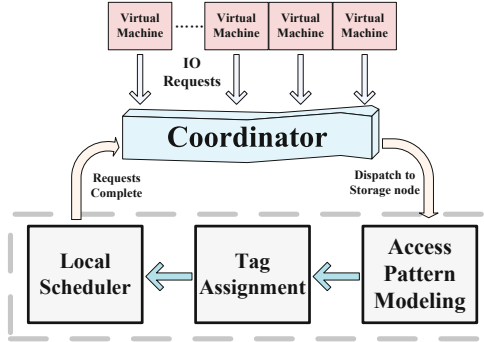


Fig. 1. The Proposed pCloud Architecture

that reflect the reserved service in each node. After that, the tagging module tags requests according to their sub-SLOs. Finally, requests are queued, waiting to be selected by the Local Scheduler.

3.1 Task Characteristics

The requests from an active VM form a flow when they are mapped to a storage node. One VM may send requests to various nodes and form one flow in each node. In the meanwhile, each node maintains multiple flows of requests that come from diverse VMs and always chooses among the first requests of flows to service (so that the sequence of requests from each VM would not be violated). The sub-SLOs of a flow are represented by a triple (σ, ρ, δ) , where σ denotes maximum burst size, ρ average throughput and δ maximum allowable latency. Within an arbitrary interval $[T_1, T_2]$, a VM issues up to $(\sigma + \rho * (T_2 - T_1))$ requests to the node. The number of available tokens is initially σ and increases at the rate of ρ . Every time a request arrives, it takes one token so the number of available tokens decreases. It is always capped at σ to ensure that the available burst size is limited. When tokens are used up the following requests are treated as if they arrived at a later time to avoid some VMs consuming too much resources. After being assigned tokens, requests start queuing, waiting to be scheduled.

We define three states of a flow: idle, pending, and backlogged. Idle means the queue length is zero. Pending suggests there are requests in the queue to be scheduled; however, none of them has violated its deadline requirement. If a flow has at least one request that missed its deadline, we assume it backlogged. By checking the status of flows, we can infer the provisioning level for proper adjustment.

3.2 Access Pattern Modeling and SLO Decomposition

Each VM_i is assigned a global SLO, and the SLO specifications are attached to the request the first time a VM is activated on new nodes. After that, we have to decide the service each VM receives locally. Local shares should be on-line

adaptive to meet the variability of workloads. Take a particular VM, VM_i , for example. Let $(\sigma_i, \rho_i, \delta_i)$ denote the global SLO, and $(\sigma_i^j, \rho_i^j, \delta_i^j)$ the local SLO for VM_i in node n_j . δ_i^j must equal δ_i in terms of worst-case latency. σ_i^j should remain identical to σ_i since a larger burst allows more flexibility in dealing with bursty workloads. Hence our goal is to decompose global average throughput ρ_i properly.

We define:

$$\sigma_i^j = \sigma_i, \rho_i^j = \mu_i^j * \rho_i, \delta_i^j = \delta_i \quad (1)$$

where μ_i^j is the portion of global SLO that is allocated locally in n_j after decomposition.

To figure out the exact value of μ_i^j , we further split it into two parts: historical share and recent behavior. Let H_i^j denotes the historical share, R_i^j recent share, and α the contribution factor H_i^j holds. Then we can model μ_i^j as a linear combination of H_i^j and R_i^j as:

$$\mu_i^j = \alpha * H_i^j + (1 - \alpha) * R_i^j \quad (2)$$

If H_i and R_i are precisely calculated for all nodes, namely:

$$\sum_j H_i^j = 1, \sum_j R_i^j = 1 \quad (3)$$

we may get from (1) (2) (3):

$$\begin{aligned} \sum_j \mu_i^j &= \sum_j (\alpha * H_i^j + (1 - \alpha) * R_i^j) \\ &= \alpha * \sum_j H_i^j + (1 - \alpha) * R_i^j = 1 \end{aligned} \quad (4)$$

and that is to say, all shares in distributed nodes just add up.

There are several reasons why we make the local share μ_i^j a linear combination. Firstly, compared to machine-learning based approaches such as neural networks, it's light-weighted and easy to implement, and even in the worst case, α can be configured as a parameter on a per-VM basis. Secondly, as is proven above, since all the shares distributed to the participating nodes add up to 1, a VM will theoretically have all of its SLO met with great flexibility. Thirdly, this function captures both long-term trends and recent behavior, which is essential for prediction.

In order to make corresponding nodes aware of the service one VM deserves locally, we let the coordinator keep a running count of two integers, C_i^t and D_i^t , which denote the total number of completed and dispatched requests from VM_i as of time t respectively. These two integers are attached to each request. In the storage end, the local scheduler in each node n_j counts the number of requests from VM_i that have completed service locally, denoted by S_i^t . Hence the historical share and recent share are computed as follows:

$$H_i^j = \frac{S_i^j}{C_i^t}(a), R_i^j = \frac{1}{D_i^t - D_i^{j,t-1}}(b) \quad (5)$$

Intuitively, equation (5)(a) means among all C_i^t completed requests, S_i^j are completed locally. And equation (5)(b) implies that VM_i has issued $D_i^t - D_i^{j,t-1}$ requests since n_j received the last request from VM_i , and only the latest one is mapped to this node.

With (1), (2) and (5), we finally get:

$$\rho_i^j = \mu_i^j * \rho_i = (\alpha * \frac{S_i^j}{C_i^t} + (1 - \alpha) * \frac{1}{D_i^t - D_i^{j,t-1}}) * \rho_i \quad (6)$$

Practically, since both H_i^j and R_i^j are scaled down by factors that are less than 1, a VM may receive insufficient allocation locally if either H_i^j or R_i^j is small. To avoid this, we set two thresholds for μ_i^j . We set μ_i^j to the lower threshold if μ_i^j is below it. And if μ_i^j exceeds the upper threshold, we assume that VM_i is hot spotting in one node, and thus allow it a full SLO locally for the instant. Finally, a large burst size σ_i^j that equals the global one would serve as a buffer and smoothen the transition during the establishment of a reliable historical share. Moreover, since the difference between D_i^t and C_i^t becomes insignificant as the base increases, it is possible to forward C_i^t only instead of both.

3.3 Normalized Revenue

We evaluate the importance of requests by comparing the predicted revenues with operating cost. Given a SLO and the corresponding bidding price, we distinguish the requests in terms of normalized revenue, i.e., the revenue gained per unit of service. The detailed amount of service reserved for VM_i is stipulated by the global SLO $(\sigma_i, \rho_i, \delta_i)$. We examine an arbitrary interval $[t, t + \delta_i]$, and the cost of throughput is:

$$GlobalCost = \frac{\sigma_i + \rho_i * \delta_i}{\delta_i} = \frac{\sigma_i}{\delta_i} + \rho_i \quad (7)$$

and μ_i^j of the throughput is now allocated for this node n_j and the local cost for performing is:

$$LocalCost = \mu_i^j * GlobalCost = \frac{\mu_i^j * \sigma_i}{\delta_i} + \mu_i^j * \rho_i \quad (8)$$

Given a bidding price Revenue, we get the normalized revenue:

$$NormRevenue = \frac{Revenue}{LocalCost} = \frac{price}{\mu_i^j * (\sigma_i + \rho_i * \delta_i)} \quad (9)$$

This equation suggests the principles in revenue based allocation: loosened deadlines, less capacity requirements and higher bidding price lead to higher priority.

3.4 Scheduling Framework

As with most existing algorithms [8, 4, 7, 6] we have discussed, pCloud uses tagging to mark the priority of requests. Each request receives a start tag, a finish tag and a priority tag when they arrive. Generally, requests that fall within the SLO constraints are assigned start tag equal to their arrival time. Otherwise they will be assigned larger time stamps to make the adjusted flow compliant with SLO constraints. The finish tags are the sum of the corresponding start tags and maximum allowable latency.

Table 2. Symbols Used and Their Description

Symbol	Meaning
$STag_r$	Start tag of request r
$FTag_r$	Finish tag of request r
$PTag_r$	Priority tag of request r
$MinS_i$	The minimum start tag of requests in flow $flow_i$

The high level description of pCloud is shown in Algorithm 1. A formal description of every component is presented in Algorithm 2. The related symbols are listed in Table 2. The functions description and process flows of different modules in Algorithm[1,2] are introduced as below:

APM: Using the Access Pattern Modeling presented earlier, the scheduler generates the new local throughput.

UpdateToken: Before a request is assigned tags, the number of available tokens must be checked. Using the new local throughput parameter generated in APM, we update the number of tokens.

AdjustTags: Two goals are achieved in tag adjustment phase. First, we detect the flows that had utilized spare capacity. This is indicated by a minimum start tag that lags current time. If there are such flows, their minimum start tags are adjusted to current time, preventing them from being penalized in the future. Second, if a request arrives in an idle flow, we grant it equal chances to be scheduled. This is done by adjusting all the priority tags with an identical shift so that the minimum priority tags of all the flows starts from current time.

AssignTags: pCloud algorithm assigns three tags to arriving requests, start tags, finish tags, and priority tags. Normally, start tags record the arrival time of requests. Otherwise, it will be delayed to a later time to align with the arrival curve when no tokens are available. The finish tag is always the sum of start time and maximum allowable latency, indicating the deadline of a request. The priority tag is spaced by the normalized revenue or the difference in arrival time, whichever is larger. This prevents bursty requests from having similar priority tags.

Algorithm 1. The Algorithm of pCloud

RequestArrival:

- 1: APM();
- 2: UpdateToken();
- 3: AdjustTags();
- 4: AssignTags();

RequestScheduling:

- 1: **if** more than p percent flows are backlogged **then**
 - 2: Schedule the request with minimum priority tag PTag;
 - 3: **else**
 - 4: Schedule the request with minimum finish tag FTag;
 - 5: Let the scheduled request be request r from $flow_i$;
 - 6: Adjust the PTags of requests from $flow_i$, so that the minimum PTag=now
 - 7: **end if**
-

Algorithm 2. The Components' Description of pCloud Algorithm

APM():

- 1: Perform Access Pattern Modeling and generate token parameters;

UpdateToken():

- 1: Let $\Delta token$ be the time difference between current time and last bucket update;
- 2: $tokennum += \rho_i * \Delta token$;
- 3: **if** $tokennum > \delta_i$ **then**
- 4: $tokennum = \delta_i$;
- 5: **end if**

AdjustTags():

- 1: Let request be r from $flow_k$ at time t ;
- 2: **for all** for each pending flow with $MinS_i > t$ **do**
- 3: adjust all the STags and FTags from $flow_i$ so that $MinS_i$ equals now
- 4: **end for**
- 5: **if** $flow_k$ is idle **then**
- 6: Adjust the PTags from all flows so that the minimum PTag equals now
- 7: **end if**

AssignTags():

- 1: Let the request from flow k arrives at time t , Δt_r between this request and the last one;
 - 2: **if** $flow_k$ is idle **then**
 - 3: STag=PTag= t ;
 - 4: **else**
 - 5: $PTag_r = PTag_{r-1} + \max\{\Delta t_r, \frac{1}{NormRevenue}\}$;
 - 6: **if** $tokennum < 1$ **then**
 - 7: $STag_r = STag_{r-1} + \frac{1}{\rho_i}$;
 - 8: **else**
 - 9: $STag_r = STag_{r-1} + \Delta t_r$;
 - 10: **end if**
 - 11: **end if**
 - 12: $FTag_r = STag_r + \delta_i$;
 - 13: $tokennum = 1$;
-

ScheduleRequest: The scheduling policy depends on the provisioning level. If the portion of flows that are currently backlogged exceeds a threshold, we infer that the capacity runs insufficient. The scheduler will then try to schedule the most valuable requests according to their priority tags. EDF scheduling is adopted otherwise.

4 Experiment and Analysis

4.1 Testbed Setup

In this section, we present the results of pCloud implemented in a distributed storage system. We implemented pCloud in an event-driven system simulator DiskSim [1]. The system simulator simulates the behavior of all pCloud components at higher level. At device level, the simulator schedules requests and issues them to DiskSim modules to obtain I/O access details. We chose a configuration with two devices representing two storage nodes. The first is a RAID-0 array backed by ten *Seagate Cheetah* disks and the second by four Cheetah disks. We evaluated the performance with two traces, OLTP and WebSearch, both from Umass Trace Repository [2]. The traces include anonymous I/O access records obtained from real services. OLTP and WebSearch traces have several parallel units running, 19 and 3, respectively. Each unit can be regarded as a process running in a VM. To simulate the access to distributed storages, we manually partition the processes in two groups and map them to either node 1 or node 2. For the OLTP trace, the first unit is mapped to node 1 and the remaining 18 to node 2. For WebSearch trace, the first 2 units are dispatched to node 1, and the remaining one accesses node 2. Since the original traces are big, we extract a section (around 90 seconds) of the traces for better clarity.

The global arrival rates of both traces are shown in Figure 2. Both curves are spiky and the global SLOs assigned to them are (50, 150, 300) and (70, 150, 1000). The average size of requests is 16KB for WebSearch and 6KB for OLTP. The arrival rates of decomposed traces in the two nodes are presented in Fig. 3(a) and (b). From the figures we can see that OLTP keeps a small amount of requests mapped to node 1 and the share of service allocated in node

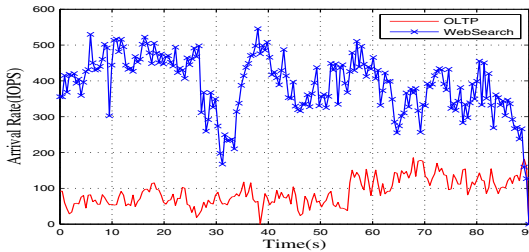


Fig. 2. The Global Arrival Curves

2 increases over time. For the WebSearch trace, the two flows in both nodes have similar arrival curves while the global SLO is roughly divided in a ratio of 2:1. Using such configurations, we simulated an over provisioned environment in node 1 and a thin provisioned one in node 2 to illustrate the properties of pCloud in both provision levels. We also implemented FCFS(First Come First Serve) and pClock [4], an EDF algorithm with excellent performance, as comparison to demonstrate the strengths of our pCloud. For simplicity, we assume all the storage nodes apply the same alpha values in APM.

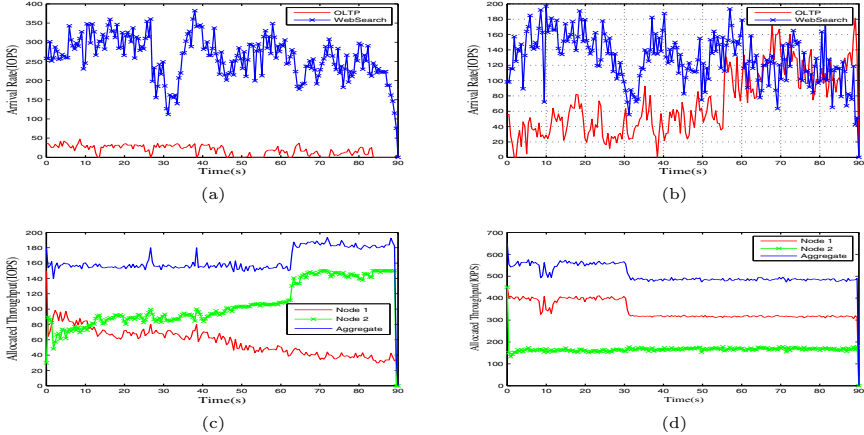


Fig. 3. Arrival rates and SLO decomposition: (a) Arrival rates in node 1; (b) Arrival rates in node 2; (c) Local SLOs for OLTP; (d) Local SLOs for WebSearch

4.2 SLO Decomposition

First we evaluate the effectiveness of Access Pattern Modeling. We set α to 0.80, the lower threshold 0.2 and the upper one 0.75, and Fig. 3(c) and (d) present the sub-SLOs assigned by APM modules in each node.

Initially, the sub-SLOs in both figures are inaccurate because the historical records are not well-established. First we analyze the results of OLTP, as shown in Fig. 3(c). Although OLTP is assigned a higher sub-SLO in node 1 which decreases over time and finally reaches a proper level only at the end of the trace, the sub-SLO assigned in node 2 never fails to meet the demands in node 2. Since $t=62s$, the sub-SLO assigned to OLTP in node 2 takes a leap and becomes equal to the global SLO for OLTP, because the calculated share exceeds the threshold and thus node 2 is assumed to be a hot node for OLTP. The aggregate SLO assigned for OLTP exceeds the global requirements since then, but the extra portion is limited. For the WebSearch trace as show in Fig. 3(d), the sub-SLO curves are relatively steady because the service is divided in a ratio of 2:1 roughly, as can be inferred from Fig. 3(a) and 3(b). In general, both the curves exhibit the trends of real access behaviors as statistics accumulate and adapt quickly.

We then altered the value of α , the two thresholds and the results complied with the underlying principles: (1) The curves of sub-SLOs get smoother as alpha increases because the bursty nature is deeper masked by a smaller factor. (2) The SLO assignment is more cautious with higher upper threshold, because the conditions in which a local node is granted with the full global SLO are limited. (3) With a smaller lower threshold, fewer services are reserved in each node and the minimum share is always the lower bound.

When α is set to 0.80, the upper threshold to 0.8 and the lower threshold to 0.2, the results are similar to this one but the over allocated parts are mostly eliminated due to more cautious decisions. Comparing the sub-SLOs generated by Access Pattern Modeling with the actual arrival rates, we may conclude that APM can adaptively adjust the sub-SLO assignment in each node in accordance with the requirements.

4.3 Latency Control

To evaluate the performance of latency control, we implemented pClock [4] and FCFS as comparison. Since pClock requires manual settings regarding sub-SLOs, we referred to the actual arrival rates and the results of SLO decomposition presented previously (Fig. 3). We then assigned SLOs as presented in Table 3. Fig. 4 show the latency observed by both traces in each node. The bidding prices here for pCloud are (4,2);

Table 3. The Configurations of pClock

VM	NODE	Sub-SLO	VM	NODE	Sub-SLO
OLTP	Node 1	50,30,300	WebSearch	Node 1	70,300,1000
OLTP	Node 2	50,120,300	WebSearch	Node 2	70,150,1000

We can observe that there is little different between the performance of the three algorithms in over-provisioned environment. However, in thin-provisioned environment, pCloud demonstrates the best performance among the three. This suggests that pCloud is capable of achieving high utilization level by adopting EDF scheduling. Moreover, pCloud achieves such utilization in an intelligent way that global SLOs are adaptively split into smaller pieces and the demands in each node are met.

4.4 Revenue Based Allocation

The intuition behind the scheduling strategies in under provisioned environments is to serve the requests in descending order of their normalized revenue. We used the same configurations for pClock as we mentioned in the previous section (Table 3) for consistency. Further, as FCFS and pClock lack mechanisms regarding revenue-based priority, the results of service remain unchanged with different bidding prices.

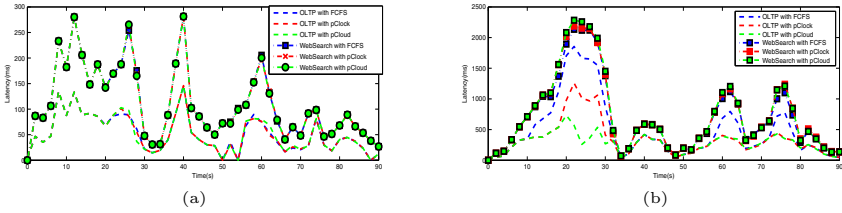


Fig. 4. Latency comparison:(a) Node 1(over-provisioned);(b) Node 2(thin-provisioned)

Fig. 4 shows the latencies observed by OLTP and WebSearch with three scheduling algorithms implemented in node 2, respectively. As shown in the figures, for pClock and FCFS, the two flows soon became backlogged after they started and both suffered severe latencies. The system began to catch up with the requests at $t=30s$ where the arrival rates of both VMs had a significant drop. At $t=50s$, all the backlogged requests were completed and the latencies for requests were low. However, as the arrival rates of both flows rose again, latencies fluctuated when the capacity were merely able to finish most requests in time.

When using pCloud, WebSearch never missed its deadline while OLTP observed only a tiny increase in the latencies from $t=20s$ to $t=30s$. Within this backlogged period, pCloud detected that WebSearch was more profitable and thus granted WebSearch more service. Although the continuous backlogged period lasted for only about 10s, the difference between the performance of pCloud and pClock is enormous. Further, we altered the bidding prices to (1, 3), (2, 2) and (3, 2), respectively. For better clarity, we take the results of bidding prices (1,3) and (4, 2) for example, as shown in Fig. 5. Apparently, the latencies of requests from OLTP were reduced and WebSearch was less favored when OLTP offered a higher bid.

We estimate the aggregate revenue as price multiplied by the percent of requests that meet their deadlines. Table 4 presents the revenues for both VMs in node 2 with the three scheduling algorithms. We omit the revenues in node 1 because that node is over provisioned and the results for all three algorithms are almost identical. The results from Table 3 suggest that pCloud can increase

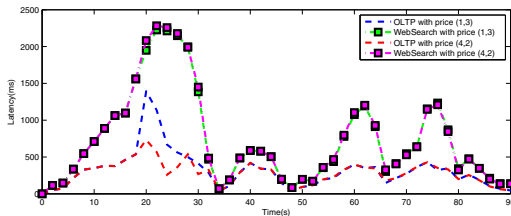


Fig. 5. Latency Comparison of pCloud with Different Bidding Prices

Table 4. Completion Rates and Total Revenue

Pricing	Revenue			Pricing	Revenue		
VM	FCFS	pClock	pCloud	VM	FCFS	pClock	pCloud
(1, 3)	2.656	2.697	2.705	(3, 2)	2.996	3.257	3.323
(2, 2)	2.471	2.632	2.675	(4, 2)	3.522	3.883	3.972

revenues by up to 10%. Given that this node is under-provisioned for only a small fraction of time(10%), such improvement is very impressive.

5 Related Works

Existing algorithms for I/O resource allocation involves several scenarios. Some works [5, 10] analyzed the market-driven characteristics of Amazon EC2 Spot Markets [3] from the cloud service provider’s point of view. The price and demand curves of different VM types are predicted by analyzing historical records. The predicted price and demand curves are then used to solve optimization problems so that total revenue is maximized. PESTO [9], from a different aspect, models both workloads and storage systems to achieve ideal placement of workloads and load balancing. Some algorithms, including Avatar [8], mClock [6], pClock [4], Nested QoS [7], etc, aims at providing robust scheduling mechanisms and dealing with storage-specific issues on a lower level. mClock [4] provides proportional allocation subject to a minimum reservation for each VM. Although VMs are isolated, the proportional allocation algorithms fail to regulate the behavior of competition and cannot achieve full utilization. More importantly, the local settings for each VM in the distributed version of mClock are preset and fixed, and can’t adapt to the variability of workloads. pClock [4] and Nested QoS [7] are latency sensitive algorithms that schedule requests based on latency requirements. pClock achieves high utilization and VMs can use spare capacity without being penalized. Nested QoS controls and sets several levels of SLOs for a VM to reduce the capacity requirements of workloads. The highest level is scheduled in an EDF manner, and requests in the lower levels are scheduled with best effort whenever there is spare capacity. The drawback of this algorithm is that the sequence of requests is altered so that it may result in storage-specific issues such as inconsistency. Moreover, both of these two latency sensitive algorithms are not implemented in distributed environments and overlook the market factors in scheduling. Compared with the algorithms above, pCloud is the first light-weighted and adaptive distributed algorithm for I/O resource allocation.

6 Conclusions and Future Works

This work presents a dynamic I/O resource allocation algorithm, pCloud over cloud environment. The SLO of a VM is expressed as a maximum burst size, an average throughput and a worst-case allowable latency. When a VM is assigned a global SLO, its requests may be mapped to different storage nodes when the

service is providing. A key feature of pCloud is that it dynamically decomposes the global SLO for a VM based on Access Pattern Modeling into smaller pieces that meet the demands in each node. pCloud also adopts auto switching between two scheduling strategies under different provision levels. We have demonstrated that pCloud achieves high utilization in over provisioned systems and maximizes revenue even under thin provisioned environment. Our future works include exploring how application-level characteristics interact with SLO assignment and how to employ similar modeling in other scenarios such as load balancing in clouds, edibility in scheduling and mostly features latency control.

Acknowledgement. We thank the anonymous reviewers of GPC for their feedback on previous versions of this paper. This Project supported by the National Basic Research Program (973) of China (No. 2011CB302303), the National Natural Science Foundation of China (No. 60933002), the Natural Science Foundation of Hubei province (NO. 2010CDB01605), the HUST Fund under Grant (Nos.2011QN053 and 2011QN032), the Fundamental Research Funds for the Central Universities.

References

1. The disksim simulation environment (version 4.0), <http://www.pdl.cmu.edu/DiskSim/>
2. Storage Performance Council (Umass Trace Repository), <http://traces.cs.umass.edu/index.php/Storage/>
3. Amazon Elastic Compute Cloud (Amazon EC2), <http://aws.amazon.com/ec2/>
4. Gulati, A., Merchant, A., Varman, P.: pClock: an arrival curve based approach for QoS in shared storage systems. In: International Conference on Measurement and Modeling of Computer Systems, pp. 13–24 (2007)
5. Zhang, Q., Zhu, Q., Boutaba, R.: Dynamic resource allocation for spot markets in cloud computing environments. In: UCC 2011 (2011)
6. Gulati, A., Merchant, A., Varman, P.: mClock: Handling throughput variability for hypervisor IO scheduling. In: 9th USENIX Symposium on Operating Systems Design and Implementation (October 2010)
7. Wang, H., Varman, P.: Nested QoS: providing flexible performance in shared IO environment. In: USENIX 3rd Workshop on IO Virtualization (June 2011)
8. Zhang, J., Sivasubramaniam, A., Wang, Q., Riska, A., Riedel, E.: Storage performance virtualization via throughput and latency control. ACM Transactions on Storage, TOS (August 2006)
9. Gulati, A., Shanmuganathan, G., Ahmad, I., Waldspurger, C.A., Uysal, M.: Pesto: online storage performance management in virtualized datacenters. In: SOCC 2011 (2011)
10. Zhang, Q., Grses, E., Boutaba, R., Xiao, J.: Dynamic resource allocation for spot markets in clouds. In: Hot-ICE 2011 (2011)

A Gossip-Based Mutual Exclusion Algorithm for Cloud Environments*

JongBeom Lim¹, Kwang-Sik Chung², Sung-Ho Chin³, and Heon-Chang Yu^{1,**}

¹ Department of Computer Science Education, Korea University

² Department of Computer Science, Korea National Open University

³ Software Platform Laboratory, CTO Division, LG Electronics, Seoul, Korea
{jblim, yuhc}@korea.ac.kr, kchung0825@knou.ac.kr,
sunghochin@gmail.com

Abstract. Mutual exclusion is a salient feature in distributed computing systems whereby concurrent access of processors to a shared resource is granted in a mutually exclusive manner. The primary aim of this study is to investigate the use of a gossip protocol to a mutual exclusion algorithm to cope with scalability and fault-tolerance problems which are fundamental issues for cloud computing systems. In this paper we present a gossip-based mutual exclusion algorithm for cloud computing systems with dynamic membership behavior. The correctness proof of the algorithm is provided. The amortized message complexity of our algorithm is $O(n)$, where n is the number of nodes. Simulation results show that our proposed algorithm is attractive regarding dynamic membership behavior.

Keywords: Mutual Exclusion, Gossip-based Algorithm, Cloud Computing.

1 Introduction

Mutual exclusion algorithms are used in distributed systems ensuring no simultaneous access of shared resources or data, by pieces of program code also known as critical sections (CSs), is granted. Many researchers have suggested ways of implementing mutual exclusion algorithms in distributed systems [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. These studies can be categorized into three basic approaches for distributed mutual exclusion algorithms: 1) Token-based approach, 2) Timestamp-based approach and 3) Quorum-based approach.

In the token-based mutual exclusion algorithms [1, 2, 3, 4], executing the critical section can be done by a process that holds the unique token that cannot be presented more than one process at a given time. Timestamp-based mutual exclusion algorithms

* This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (No. 2011-0026210).

** Corresponding author.

[5, 6, 7, 8] require some number of message exchanging rounds among processes to decide which process can be in the critical section next. In the quorum-based approach [3, 9, 10], any process wishing to execute the critical section requests consent from a subset of processes called a quorum. Because of intersection and minimality properties, at least one process receives more than one request for permission for concurrent requests; the process gives a response to one of the requesting processes to make sure only one process can execute the critical section at any time.

More recently, a group mutual exclusion (GME) problem, an extension of the basic mutual exclusion problem, has been proposed [11]. In the group mutual exclusion problem, every critical section is associated with a particular type or a group. Processes belonging to the same group can execute a critical section concurrently, whereas processes belonging to the different groups must execute the critical section in a mutually exclusive way.

A system that mutual exclusion algorithms can be used in is the cloud computing system where constituent nodes can be easily added to or removed from with dynamic behavior due to loosely-coupled environments. However, although much research for the mutual exclusion problem in recent years mainly focuses on reducing message complexity, little attention has been paid to the aforementioned dynamic behavior. Most of the studies assumed that the system does not change anymore without considering node failures and joining which are vital aspects in cloud computing environments that should not be dismissed.

Recently, gossip-based algorithms have received much attention due to its inherent scalable and fault-tolerant properties, which offer additional benefits in distributed systems [12]. Correctness of a gossip-based protocol is presented in [13, 14]. In gossip-based algorithms, each node maintains some number of neighbors called a partial view. With this partial view, at each cycle (round), every node in the system selects f (fanout) number of nodes at random and then communicates using one of the following ways: 1) Push, 2) Pull, and 3) Push-pull mode. Gossip-based algorithms guarantee message delivery to all nodes with high probability and their variation can be found in [15, 16, 17, 18, 19]. Applications of gossip-based algorithms include message dissemination, failure detection services, data aggregation etc.

In this paper, we take the timestamp-based approach for the basic mutual exclusion problem based on the gossip-based algorithm. We conjecture that using the gossip-based algorithm for the basic mutual exclusion problem is a desired approach to deal with scalability and dynamic behavior in cloud computing systems. Having partial view in the gossip-based algorithm is the essential key to achieve the scalability issue. In other words, each node does not have to maintain all the nodes in the system, but the small number of nodes.

The rest of the paper is organized as follows. We present the system model and formally describe the mutual exclusion problem in Section 2. Section 3 provides our gossip-based mutual exclusion algorithm with proof. Simulation results for the algorithm and their interpretation are given in Section 4; this section also analyzes the message complexity. Finally, Section 5 gives our conclusions.

2 Model and Problem Specifications

2.1 System Model

We assume that the cloud computing infrastructure consists of numerous nodes of resources, and individual nodes process arbitrary programs to achieve a common goal. Because of the absence of shared memory, each process or node should communicate with other nodes only by message passing through a set of channels. In addition, we assume that all channels are reliable and FIFO (first-in, first-out), meaning all messages within a channel are received in the order they are sent to. And the message delay is bounded. There is no global clock. However, it is assumed that each node synchronizes its time by gossiping with other nodes. This approach has been justified by [20]. Furthermore, the communication model is asynchronous. In other words, a sender does not have to wait for acknowledgements of receivers (non-blocking). When we consider the dynamic scenario regarding node failures, the fail-stop model is conceived, that is, other processes can learn about whether processes are failed.

2.2 Specifications of the Problem

Mutual Exclusion is a fundamental problem in distributed systems; it is not an exception in the cloud computing systems. With a mutual exclusion algorithm, shared resources or data can be accessed in a consistent way, allowing only one process to execute the critical section (CS). Because there is no shared memory, shared variables such as semaphores cannot be used to implement a mutual exclusion algorithm. In this regard, message passing is the only way to deal with the mutual exclusion problem satisfying following properties:

- **Safety:** Two or more processes are not allowed to execute the critical section simultaneously; only one process can execute the critical section at any given time.
- **Liveness:** Every request for the critical section is eventually granted in the finite time.
- **Fairness:** Concurrent requests for the critical section must be granted in the order they were sent to.

Formal descriptions of above three properties are as follows (It is noted that in order to help more intuitive understand, the properties are expressed from a process point of view):

$$\forall i,j \exists P_i, P_j \neg \exists t_u [P_i \neq P_j : i, j \in \{1 \dots n\} \Rightarrow cs(P_i, t_u) \wedge cs(P_j, t_u)] \text{ (safety)}$$

where $cs(P_i, t_u)$ means that process P_i is executing the critical section at time t_u .

$$\forall P_i, t_u \exists t_v [req_cs(P_i, t_u) \Rightarrow cs(P_i, t_v) \wedge t_u < t_v] \text{ (liveness)}$$

where $req_cs(P_i, t_u)$ means that process P_i has requested the critical section at time t_u .

$$\forall i, j \exists P_i, P_j, t_u, t_v \neg \exists t_w, t_x [req_cs(P_i, t_u) \wedge req_cs(P_j, t_v) \wedge (t_u < t_v) \Rightarrow cs(P_i, t_w) \wedge cs(P_j, t_x) \wedge (t_w > t_x)] \text{ (fairness)}$$

It is noted that that the safety property must be satisfied, while the other two properties should be satisfied.

2.3 Performance Metrics

Traditionally, the following metrics have been used to measure the performance of mutual exclusion algorithms:

- **Message complexity:** The number of messages required to execute the critical section by a process.
- **Synchronization delay:** The time elapsed between the latest critical section exit and the entrance of the critical section for the current request.
- **Response time:** The time elapsed between the request message of a process sent out and the exit of the process from the critical section.

Furthermore, a low and high load performance, the best and the worst case performance could be considered for mutual exclusion algorithms

3 Gossip-Based Mutual Exclusion Algorithm

In this section, we first review the basic gossip-based protocol based on [21] to describe our gossip-based mutual exclusion algorithm. The mutual exclusion algorithm proposed in this section can be viewed as an extension of the gossip-based algorithm to support the mutual exclusion functionality.

3.1 Review of the Gossip-Based Algorithm

The basic gossip-based algorithm using the push-pull mode is illustrated in Figure 1. There are two different kinds of threads in each node: active and passive. At each cycle (round), an active thread selects a neighbor at random and sends a message. The active thread then waits for the message from the receiver. Upon receiving the message from the neighbor, the active thread updates local information with the received message and its own information. A passive thread waits for messages sent by active threads and replies to the senders. Afterwards, the passive thread updates its local information with the received message from the sender accordingly.

The function *getNeighbor()* returns a random neighbor identifier from its partial view, not from the entire set of nodes in our algorithm. It is noted that according to the system parameter *f* (fanout), *getNeighbor()* returns *f* number of neighbor identifiers. Additionally, before the gossiping is initiated, the partial view of nodes is constructed by the middleware called peer sampling service [21], which returns a uniform random sample from the entire set of nodes in the system.

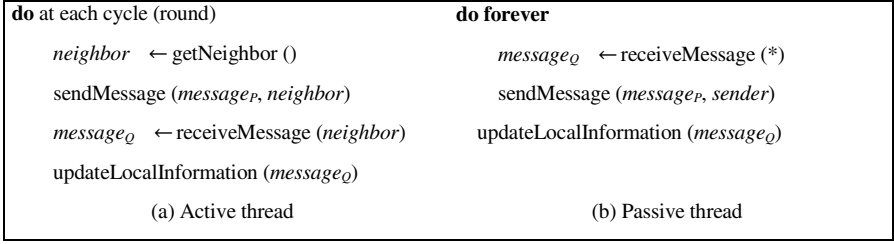


Fig. 1. Push-pull based gossip algorithm

3.2 Basic Idea

The simplest way to solve the mutual exclusion problem is to use the centralized mutual exclusion algorithm. However, it violates our assumptions in two aspects. In the centralized algorithm, the one node (i.e., coordinator) plays a special role that others cannot do. For instance, any node wishing to execute the critical section requests to the coordinator and the coordinator decide which node to be executing the critical section. In our context, nodes are functionally equal to each other.

Another violation is that fairness could not always be achieved. For example, let's assume that one of nodes is designated as the coordinator for the critical section. In this case, the coordinator permits a node to be executing the critical section if the queue is empty. This signifies that the coordinator permits a node to execute the critical section for requests in the order that the coordinator receives, not requests are sent to. As far as the communication delay is concerned, the probability violating fairness exists. Furthermore, when the coordinator fails, the mutual exclusion algorithm is not functional (i.e., Single point of failure).

Hence, we take the distributed approach with the gossip-based algorithm. To let a process decide when it can execute the critical section, we use the piggybacking mechanism by which a node adds additional information of neighbors to the message during gossiping. By using the piggybacking mechanism, any node wishing to execute the critical section can eventually decide when the process can execute the critical section without the coordinator.

In the previous researches using the distributed approach, however, they assumed that the number of nodes is static. Few studies have focused on the dynamic behavior such as adding and removing nodes while request operations are ongoing, which is that we want to deal with. In the dynamic scenario, it is assumed that each node can learn about newly added and removed nodes by the middleware before each cycle begins.

3.3 Proposed Algorithm

The gossip-based mutual exclusion algorithm is summarized in Figure 2. We explain only our extensions to the gossip algorithm. We assume that each process has a unique identifier and can be indexed by from 1 to n, where n is the number of processes (nodes) in the system. Henceforth, the terms a node and a process are used interchangeably.

- *Initial local state for process P_i*
 - **array of request tuple** $RT_i[j] = null, \forall j \in \{1 \dots n\}$
 - **extant request** $ER_i = null$
 - **array of extant request tuple** $ERT_i[j] = null, \forall j \in \{1 \dots n\}$
- *Request for CS: Process P_i executes the following for the Critical Section at a certain cycle:*
 1. **IF** (ER_i is null) **THEN** $ER_i.timestamp = LC_i, ER_i.requester = i$;
 2. **ELSE IF** $ER_i.timestamp < LC_i$ **THEN** $RT_i[i].timestamp = LC_i; RT_i[i].requester = i; RT_i[i].wish = true$;
 3. **ELSE** $RT_i[ER_i.requester] = ER_i, ER_i.timestamp = LC_i, ER_i.requester = i$
- *During gossiping: Process P_i executes the followings during gossiping with target P_j (where $j \neq i$):*
 1. *Updating extant request:*
 - (a) **IF** ($ER_i.timestamp < ER_j.timestamp$) **THEN** $ER_j = ER_i$;
 - (b) **ELSE** $ER_i = ER_j$;
 2. *Updating extant request tuple:*
 - (a) $ERT_i[i] = ER_i$;
 - (b) $ERT_j[j] = ER_j$;
 - (c) Update each element of $ERT_i[k]$ and $ERT_j[k]$, where $\forall k \in \{1 \dots n\}$, according to timestamp
 3. *Updating request tuple:*
 - (a) Update each element of $RT_i[k]$ and $RT_j[k]$, where $\forall k \in \{1 \dots n\}$, according to timestamp
 4. *Deciding for the critical section:*
 - (a) **IF** ($i == ER_i.requester$) **THEN** count the number of elements in $ERT_i[j]$, where $\forall j \in \{1 \dots n\}$ and $ERT_i[j].requester == i$
 - (b) **IF** (count == network.size()) **THEN** process P_i execute the critical section
- *Relinquishing the critical section: Process P_i finishes executing the critical section*
 1. *Removing from queue:*
 - (a) Nullify both ER_i and $ERT_i[j]$, where $\forall j \in \{1 \dots n\}$
 - (b) Set $RT_i[i].timestamp = (current) LC_i; RT_i[i].wish = false$;
 2. *Finding the next process for the critical section:*
 - (a) Find the element of j that has the smallest timestamp in RT_i (if exist) whose *wish* value is true
 - (b) Set $ER_i = RT_i[j]$;

Fig. 2. The gossip-based mutual exclusion algorithm

Each process P_i maintains the following data structures:

- $RT_i[1 : n]$: Request tuple array for P_i . This data structure consists of three elements for each array: *requester*, *timestamp* and *wish*. This request tuple acts as a request queue. It is noted that for the relinquish step, *wish* parameter is checked to select the next process for the critical section.
- ER_i : Extant (ongoing) request that has the highest priority of the request tuple array. For the sake of clarity, extant request is taken apart from request tuple. This structure includes *requester* and *timestamp*.

- $ERT_i[1 : n]$: Extant request tuple array for P_i . This data structure is necessary for the local decision to execute the critical section. The elements for each array are same as extant request.

We describe our extensions as follows:

1. If a process P_i wants to execute the critical section, ER_i is checked. When ER_i is null, set $ER_i.timestamp$ to LC_i (clock for P_i) and $ER_i.requester$ to i . When ER_i contains some values, check the timestamp. If $ER_i.timestamp$ is less than LC_i ($ER_i.requester$ has higher priority than P_i), P_i puts its request to $RT_i[i]$. Otherwise (P_i has higher priority than that of $ER_i.requester$), P_i puts ER_i to $RT_i[ER_i.requester]$ and ER_i is set for P_i .
2. During gossiping with randomly chosen target P_j from the partial view, following four steps are performed.
 - (a) Updating extant request: Compare timestamp values of the two extant requests and then update the elements of ER_i or ER_j one by one with the one that has a higher priority whose *wish* value of its request tuple is true.
 - (b) Update extant request tuples with updated extant request values. After that, update each of the elements of ERT_i and ERT_j according to timestamp.
 - (c) Update request tuples of each other (i.e., RT_i and RT_j) according to timestamp.
 - (d) Deciding for the critical section: If extant request is for P_i , P_i counts the number of elements in $ERT_i[j]$ whose requester value is equal to i . If the count value is the same as the total number of processes, process P_i can execute the critical section.
3. To relinquish the critical section, P_i nullifies ER_i and ERT_i and set $RT_i[i].wish$ to false and $RT_i[i].timestamp$ to LC_i . Finally, P_i set ER_i to $RT_i[j]$ where j 's element has the smallest timestamp in RT_i among elements that *wish* value is true.

3.4 Proof of the Algorithm

We formally prove the gossip-based mutual exclusion algorithm, showing the satisfaction of the three properties: safety, fairness, and liveness.

Theorem 1. The gossip-based mutual exclusion algorithm achieves mutual exclusion.

Proof. The proof is by contradiction. Suppose two processes P_i and P_j are executing the critical section concurrently. This means that both P_i 's ERT and P_j 's ERT are full with their own requests at some time. Formally,

$$\forall i, j \exists P_i, P_j, t_u [cs(P_i, t_u) \wedge cs(P_j, t_u) \Rightarrow R_1(i, t_u) \wedge R_1(j, t_u)]$$

where $R_1(i, t_u)$ is the relation that indicates all requester element values of extant request tuple (ERT) for P_i are i at time t_u .

Suppose that P_i 's request has a higher priority than that of P_j . This implies that when P_i 's request is in progress, P_j can execute the critical section while P_j 's ERT is full with P_j 's request. Formally,

$$\forall i,j \exists P_i, P_j, t_v, t_w [req_cs(P_i, t_v) \wedge req_cs(P_j, t_w) \wedge t_v < t_w]$$

Because P_j 's request has a higher timestamp value than that of P_i , P_i handles P_j 's request by updating its RT rather than ER containing its own request. Formally,

$$\forall i, t_u \exists P_i \neg \exists j [cs(P_i, t_u) \Rightarrow R_1(j, t_u)]$$

This means that the process P_j does not have extant request tuple (ERT) satisfying R_1 at t_u . Therefore, P_j cannot execute the critical section while P_i is executing the critical section. Formally,

$$\forall i,j \exists P_i, t_u \neg \exists P_j [cs(P_i, t_u) \Rightarrow cs(P_j, t_u)]$$

This is a contradiction. □

Therefore, the gossip-based mutual exclusion algorithm achieves mutual exclusion.

Theorem 2. The gossip-based mutual exclusion algorithm is fair.

Proof. The proof is by contradiction. Suppose that P_i 's request has a smaller timestamp value than that of P_j 's request and P_j is able to execute the critical section before P_i . Formally,

$$\begin{aligned} \forall i,j \exists P_i, P_j, t_u, t_v, t_w, t_x [req_cs(P_i, t_u) \wedge req_cs(P_j, t_v) \wedge t_u < t_v \\ \Rightarrow cs(P_i, t_w) \wedge cs(P_j, t_x) \wedge t_w > t_x] \end{aligned}$$

Suppose that P_j 's ERT is full with its own request before P_i executes the critical section after t_v and before t_w . Formally,

$$\forall i,j \exists t_v, t_w, t_x [R_2(i, t_x) \wedge R_1(j, t_x) \wedge (t_v < t_x < t_w)]$$

where $R_2(i, t_x)$ is a relation that indicates *not* all requester element values of extant request tuple (ERT) for P_i are i at time t_x .

However, ERT is updated with the request that has the highest priority among requests whose *wish* value is true. Because both of requests' *wish* is true and timestamp of P_i 's request has higher priority, P_i puts P_j 's request into RT , not ER . Likewise, P_j puts its own request into RT and P_i 's request into ER . After some number of gossip cycles, P_i 's ERT is full with its own request after t_v and before t_w . Formally,

$$\forall i \exists t_v, t_w, t_x \neg \exists j [R_1(i, t_x) \wedge R_1(j, t_x) \wedge (t_v < t_x < t_w)]$$

Subsequently, P_i executes the critical section and P_i relinquishes the critical section. Afterwards, P_i puts P_j 's request into the ER . Since P_i 's *wish* value of the request tuple is false after P_i relinquishes the critical section, P_j eventually can execute the critical section after some number of gossip cycles. Formally,

$$\forall i,j \exists P_i, P_j, t_w, t_x [cs(P_i, t_w) \wedge cs(P_j, t_x) \wedge t_w < t_x]$$

This is a contradiction. □

Hence, the gossip-based mutual exclusion algorithm is fair.

Theorem 3. The gossip-based mutual exclusion algorithm ensures the liveness property.

Proof. We show that $req_cs(P_i, t_u)$ implies that there exists t_v such that $cs(P_i, t_v)$, where $t_u < t_v$ by induction.

Basis: There is only one request for the critical section.

If only a process P_i wants to execute the critical section, P_i simply puts its own request to ER . After the requisite number of gossip cycles for the critical section, P_i 's ERT is full with its own request. This means that P_i can execute the critical section in the finite time.

Induction Step (1): There are two distinct requests for the critical section at some time.

When we consider the induction step (1), suppose that P_i and P_j have requested the critical section before one of the two processes gets the consent from all of the other processes. If P_i 's request has the higher priority than that of P_j 's, P_i 's request gets placed on ER . After the requisite number of gossip cycles, P_i can execute the critical section. Then, ERT is gradually filled with P_j 's request no sooner than P_i relinquishes the critical section. Lastly, P_j can execute the critical section. This means that, in this induction step, two distinct requests are eventually granted.

Induction Step (2): There are more than two distinct requests for the critical section at some time.

In induction step (2), two processes that have the higher priority than others can execute and relinquish the critical section in a mutually exclusive manner like in the induction step (1). After that, the request of a process that has the highest priority except for the two processes will be exchanged during the gossiping using ERT data structure. It signifies that all requests made by processes eventually be granted even if the m number of requests have made concurrently (where $m > 2$). \square

To summarize, when gossiping at each cycle, each node exchanges ERT ; after some number of gossip cycles, the process that has the highest priority execute the critical section and relinquishes it in the finite time then the entry is removed from ERT . After that, the process that has the next highest priority can execute the critical section and relinquishes it. These steps are repeated until requests exist. Hence, every request for the critical section is eventually granted.

4 Evaluation

In this section, we compare the theoretical results and simulation results with varied number of nodes. We present detailed simulation results for the gossip-based mutual exclusion algorithm using the PeerSim simulator [22]. Afterwards, we experimentally analyze the performance results for the consecutive requests with dynamic membership behavior by adding or removing some number of nodes during gossip cycles. For simplicity, it is assumed that a process executing the critical section relinquishes that before next cycle begins.

4.1 Simulation Results

The simplest form of the gossip (or epidemic) protocol comes in two states: susceptible and infected. This form of the gossip protocol is called the SI model [23]. To apply the SI model to our algorithm, any node that is in the susceptible state is considered as the non-granted node for the request. Whereas, any node that is in the infected state can be viewed as the granted node for the request. The logistic growth function for the SI model is mathematically described as follows:

$$I(c) = \frac{i_0 e^{fc}}{1 - i_0 + i_0 e^{fc}} \quad (1)$$

where $I(c)$ is the function that returns the fraction of nodes infected, i_0 is the value of $I(c)$ at $c = 0$, f is a fanout, and c is a cycle.

The theoretical results for Eq. (1) are presented in Figure 3 (i_0 is set to $1/n$). And their simulation results are presented in Figure 4. When we compare these two results, they are not exactly matched due to the nature of random uncertainty. However, notice that, in both of the results, the requisite number of cycles grows linearly as the total number of nodes increases exponentially.

Table 1. Simulation Settings

Scenario	Parameter	Value
All	The number of nodes	10,000
All	Fanout	1
All	The size of partial view	20
All	Cycles for requests	1~10
All	The number of requests at a cycle	1
2	Cycles for adding	1~10
2	The number of nodes for adding at a cycle	500
3	Cycles for removing	1~10
3	The number of nodes for removing at a cycle	100

The rationale for the comparison between the theoretical and the simulation results is: 1) Although the theoretical results imply that a fraction of infected nodes is global information, the simulation results of our algorithm show that the fraction data through the gossiping from one node (that has identifier 1). 2) Through experiments, we have confirmed that the fraction data from local information and from global information are not much different. In this regard, we can say that a local decision for the critical section is equivalent to a global-information-based decision eventually. And 3) even though the mathematical function provides the basis for expecting synchronization delay for the critical section, the actual result can vary.

For extended experiments, we also consider the following three scenarios:

1. Static membership behavior for consecutive requests: the number of nodes during gossiping is static and nodes do not fail.

2. Dynamic membership behavior for consecutive requests with joining: from cycle 1 to 10, some numbers of nodes are added during gossiping.
3. Dynamic membership behavior for consecutive requests with failing: from cycle 1 to 10, some numbers of nodes are removed during gossiping.

In dynamic scenarios, we assume that adding and removing nodes can be known to whole nodes in the system by the middleware (peer sampling service). Furthermore, during the cycle 1 through 10, requests for the critical section are made by one node at random at each cycle (total of 10 requests). Simulation settings and parameters are summarized in Table 1.

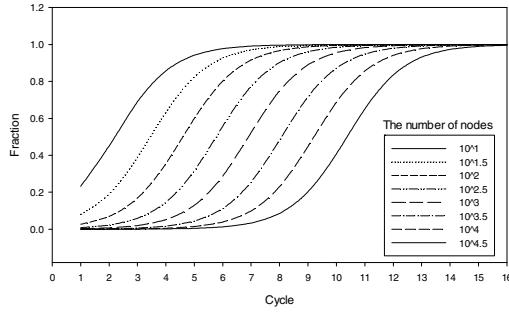


Fig. 3. Theoretical results for the SI model with varied number of nodes

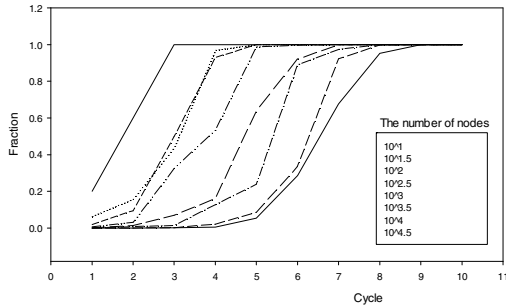


Fig. 4. Simulation results for the algorithm with varied number of nodes

Figure 5 shows the simulation results for scenario 1. Despite the consecutive requests for the critical section, we have confirmed that our proposed algorithm works correctly satisfying safety, liveness, and fairness properties according to the piggybacking mechanism. The requisite number of cycles for the critical section is 8 to 10 in scenario 1. The variability of the requisite number of cycles for the critical section (in spite of static membership behavior) is due to the nature of randomness. Recall that requesting nodes for the critical section are also set at random. The varying ranges with respect to the requisite number of cycles are relatively small.

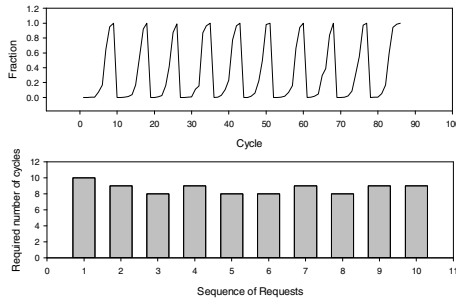


Fig. 5. Simulation results for scenario 1

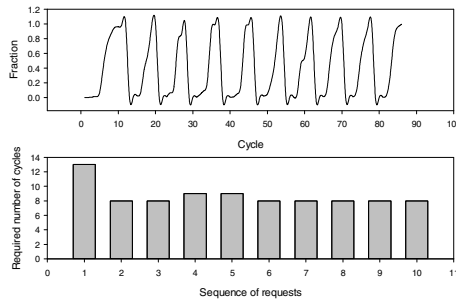


Fig. 6. Simulation results for dynamic scenario 2

In scenario 2, however, the requisite number of cycles for the critical section for the first request is 13 (see Figure 6) because some numbers of nodes are added during cycle 1 through 10. This means that if we add some number of nodes at every cycle, then a requesting node couldn't execute the critical section; some numbers of stable cycles (in terms of the number of nodes) are required. On the other hand, this signifies that the algorithm works correctly since in order to execute the critical section, all nodes (including added ones) have to agree upon the request.

When we consider failures of nodes, the situation is different because the nodes that are supposed to grant the request might be removed. In other words, the node requesting the critical section requires fewer number of consent. Furthermore, the requisite number of cycles for the critical section more fluctuates than other scenarios among the requests (see Figure 7.) because the probability that some nodes might attempt to contact with failed nodes exists if membership management of failed nodes is not implemented. Again, however, the results show that the algorithm satisfies the three properties.

On the whole, except for the scenario 3, the requisite number of cycles for the critical section is relatively stable when the number of nodes is stationary (the variance of the requisite number of cycles is less than 2). It is apparent that extrapolating from the results of the last scenario, membership management of partial

view is appealing when the number of nodes is decreasing because the variance of the requisite number of cycles is dramatic (the maximum variance of the requisite number of cycles is 5 in scenario 3 when the number of nodes is stationary).

If individual nodes select non-existing nodes frequently, synchronization delay for the critical section is prolonged. Future work should therefore include membership management of partial view for failed nodes. Furthermore, to reduce the number of messages and the requisite number of cycles for the critical section, the combining of our algorithm and efficient dissemination algorithm like in [19] would be of considerable interest.

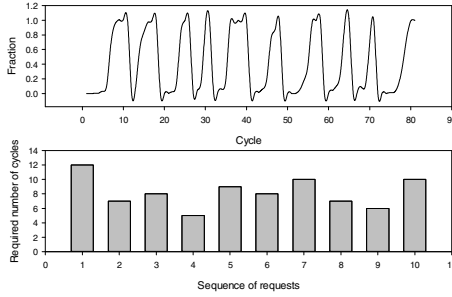


Fig. 7. Simulation results for dynamic scenario 3

4.2 Complexity Analysis

We analyze the performance of our algorithm in terms of message complexity and synchronization delay. We count the number of push-pull message exchange for the proof.

Theorem 4. The message complexity of the gossip-based mutual exclusion algorithm is ncf where n is the number of nodes, c is the requisite number of cycles for the critical section, and f is a fanout.

Proof. For each cycle, the algorithm generates nf messages. Suppose that process P_i 's request has the highest priority among requests. In order to execute the critical section, P_i must get permission from all the nodes in the system. To this end, the requisite number of cycles is c . (In this context, although c can vary according to the number of nodes, it increases linearly as the number of nodes increases exponentially.) If we set the f value to 1 (like in our simulations), the message complexity of the algorithm is nc . When we set the f value to higher than 1, the requisite number of cycles could be shortened. To generalize above arguments, the message complexity of the algorithms is ncf . \square

Theorem 5. The amortized message complexity of the gossip-based mutual exclusion algorithm is $O(n)$.

Proof. In theorem 4, we argued that the message complexity of the algorithm is ncf . However, in the gossip algorithm, gossiping cycles are periodic events and will happen infinitively if the gossip algorithm is used for (for example) the failure detection service. By amortizing the message complexity by a cycle, the message complexity of the gossip-based mutual exclusion algorithm is nf . Furthermore, disregarding the coefficient f (fanout), we can say that the amortized message complexity of our algorithm is asymptotically at most n . \square

Thus, the amortized message complexity of the algorithms is $O(n)$.

Theorem 5. The synchronization delay of the gossip-based mutual exclusion algorithm is c cycles.

Proof. For the critical section, a process P_i requires c cycles to get permission from all nodes in the system. If P_j 's request has the highest priority after P_i relinquishes the critical section, then P_j should wait for c' cycles as well. Because our algorithm does not generate relinquish messages (*wish* value in RT_i is used instead), the synchronization delay of the algorithms is c cycles. \square

5 Conclusions

In this work, we have presented the mutual exclusion algorithm based on the gossip-based algorithm to cope with scalability and fault tolerance issues with proof. A cloud environment where the behavior of their constituting nodes is active and dynamic (i.e., joining and leaving at any time) is one that our algorithm will be applied to. The amortized message complexity of our proposed algorithm is $O(n)$, which is worse than previous research; however, from the requester point of view for the critical section, the message complexity is $2c$, c for the active thread, c for the passive thread because the probability that each node being selected by other nodes at each cycle is $1/n$. Nonetheless, our approach is promising and should be explored with other algorithms and applications. For example, our gossip-based mutual exclusion algorithm could be embedded seamlessly into other existing gossip-based algorithms. In other words, if a gossip-based algorithm is implemented for a failure detection service, then the mutual exclusion algorithm proposed in our work can be embedded in the existing gossip-based algorithm. In addition, maintenance of replicated data is a suitable application to which our algorithm can be applied where write operations rarely occur compared to read operations. Our simulation results show that our proposed gossip-based algorithm in dynamic and loosely-coupled environments is scalable and fault tolerant.

References

1. Suzuki, I., Kasami, T.: A distributed mutual exclusion algorithm. ACM Trans. Comput. Syst. 3, 344–349 (1985)
2. Raymond, K.: A tree-based algorithm for distributed mutual exclusion. ACM Trans. Comput. Syst. 7, 61–77 (1989)

3. Mizuno, M., Neilsen, M.L., Rao, R.: A token based distributed mutual exclusion algorithm based on quorum agreements. In: 11th International Conference on Distributed Computing Systems, pp. 361–368 (1991)
4. Neilsen, M.L., Mizuno, M.: A DAG-based algorithm for distributed mutual exclusion. In: 11th International Conference on Distributed Computing Systems, pp. 354–360 (1991)
5. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* 21, 558–565 (1978)
6. Ricart, G., Agrawala, A.K.: An optimal algorithm for mutual exclusion in computer networks. *Commun. ACM* 24, 9–17 (1981)
7. Singhal, M.: A Dynamic Information-Structure Mutual Exclusion Algorithm for Distributed Systems. *IEEE Trans. Parallel Distrib. Syst.* 3, 121–125 (1992)
8. Lodha, S., Kshemkalyani, A.: A fair distributed mutual exclusion algorithm. *IEEE Transactions on Parallel and Distributed Systems* 11, 537–549 (2000)
9. Maekawa, M.: A \sqrt{N} algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comput. Syst.* 3, 145–159 (1985)
10. Agrawal, D., Abbadi, A.E.: An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Trans. Comput. Syst.* 9, 1–20 (1991)
11. Joung, Y.-J.: Asynchronous group mutual exclusion. *Distrib. Comput.* 13, 189–206 (2000)
12. Ganesh, A.J., Kermarrec, A.M., Massoulié, L.: Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers* 52, 139–149 (2003)
13. Allavena, A., Demers, A., Hopcroft, J.E.: Correctness of a gossip based membership protocol. In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing*, pp. 292–301. ACM, Las Vegas (2005)
14. Gurevich, M., Keidar, I.: Correctness of gossip-based membership under message loss. In: *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, pp. 151–160. ACM, Calgary (2009)
15. Ganesh, A.J., Kermarrec, A.-M., Massoulié, L.: HiScamp: self-organizing hierarchical membership protocol. In: *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, pp. 133–139. ACM, Saint-Emilion (2002)
16. Voulgaris, S., Gavidia, D., van Steen, M.: CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays. *Journal of Network and Systems Management* 13, 197–217 (2005)
17. Matos, M., Sousa, A., Pereira, J., Oliveira, R., Deliot, E., Murray, P.: CLON: Overlay Networks and Gossip Protocols for Cloud Environments. In: Meersman, R., Dillon, T., Herrero, P. (eds.) *OTM 2009. LNCS*, vol. 5870, pp. 549–566. Springer, Heidelberg (2009)
18. Jelasity, M., Montesor, A., Babaoglu, O.: T-Man: Gossip-based fast overlay topology construction. *Comput. Netw.* 53, 2321–2339 (2009)
19. Lim, J.B., Lee, J.H., Chin, S.H., Yu, H.C.: Group-Based Gossip Multicast Protocol for Efficient and Fault Tolerant Message Dissemination in Clouds. In: Riekk, J., Ylianttila, M., Guo, M. (eds.) *GPC 2011. LNCS*, vol. 6646, pp. 13–22. Springer, Heidelberg (2011)
20. Iwanicki, K., van Steen, M., Voulgaris, S.: Gossip-Based Clock Synchronization for Large Decentralized Systems. In: Keller, A., Martin-Flatin, J.-P. (eds.) *SelfMan 2006. LNCS*, vol. 3996, pp. 28–42. Springer, Heidelberg (2006)
21. Jelasity, M., Guerraoui, R., Kermarrec, A.-M., van Steen, M.: The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations. In: Jacobsen, H.-A. (ed.) *Middleware 2004. LNCS*, vol. 3231, pp. 79–98. Springer, Heidelberg (2004)
22. Montesor, A., Jelasity, M.: PeerSim: A scalable P2P simulator. In: *IEEE Ninth International Conference on Peer-to-Peer Computing, P2P 2009*, pp. 99–100 (2009)
23. Newman, M.: *Networks: An Introduction*. Oxford University Press, Inc., New York (2010)

An Effective Partition Approach for Elastic Application Development on Mobile Cloud Computing

Zhuoran Qin¹, Jixian Zhang¹, and Xuejie Zhang^{1,2}

¹Dept. of Computer Science and Technology, Yunnan University, Kunming, China
zhuoran.qin@gmail.com, denonji@163.com

²High Performance Computing Center, Yunnan University, Kunming, China
xjzhang@ynu.edu.cn

Abstract. Mobile cloud computing is the cloud infrastructure where the computation and storage are moved away from mobile devices. The elastic partition in according to context-awareness could break through the resource constrain of mobile devices. The improved (K+1) coarse partition algorithm is used to partition the cost graph, where the vertexes are represented by the execution cost on mobile device and offloading cost to cloud. The two factors are represented by some contextual information including execution time, current power, network and the probabilities which are obtained by the statistical analysis of historical results. The levels of context-awareness could adjust the weight of the contextual information and lead to partition again. Partition cost module is used to store and compute the contextual information. The extensive experiments deployed the OCR project on the proposed architecture demonstrate a good performance in different input and network environments.

Keywords: Mobile Cloud Computing, Elastic Mobile Application, Elastic Partition Algorithm, Context-Awareness.

1 Introduction

Mobile devices and mobile applications have enjoyed rapid development in recent years. Compared with current PC, mobile devices still cannot run data intensive applications, such as search, large-scale data management and mining, etc., and have limitations in battery power, screen size, wireless communication etc.

[1]The most attractive features of cloud computing lie on the capability in powerful computing capability and massive data storage as well as a new business model, which delivers the computing resources as a utility. [2, 8] Mobile cloud computing is defined as an extension of cloud computing in which foundation hardware consists at least partly of mobile devices. This definition recognizes the opportunity to harness collective sensing, computational capabilities of multiple networked wireless devices to create a distributed infrastructure that supports a wealth of new applications.

It is a complex issue that mobile applications move computing power and data storage away from mobile phones into the cloud. First and foremost, it will face to the partition problem. With the changes of the mobile environment, partition algorithm

should achieve elastic partition in according to context-awareness. A variety of contextual information could impact the elastic partition results, such as battery level, connection quality, device loads, etc. [1]Zhang proposed elastic application model, a new application model supporting applications partitioned into multiple components, each of which can run autonomously from others. The model is effective in leveraging cloud computing for the resource constrained mobile device. Based on elastic application model, we proposed the elastic partition algorithm and the partition cost module.

The rest of the paper is organized as follows. Section 2 is the discussion of related work. Then Section 3 will present elastic partition algorithm, following that, respectively in the next two sections, would be the description of partition cost module and the evaluation. Fundamentally the conclusion lies in the last section.

2 Related Work

In [3], an undirected graph is used to represent a partition model. 3-tuple including memory, CPU time and bandwidth, represents the vertexes' weight. All the vertexes will be partitioned by (K+1) Coarse Partition Algorithm. Although the partition algorithm is accurate, the computing process is complicated and consumes a large number of computing resources. In [4] Gu uses graph structure and OLIE algorithm in order to find all possible 2-way cuts of execution graph. The migration will be triggered, only if mobile device doesn't have enough computing resources. In [5], the vertex's weight is the size of code and the edge's weight is the number of two vertexes' interactions. The two algorithms [4, 5] don't consider the changes of the context, migration cost or users' behaviors. [6]Chun uses tree structure as the partition model. The nodes represent time cost, which can also be used to calculate the energy consumption. In practice, the time of display state (on/off) is random, while the display cost generally takes a greater proportion than others. Hence, the calculation of energy consumption is not accurate. [7, 8]Considering that mobile device has sensing abilities and the number is large, researchers make mobile devices as a part of cloud computing. [7]Gonzalo adopts P2P technologies, but it doesn't implement how to split the elements of the tasks. Hyrax [8] which is derived from Hadoop shows that the data storage and processing can carry out on mobile device. It is hard to promote because the two algorithms will bring a lot of security issues, and in some cases, some users have to share their resources first.

3 Elastic Partition Algorithm

In this section, we will introduce elastic partition algorithm as figure 1 shows. Partition granularity is one app-component which can be functions, classes, or components. [9] Any application unit which is launched independently on mobile devices or cloud can be called as app-component. The partition granularity can be decided by developers under different conditions.

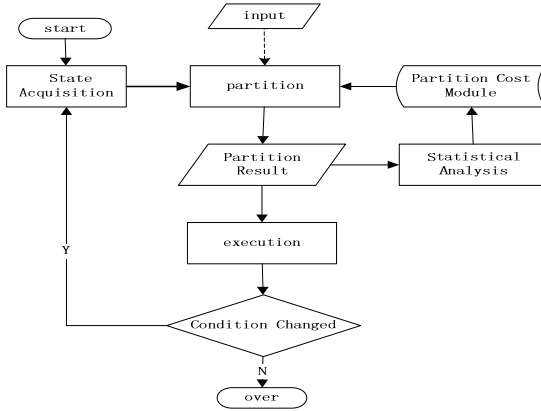


Fig. 1. The process of partition algorithm

At the beginning, it will acquire the status information of mobile device, such as battery power, signal intensity, etc. After the input and status information are passed to the partition module, the application can get into computing according to the partition algorithm. If the context is changed over the threshold, it will partition again so as to achieve the context-awareness. When it is partitioned again, the process could not need to input again. So it is represented in dash line under the input module. Context refers to the computing resources, battery power, and sensing ability, etc. inside the devices or mobile environments, such as network connection. In [7, 10], researchers think offloading computing often occurs with place-bound activities which are at a fixed location, such as museum, coffee shop. In that case, network connection is more stable, leading to fewer disconnections and faults, and the computing performance is stable and regular. According to this characteristic, we deem that the probability, obtained by the statistical analysis of historical partition results, is one of the most important partitioning factors. By probabilities, it can decrease the number of computation, and save time.

3.1 Cost Graph

$$w_2 = (\varepsilon_1 power_2 + \varepsilon_2 time_2 + \varepsilon_3 memory) \cdot PMobile \quad (1)$$

$$\varepsilon_1 + \varepsilon_2 + \varepsilon_3 = 1, \text{ and } 0 < \varepsilon_i < 1 (i = 1, 2, 3)$$

w_2 represents the execution weight in mobile devices. $power_2$ is the current battery power of mobile device. $time_2$ which tightly associates with computing environments is the execution time of app-component in mobile device. $memory$ represents the memory cost of every app-component which can be measured by a large number of experiments. $PMobile$ is the execution probability in mobile device. The parameters represent the weight of the three factors in w_2 , and can be adjusted according to the levels of context-awareness in different conditions. So does the parameters in formula 2.

$$w_1 = (\lambda_1 power_1 + \lambda_2 time_1 + \lambda_3 network) \cdot PCloud \quad (2)$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1, \text{ and } 0 < \lambda_i < 1 (i = 1, 2, 3)$$

w_1 represents the weight of execution in cloud. $power_1$ is the current battery power of mobile device. $time_1$, composed of cloud execution time and transmission time, represents time cost of offloading. Transmission time associates with location as we mentioned above. At the beginning, the application will connect the cloud server, which not only ensures the connection, but also measures the value of $time_1$. $network$ represents the present status of network. $PCloud$ is the execution probability in cloud.

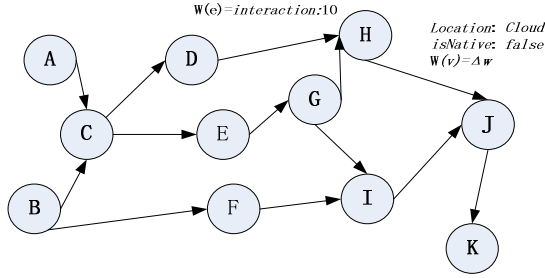


Fig. 2. Cost Graph

Figure 2 is the cost graph and the weight of vertex equals Δw as formula 3 shows.

$$w(v) = \Delta w = w_2 - w_1 \quad (3)$$

$\Delta w = w_2 - w_1$, Δw is the weight of vertex. Offloading computing is associated with Δw and interactions between the two vertexes, software authority, privacy, and safety problems, etc. Only $\Delta w \geq 0$ is out of right to decide the offloading. The weight of edges equals interaction as formula 4 shows.

$$w(e(v_i, v_j)) = interaction \quad (4)$$

$interaction$ is the total number of interactions between v_i and v_j , including method invocation and data access. Every vertex is denoted by another 2 labels in figure 2, $isNative$ and $Location$. The app-component which could not offload to cloud, such as GPS, Photos, etc. can be labeled as true. $Location$ represents the execution location, on cloud or mobile device.

3.2 Optimization Problem

This type of graph partition problem is known to be NP-complete. Then our algorithm attempts to find the optimized solution. The graph model is Graph $G = (V, E)$.

$$C_M \cap C_N = \emptyset \quad (1 \leq M, N \leq K \text{ and } M \neq N) \tag{5}$$

$$\bigcup_{i=1}^K C_i = C_N, \quad \bigcup_{i=1}^K C_i = P \setminus C_M$$

It partitions the app-component into $K+1$ groups, of which K groups will be executed in cloud and one group will be run in mobile device. In formula 5, C_N represents entire groups running on cloud. C_M is the group running on mobile device. C_i represents the group i which is executed on cloud. P represents all the groups.

$$CW(U, v) = \lambda_1 w(U) + \lambda_2 w(e(U, v)) \tag{6}$$

$U=C_M$ or $C_i(i=1, 2, \dots, k)$, Ω =all the vertexes of U , $\lambda_i(i=1,2)$ ($0<\lambda_i<1$ and $\lambda_1+\lambda_2=1$), and $w(U) = \sum_{v \in \Omega} w(v)$ $w(e(U, v)) = \bigcup_{v \in \Omega} w(e(u, v))$ CW is the cost weight.

Formula 6 means all the vertexes' and edges' weight equals to $CW(U, v)$. The sum of all the vertexes' weight is $w(U)$ and the sum of all edges' weight is $w(e(U, v))$.

$$\min Objective = CW(C_M, v) + \sum_{i=1}^K CW(c_i, v) \quad (0 < K < n) \tag{7}$$

if $K=0$ $\min Objective = CW(C_M, v)$

Formula 7 is the objective function represented by *Objective*, expressing the general goal which means the minimum cost. If there is no app-component in cloud, then $K=0$, the objective equals to the cost in mobile device.

3.3 Improved (K+1) Coarse Partition Algorithm

Figure 3 depicts the partition results by Improved (K+1) Coarse Partition Algorithm, which is based on (K+1) Coarse Partition Algorithm [3]. Our algorithm requests the Heavy-Edge-and-Weight-Vertex, which is represented by CW as formula 8 shows, whereas the original algorithm requests Heavy-Edge-and-Light-Vertex.

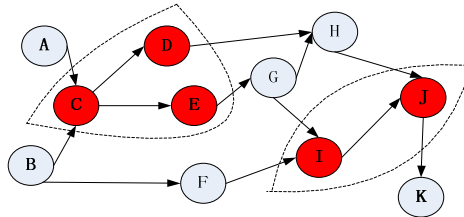


Fig. 3. Improved (K+1) Coarse Partition Algorithm

$$\begin{aligned}
 CW(u, v) &= \lambda_1 w(v) + \lambda_2 w(e(u, v)) \\
 \lambda_i (i = 1, 2) &(0 < \lambda_i < 1 \text{ and } \lambda_1 + \lambda_2 = 1)
 \end{aligned}
 \tag{8}$$

Formula 8 means that merging the vertexes which are connected tightly and easy to offload as a group. As figure 3 shows, the vertexes in dash line mean a partition group, distinguished in red.

It can achieve dynamic partition by partition again. The figure 4 tries to demonstrate the scenario that the context has changed after A,B,C,D,E,F run over, as a result that the cost graph will be partitioned again with context-awareness by running the algorithm again. The vertexes on the left side are in grey, which means the execution of vertex is over. H and J are partitioned into a new group, instead of I and J.

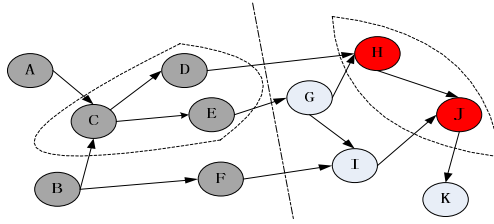


Fig. 4. Partition Again

At the end of the algorithm, it will modify the parameters in partition cost module, such as *PMobile*, *PCloud*, etc.

4 Partition Cost Module

Partition cost module is used to store and compute the context-awareness factors. We can classify those factors into 3 classes. **Fixed Value** is determined by developers' the number of experiments. Once the values are fixed, they cannot be changed within the context. **Fixed Value** includes *memory*, *bandwidth*, *throughput* and *interaction*. *throughput* represents the amount of transmission between app-component. **Current Value** signifies the current state value of mobile device, such as *power₂*, *power₁*, *networkstate*, *time₁*. *networkstate* represents the current signal intensity. **Computing Value** involves *time₂*, *network*, *PMobile*, *PCloud*. *network* represents the current state of network, which can be the functional value of throughput, bandwidth and network state, e.g. $network = f(\text{throughput}, \text{bandwidth}, \text{networkstate})$. *PMobile*, *PCloud* represent the probabilities of execution on cloud and mobile device respectively. *PMobile* equals the execution number on mobile devices divided by the total execution amounts. *PMobile* plus *PCloud* are equal to 1. [3]The parameters in formula 1 and 2 can be determined by different means: (1) the values can be set by the application developer; (2) they can be chosen by end users according to the real-time scenarios; (3) these values can be dynamically decided by the offloading systems according to resource availabilities in the mobile device.

5 Implement and Evaluation

We test the system by using MOTO ME525. Cloud server is composed of 3 desktop computers, which are HP Compaq 8000 Elite equipped with Ubuntu Linux 11.10 and Hadoop 0.20.203.0. Tesseract-ocr [11] is deployed on cloud. The OCR engine will read a binary, grey or color image and output text. [12]The program includes Page Layout Analysis, Blob Finding, Find Text Line and Words, Recognize, etc. We use the English project, and neglect the substitutions, deletions, insertions, lines and other problems. The only factor to influence computing results is the number of words.

5.1 Experiment 1

Table 1 shows the execution time in different amounts of input and communication channels. MD represents the execution time on mobile device. The time unit is second. With the amounts of input increased, the performance will be enhanced. According to the partition algorithm, more parts will be loaded on mobile device under the low transmission rate, and the program will execute on Android device from 20,000 to 40,000 on GPRS.

Table 1. The execution time in different number of input and communication channels

Input	3G	WiFi	GPRS	MD
10,000	20.45	17.31	75.74	84.46
20,000	32.77	27.18	152.68	152.68
30,000	42.84	35.65	225.37	225.37
40,000	51.39	42.72	296.89	296.89

5.2 Experiment 2

Experiment 2 will prove the efficiency when the context is changed. The communication channels changed between 3G and WiFi, WiFi and GPRS respectively. The data in third column is less than that of the forth column. Because of the low transmission rate, a large number of computations will run in the device.

Table 2. The execution time on the communication channels exchange

Input	MD	3G↔WiFi	WiFi↔GPRS
10,000	80.46	24.02	50.52
20,000	152.68	34.29	85.67
30,000	225.37	43.85	173.91

6 Conclusion and Future Work

In this paper, we proposed the elastic partition algorithm and partition cost module. The partition algorithm achieves elastic partition according to the context-awareness. Our algorithm not only considers the device loads and cloud features, but also the offloading cost to the cloud, and users' preferences which are represented by the probabilities.

The fault-tolerance and data security mechanism are under development. The integrated scheduling and the data synchronization mechanism are desirable. Besides, we need an effective method to count recent status, especially the probabilities, on the condition of moving to another place which is considerably far and inexperienced.

Acknowledgements. This project is supported by the National Natural Science Foundation of China (GrantNo:61170222).

References

1. Zhang, X., Jeong, S., Kunjithapatham, A., Gibbs, S.: Towards an Elastic Application Model for Augmenting Computing Capabilities of Mobile Platforms. In: The 3rd International ICST Conference on Mobile Wireless Middleware, Operating Systems, and Applications (MobileWare), vol. 48(4), pp. 161–174 (2010)
2. Fan, X., Cao, J.: A Survey of Mobile Cloud Computing. *ZTE Communications* 9(1), 4–8 (2011)
3. Ou, S., Yang, K., Zhang, J.: An effective offloading middleware for pervasive services on mobile devices. In: *Pervasive and Mobile Computing*, pp. 362–385 (2007)
4. Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., Milojicic, D.: Adaptive Offloading Inference for Delivering Applications in Pervasive Computing Environments. In: *Proc. of PerCom*, pp. 107–114 (2003)
5. Giurgiu, I., Riva, O., Juric, D., Krivulev, I., Alonso, G.: Calling the Cloud: Enabling Mobile Phones as Interfaces to Cloud Applications. In: Bacon, J.M., Cooper, B.F. (eds.) *Middleware 2009*. LNCS, vol. 5896, pp. 83–102. Springer, Heidelberg (2009)
6. Chun, B.G., Ihm, S., Maniatis, P., Naik, M., Patti, A.: CloneCloud: Elastic Execution between Mobile Device and Cloud. In: *Proc. of the 6th European Conference on Computer Systems (EuroSys 2011)*, pp. 301–314 (2011)
7. Huerta-Canepa, G., Lee, D.: A virtual cloud computing provider for mobile devices. In: *Proc. of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond* (2010)
8. Marinelli, E.E.: Hyrax: cloud computing on mobile devices using MapReduce. Master Thesis Draft, Computer Science Dept., Carnegie Mellon University (2009)
9. Zhang, X., Schiffman, J., Gibbs, S., Kunjithapatham, A., Jeong, S.: Securing Elastic Applications on Mobile Devices for Cloud Computing. In: *ACM Cloud Computing Security Workshop (CCSW)*, pp. 127–134 (2009)
10. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The Case for VM-Based Cloudlets in Mobile Computing. In: *Proc. IEEE Pervasive Computing*, vol. 8(4), pp. 14–23 (2009)
11. Smith, R.: An Overview of the Tesseract OCR Engine. In: *Proc. of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 02, pp. 629–633 (2007)
12. Smith, R., Antonova, D., Lee, D.S.: Adapting the Tesseract open source OCR engine for multilingual OCR. In: *Proc. of the International Workshop on Multilingual OCR, MOCR 2009* (2009)

Memory Virtualization for MIPS Processor Based Cloud Server

Li Ruan^{1,2}, Huixiang Wang¹, Limin Xiao^{1,2}, Mingfa Zhu¹, and Feibo Li

¹School of Computer Science and Engineering, Beihang University, Beijing 100191, China

²State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University,
Beijing, 100044, China
ruanli@buaa.edu.cn

Abstract. Although the Loongson 3 processor, one representative of RISC processors based on MIPS64 instruction set, nowadays gains increasing focus and applications in Servers, memory virtualization, a critical part of a system virtual machine for such MIPS processor based Cloud Servers, meets challenges such as non-virtualizable instruction set architecture, without hardware assistant and addresses translation from guest virtual address (GVA) to host physical address (HPA) supports. However, there are few virtualization researches on MIPS processor based Cloud Server and practical virtual systems are much fewer. This paper introduces a shadow TLB based memory virtualization method. We summarize the challenges and present the key technologies such as the memory architecture, address space construction and mapping mechanism, the shadow TLB maintenance method, etc. Experimental results show that our approach can provide effective memory virtualization support for MIPS processor based Cloud Servers.

Keywords: MIPS, cloud computing, sever virtualization, memory virtualization.

1 Introduction

With the performance improvement, MIPS[1] processors gain increasingly application to servers. Recently, the virtualization of such servers has been played great emphasis on as a backbone technology to improve their resource utilization with the revival of virtualization and the advent of Cloud computing. Although x86 based server virtualization has achieved many fruits, there are comparatively few virtualization researches on MIPS processor based Cloud Server and practical systems are much fewer.

Loongson 3 processor is a general-purpose RISC processor based on the MIPS64 instruction set developed by the Institute of Computing Technology (ICT), Chinese Academy of Sciences (CAS)[2]. The newest 65 nm Loongson 3A is able to run at a clock speed near 1 GHz, with 4 CPU cores (~15W). In 2012, we developed a Server for Cloud computing based on Loongson 3A processors (for convenience, Loongson 3 Cloud Server in short) from its hardware to system software under the support of

Chinese EIIIF (Electronic Information Industry Fund). To improve the resource utility, a system machine was implemented for the Server based on kernel virtual machine (KVM)[3]. Our system virtual machine included three key modules: CPU virtualization, memory virtualization and I/O virtualization. In this paper, we summarize challenges of memory virtualization for MIPS processor (especially Loongson 3) based Server, compare the related work, propose an address space construction and mapping mechanism, TLB based address mapping and maintenance methods. Experimental results demonstrate that our approach is feasible.

2 Analysis of Challenges in Memory Virtualization

As Loongson 3 processors is a typical general-purpose RISC processor based on the MIPS64 instruction set, to implement memory virtualization for Loongson 3 Cloud Server, there exists the following challenges to be solved:

- For Loongson 3 processors, only Loongson 2E has the QEMU support which is not a traditional sense of the virtualization solution. i.e., Loongson 3 is without hardware assistant support.
- As Loongson 3 is based on the MIPS architecture, virtualization support meets many difficulties, such as not all sensitive instructions are privileged instructions, the kernel address space mapping relationship is not flexible. As a MIPS based server, because the MIPS processor's kernel address space is stationary, it brings a big problem that the host kernel can run in the stationary address space but the guest kernel cannot. This causes a challenge of how to design a strategy to construct the host kernel and guest kernel in different address space.
- To make GOS own a separate continuous memory space which starts from zero, VMM introduces a new hierarchy of address space- Guest Physical Address (GPA) space. Thus, a three-tiered address mapping mechanism from Guest Virtual Address (GVA), GPA to Host Physical Address (HPA) should be established. As the traditional Memory Management Unit (MMU) can only finish one time of mapping from virtual address to physical address, the original MMU is ineffective to deal with such three-tiered address space and especially dynamically maintain the mapping for GPA to HPA for each virtual machine.

3 Related Work

We briefly survey related work from two perspectives: MIPS virtualization and memory virtualization.

In the past few years, there are fewer researches on MIPS virtualization than X86 virtualization. The representative works are Disco[4] and OKL4 [5] which proposed virtualization solutions to MIPS. However, they had a big gap to be utilized in practice. What's more, they did not conduct implementation and experiments on Loongson 3 processors. For memory virtualization, the most mature method is

shadow page table [6-8] for the X86 architecture. Effective memory virtualization methods on MIPS architecture still lack. There are some optimization technologies such as “balloon module” and “page sharing”. As Loongson 3 server has a TLB hardware, so one problem is how to implement the shadow TLB. We introduced the design and implementation of a system virtual machine for Loongson 3 Cloud Server in [3]. However, [3] just gave a bird-eye of the whole virtual machine, details of its memory virtualization together with its experiments was not introduced.

4 The Memory Architecture of Loongson 3

4.1 Virtual Address Space of Loongson 3 Processors

Loongson 3 processor uses a special MMU to translate the virtual address to physical address and also provides a TLB for each core to accelerate the speed of translation. We use $M = \{kernel\ mode, supervisor\ mode, user\ mode\}$ to denote the set of operating modes and $S = \{Kuseg, Kseg0, Kseg1, Kseg2, Kseg3\}$ to denote the set of address spaces of Loongson 3 processor. Table 1. shows the access right matrix of M to S.

As Table 1 and Fig.1 shows, each Loongson 3 processor has 3 modes which have different priorities and own different address spaces. We will use 32-bit virtual address space as an example to illustrate the address mapping space mechanism. The 32-bit address space S is divided into 5 segments. The *kernel mode* can access all of the five segments, the *supervisor mode* can access the *kseg2* and *kuseg* segments and the *user mode* can only access the *kuseg* segment. As the *kseg0* and *kseg1* are unmapped, that virtual address in these two segments does not use the page table and

Table 1. The access right matrix of M to S

		S				
		<i>Kuseg</i>	<i>Kseg0</i>	<i>Kseg1</i>	<i>Kseg2</i>	<i>Kseg3</i>
M	<i>Kernel mode</i>	Y	Y	Y	Y	Y
	<i>Supervisor mode</i>	Y	N	N	Y	N
	<i>User mode</i>	Y	N	N	N	N

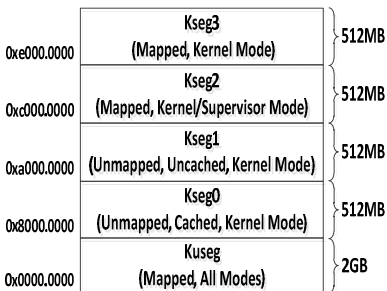


Fig. 1. 32-bit Loongson 3 memory mapping relationship

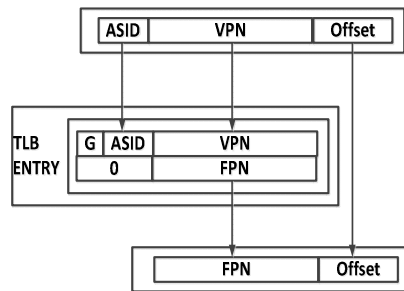


Fig. 2. The translation between virtual address and physical address

TLB to translate the address to the physical address. And according to MIPS architecture, the kernel of operating system usually runs in the *kseg0* segment.

4.2 The Translation Lookaside Buffer of Loongson 3

In order to enhance the speed of the translation of the virtual address to physical address, Loongson 3 uses two kinds of TLB: the data TLB for data mapping with 128 entries and the instruction TLB for instruction mapping with 16 entries. Fig.2 illustrates how virtual address is translated to physical address. From the view of address translation, the 8-bit address space identifier (ASID) extends the virtual address. This method will reduce the TLB refreshing frequency when the context is switched.

5 Address Space Construction and Mapping Mechanism

As is discussed in the above, the *kseg0* and *kseg1* are unmapped. When the host OS boots, some initial work has not been done and the page table can't be used. Therefore, the system must choose a directly-mapped address space to load the kernel. As the *kseg0* and *kseg1* segments are unmapped and occupied by the host OS, the GOS of the virtual machine can't run in those segments. Therefore, we must design a mechanism to move the guest OS's kernel and I/O registers to other address spaces. The requirements of the address segments are defined in Table 2.

Now we analyze which address segment in S can be used as the suitable address for the GOS's kernel and I/O registers. *kseg0* and *kseg1* do not satisfy the Req.1 and Req.2 because host kernel is running in *kseg0* and they are unmapped. If we use *kseg0* and *kseg1* as the address, after translated to the physical address, these two fragments may conflict with the host. Although *kuseg* meets the Req.3 and Req.4, this segment

Table 2. Requirements of the address space construction

Req.#	Description of the requirements
Req.1	Guest operating system's kernel running in a special mode which priority is lower than the VMM's mode and higher than the user mode.
Req.2	This special mode can access adequate address space.
Req.3	This address space is large enough to running the system kernel and its I/O registers.
Req.4	This address space can be transformed to the physical address space which does not incur conflict with the host.

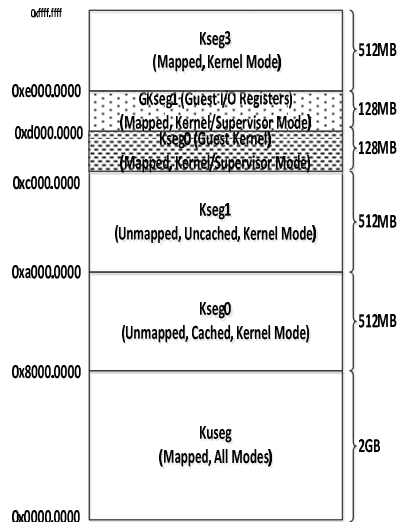


Fig. 3. 32-bit Loongson 3 memory map with memory virtualization

has the lowest priority and thus Req.1 and Req.2 are not satisfied. *kseg3* segment's priority is the highest but it does not satisfies Req.1. Fortunately, *Kseg2* segment can meet all requirements. The proof is as following: (1) *kseg2* can be accessed in the *supervisor mode*(Req.1 is satisfied). (2) If GOS's kernel and I/O registers run in *kseg2*, it can access other adequate address space(Req.2 is satisfied). (3) The address of *Kseg2* ranges from *0xc0000000* to *0xe0000000*, totally 512MB which is enough to be assigned to GOS's kernel and I/O registers(Req.3 is satisfied).(4) *Kseg2* can transform the virtual address to physical address by the page table or the TLB(Req.4 is satisfied).Thus, we propose to use *kseg2* segment as GOS's kernel and I/O register's address space(Fig.4 is satisfied).

By dividing the *kseg2* into *Gkseg0* for *guest kernel* and *Gkseg1* for *guest I/O registers*, our address space construction mechanism is based on the following rules:

$$\begin{aligned} Smips &= \{Kuseg, Kseg0, Kseg1, Kseg2, Kseg3\}; Kseg2 = \{Gkseg0, Gkseg1\}; \\ Kuseg &= [0x0000.000, 0x8000.000]; Kseg0 = [0x8000.000, 0xa000.0000]; \\ Kseg1 &= [0xa000.0000, 0xc000.0000]; GKseg0 = [0xc000.0000, 0xd000.0000] ; \\ GKseg1 &= [0xd000.0000, 0xe000.0000]; Kseg3 = [0xa000.0000, 0xffff.ffff]. \end{aligned}$$

6 The TLB Maintenance Method

As Loongson 3 processor has a TLB support, the maintenance of TLB is the core procedure of our virtualization methods. In the following sections, we will introduce the architecture, workflow and exception handling mechanisms of TLBs in detail.

6.1 The Architecture and Workflow of the TLBs

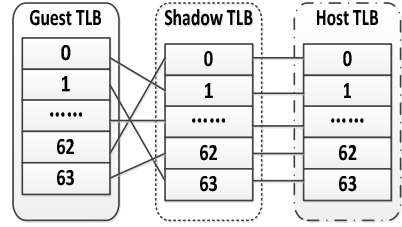
Three kinds of TLBs are defined in our system. Their functions, characteristics and relationships are as following:

- 1) **Guest TLB(GTLB):** GTLB is a structure maintained by the GOS and the basis for generating the shadow TLB(STLB). Its responsibility is to store the mapping relationship between GVA and GPA. When GOS runs a TLB-related instruction, the VMM will capture a TLB instruction and modify GTLB. By this, GOS thought that it changed the hardware TLB.
- 2) **Shadow TLB(STLB):** The STLB is the core in the memory virtualization module. Its responsibility is to store of the mapping relationship between GVA and HPA.
- 3) **Host TLB(HTLB):** The HTLB is a structure maintained by the host OS. Its responsibility is to store of the mapping relationship between host virtual address (HVA) and host physical address (HPA). With HTLB, our system virtual machine can translate GVA to HPA.

The mapping relationships of TLBs are defined in Table 3, where the HTLB and STLB hold one to one relationship and the relationship between the STLB and GTLB is out of order. Fig. 4 shows an example among GTLB, STLB and HTLB.

Table 3. The mapping relationships

TLBs	Mapping
< STLb, HTLB >	1:1
< STLb, GTLB >	Out of order

**Fig. 4.** An example mapping

6.2 The Shadow TLB

As the TLB architecture shows, STLb is the main connection between the GVA and HPA. Therefore, how to maintain the STLb is the core in Loongson 3 server' memory virtualization. When GOS runs, it will take the mapping relationship between GVA and HPA maintained by the STLb out from the TLB hardware. In our virtual system, the STLb's workflow is defined as follows:

Step1(Initialize TLB): Before the GOS runs, entries of STLb whose valid bit (V-bit) is 1 are written into the TLB hardware by KVM.

Step2(Capture TLB instruction): When the GOS runs, the TLB instruction of GOS is captured by the VMM.

Step3(Store new mapping relation): The VMM modifies GTLB. Then Based on the new mapping relationship between the GVA and GPA stored in the GTLB and the reflection from HVA to HPA in the HTLB, VMM generates a new mapping relationship between GVA and HPA and stores it in STLb structure.

Step4(Update new STLb entry and write STLb to hardware): VMM will set the valid bit (V-bit) to 1 to update the new STLb entry. And finally it writes the shadow entry into TLB hardware.

6.3 TLB Miss Exception Handler

When the GOS triggers a *TLB miss* exception, VMM will capture this exception and handle it. In our system, we need to deal with 3 cases of *TLB miss* exception of the virtual machine.

- **Case1: Shadow TLB miss, guest TLB hit**

VMM handles this case by itself. Based on the *GTLb hit* entry, VMM first builds a mapping between the GVA and HPA and then refills it to the STLb.

- **Case2: Shadow TLB miss, guest TLB miss, guest page table hit**

VMM first informs GOS to refill the GTLB according to the guest page table so that the GTLB instruction can be used. Secondly, combined with the HTLB, VMM generates the relationship between GVA and HPA and filled it into the STLb.

- **Case3: Shadow TLB miss, guest TLB miss, guest page table miss**

In this case, as there is an error of guest of its own, memory virtualization has nothing to do with it. To handle *TLB miss* exception, we must remove an existing entry to fill in a new mapping relationship.

A Round-Robin based exception handling mechanism (Fig. 5) is proposed in our virtual system. The TLB fill process is as following:

Step1: When GOS issues a *shadow TLB miss* exception, our system virtual machine will capture this exception.

Step2: Then VMM searches a hit entry for the *TLB miss* virtual address from the GTLB. If the GTLB misses, VMM refills this exception back into the GOS. If the GTLB hits, the VMM will fill the STLTB according to GTLB and HTLB.

Step3: When the exception is filled back into the GOS, guest will search guest page table to guarantee that GTLB can be refilled.

Step4: Then guest runs a TLB instruction of writing the mapping between GVA and GPA into GTLB. At the same time, VMM will capture the TLB instruction and handle it. After the TLB instruction exception has been handled, VMM will repeat the STLTB’s refilling process.

7 Experiments and Results Analysis

Both the function and performance tests are performed to verify the correctness of our memory virtualization method.

- Experimental environment

The experiment is performed on our system virtual machine which is running on a 4-core MIPS 64 R2 compatible platform (Fig. 6) for Loongson 3 Cloud Server. Experimental environments are shown in Table 4. Our virtual machine supports three modules: the kernel module running on the host OS, QEMU as a host process and GOS. QEMU is modified from the standard qemu-0.14.0 by adding supports to our system machine for Loongson 3 Cloud Server. Host OS is a modification of standard NeoKylin OS V.2.6.36.4 kernel which supports Loongson 3A processor by adding our virtual machine’s kernel module. GOS is a modification of standard NeoKylin OS V.

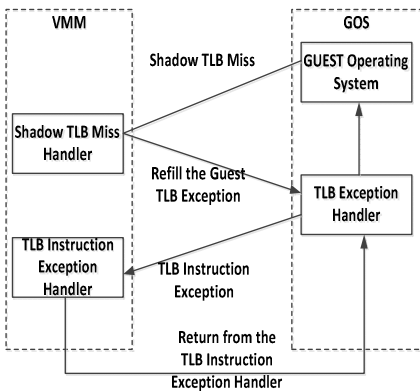


Fig. 5. STLTB miss exception handler

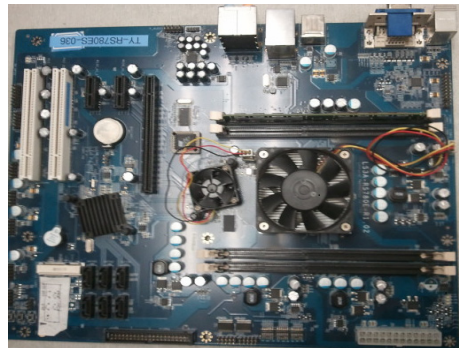


Fig. 6. Test board of a 4-core MIPS 64 R2 compatible platform for Loongson 3 Cloud Server

Table 4. Experimental environment

Hardware		Software	
Type	Configuration	Type	Configurations
The Type of CPUs	Loongson 3A	Release version of Host OS	NeoKylin
The number of CPUs	1	Kernel version of Host OS	Modified version of V2.6.36.4
The core number of one CPU	4	Release version of GOS	NeoKylin
Memory size	2GB	Kernel version of GOS	Modified version of V2.6.33.3
Chip set	AMD RS780E/SB700	QEMU of user space	Modified version of qemu-0.14.0
NIC	RTL8193	Compiler	Loongson compiler based on gcc
Disk	WESTDIGITAL 320G SATA	Benchmarking programm	nbench 2.2.3

2.6.33.3 kernel which supports Loongson 3A processor by moving its kernel address space and I/O space.

● Functional test

As our memory virtualization method is based on the 2.6.36 kernel running on a 4-core MIPS 64 R2 compatible platform, from Fig 7 and Fig 8, we can see that the

```
busClock=33000000, cpuclock=10000000, memsize=256
, highmemsize=0
bootconsole [early0] enabled
CPU revision is: 0006303 (ICT Loongson-2)
FPU revision is: 00770501
Checking for the multiply/shift bug... no.
Checking for the daddiu bug... no.
Determined physical RAM map:
 memory: 0000000100000000 @ 0000000000000000 (usable)
 memory: 0000000040000000 @ 0000000010000000 (reserved)
 memory: 0000000003ffffff @ 000000001c000001 (reserved)
```

Fig. 7. Guest booting process 1

```
Dentry cache hash table entries: 32768 (order: 2,
262144 bytes)
Inode-cache hash table entries: 16384 (order: 1, 1
31072 bytes)
Primary instruction cache 64kB, VIPT, direct mapped,
linesize 32 bytes.
Primary data cache 64kB, 4-way, VIPT, no aliases,
linesize 32 bytes
Unified secondary cache 512kB 4-way, linesize 32 b
ytes.
Memory: 251264k/262144k available (5893k kernel co
de, 10880k reserved, 1799k data, 320k init, 0k high
mem)
```

Fig. 8. Guest booting process 2

```
VFS: Mounted root (ext3 filesystem) on device 253:
1.
Freeing unused kernel memory: 320k freed
sh-3.2# ls
bin cdrom etc lib media opt root
selinux sys usr vmlinux
boot dev home lost+found mnt proc sbin
srv tmp var ~
```

Fig. 9. Guest boots successfully

system virtual machine successfully boots a Linux kernel with a simple user space init environment. By running some small programs in the guest, we prove the correctness of CPU virtualization and memory virtualization. In Fig. 7, there shows some memory information about memory size and high-memory. It is provided by the VMM. Fig. 8 shows some information about the cache and memory system. As VMM provides 256M memory to the guest, the last 3 lines list out the memory usage when the guest boots. Fig. 9 is a GOS screenshot and GOS boots up and runs well. Therefore, from Fig.7-9, it can conclude that our memory virtualization module function well to support the virtual machine.

- Performance test

We also had performance tests with NBench [9]. NBench is a synthetic computing benchmark program developed in the mid-1990s by the now defunct BYTE magazine intended to measure a computer’s CPU, FPU and Memory System speed. By completing 10 different types of benchmarks, the NBench tests the memory performance, the integer performance and floating point performance. The test result was then compared with an AMD K6-233 computer which runs Linux and takes the ratio as the performance measure.To highlight the work of memory virtualization, we only list the memory system testing data. The performance testing are conducted in two cases: 1-core host (Table 5) and 4-core host(Table 6).

Table 5. NBench memory test with single core host

Test case	Host (s)	Guest (s)	Performance percentage
1	2.449	0.288	
2	2.443	0.288	
3	2.465	0.289	
Ave.	2.452	0.288	11.75%

Table 6. NBench memory test with 4-core host

Test case	Host (s)	Guest (s)	Performance percentage
1	2.546	0.286	
2	2.505	0.288	
3	2.519	0.287	
Ave	2.523	0.287	11.38%

As the result listed in Table 5 and Table 6, we can see the memory system of host Loongson 3 server score in 2.44 to 2.55, and the guest with our memory virtualization method fraction at 0.288. The radio of the guest memory system and the host memory system is slightly more than 11.3%. As the first goal is to implement the VM and make it function well and no optimization has been conducted. In the performance tests, the performance of our memory virtualization system of the virtual machine does not meet our requirements. Now we are performing its performance optimization work. More macro-benchmark will be tested. Also, more multi-core system evaluation tools will be conducted further.

8 Conclusions

In this paper, by carefully summarizing the virtualization challenges of MIPS processor (Loongson 3) based Cloud Server, we proposed a memory virtualization method, which effectively utilized the Loongson 3 processor virtualization support characteristics like TLB, so as to overcome the problems such as non-virtualizable instruction set architecture, without hardware assistant and addresses translation from guest virtual address (GVA) to host physical address (HPA) supports. The functional and performance experimental results verified that our method functioned well.

Based on the previous work, we are now performing performance optimization work on MIPS processor based Cloud server. We believe that our research results will help researchers to better understand the critical issues of memory virtualization and system virtualization for not only Loongson but also MIPS processor based Cloud Servers. We hope that this work will also motivate virtual system designers to implement more system virtual machine for MIPS processors based Cloud Servers.

Acknowledgments. This work was supported by the Hi-tech Research and Development Program of China (863 Program) under Grant No. 2011AA01A205, National Natural Science Foundation of China under Grant No. 60973008, 60973007, 61003015, the fund of the State Key Laboratory of Rail Traffic Control and Safety (Contract No. RCS2008K001), Beijing Jiaotong University; Beijing Natural Science Foundation (4122042). We are grateful to those students and colleagues who participated in our Cloud Servers project from Beihang University, Institute of Computing Technology (ICT), Chinese Academy of Sciences and Lenovo Company.

References

1. MIPS architecture, http://en.wikipedia.org/wiki/MIPS_architecture
2. Institute of Computing Technology Chinese Academy of Sciences: Loongson 3A processor core manual (2009)
3. Wei, X., HuiXiang, W., LiMin, X., Li, R.: KVM for MIPS. In: The 2nd International Conference on Computer and Management (accepted)
4. Edouard, B., Scott, D.: Disco: Running Commodity Operating Systems on Scalable Multiprocessors. *ACM Transactions on Computer Systems* 4, 143–156 (1997)
5. Virtualization, <http://en.wikipedia.org/wiki/Virtualization>
6. Adams, K., Agesen, O.: A Comparison of Software and Hardware Techniques for x86 Virtualization. VMware (2010)
7. VMware Technology: Virtualization: Architectural Considerations and Other Evaluation Criteria. VMware (2010)
8. Eyad, A., Ernie, C., et al.: Verifying shadow page table algorithms. In: Proceedings of the 2010 Conference on Formal Methods in Computer-Aided Design (FMCAD 2010), pp. 267–270. FMCAD Inc., Austin (2010)
9. <http://en.wikipedia.org/wiki/NBench>

Implementation of a Distributed Data Storage System with Resource Monitoring on Cloud Computing^{*}

Chao-Tung Yang^{1,**}, Wen-Chung Shih², and Chih-Lin Huang¹

¹ Department of Computer Science, Tunghai University, Taichung, 40704, Taiwan ROC
ctyang@thu.edu.tw

² Department of Applied Informatics and Multimedia, Asia University, Taichung, 41354
Taiwan ROC
wjshih1@gmail.com

Abstract. This paper focuses on cloud computing infrastructure, and especially data services. The goal of this paper is to implement a high performance and load balancing, and able-to-be-replicated system that provides data storage for private cloud users through a virtualization system. The DaaS extends the functionality of the Hadoop distributed system (HDFS). The proposed approach also implements a resource monitor of machine status factors such as CPU, memory, and network usage to help optimize the virtualization system and data storage system. To prove and extend the usability of this design, a synchronize app was also developed running on Android based on our distributed data storage (DDS).

Keywords: Cloud Computing, Distributed data storage, DaaS, Distributed file system, Resource monitor.

1 Introduction

Cloud computing is a current trend and emerging computing platform and service mode that organizes and schedules services on the Internet. Once Internet protocols have been defined, resources can be connected and share information between layers [1]. The first layer is the cloud client. In the concept of cloud computing, the cloud client does not need powerful machines because the cloud takes care of most of the computing work or data. The cloud client may be a browser, lightweight program, or even a mobile device. The user can submit jobs using cloud client through a specified protocol. The second layer of the cloud is the cloud software, which may have complete functionalities. People can use the cloud software to send e-mail, modify photos, listen to music, and so on. The third layer is the cloud platform. The difference between the cloud software and the cloud platform is that the cloud

^{*} This paper was supported in part by the National Science Council, Taiwan ROC, under grant numbers NSC 100-2218-E-029-004 and NSC 100-2622-E-029-008-CC3.

^{**} Corresponding author.

platform provides a space in which developers can build their software. Developers do not need to focus on the systems' health. The final layer is the cloud infrastructure. Most of the cloud service aims to provide a mass of users, which requires high network traffic and much computing. Tens of thousands of racks must be connected together to fit the needs.

Cloud storage is a service that provides storage resource service through remote storage servers based on cloud computing. Cloud storage can provide storage services at a low cost with high reliability and security. A cloud storage system is a cooperative storage service system with multiple devices, many application domains, and many service forms. The development of a cloud storage system benefits from broadband networks, Web 2.0, storage virtualization [2-4], storage networks, application storage integrated with servers and storage devices, cluster technology, grid computing, distributed file systems, content delivery networks, peer-to-peer networks, data compression, and data encryption.

This paper focuses on cloud computing infrastructure, and particularly data services. The goal of this study is to implement a system that can provide data storage services for a private cloud used for a virtualization system and a public cloud for DaaS [5-8]. DaaS extends the functionality of a block distributed file system using HDFS [9-11, 45] and implements distributed data storage (DDS). The proposed approach implements a distributed resource monitor [12-16] of machine runtime factors such as CPU utilization, memory usage, and network flow. A block distributed file system enables web storage that can provide cloud software to store data. This system also requires high-performance data storage for creating redundant, scalable object storage using clusters of standardized servers to store petabytes of accessible data. It is not a file system or real-time data storage system, but rather a long-term storage system for more permanent, static data that can be retrieved, leveraged, and updated as necessary. This paper presents the details of the proposed design. To prove and extend its usability, a cloud program (app) running on Android used the proposed data storage services. This app can synchronize files, contact lists, messages, and phone settings between phones or PCs. Whenever the user uploads files, they are saved on the DaaS server. As soon as the file is modified, the app notifies the user or automatically updates the data.

2 Background

Data grid [17-23] or distributed storage systems focus on how storage can be used efficiently, and how users can share their resources in different regions. This type of system requires middleware to integrate and manage the distributed resources shared by users. Metadata are needed to record these distributed resources, and is used when a user queries a file or a resource.

This study proposes a next generation storage service to solve this problem. The cloud storage model provides online data storage services. The data are stored in multiple virtual or real machines called clusters. The service is usually managed by third-party companies who own a large data center. People pay according to their

usage and are not concerned with maintenance problems. Some examples include Amazon S3 [24, 25], EdgeCast [26], Ceph [29], Sector [30, 31], and HDFS [45], which is part of the Hadoop project in the Apache Software Foundation. HDFS is a Hadoop distributed file system based on Google GFS's [27, 28] approach. However, the HDFS is not suited for virtualization because of its authentication strategy. Ceph is a distributed network storage and file system designed to provide excellent performance, reliability, and scalability. Sector can be regarded as a distributed storage/file system. However, Sector is not a generic file system like NFS. Sector is designed for read-intensive scenarios. Because Sector makes replicas of files on different nodes within the system, reading is much more efficient than writing. This paper only compares the proposed approach to HDFS because it is more stable than Ceph and Sector.

Storage area network (SAN) [32] is an architecture that connects external storage and servers. The characteristic feature of this architecture is that a device connected to the server has direct access to the storage equipment. However, the cost of this approach remains too high for general use. Network attached storage (NAS) [33] is another data storage technology that can connect to the computer network directly and provide centralized data access to a heterogeneous network.

Green computing [38-40] attempts to effectively use energy through energy-efficient CPUs, servers, and peripherals, and reducing resource consumption. Green computing uses virtualization technology and power management to achieve energy saving and reduce carbon emissions. Virtualization [2-4] is one of the most effective tools in cost-effective, energy-efficient computing. This approach divides each server into multiple virtual machines that run different applications. This approach is so energy friendly that companies can increase their server utilization rates.

In virtualization computing, there is often the need to move a virtual machine (VM) from one computer to another. Therefore, both of the computers must obtain the same image from shared storage like SAN or NAS. The most well-known packages are iSCSI [34] and NFS [35]. Famous open source toolkits for cloud computing include OpenNebula [36] and Citrix XenServer [37], which use iSCSI or NFS to move the VM. However, both of the iSCSI and NFS are centralized storage. This creates a bottleneck in the network and often reduces system performance.

Representational State Transfer (REST) [41] is a type of software architecture for distributed hypermedia systems such as the World Wide Web. The concept of REST is to combine the two protocols HTTP and URL and how to apply in network software architecture design. REST treats software as a resource and addresses the position of resources using URL. The users can operate the resources by the methods defined by the HTTP protocol [42, 43]. The software that the REST called is a package contains data and methods of data processing. The HTTP defines six operations, and people often use four of them: POST, GET, PUT, and DELETE. The return method also consists of two parts: status code and content.

In the past, JAVA users could only use blocking I/O to build their applications. Every read/write operation is an independent blocking thread. This approach is easy to use and involves simple logic, but this comes at the expense of poor performance are more threads are created. New I/O, usually called NIO [44, 45], is a collection of

Java APIs that provide some features for intensive I/O operations. NIO was announced by Sun Microsystems with the JDK 1.4 release to complement an existing standard I/O. The NIO APIs were designed to provide access to the low-level I/O operations of modern operating systems. To avoid busy loops, NIO usually use “select” to dispatch operations. Each channel must register operation to the selector. The selector monitors the availability of these operations. This programming model can use only one thread to complete all functions, limiting the number of threads.

3 System Design and Implementation

Figure 1 shows the main components of the proposed system. This system uses web-based virtual machine management system to control the virtual machine cluster. Each node in the clusters has a daemon virtual machine controller. This system also contains a block distributed file system, distributed data storage (DDS), and file system management system. The file system management system controls the user account authentication and the space quota. The block distributed file system provides a web-based storage, and the DDS stores virtualization images. To monitor the whole system’s healthy and loading, the system also uses a resource monitor to gather information from each node. This paper takes over the file system, data storage, and resource monitor.

3.1 Block Distributed File System

We used HDFS to build the block distributed file system. Because of the authentication limitations of HDFS, it is unsuitable to connect to public cloud. Therefore, we designed an interface between HDFS and outside of the cloud. Developers may input or obtain their files through this interface. To be more efficient with HDFS, the system was optimized to speed up the transfer rates. HDFS was not designed for the public cloud. Thus, we developed an interface to exchange the data between private cloud and public cloud. The interface includes FTP protocol and JAVA library. However, most of the important commands were implemented in RFC 959.

Figure 2 shows how the proposed system supports HDFS over FTP. During an FTP connection, two transmission channels are open:

- A channel for commands (control channel).
- A channel for data.

Both the client and server have two processes that allow these two types of information to be managed:

- DTP (Data Transfer Process) is the process in charge of establishing the connection and managing the data channel. The server side DTP is called SERVER-DTP, and the client side DTP is called USER-DTP
- PI (Protocol Interpreter) interprets the protocol allowing the DTP to be controlled using commands received over the control channel. It is different on the client and the server.

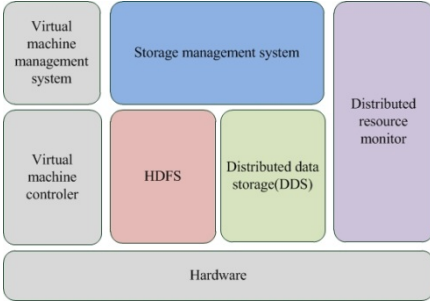


Fig. 1. Cloud Infrastructure system stack

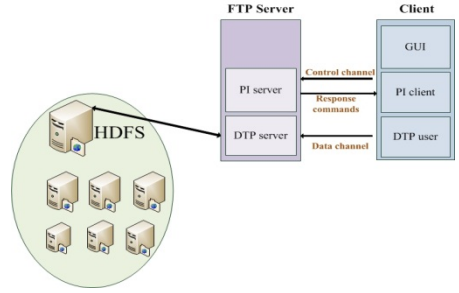


Fig. 2. FTP network flow

3.2 System Architecture of Distributed Data Storage

The reason for using block DFS to develop DDS stems from the different purposes of the users. The first scenario of the operation is virtual image placement. A user requests a VM and the virtual machine management system then obtains an image from the DDS. The second scenario is when a normal user uploads to or downloads data from the DaaS. The difference between the two scenarios is the data size. Suppose each of the images is at least 1 GB. In the second scenario, assume the data most of the users want to store in the DaaS is a document file, music, photos, or even high-quality pictures. With block DFS, the data are split into fixed chunks wherever DDS does not split data. This is one of the reasons for using block and DDS in the proposed DaaS. Equation (1) shows that if the data size S_d is larger than the chunk size S_c , the total transfer time T_{tc} increases, where S_t is transmit rate, T_{cl} is connection latency time, and finally, S_b is block size.

$$T_{tc} = \frac{S_d}{S_t} + T_{cl} \times \frac{S_d}{S_b} \quad (1)$$

In the real world, the image size is much larger than 1 GB and may be 10 GB to 50 GB. The block size might be 64 MB, using HDFS as an example. To obtain a 50 GB image takes $T_t + T_{cl} \times 800$ assuming that the transfer time is T_t and DDS only costs $T_t + T_{cl}$. Thus, it is much cheaper than block DFS and reduces the overhead to the data nodes.

The other reason for not simply using DDS and eliminating the block distributed file system is because DDS does not support the append feature and it is difficult to integrate with map-reduce. In the second scenario, the user may want to modify data or append new data at the end of the original data. It is a difficult task to add a feature to the DDS. We have to care for the consistence between each replica and lots of synchronization problems. In addition, when the system is big enough, there are many logs to be analyzed. It is unreasonable for a cloud provider to use a single machine for this. Thus, the proposed approach uses Map-Reduce technology to help reduce the time. As a result, the communication between the file systems to the map-reduce system is critical. One of the reasons map-reduce can be faster than a traditional

computing framework like MPI is because the scheduling is tight with data instead of managing tasks or jobs by grid. Lots of works remain to provide the data locality to map-reduce, and so on. The following sections introduce some special DDS strategies.

4 Experimental and Results

4.1 Experimental Environment

The previous sections demonstrate the design principle and implementation methods. This section presents several experiments conducted on seven machines within two switches. Each nodes contained 2-GE NICs, but had different CPU and memory levels. Figure 3 shows the network topology of the testbed. This experiment compared six topologies: DDS, DDS-Green mode, DDS-Net optimize, NFS, iSCSI, and HDFS. The DDS-Green mode means enable green strategy and set the “leastTotalSize” to 219902325552 bytes and set the “leastNodeSize” to 107374182400 bytes. This experiment reduced the data node to three. The DDS-Net optimize step split the network into upload channel and replication channel. The NFS was ver. 4 and the mount option was “rw, sync, no_subtree_check”. This experiment used Open-iSCSI ver. 2.0-871 and Hadoop 0.20.203.0. We used Linux command “dd” [46] to generate test data from 1KB to 32 GB. We also used dd to test the disk write performance for each node. The first experiment compared the performance of zero-copy IO and traditional IO to illustrate the level of speed enhancement provided by the zero-copy IO. The second and the third experiments is compared the performance of DDS, HDFS, iSCSI, and NFS in terms of upload and download speed. Both of the DDS and the HDFS were set to have 3 replicas.

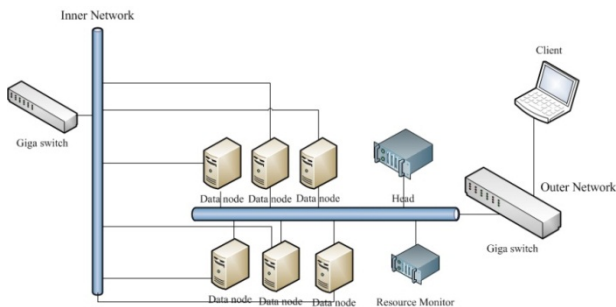


Fig. 3. Experiment network topology

4.2 Experimental Results

Figure 4 shows the performance enhancement using zero-copy IO. The **transferTo()** API decreases the time by approximately 65% compared to the traditional approach. This has the potential to increase performance significantly for

applications involving a great deal of copying of data from one I/O channel to another, such as storage systems. Fig. 5 shows the upload performance for a small data set, and Fig. 6 depicts a large data set. The DDS has six data nodes, the DDS-Green mode reduces number of data nodes to three, and the DDS-Net Optimize splits the upload channel and replication channel and has six data nodes. The HDFS based on Hadoop version 0.21.0 has six data nodes. Both the NFS and iSCSI have only two nodes during the test. Both DDS and Hadoop used their own copy commands, whereas the NFS and the iSCSI were mounted as a folder and used system copy command. The iSCSI was the fastest in the small file upload test. DDS with three different modes achieved similar results.

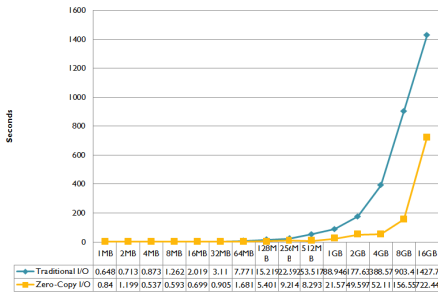


Fig. 4. Performance comparison between traditional IO and Zero-copy I/O

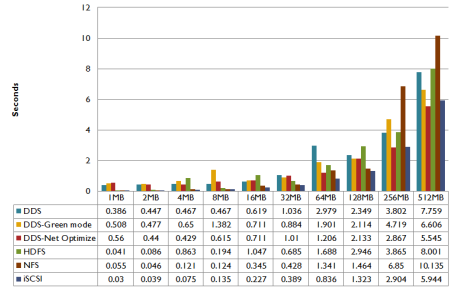


Fig. 5. Performance comparison between DDS, HDFS and NFS on upload data with small

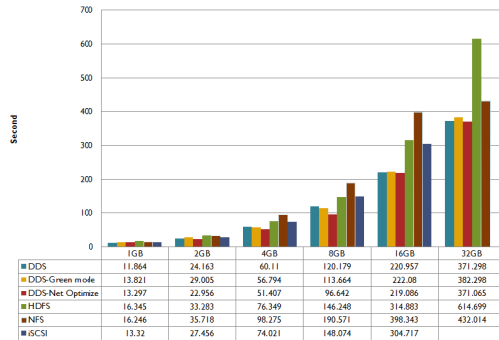


Fig. 6. Performance comparison between DDS, HDFS and NFS on upload data with large file

These figures show that the HDFS with one replica achieved the fastest performance. However, if we set the number of replicas to three, the performance decreases significantly, and the HDFS has the worst performance in the experiment. The DDS was the fastest one when the data size was up to 1GB, and the green mode achieved almost the same speed. The iSCSI achieved impressive performance under

512MB. In this experiment, only the HDFS split the data, suggesting that it incurs some overhead in splitting the data.

The third experiment tested the download speed (Fig. 7-8). Before the test, we assumed that the systems should have results similar to the upload test. With its write-once, read-many strategy, HDFS should have a faster download speed than upload speed. The experiment results are close to these assumptions. The HDFS reduces 20%~30% time in downloading when the data exceed 1GB, but shows little difference for data less than 512MB. When the data exceed 1GB, the NFS's performance deteriorates. The DDS and DDS with green mode achieved almost the same speed. The HDFS is a little slower and stable.

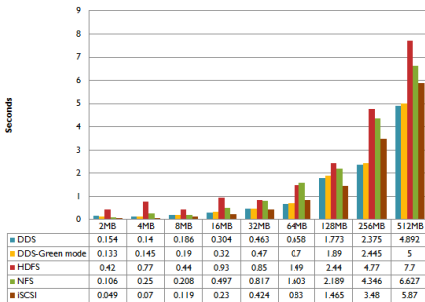


Fig. 7. Performance comparison between DDS, HDFS and NFS on download data with small files

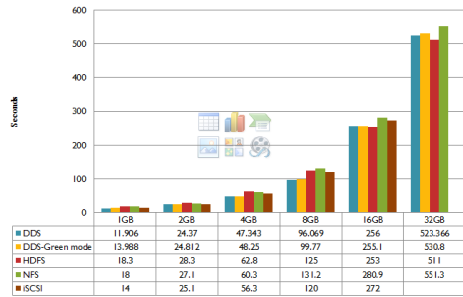


Fig. 8. Performance comparison between DDS, HDFS and NFS on download data with large files

5 Conclusions and Future Work

This study develops a high-speed, load-balanced, power-saving and reliable distributed data storage method to meet the needs of a virtualization management system. The DDS approach can use some special strategy like zero-copy to reduce transfer time and split the network to avoid unnecessary noise, whereas the green strategy saves power consumption and replication to increase reliability. For the public cloud, the proposed approach extends the functionality of HDFS by providing FTP and REST-like protocols for those who need cloud storage. This study also presents an Android app that helps people synchronize their data. We hope the proposed system can help administrators or developers to cut their work and double their output.

References

1. Cloud computing, http://en.wikipedia.org/wiki/Cloud_computing#Infrastructure
2. Milojićić, D., Llorente, I.M., Montero, R.S.: OpenNebula: A Cloud Management Tool. IEEE Internet Computing 15(2) (2011)

3. Sempolinski, P., Thain, D.: A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), December 3, pp. 417–426 (2010)
4. Cordeiro, T., Damalio, D., Pereira, N., Endo, P., Palhares, A., Gonçalves, G., Sadok, D., Kelner, J., Melander, B., Souza, V., Mångs, J.-E.: Open Source Cloud Computing Platforms. In: 2010 9th International Conference on Grid and Cooperative Computing (GCC), pp. 366–371 (2010)
5. Truong, H.L., Dustdar, S.: On analyzing and specifying concerns for data as a service. In: IEEE Asia-Pacific on Services Computing Conference, APSCC 2009, pp. 87–94 (2009)
6. Truong, H.-L., Dustdar, S.: On Evaluating and Publishing Data Concerns for Data as a Service. In: 2010 IEEE Asia-Pacific Services Computing Conference (APSCC), pp. 363–370 (2010)
7. Ju, D., Liu, C., Wang, D., Liu, H., Tang, Z.: Performance Comparison of IP-Networked Storage. *Tsinghua Science & Technology* 14(1), 29–40 (2009)
8. Wang, D.: Meeting Green Computing Challenges. In: International Symposium on High Density packaging and Microsystem Integration, HDP 2007, pp. 1–4 (2007)
9. Mackey, G., Sehrish, S., Wang, J.: Improving metadata management for small files in HDFS. In: IEEE International Conference on Cluster Computing and Workshops, CLUSTER 2009, pp. 1–4 (2009)
10. Shafer, J., Rixner, S., Cox, A.L.: The Hadoop Distributed Filesystem: Balancing Portability and Performance, pp. 122–133. IEEE, Houston (2010)
11. Jiang, L., Li, B., Song, M.: THE optimization of HDFS based on small files. In: 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT), pp. 122–133 (2010)
12. Barlet-Ros, P., Iannaccone, G., Sanjuas-Cuxart, J., Sole-Pareta, J.: Predictive Resource Management of Multiple Monitoring Applications. *IEEE/ACM Transactions on Networking* 19(3), 788–801 (2011)
13. Cheng, G., Gong, J.: A Resource-Efficient Flow Monitoring System. *IEEE Communications Letters* 11(6), 558–560 (2007)
14. An Adaptive Resource Monitoring Method for Distributed Heterogeneous Computing Environment. In: 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, pp. 40–44. Sch. of Comput. Sci., Northwestern Polytech. Univ., Xi'an, China (2009)
15. Miettinen, T., Pakkala, D., Hongisto, M.: A Method for the Resource Monitoring of OSGi-based Software Components. In: 34th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2008, pp. 100–107 (2008)
16. Wang, C.-C., Chen, Y.-M., Weng, C.-H., Chung, T.-Y.: An overlay resource monitor system. In: The 8th International Conference on Advanced Communication Technology, ICACT 2006, vol. 3, p. 5 (2006)
17. Düllmann, D., Hoschek, W., Jaen-Martinez, J., Segal, B.: Model for Replica Synchronization and Consistency in a Data Grid. In: The IEEE International Symposium on High Performance Distributed Computing, San Francisco, CA, USA, pp. 67–75 (2001)
18. Xu, P., Huang, X., Wu, Y., Liu, L., Zheng, W.: Campus Cloud for Data Storage and Sharing. In: Eighth International Conference on Grid and Cooperative Computing, GCC 2009, pp. 244–249 (2009)
19. Zeng, W., Zhao, Y., Song, W.: Research on Cloud Storage Architecture and Key Technologies. In: ICIS 2009, November 24–26. ACM (2009)
20. Zhan, Y., Sun, Y.: Cloud Storage Management Technology. In: Proceedings of the 2009 Second International Conference on Information and Computing Science, May 21–22, pp. 309–311 (2009)

21. Hirofuchi, T., Nakada, H., Ogawa, H., Itoh, S., Sekiguchi, S.: A live storage migration mechanism over wan and its performance evaluation. In: Proceedings of the 3rd International Workshop on Virtualization Technologies in Distributed Computing, Barcelona, Spain, June 15 (2009)
22. Bertino, E., Maurino, A., Scannapieco, M.: Guest editors' introduction: Data quality in the internet era. *IEEE Internet Computing* 14, 11–13 (2010)
23. Carns, P., Lang, S., Ross, R., Vilayannur, M., Kunkel, J., Ludwig, T.: Small-File Access in Parallel File Systems. In: Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium, pp. 1–11 (April 2009)
24. Amazon S3, http://en.wikipedia.org/wiki/Amazon_S3
25. Amazon Simple Storage Service, <http://aws.amazon.com/s3/>
26. EgeCast, <http://www.edgecast.com/>
27. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data. *ACM Transactions on Computer Systems* 26(2), Article 4 (June 2008)
28. Ghemawat, S., Gobiuff, H., Leung, S.-T.: The google file system. In: *SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, pp. 29–43. ACM Press, New York (2003)
29. Ceph, <http://ceph.newdream.net/>
30. Gu, Y., Lu, L., Grossman, R., Yoo, A.: Processing Massived Sized Graphs using Sector/Sphere. In: 3rd Workshop on Many-Task Computing on Grids and Supercomputers, co-located with SC10, New Orleans, LA, November 15 (2010)
31. Gu, Y., Grossman, R.: Sector and Sphere: The Design and Implementation of a High Performance Data Cloud. Theme Issue of the *Philosophical Transactions of the Royal Society A: Crossing Boundaries: Computational Science, E-Science and Global E-Infrastructure* 367(1897), 2429–2445 (2009)
32. SAN, http://en.wikipedia.org/wiki/Storage_area_network
33. NAS, http://en.wikipedia.org/wiki/Network-attached_storage
34. iSCSI, <http://en.wikipedia.org/wiki/iSCSI>
35. NFS, [http://en.wikipedia.org/wiki/Network_File_System_\(protocol\)](http://en.wikipedia.org/wiki/Network_File_System_(protocol))
36. OpenNeBula, <http://opennebula.org/>
37. Citrix XenServer, <http://www.citrix.com/>
38. Lo, C.-T.D., Qian, K.: Green Computing Methodology for Next Generation Computing Scientists. In: 2010 IEEE 34th Annual Computer Software and Applications Conference (COMPSAC), pp. 250–251 (2010)
39. Giroire, F., Guinand, F., Lefevre, L., Torres, J.: Energy-aware, power-aware, and Green Computing for large distributed systems and applications. In: 2010 International Conference on High Performance Computing and Simulation, HPCS, pp. lv – lxvii (2010)
40. Zhong, B., Feng, M., Lung, C.-H.: A Green Computing Based Architecture Comparison and Analysis. In: 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom), Green Computing and Communications (GreenCom), pp. 386–391 (2010)
41. Richardson, L., Ruby, S.: *Restful Web Services*, 1st edn., O'Reilly Media (May 15, 2007)
42. RFC 2616, <http://tools.ietf.org/html/rfc2616>
43. Fielding, R.T., Gettys, J., Mogul, J.C., Nielsen, H.F., Masinter, L., Leach, P.J., Berners-Lee.: RFC 2616: Hypertext Transfer Protocol – HTTP/1.1
44. NIO, http://en.wikipedia.org/wiki/New_I/O
45. Hadoop, <http://hadoop.apache.org>
46. dd, [http://en.wikipedia.org/wiki/Dd_\(Unix\)](http://en.wikipedia.org/wiki/Dd_(Unix))

Design, Verification and Prototyping the Next Generation of Desktop Grid Middleware*

Leila Abidi^{1,2}, Christophe Cérin¹, and Kais Klai¹


¹ Université de Paris 13, LIPN UMR CNRS 7030, 99, avenue Jean-Baptiste Clément, 93430 Villetaneuse, France

² Université de Tunis, LaTICE ESSTT, 5 Avenue Taha Hussein, BP, 56, Bâb Manara, Tunis, Tunisie
{leila.abidi,christophe.cerin,kais.klai}@lipn.univ-paris13.fr

Abstract. This paper proposes a formal framework for the design and verification of a new Desktop Grid (DG) prototype which is currently developed with Web 2.0 technologies and only with this technology. The paper is an approach for developing a new generation of Desktop grid middleware, in our case based on Redis, a key-value no-SQL Web 2.0 tool with capability for managing the Publish-Subscribe asynchronous paradigm. We propose to revisit the Desktop Grid paradigm based only on concepts from Web 2.0 tools. It is different from previous approaches that have required to build software layers before the layer of the DG middleware. We demonstrate that this corresponds to a progress in freeing time for modeling and verification, that is, to build safe middleware. This work proposes (1) a modeling and a verification of a DG protocol based on the Publish-Subscribe paradigm (2) a prototype of a new generation of DG middleware that we are developing, concurrently with the modeling. A simulation, according to a prototype is conducted on a local cluster and demonstrate that our system is operational, light in terms of coding lines and used resources. Thus, it offers remarkable properties in order to implement DGs on tablets and Smartphones, we mean on resource constrained systems.

Keywords: Desktop grid computing, Grid middleware, Volunteer Computing, Service-oriented computing, resource management, Redis, Web 2.0, Publish-Subscribe paradigm, Formal Models, Colored Petri Nets.

1 Introduction

Desktop Grid  systems represent an alternative to supercomputers and parallel machines and they offer computing power at low cost. Desktop grids (DGs) are made with PCs and Internet as the communication layer. DGs aim to exploiting the resources of idle machines over Internet. Indeed, Desktop Grids have important features that explain the large number of international projects aiming to better exploit this computational potential. Many Desktop Grid systems

* Experiments presented in this paper were carried out using the Paris 13 experimental testbed.

have been developed using a centralized model. These infrastructures run in a dynamic environment and the number of resources may increase dynamically.

The Seti@Home [2] project is among the large amount of success stories. While the increasing number of users of such systems demonstrates the potential of Desktop Grid, current implementations, for instance Boinc [3], United Devices [4], Distributed.Net [5] and XtremWeb [6] still follow the client-server or master/slave paradigm. Theoretically, the computing power that can be obtained from these systems is constrained by the performance of the master node.

The BonjourGrid [7-9] middleware has appeared in this context. The basic idea is to exploit, dynamically, different instances of DG middleware. The coordination is fully distributed through Publish-Subscribe mechanisms. BonjourGrid is the first middleware of this kind, to the best of our knowledge, and it provides a view of the architecture and of the execution of applications running inside the middleware that match the ideas of decentralization.

In this paper, we provide graphic models based on Petri Nets to verify that the Publish-Subscribe system (BonjourGrid is just a case study introducing our Redis based prototype) is correct. Our research has been built around the desire to develop the BonjourGrid protocol using colored Petri Nets as a technique for modeling and verification and to receive a feedback about the good practices in developing DGs in the context of Web 2.0 tools.

This paper is organized as follows. After an introduction of the context of this work, we set in Section 2 our problem in the form of key issues that summarize the different aspects to what we are interested. Therefore, we introduce the principle of resource coordination (as implemented in BonjourGrid) and the benefit in using Publish-Subscribe systems. We also introduce our motivations for the design and the formal verification of BonjourGrid. We conclude this section by presenting some related work. Section 3 is related to our contributions, and describes the different steps of our work in order to provide a formal specification of the Publish-Subscribe paradigm. We also present our views to mix our Publish-Subscribe substrate with the core part of the BonjourGrid protocol, and finally we present the software prototype based on Redis. Section 5 concludes the paper.

2 Context and Motivations

2.1 Key Issues in Designing DG Middleware

In this section, we introduce the different issues facing the design of a DG middleware:

- Heterogeneity and volatility of resources are the main characteristics of DG environment that make communication, coordination, and scheduling difficult tasks. That's why we need a powerful mechanism in the middle of our system in order to guarantee a minimum of robustness and safety;
- The communication paradigm (for coordination, not for exchanging data) adopted should provide a high level of asynchronism in order to promote scalability; The question is: what is the appropriate model for controlling and coordinating the components of a DG middleware?

- The systems become so complex that we must think to verify them formally. This will allow us to have more confidence in what we code. We promote a co-design between specification and implementation parts, and we want to isolate pieces of code (the generic patterns) that will be generated automatically from the specification; The question is: how to specify and verify grid middleware?
- Web 2.0 technologies are the future, hence it would be a benefit to take advantage of them. On one hand, the Web 2.0 technologies assist to advertise desktop grid goals and attract computational resources for desktop grid communities. On the other hand, Web 2.0 systems should handle heavy data traffic and complex relations that need extraordinary large computational power: grid technologies. The question is: how Web 2.0 and grids technologies may merge?
- Grid technologies may serve as building blocks for Cloud technologies. In [10], we have explained how the DG paradigm is reused for the SlapOS system which is a provisioning and billing system for the cloud. SlapOS¹ is part of a 2.3M euros FUI project in which we are working on the coordination of servers. The question is to isolate problems in clouds that could be solved with grid technologies.

2.2 Resources Coordination

Desktop grids are characterized by a dynamic environment due to the heterogeneity and volatility of resources, in our case PCs at home. User's machines can join or leave the grid at any time, without any constraint. Each machine has its own properties such as its memory size, bandwidth, CPU/core number... that makes difficult the scheduling of tasks.

Consequently, the power of DGs that resides in the participation of volunteers, constitutes also a weakness in terms of resources orchestration when a job is submitted. Thus, the main problem with DGs is coordination, in particular when we have to execute communicating applications i.e., applications that are modeled by a task graph with precedence.

To bypass these problems, BonjourGrid counts on a distributed vision for the coordination and the execution of applications based on existing DG middleware. Moreover, the coordination mechanism is based on the Publish-Subscribe paradigm.

2.3 The Publish-Subscribe Paradigm

The Publish-Subscribe paradigm is an asynchronous mode for communicating between entities. Some users, namely the subscribers, or clients, or consumers, express and record their interests under the form of subscriptions, and are notified later by another event produced by other users, namely the producers [11].

¹ <http://www.slapos.org/>

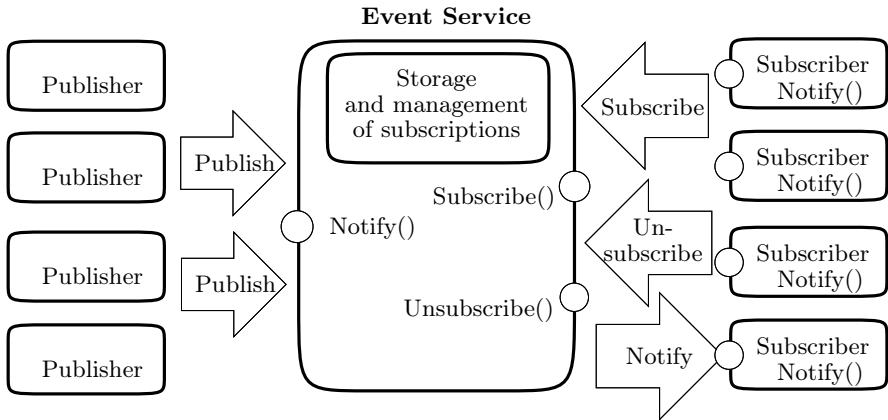


Fig. 1. The Publish-Subscribe paradigm [11]

As stated in Figure 1, subscribers record their interest by a call to the `subscribe()` operation inside the event service management system, without knowing the source of events. The `unsubscribe()` operation allows us to stop a subscription. The `notify()` (or `publish()`) operation is called by `publishers` in order to generate events that will be propagated to subscribers, and such events are managed by the event service management system too. Each subscriber will receive a notification for every event that is conform to its interest.

This communication mode is thus multi-point, anonymous and implicit. It is a multi-point mode (one-to-many or many-to-many) because events are sent to the set of clients that have declared an interest into the topic. It is an anonymous mode because the provider does not know the identity of clients. It is an implicit mode because the clients are determined by the subscriptions and not explicitly by the providers.

It is also known that this asynchronous communicating mode allows spatial decoupling (the interacting entities do not know each other), and time decoupling (the interacting entities do not need to participate at the same time). This total decoupling between the production and the consumption of services increases the scalability by eliminating many sorts of explicit dependencies between participating entities. Eliminating dependencies reduces the coordination needs and consequently the synchronizations between entities. These advantages make the communicating infrastructure well suited to the management of distributed systems and simplify the development of a middleware for the coordination of DGs.

2.4 BonjourGrid

BonjourGrid is an approach for the decentralization and the self organization of resources in DG systems [7-9]. The key idea is to exploit existing DG middleware (Boinc, Condor, XtremWeb) and concurrently to manage multiple instances of

DG middleware. The notion of meta desktop grid middleware has been introduced with BonjourGrid and the Publish-Subscribe paradigm is used intensively for the coordination of the different DG middleware.

Each user, behind a desktop machine in his office, can submit an application. BonjourGrid deploys a master (coordinator), locally on the user machine, and requests for participants (workers). Negotiations to select them should now take place. Using a Publish-Subscribe infrastructure, each machine publishes its state (idle, worker or master) when changes occur as well as information about its local load, or its use cost, in order to provide useful metrics for the selection of participants. Under these assumptions, the master node can select a subset of workers nodes according to a selection criteria. The master and the set of selected workers build the Computing Element (CE) that will execute and manage the user application. When the execution of an application of a CE terminates, its master becomes free, returns in the idle state, and it releases all workers who return to the idle state. Then, the nodes can participate to others projects.

To implement this approach, BonjourGrid has been decomposed in three fundamental parts: a) A fully decentralized resources discovery layer, based on Bonjour protocol [12]; b) A CE, using a Desktop Grid (DG) middleware such as XtremWeb, Condor or Boinc, which executes and manages the various tasks of applications; c) A fully decentralized protocol of coordination between a) and b) to manage and control all resources, services and CEs.

2.5 Related Work

Papers about Publish-Subscribe systems [13-15] are invitations to investigate more deeply the BonjourGrid protocol, in particular under the point of view of the verification of a distributed system. In this section we review the related works about publication-subscription systems and approaches for modeling and verifying formally such systems.

Publish-Subscribe systems concern both companies and researchers. Standards and industrial products are directly based on this paradigm, for instance the Bonjour protocol from Apple. For researchers, most of the works concern the problem of constructing a system as perfect as possible in terms of scalability, efficiency and safety. But they do not focus enough on the problem of the formal analysis of the accuracy of such systems. However, some research papers, such as [13-15], investigate the formal verification of Publish-Subscribe systems.

The work of Abidi and al. [16] focuses on the modeling of the BonjourGrid protocol. Authors have isolated the generic mechanisms of construction for the Publish-Subscribe approach. Then, they have modeled and verified, based on those mechanisms, the BonjourGrid protocol that allows the coordination of multiple instances of desktop grid middleware. Formal modeling allowed them to verify the adequacy of BonjourGrid with respect to the coordination of resources and to have a "composition" mechanism for integrating any protocol based on the Publish-Subscribe paradigm. All these ideas were illustrated along the BonjourGrid case study and they constitute a methodology for building Publish-Subscribe systems.

In this paper we continue this work, and we propose a more sophisticated formal model. We remind that in [16] we did not cover all the BonjourGrid protocol for the sake of simplicity whereas, in this paper we cover different internal details in order to have a realistic simulation of the actual behavior of the system.

In [13, 14], the authors propose an approach for modeling and validating systems. This approach is based on an architecture of components that react to events. In these works, the components are specified with UML state-transition diagrams. Formal verification is achieved through model checking (using SPIN). But instead of using the formulas of linear temporal logic (LTL) of SPIN, the authors have interpreted the properties as automata. According to them, this will represent more complex properties required to validate the modeled system.

Although our modeling approach is also based on component reacting on events, we do prefer to enjoy the advantages of temporal logic for the formal verification, in particular by using the ASK-CTL library.

In [15], the authors describe a generic framework dedicated to modeling and formal verification of Publish-Subscribe mechanisms. Their system is based on a model of states machine providing management of events during the execution of the publication-subscription protocol. The framework takes as input a set of components and a set of properties for the Publish-Subscribe mechanism. The matching of the two sets is subsequently validated using model checking tools. This system is regarded primarily as a generic framework in which there is always the risk of not providing a model and an audit tailored to each specific case for the Publish-Subscribe mechanism. Our approach is a successful modeling and formal verification. It is suitable for BonjourGrid while isolating the Publish-Subscribe mechanism.

In [17, 18], the authors, motivated by the benefits of formal analysis, build coordination protocol for a formal model using colored Petri Nets. To evaluate the accuracy of their model and as a result of their protocol, they checked the behavioral properties formally, and implemented a mechanism of CTL model checking. Our work is built around the protocol proposed by the authors. From our side, we also capitalize on the use of colored Petri Nets and CTL logic.

In [19], the authors present a new approach to modeling and formal verification, dedicated to software components. Their methodology is based on a software architecture-driven and the reuse of Petri Nets models. Their contribution is rather a new approach for visual composition, formal verification and validation of software systems. The work is built primarily around software components.

In [20], the authors present an overview of the analytical performance of colored Petri Nets in particular by using the CPN Tools. They use it to collect data during simulations, to generate different results on the performance and to implement several cases of simulation. A simple protocol is used to illustrate these aspects.

3 Contributions

3.1 Analysis and Criticisms

Modeling may guide the development of our forthcoming prototype that will serve for the validation of ideas and choices made during the design part.

The verification is for proving that the protocol is correct, then that any analyzed configuration will produce the "good" answer. This goal requires the formal verification of properties that we expect for the protocol: safety, for which the absence of deadlock is an example, and liveness.

In [16], we have modeled the BonjourGrid protocol according to the colored Petri Net [21] formalism and we have used the CPN Tools² for that purpose. CPN Tools is a fast and efficient simulator that handles both untimed and timed nets. Full and partial state spaces can be generated and analyzed, and a standard state space report contains information such as liveness properties. By means of a simple query language it is possible to specify and to check system specific properties.

The work introduced in [16] suffer from the fact that it is too specific to a dedicated protocol. In the current work, we have a more agnostic approach: we separate the Publish-Subscribe mechanisms and what is specific to the Bonjour-Grid protocol. The BonjourGrid serves as a guideline, a concrete example. We are looking for a "universal" Petri Net for the Publish-Subscribe paradigm on top of which any protocol based on Publish-Subscribe could be built and verified.

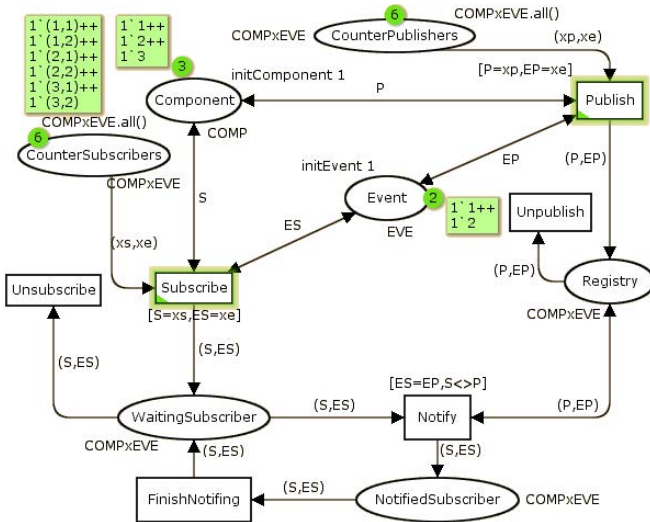


Fig. 2. Colored Petri Net for the Publish-Subscribe protocol

² See <http://www.daimi.au.dk/CPNtools>

3.2 A Colored Petri Net Model for the Publish-Subscribe Paradigm

For pedagogical reason, we start by explaining the Petri Net model we obtained for SCC in [16]. We show how the initial core idea was captured and it is essential to understand the concept and the model.

The colored Petri Net model in Figure 2 models the Publish-Subscribe paradigm. It introduces an "initial" state with a definite number of components and events. Each component can be a publisher (represented by **EP** for Event-Published on the figure), or a subscriber (represented by **ES** for EventSubscribed on the figure) or both.

This model is compliant with the Publish-Subscribe protocol, since a component can publish an event as many times as it wants; it can also subscribe to an event as many times as it wants. An event can be issued by one or more components, and one or more components can subscribe to this event.

Published events are saved in a directory that is modeled by the place "Registry". When a component subscribes to an event **E**, it goes to the place "WaitingSubscriber". Once the event is published, the transition "Notify" can be fired. A condition should be checked when we fire the transition "Notify": "a component cannot subscribe to the event it published", which was modeled using the guard $[S <> P]$.

Hence, we have successfully achieved our first aim by defining a colored Petri Net for which we can build representations of any protocol that is written with the publish-subscribe paradigm in mind. In the next section, we model the BonjourGrid protocol built on top of that Publish-Subscribe Petri Net model.

3.3 A Colored Petri Net Model for the BonjourGrid Protocol

Figure 3 illustrates the current methodology used to compose any Publish-Subscribe protocol. This figure is related to the central part of the BonjourGrid protocol and constitutes a contribution of this paper. Indeed, this paper exhibits a major refinement of the initial specification as published in [16].

The new refinements cover different internal details of the BonjourGrid protocol around the central part. The difficulty for the designer is to have a global view of the overall behavior of such a system where asynchronism is the essential criterion.

In Publish-Subscribe architectures, components communicate with each other through the exchange of events. Thus, any model of these architectures must explicitly consider the two main actors: components and events, and the three main services: publish, subscribe and notify.

The first actor stands for publishing an event **ep** from a machine **c**, the second one is for subscribing an event **es** coming from a machine **c**, and the third one is for notifying events to interested machines.

BonjourGrid modeling is focused on that aspect.

In fact, the challenge was to consider a "black box" (the Petri Net for the Publish-Subscribe model which is represented in the center of Figure 3) that cannot be modified and to try to specify the behavior of BonjourGrid as external events of the black box. By doing this, we exhibit a general methodology.

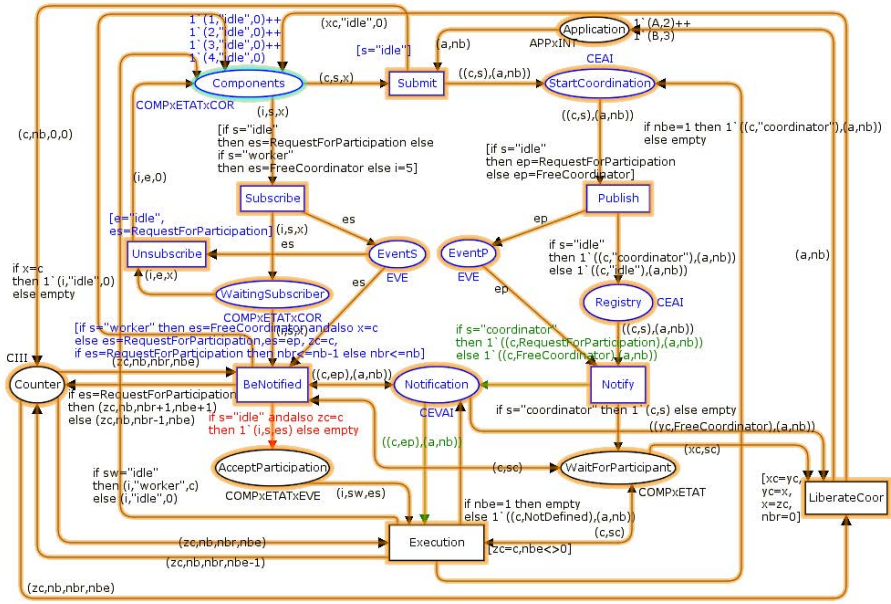


Fig. 3. BonjourGrid model, composed on top of a Publish-Subscribe Petri Net

The main idea is to plug the BonjourGrid protocol on top of the Publish-Subscribe protocol. The later is mainly presented by the cycle: "publication", "subscription" and "notification". BonjourGrid elements are plugged on inputs and outputs of this cycle.

In BonjourGrid model we just blow up the state "Event", previously represented in Publish-Subscribe model, into two states "EventS" and "EventP" to differentiate between events that are published and those for which components can subscribe, in order to have more clarity.

We represent each component in the place *component* by the tuple $(compId, state, coordId)$ where $compId$ is the identifier of the component, $state$ can take the values : "idle", "worker", "coordinator" that represent the different states a component may take, $coordId$ is the identifier of the coordinator attached to the component/worker; it takes the value 0 when the component is not a worker.

Firstly, a component may submit an application a , and publish a "RequestForParticipation" event associated to that application. the state *Application* is represented by the couple (a, nb) where a is the identifier of the application and nb is number of participant required to execute this application.

In parallel, other components (in "idle" states) may subscribe to that "RequestForParticipation" event. A coordination starts when subscribed machines are notified by their corresponding events and accept to participate to the execution of the application. All the subscribers on that event are notified by its publication, but only the number nb required by the application can move to

"AcceptParticipation" state to assist in its execution. The place "counter" serves to ensure that task. In this step, the state "idle" becomes "worker". Coordinators who have required in their support applications a number of participants that is not provided yet, must wait in the place "WaitForParticipant" until the required number \mathbf{nb} becomes available (simulated with the place "WaitForParticipant"). The execution of the application can start when the required number \mathbf{nb} of participants is reached. All these coordinating steps will be locked in the place "WaitForParticipant" until the application is completed.

Once the execution is completed, we move to the step of releasing the coordinator and the components that are attached. Thus, workers may subscribe to the "FreeCoordinator" event that would allow to release them. When coordination is finished, the coordinator component publishes the "FreeCoordinator" event then subscribed workers are notified. They are released and the state moves from "worker" to "idle".

By doing this, we have successfully achieved our second aim by formally modeling the BonjourGrid protocol built on top of that Publish-Subscribe Petri Net. Using CPN Tools, we then performed simulations of the net to gain more confidence in our model. During this first step of the simulation, no straightforward problems was discovered. The next step was naturally to perform an exhaustive simulation exploring all the possible states of the system, i.e., its state space.

For the verification of the desired properties (liveness for instance), our goal was to keep the same level of abstraction as the modeling of the CPN. For these reasons, we did not use any temporal logic formalisms since they work at a different level of abstraction and, thus, it can be difficult to use. We exploited the tools provided by CPN Tools, which allow us to calculate and analyze state spaces. With these tools, the standard queries for the verification require no programming at all.

Fundamental properties we want to verify on that model were:

- Any event produced (published) must be received by all interested consumers (subscribers).
- A coordinator \mathbf{C} begins execution of its implementation if and only if there exists at least one machine \mathbf{M} that agree to participate with \mathbf{C} .
- If a coordinator \mathbf{C} publishes the event "FreeCoordinator" then all Workers for that coordinator will eventually switch to the "idle" state.
- Any worker \mathbf{W} can be attached to a single coordinator \mathbf{C} .

Figure 4 provides a sample of the state space report provided by CPN Tools.

The analysis of these small configurations gives us more confidence in the BonjourGrid system especially considering the following facts:

- We have not found any deadlock states (i.e., states that do not admit executable transitions). The absence of such state is obviously required in our context.
- All possible transitions are executable. Hence, our specification seems to be correct from the perspective of event triggering: all possible events can eventually happen.

- All state spaces built are composed of a single strongly connected component. It seems therefore impossible to be trapped in a specific or undesired system configuration. This liveness property is indeed a crucial prerequisite for our system.

```

State Space
  Nodes:11296  Arcs:38044
  Secs:22      Status:Full
Scg Graph
  Nodes:1  Arcs:0  Secs:1
Home Properties
-----
Home Markings  All
Liveness Properties
-----
Dead Markings  None
Dead Transition Instances  None
Live Transition Instances  All

```

Fig. 4. A partial view of the report of the BonjourGrid model

3.4 A Prototype Based on Redis

In parallel with the modeling and verification parts, as introduced in the previous section, we have built a prototype of a Desktop Grid middleware based on BonjourGrid and Redis and on top of Python. Redis³ is an open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings, hashes, lists, sets and sorted sets. Moreover, it implements a Publish-Subscribe layer. Indeed, we have an "all-in-one" tool that fulfills our initial needs: storage of codes we have to execute, storage of input/output data, support for Publish-Subscribe. This explains why we have declined the opportunities to work with XMPP, Nodejs like tools, or the promising Hookbox interface which is a Comet server and message queue that tightly integrates with Web application frameworks.

Redis do not offer a strong support for authentication, cryptographic communication or data protection but it is well targeted for our work related to prototyping. We assume that in the future, the community developers will offer such properties. However, notice that Redis supports some sort of server redundancy to make the system fault tolerant.

Our prototype does not include yet all the components of DG middleware such as those we find in Boinc, Condor, or XtremWeb. For instance we do not have a component for doing result certification or we do not manage fault tolerance issues by task duplication. Again, the prototype is currently devoted to the understanding of the core decentralized protocol for the coordination of resources and it would be too challenging to hope that we will solve all the problems in one shot. We do prefer to work "step by step".

Our prototype is organized according to the following Python classes:

³ See <http://www.redis.io>

- `ServerClass`. There are three possibilities for servers: the main server for the protocol itself, the data server storing input/output data for the applications and `CodeServer` which is the name of the server for retrieving codes of applications; The three servers can point on the same name;
- `DataManager`: set-up an instance of `ServerClass`; In this class, we also define functions for loading files into a Redis server, for executing a code (binary or script);
- `FormMultiprocessingTasks`: it is the core of the "workflow" engine because the method in this class "executes" the task graph; Note that we create threads to execute, concurrently, all the tasks (descendants in the task graph) attached to a node;
- `EngineClass`. It starts and instance of the `FormMultiprocessingTasks` class in one thread, and wait that all nodes of the task graph are visited in another thread;
- `MachineClass`: it allows to set the properties of a machine (operating system type, amount of memory on the machine, processor type...); In this class we have also two methods for launching a worker and a coordinator respectively;
- `WorkerClass` and `CoordinatorClass` define the behavior of a worker, or a coordinator;

Our prototype executes series-parallel graphs (SPG). Intuitively, a SPG is built from a sequence of compositions (parallel or series) of smaller-size SPGs. The smallest SPG consists of two nodes connected by an edge. The first node is the source of the SPG while the second is its sink. When composing two SPGs in series, we merge the sink of the first SPG with the source of the second. For a parallel composition, the two sources are merged, as well as the two sinks. In our prototype we execute the graph depicted on Figure 5.

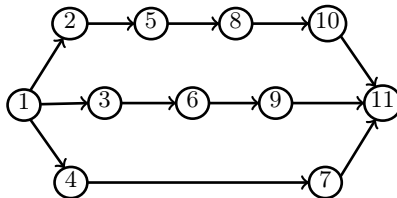


Fig. 5. The graph executed by our prototype. The initial state is 1, the final state is 11.

Classical workflow applications usually consists of a directed acyclic graph: the application is made of several tasks, and there are dependencies between these tasks. However, it turns out that many of these graphs are series-parallel graphs. For instance, in [22], McClatchey et al. introduce a prototype scientific workflow management system called CRISTAL, and the distributed scientific workflow applications that they consider are SPGs. In [23], Qin and Fahringer

use scientific grid workflow applications, which are all structured as SPGs: the WIEN2k workflow performs electronic structure computations of solids; the Me-teoAG workflow is a meteorology simulation application [24], and the GRASIL workflow computes the spectral energy distribution of galaxies [25]. A last, the fMRI workflow [26], which is a cognitive neuroscience application is also based on SPGs.

The simulation implemented in the prototype⁴ is as follows. A series-parallel graph with 11 vertices is defined. Each edge represents an application. The application is the same for all edges: it is a Bash script that should be located on /tmp and echoing a message.

The program forks and we start a coordinator as well as a worker (we use only one worker which is created for executing one application, one by one). The program is multi-threaded in the sense that we execute the descendants of a node (the applications) in different threads. We manage locks so that we cannot predict which is the order of execution of the descendants.

The coordinator is in a loop doing the work described in this paragraph. It publishes a message to advertise that he needs a worker for executing a task (according to some properties). Worker(s) listen on the channel dedicated to this message, then reply by publishing a random channel name for future communication between him and the coordinator. The coordinator accepts the first response (the worker who is arriving the first) and he publishes on the random channel name the name and the location of the application. The worker keeps the application name and executes it.

It is important to notice that the protocol is entirely depicted by the exchange of publication and subscribe messages and the development is guided by our different modeling. The prototype serves for the validation of ideas and choices made during the design and modeling parts.

Note also that in the prototype we do not manage files representing the input/output data of the applications. This point is related to scheduling strategies and we plan to include a more elaborated scheduling class in our prototype in the future. In the context of Desktop Grid, scheduling should serve also to check the results computed on (hostile) workers. The issue is to write scheduling algorithms in terms of the Publish-Subscribe paradigm and in such a way that the strategies could be composed with the current Petri Net.

4 Conclusion

In this paper, we have introduced the context of our work about the coordination of resources using the Publish-Subscribe paradigm. We have also demonstrated the usefulness of modeling and formal verification of such a specific mechanism for the BonjourGrid system, dedicated to the management of multiple instances of Desktop Grid middleware.

This work is a step toward the development of DG middleware based on Web 2.0 technologies. Furthermore, this effort has been consolidated in this paper

⁴ See <http://www.lipn.fr/~cerin/ProtoRedis.tar>

with another facet of the problem, namely the definition of a mechanism for composition of the basic scheme introduced in this paper with any protocol that is written with the Publish-Subscribe paradigm in mind. Recall that BonjourGrid is only a case study that allows us to discuss the problems and devise solutions that we have implemented, concurrently with the design, in Redis.

We are currently working on the programming effort for introducing a Python module for scheduling jobs and later, we will conduct experiments on large clusters running Redis. Scheduling is an important issue because we want to do result certification: the computation are done on (hostile) workers so we need to do redundant computation. We have imagined an algorithm based on tickets that we duplicate and managed by the Publish-Subscribe paradigm. Again, the scheduling itself is made exclusively through a Publish-Subscribe approach which is unconventional, but allows to build a fully distributed protocol where asynchronism is maximized because of scalability requirement of systems we build nowadays.

References

1. Kondo, D.: Preface to the special issue on volunteer computing and desktop grids. *J. Grid Comput.* 7, 417–418 (2009)
2. University of California: SETI@Home (October 2011), <http://setiathome.berkeley.edu/>
3. University of California: BOINC (October 2011), <http://boinc.berkeley.edu/>
4. Univa: United Devices (October 2011), <http://www.unicluster.org/>
5. DistributedNet: Distributed.Net (October 2011), <http://www.distributed.net/>
6. Univa: XtremWeb (October 2011), <http://www.xtremweb.net/>
7. Abbes, H., Cérin, C., Jemni, M.: Bonjourgrid as a decentralised job scheduler. In: APSCC, pp. 89–94. IEEE (2008)
8. Abbes, H., Cérin, C., Jemni, M.: Bonjourgrid: Orchestration of multi-instances of grid middlewares on institutional desktop grids. In: IPDPS, pp. 1–8. IEEE (2009)
9. Abbes, H., Cérin, C., Jemni, M.: A decentralized and fault-tolerant desktop grid system for distributed applications. *Concurrency and Computation: Practice and Experience* 22, 261–277 (2010)
10. Smets-Solanes, J.P., Cérin, C., Courteaud, R.: Slapos: A multi-purpose distributed cloud operating system based on an erp billing model. [27] , 765–766
11. Eugster, P.T., Felber, P., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/subscribe. *ACM Comput. Surv.* 35, 114–131 (2003)
12. Cheshire, S., Steinberg, D.H.: Zero configuration networking - the definitive guide: things that just work: covers Apple's Bonjour APIs. O'Reilly (2005)
13. Zanolin, L., Ghezzi, C., Baresi, L.: An approach to model and validate publish/subscribe architectures (2003)
14. Harrison, M.D., Kray, C., Sun, Z., Zhang, H.: Factoring user Experience into the Design of Ambient and Mobile Systems. In: Gulliksen, J., Harning, M.B., van der Veer, G.C., Wesson, J. (eds.) EIS 2007. LNCS, vol. 4940, pp. 243–259. Springer, Heidelberg (2008)
15. Garlan, D., Khersonsky, S., Kim, I.: Model Checking Publish-Subscribe Systems. In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 166–180. Springer, Heidelberg (2003)

16. Abidi, L., Cérin, C., Evangelista, S.: A petri-net model for the publish-subscribe paradigm and its application for the verification of the bonjourgrid middleware. [27], 496–503
17. Kacem, N.H., Kacem, A.H., Jmaiel, M., Drira, K.: Towards modelling and analysis of a coordination protocol for dynamic software adaptation. In: Chbeir, R., Badr, Y., Abraham, A., Laurent, D., Köppen, M., Ferri, F., Zadeh, L.A., Ohsawa, Y. (eds.) CSTST, pp. 499–507. ACM (2008)
18. Kacem, N.H., Kacem, A.H., Drira, K.: A formal model of a multi-step coordination protocol for self-adaptive software using coloured petri nets. *International Journal of Computing and Information Sciences* (2009)
19. Silva, L.D.D., Perkusich, A.: Formal verification of component-based software systems. In: Isaías, P.T., Sedes, F., Augusto, J.C., Ultes-Nitsche, U. (eds.) NDDL/VVEIS, pp. 113–124. ICEIS Press (2003)
20. Wells, L.: Performance analysis using cpn tools. In: Lenzini, L., Cruz, R.L. (eds.) VALUETOOLS. ACM International Conference Proceeding Series, vol. 180, p. 59. ACM (2006)
21. Jensen, K., Kristensen, L.M.: *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st edn., July 1. Springer, Heidelberg (2009)
22. McClatchey, R., Estrella, F., Le Goff, J.M., Kovacs, Z., Baker, N.: Object databases in a distributed scientific workflow application. In: *Proceedings of the 3rd Basque International Workshop on Information Technology (BIWIT 1997)*, p. 11. IEEE Computer Society, Washington, DC (1997)
23. Qin, J., Fahringer, T.: Advanced data flow support for scientific grid workflow applications. In: *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC 2007*, pp. 42:1–42:12. ACM, New York (2007)
24. Schüller, F., Qin, J., Nadeem, F., Prodan, R., Fahringer, T., Mayr, G.: Performance, scalability and quality of the meteorological grid workflow meteog. In: *Proceedings of the 2nd Austrian Grid Symp., Univ. Innsbruck* (2006)
25. Silva, L., Granato, G.L., Bressan, A., Lacey, C.G., Baugh, C.M., Cole, S., Frenk, C.S.: Modeling dust on galactic sed: Application to semi-analytical galaxy formation models (1999)
26. Zhao, Y., Wilde, M., Foster, I., Voekler, J., Jordan, T., Quigg, E., Dobson, J.: Grid middleware services for virtual data discovery, composition, and integration. In: *2nd Workshop on Middleware for Grid Computing*, p. 57. ACM Press (2004)
27. Jacobsen, H.A., Wang, Y., Hung, P. (eds.): *IEEE International Conference on Services Computing, SCC 2011, Washington, DC, USA, July 4-9*. IEEE (2011)

A Request Multiplexing Method Based on Multiple Tenants in SaaS*

Pingli Gu¹, Yanlei Shang¹, Junliang Chen¹, Bo Cheng¹, and Yan Jiang²

¹Key Laboratory of Networking and Switching Technology
Beijing University of Posts & Telecommunications, Beijing, China
gplqy98@gmail.com, {chjl, shangyl, chengbo}@bupt.edu.cn

²Shandong Polytechnic University, Jinan, Shandong, China
JiangYan@gmail.com

Abstract. As one of the key characteristic of SaaS, multi-tenant aims to support massive customers. To achieve the high economies of scale, SaaS provider hope to minimize the overall infrastructure cost without adversely affecting the customers or maximize the number of customers in the context of a given infrastructure cost. To maximize the number of customers in the context of a given infrastructure cost, that is maximize throughput of SaaS application, we propose request multiplexing method based on multiple tenants, in this method, we use the ideal of network coding to encode or decode information in multiplexing request or reply. Based on our method, the simulation experiment result shows the high throughput and better performance of SaaS application.

Keywords: Multi-tenants, request multiplexing, network coding, SaaS.

1 Introduction

With development of cloud computing, more and more software providers begin to provide SaaS application. As one of the key characteristic of SaaS, multi-tenant aims to support massive customers [1]. To achieve the high economies of scale, SaaS provider hope to minimize the overall infrastructure cost without adversely affecting the customers[2] or maximize the number of customers in the context of a given infrastructure cost. To maximize the number of customers, SaaS providers must deal with tenant requests as much as possible, and ensure reliability of SaaS application. Normally, when the number of client requests is the maximum number of requests processed by system in unit time, the SaaS application goes to the maximum throughput. In this case, SaaS application is easy to overload. In order to prevent overloading, [8] proposed a performance regulator based on feedback-control, and

* This work was supported by the National Grand Fundamental Research 973 Program of China under Grant No.2011CB302506; National Natural Science Foundation of China under Grant No. 61001118, 61132001; Project of New Generation Broadband Wireless Network under Grant No.2010ZX03004-001; Program for New Century Excellent Talents in University (Grant No. NCET-11-0592); Fundamental Research Funds for the Central Universities under Grant No.2011RC0502.

[5][6][7] proposed some methods about resource allocation. From these methods, we can see that, each request is handled separately, and each request gets an answer from system separately, independent client requests may share system resources, but the information in client requests or in system replies is still separate, and then the throughput of system is always limited.

Network coding [3] is a recent field in information theory. Instead of simply forwarding data, nodes may recombine several input packets into one or several output packets [4]. A simple example is butterfly network, as shown in Figure 1.[3]

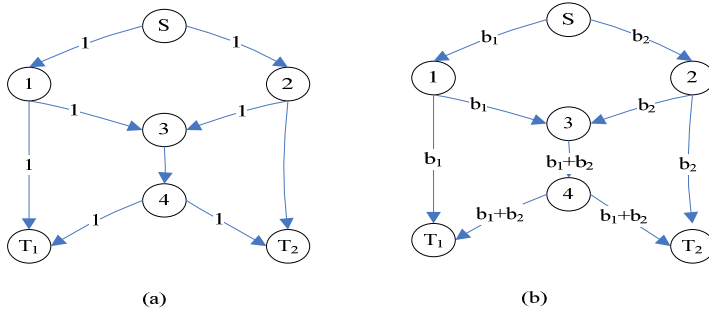


Fig. 1. Butterfly network with network coding

Fig. 1(a) shows the capacity of each edge, and Fig. 1(b) shows such a scheme, where “+” denotes modulo addition. At T_1 , b_2 can be recovered from b_1 and b_1+b_2 . Similarly, b_1 can be recovered at T_2 . In this example, information is coded at the node 3, which is unavoidable. For network throughput $L \geq 2$, network coding is in general necessary in an optimal multicast scheme. Because of network coding, we can save bandwidth, and throughput can be increased when network coding is allowed.

This paper aims to improve throughput of SaaS application by sharing requests and replies on the basis of theory about network coding. In SAAS application, tenant requests always access the same system resource, such as the same DB, the same files and so on, for example, for DB, there are some restrictions such as connection number and so on, because SaaS system processes these requests separately, it can only meet limited requests. Therefore, in order to achieve better throughput, our work proposes request multiplexing method to share requests and replies of different tenants by encoding or decoding information, and hope to recombine several requests into one request or reply, it means that one request may includes more information from other requests, and system will handle all these requests in one request. In this method, we primarily focus on two key issues, one is how to share requests, and the other is how to separate yourself response from a share reply. To solve the two problems, we use the ideal of network coding to encode or decode information in request or reply for improving throughput of SaaS application. A performance analysis of the proposed method is also presented in this paper.

The rest of the paper is organized as follows. Section 2 introduces request multiplexing method and description for multi-tenants, and gives the algorithm

description for certain request about competitive resource. In section 3, we implement this method and compare difference in general. We conclude our work and future work in section 4.

2 Request Multiplexing Method for Multi-tenants

2.1 Request Multiplexing Method

We consider competitive resource, for example, database, if there are two requests from different tenants simultaneously to access database, as shown in Figure 2 (a), let T_1, T_2 be different tenants, let R_1, R_2 be requests from T_1, T_2 , let A_1, A_2 be the answer for R_1, R_2 to T_1, T_2 . Assuming the maximum number of connections of DB is 1 (we simplify Figure 2(a), and marked the maximum number of connections to the DB, as shown in Figure 2(b)).

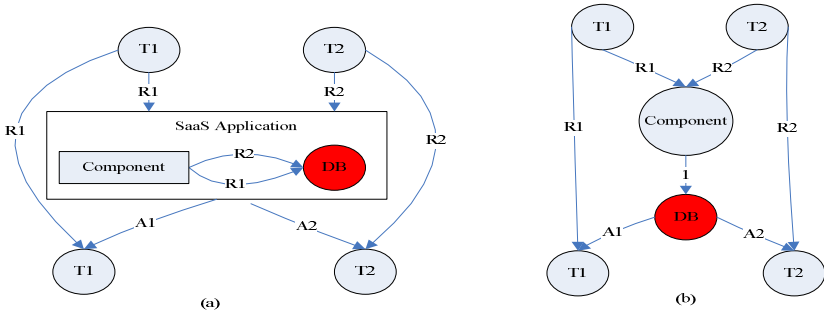


Fig. 2. Two-request two answer without coding

From Fig.2 (b), we can see that resource DB only can meet one request, so the other request must be wait. At this point, system throughput is 1.

We suppose that there is a request queue, we can get requests from the request queue, if we can make some requests into one requests, then it means that many requests can use the only one connection together. We call this process as request multiplexing process. When resource finished the request, it would return a reply including all information that all tenants' want. All these data must be organized in some way to distinguish different tenants, and when each request get the reply, each request can get its own information from the sharing reply. We call this process as data encoding & decoding process. Fig. 3 (a) shows the SaaS application with request multiplexing. In Fig. 3 (b), R_1+R_2 means that request R_1 and R_2 are merged into one request, A_1+A_2 means that the sharing reply includes answer A_1 for request R_1 and answer A_2 for request R_2 , and tenant T_1 and T_2 both receive the reply A_1+A_2 .

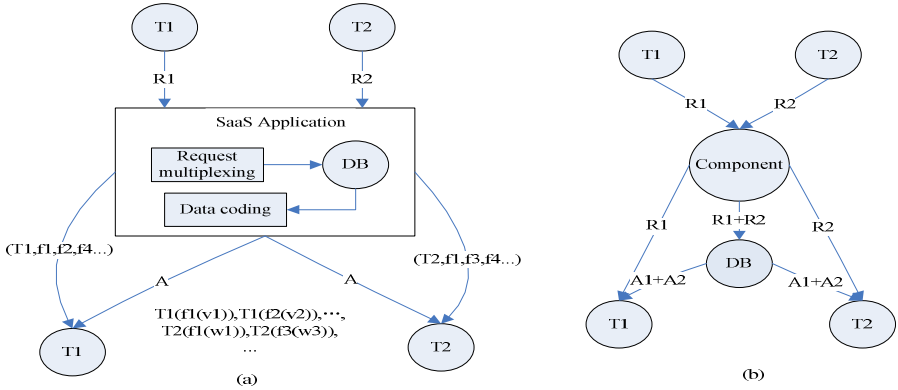


Fig. 3. Two-request one-answer with coding

From Fig.3 (b), we can see that when there is only one available connection to DB, the two requests can be treated as one request, and then they get their own information from the sharing reply. The throughput of SaaS application is increased to 2.

In theory of network coding, nodes in network need to obtain information about other nodes, but in request multiplexing method, each tenant needs to obtain information about itself, so, here, we consider (inclusive) OR operation.

First, we define truth table about OR:

	<i>a</i>	<i>b</i>
<i>a</i>	1	0
<i>b</i>	0	1

Let set $T = \{T_1, T_2, \dots, T_i, \dots, T_n\}$ be tenants set involved in coding, let set $X = \{X_1, X_2, \dots, X_j, \dots, X_p\}$ be information set, and

let information vector corresponding to tenant T_i be $T_i = (x_1, x_2, \dots, x_j, \dots, x_m), T_i \in T, 1 \leq i \leq n; x_j \in X, 1 \leq j \leq m$. , it means that tenant T_i wants to get information $x_1, x_2, \dots, x_j, \dots, x_m$. We call $item_i = T_i(x_j(v_j))$ as a data item, here v_j is the value corresponding to x_j , then coding data in sharing reply is the set of data items. Let V denote coding data, then:

$$\begin{aligned}
 V &= \{item_i\} = T_1(f_1(v_1)) \cup T_1(f_2(v_2)) \cup \dots \cup T_i(d_l(w_l)) \cup \dots \cup T_n(e_h(u_h)) \\
 &= \bigcup_{j=1}^m T_1(f_j(v_j)) + \bigcup_{l=1}^k T_2(d_l(w_l)) + \dots + \bigcup_{h=1}^s T_n(e_h(w_h)) \\
 &= \{\{ietm_{T_1}\}, \{ietm_{T_2}\}, \dots, \{ietm_{T_i}\}, \dots\}
 \end{aligned}
 \tag{1}$$

here, $item_{T_i}$ means that the data item belongs to tenant T_i .

When we get coding data, according to our data item, we do OR operation on each data item, this operation includes 2 steps: tenant OR operation and vector OR

operation. In tenant OR operation, if we use a tenant code to do OR operation, then we can get all data item about this tenant. In vector OR operation, if we use vector item in vector X , then we can get the value of this vector item. All that said, tenant can get their own data and shield unrelated data by OR operation. First, we define the operation $A \odot item_i$ as:

$$\begin{aligned} A \odot item_i &= A \odot T_i(x_j(v_j)) = (A \odot T_i)(x_j(v_j)) \\ &= \begin{cases} 1 \cdot (x_j(v_j)), A = T_i \\ 0 \cdot (x_j(v_j)), A \neq T_i \end{cases} = \begin{cases} x_j(v_j), A = T_i \\ 0, A \neq T_i \end{cases} \end{aligned} \quad (2)$$

For example:

$$x_i \odot x_j(v_j) = (x_i \odot x_j)(v_j) = \begin{cases} v_j, x_i = x_j \\ 0, x_i \neq x_j \end{cases}$$

According to formula (1) and (2), if tenant T_1 wants to get the value of x_j in his request, tenant T_1 can get it by OR operation:

$$\begin{aligned} x_j &= x_j \odot (T_i \odot V) \\ &= x_j \odot (T_i \odot (\bigcup_{j=1}^m T_1(f_j(v_j)) + \bigcup_{l=1}^k T_2(d_l(w_l)) + \dots + \bigcup_{h=1}^s T_n(e_h(w_h)))) \\ &= x_j \odot (T_i \odot \bigcup_{i=1}^a T_i(x_a(b_a))) \\ &= x_i \odot (x_1(b_1), \dots, x_i(b_i), \dots) \\ &= x_i \odot x_i(b_i) \\ &= b_i \end{aligned}$$

For example, there are some data items for multiple tenants in coding data, and tenant T_1 wants to get his reply: first, we use coding T_1 to get all information about T_1 :

$$\begin{aligned} T_1 \odot \{ \{ietm_{T_1}\}, \{ietm_{T_2}\}, \dots, \{ietm_{T_i}\}, \dots \} \\ &= T_1 \odot (\bigcup_{j=1}^m T_1(f_j(v_j)) + \bigcup_{l=1}^k T_2(d_l(w_l)) + \dots + \bigcup_{h=1}^s T_n(e_h(w_h))) \\ &= \bigcup_{j=1}^m f_j(v_j) = \{f_1(v_1), f_2(v_2), \dots\} \end{aligned}$$

Next, we get the final information:

$$\begin{aligned} f_1 &= f_1 \odot \{f_1(v_1), f_2(v_2), \dots\} = f_1 \odot f_1(v_1) = v_1 \\ f_2 &= f_2 \odot \{f_1(v_1), f_2(v_2), \dots\} = f_2 \odot f_2(v_2) = v_2 \\ &\dots \end{aligned}$$

2.2 Description of Algorithm

Assuming that we can get requests from the request queue Q and the queue is a queue of requests that access the same resource. We extract m requests every time to handle, and keep the order of requests. It means that, when these requests share one request R , information in these requests will be executed with the order of requests. The formal description of request multiplexing method is given below:

Algorithm 1 request multiplex method:

```

1. Initial  $m, R, T[], A.$ 
   //  $R$  is the sharing request,  $T[]$  includes content of requests, and  $A$  is reply of  $R$ 
2. For ( $i=1; i \leq m; i++$ )
   2.1 Get the  $i$ 'th request in request queue  $Q$  and denote it as  $r.$ 
   2.2 Keep tenant information and request content in  $r$  as  $C.$ 
   2.3 Add  $C$  into  $R.$ 
   End For.
3. Construct new request  $R.$ 
4. Connection resource( $R$ );
5. Extract request content and tenant information from  $R$  as  $T;$ 
   For ( $j=1; j \leq m; j++$ )
   5.1 execute  $T[j]$  and return result set.
   5.2 use formula (1) to code data in result set.
   End for.
6. Close connection( $R$ ).
7. Get reply of  $R$  and denote it as  $A.$ 
8. Decode  $A$  using formula (2).
9. Answer tenants with decoding data.

```

3 Implementation and Compare

We simulated a competitive resource, which can only handle 5 requests at the same time, and the processing time for each request is 1 second. In 2 minutes, set up 12 virtual users to access the resource continuous, we compare the number of request processing and response time in both cases of requests handing with the multi-tenant request multiplexing algorithms and requests handing without the request multiplexing algorithm.

Test environment is: windows XP, 1G memory, pentium(R) 4, cpu 3.00GHz, LoadRunner 9.0.

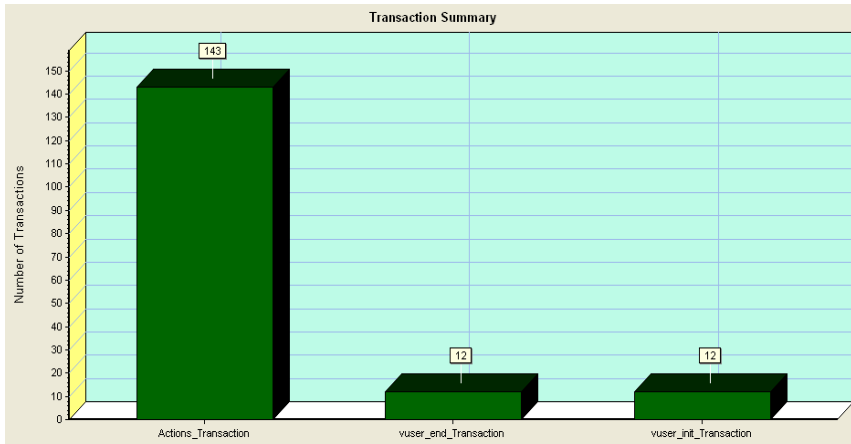


Fig. 4. Process request without request multiplexing algorithm

Fig.4 is about algorithm without request multiplexing. The figure shows, in 2 minutes, 12 virtual users, handled a total of 143 requests.

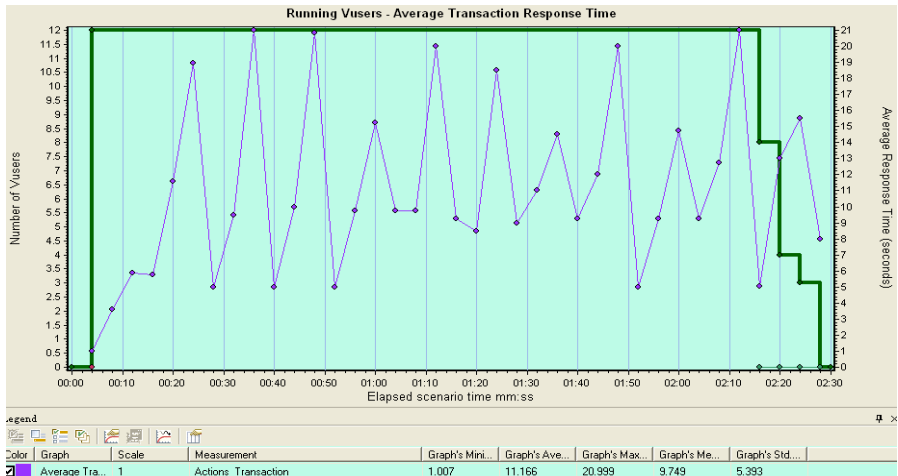


Fig. 5. Process request without request multiplexing algorithm

Fig.5 is about algorithm without request multiplexing. The figure is about 12 virtual users, average response time of 143 requests. Average response time: 11.166 seconds. (Because one request needs 1 seconds to process, and the competitive resource only handles 5 requests at the same time, and other requests must wait, so the test gets a longer average response time: 11.166 seconds.)

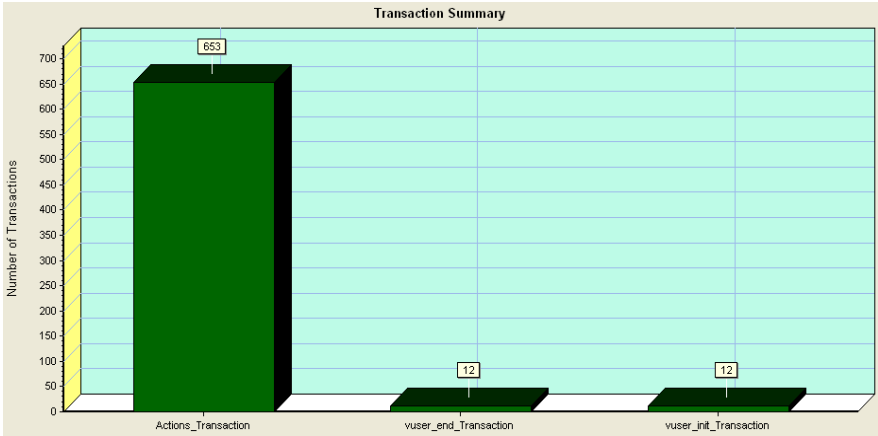


Fig. 6. Process request with request multiplexing algorithm

Fig.6 is about algorithm with request multiplexing. The figure shows, in 2 minutes, 12 virtual users, handled a total of 653 requests.

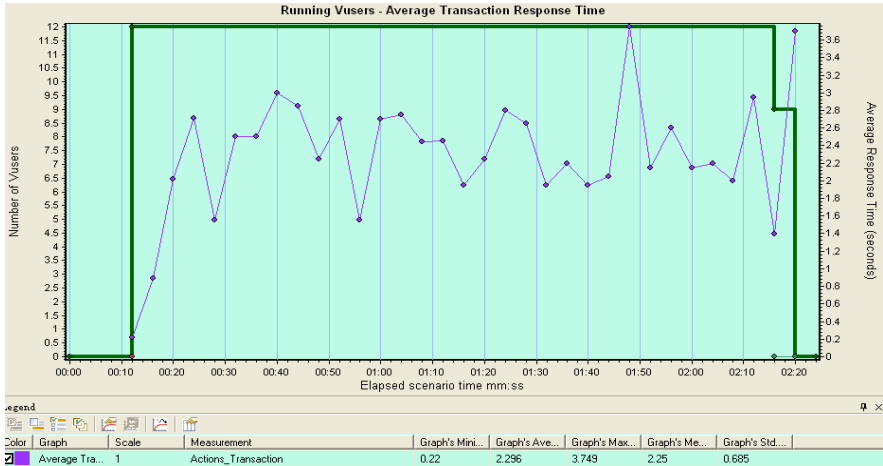


Fig. 7. Process request with request multiplexing algorithm

Fig.7 is about 12 virtual users, response time of 653 requests with request multiplexing algorithm. Average response time: 2.296 seconds.

From our experiment, we can see that in the same context our request multiplexing algorithm can process more requests (143:653) and get faster response time (11.166: 2.296).

4 Conclusion and Future Work

This paper discuss problem about multiple tenants requests multiplexing to maximize the number of customers in the context of a given infrastructure cost. To improve throughput of SaaS application, we propose request multiplexing method based on multiple tenants, in this method, we primarily focus on two key issues, one is how to share requests, and the other is how to separate yourself response from a share reply. To solve the two problems, we use the ideal of network coding to encode or decode information in request or reply. An experiment and performance analysis of the proposed method is also presented in this paper.

Our future work will further discuss the quantitative relationship about multiplexing, in order to achieve the best performance of SaaS application.

References

1. Zhang, Y., Wang, Z., Gao, B., Guo, C., Sun, W., Li, X.: An Effective Heuristic for On-line Tenant Placement Problem in SaaS. In: 2010 IEEE International Conference on Web Services (2010)
2. Wu, L., Garg, S.K., Buyya, R.: SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud. In: 2011 11th IEEE ACM International Symposium on Cluster, Cloud and Grid Computing (2011)
3. Ahlswede, R., Cai, N., Li, S.R., Yeung, R.W.: Network information flow. *IEEE Transactions on Information Theory* (July 2000)
4. Fragouli, C., Le Boudec, J.Y., Widmer, J.: Network Coding: An Instant Primer. *ACM SIGCOMM Computer Communication Review* 36(1), 63–68 (2006)
5. Caron, E., Desprez, F., Loureiro, D., Muresan, A.: Cloud Computing Resource Management through a Grid Middleware: A Case Study with DIET and Eucalyptus (2009)
6. Lee, Y.C., Zomaya, A.Y.: Energy efficient utilization of resources in cloud computing systems. *J. Supercomput.* (2010)
7. Goiri, N., Juliá, F., Nou, R., Berral, J.L., Guitart, J., Torres, J.: Energy-aware Scheduling in Virtualized Datacenters. In: 2010 IEEE International Conference on Cluster Computing (2010)
8. Lin, H., Sun, K., Zhao, S., Han, Y.: Feedback-Control-based Performance Regulation for Multi-Tenant Applications. In: 15th International Conference on Parallel and Distributed Systems (2009)

An Adaptive Design Pattern for Genetic Algorithm-Based Composition of Web Services in Autonomic Computing Systems Using SOA

Vishnuvardhan Mannava¹ and T. Ramesh²

¹Department of Computer Science and Engineering,
K L University, Vaddeswaram, 522502, A.P., India
vishnu@kluniversity.in

²Department of Computer Science and Engineering,
National Institute of Technology, Warangal, 506004, A.P., India
rmesht@nitw.ac.in

Abstract. Web services composition has been an active research area over the last few years. However, the technology is still not mature yet and several research issues need to be addressed. In this paper, we propose Genetic Algorithm based Design Pattern. This system provides tools for adaptive service composition and provisioning. We introduce a composition model where service context and exceptions are configurable to accommodate needs of different users. This allows for reusability of a service in different contexts and achieves a level of adaptive and contextualization without recoding and recompiling of the overall composed services. The proposed system will compose web services based on user request using Service oriented Architecture (SOA). Genetic Algorithm based composition Design Pattern satisfies properties of autonomic system. We use different Design Patterns for designing the system like, Master slave Design Pattern and Chain of responsibility Design Pattern. Our proposed system will satisfy all properties of autonomic system, for monitoring we have used context based monitoring, for decision making we use Master Slave which is based on decision making system that will reconfigure itself and Thread per connection is used of executing different services in different threads. A simple UML class and Sequence diagrams are depicted.

Keywords: Web service composition, Design Pattern, Autonomic system, Service Oriented Architecture (SOA), Web Services, Web Service Description Language (WSDL) and Genetic Algorithm (GA).

1 Introduction

Genetic Algorithm (GA) is a problem solving method inspired by Darwin's theory of evolution: a problem is solved by an evolutionary process resulting in a best (fittest) solution (survivor). In a GA application, many individuals derive, independently and concurrently, competing solutions to a problem. These solutions are then evaluated

for fitness and individuals survive and reproduce based upon their fitness. Eventually, the best solutions emerge after generations of evolution.

The flow of a typical GA simulation is as follows: First, a GA server creates many individuals randomly. Each of these individuals is tested for fitness. Based on their fitness, measured by a fitness function that quantifies the optimality of a solution, the server selects a percentage of the individuals that are allowed to crossover with each other, analogous to gene sharing through reproduction in biological organisms. The crossover between two parents produces offspring, which have a chance of being randomly mutated. A child thus produced is then placed into the population for the next generation, in which it will be evaluated for fitness. The process of selection, crossover, and mutation repeats until the new population is full and the new generation repeats the behavior of the previous generation. After many generations, the individuals are expected to become more adept at solving the problem to which the GA is being applied.

Web service is defined as an interface which implements the business logic through a set of operations that are accessible through standard Internet protocols. The conceptual Web services architecture [1] is defined based upon the interactions between three roles: service provider, service registry and service requester. The requester search for suitable Web services in the registry which satisfy his functional and nonfunctional requirements. The requester's service request sometimes includes multiple related functionalities to be satisfied by the Web service. In many cases the Web service has a limited functionality which is not sufficient to meet the requester's complex functional needs. To achieve complex business goals in real world applications, the execution of multiple Web services should be orchestrated through service composition.

The Web service composition can be defined as the creation of new Web service by combining the available services that realizes the complex service request. The service composition strategies are broadly classified as Static and Dynamic composition based on the time when the Web services are composed [2]. Static composition takes place during design time when the architecture and the design of the system is planned. Dynamic composition takes place at run time when the requested service is not provided by the single provider. The effective dynamic Web service composition is a major challenge towards the success of Web services.

Autonomic Computing is an initiative started by IBM in 2001 with an ultimate aim to develop computer systems capable of self-management, to overcome the rapidly growing complexity of computing systems management, and to reduce the barrier that complexity poses to further growth. So, the system, to be autonomic, must have the following properties:

- self-configuring: Automatic configuration of components;
- self-healing: Automatic discovery, and correction of faults;
- self-optimizing: Automatic monitoring and control of resources to ensure the optima functioning with respect to the defined requirements;
- Self-protecting: Proactive identification and protection from arbitrary attacks;

Our proposed system composes the web services dynamically using SOA. We use this Design Pattern for constructing the system/application and it has satisfied the properties of autonomic system. Initially client will request to all service providers for WSDL files, and then all the service providers will send WSDL files to the respective clients. The Client will search for some matching constrains in WSDL file and it will chooses all matched service providers WSDL files for composing. Based on rule generated by master slave Design Pattern, client generate XML file. This pattern uses the Genetic Algorithm for decision making; we use parallel Genetic Algorithm for evaluating population. Each population is executed in different clients and out of all returned results of clients; GA will pick best result as composition rule. We use Mater slave Design Pattern for evaluating Genetic Algorithm. Based on the rule selected, service provider will compose new web service based on XML file constrains.

2 Related Work

In this section we present some works that deal with different aspects of autonomic systems and their design. Freddy Lecue and Nikolay Mehandjiev in their paper [1] discuss Genetic Algorithm based optimization for web service composition. Demian Antony D'Mello [2] exploits the analogy of different type of web service composition techniques based on those techniques we propose this paper. In [3] is presented a SOA based composition of web service based on user demand. [4] vishnuvardhan mannava paper give an architecture for service invocation pattern we take this paper for invocation of web service.

In Vishnuvardhan Mannava and T. Ramesh paper [8] [9] [10] they propose a design pattern for Autonomic Computing System which is designed with Aspect-oriented design patterns. They also studied about the amalgamation of the Feature-oriented and Aspect-oriented software development methodology and its usage in developing a self-reconfigurable adaptive system.

Based on the previous papers we propose new approach for web service composition using Design Pattern "Genetic Algorithm based composition Design Pattern".

3 Parallel Genetic Algorithm Based Composition Design Pattern Template

To facilitate the organization, understanding, and application of the adaptation Design Patterns, this paper uses a template similar in style to that used by Ramirez et al. [2]. Likewise, the Implementation and Sample Code fields are too application-specific for the Design Patterns presented in this paper.

3.1 Pattern Name

Adaptive Design Pattern Design Pattern

3.2 Classification

Structural - Monitoring - Decision making.

3.3 Intent

In order to design the system/application for composing dynamic web services based on client request without modifying already running service code in the main memory. We model a new Design Pattern which is an amalgamation of different Design Patterns like Master Slave and Chain of Responsibility.

3.4 Motivation

Main objective of Genetic Algorithm based composition Design Pattern is to compose web services based on the user request. Our Pattern will compose web services dynamically by using SOA based service composition techniques.

3.5 Proposed Design Pattern Structure

UML class diagram for the Constrain based composition Design Pattern can be found in Figure 3.

This proposed system will compose a new web service dynamically based on user requests. We use two Design Patterns for modeling this pattern they are Mater slave and Chain of responsibility. Client will request to all service providers for the respective WSDL files of services that they provide globally, and then all service providers will send WSDL files of the services that they provide to the clients. A client search for matching constrains in WSDL files depending on the user’s request, and client chooses all matched service provider files for composing. Based on rule generated by master slave Design Pattern client generate XML file. After that we use parallel Genetic Algorithm concept for evaluating population and which serves as a decision maker. Each population is executed in different client and then out of all the obtained results from the respective clients the Genetic Algorithm (GA) will pick best result as composition rule. We use Mater slave Design Pattern for evaluating Genetic Algorithm. Based on generated rule the service provider will compose new web services depending on XML file constrains. The Composition of services at the service providers can be realized with the help of this proposed structure of composing the web services with SOA see Figure 1.

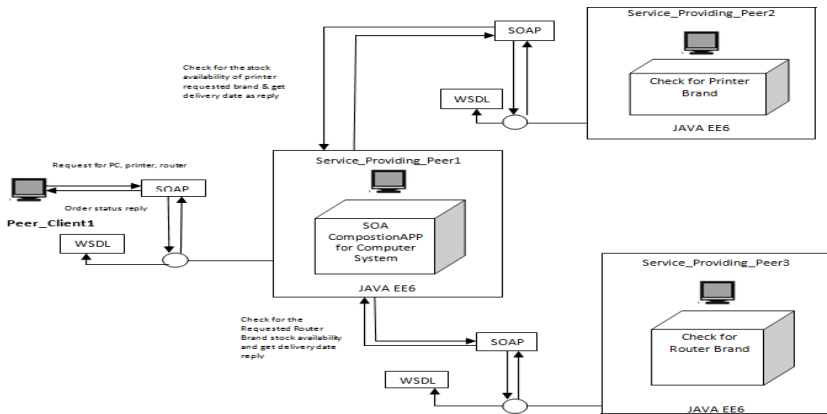


Fig. 1. Composition of the Services with Service Oriented Architecture using web services

3.6 Participants

(a) Client

Client class sends the WSDL file request to service providers, based on WSDL response client run Genetic Algorithm for finding appropriate plan for composing new service. Based on Genetic Algorithm decision client generate XML file for service provider.

(b) Service Provider

Service provider give response to client based on constrain of user. Service provider composes new web service based on XML file generated by client.

(c) Master

Master class run Genetic Algorithm based on the client constrain. Master will generate populating base in constrains that are provided by the client. Master will provide plan for composing web service after evaluating Genetic Algorithm, master will run population in different client to reduce time of server. Based on the responses of client master will pick one plan for composing web service.

(d) Slave

Slave will run population of Genetic Algorithm; each slave will execute on population at a time based on the slave results master will pick plan for composing web service.

(e) Population

Every problem have many solutions in out of all possible Genetic Algorithm will pick best as solution. All possible solution is called population in Genetic Algorithm. In population class will generate all possible solutions for composition for web service based on fitness function.

(f) Service repository

Service repository will store web service in it, based on client request service provider will invoke form service repository, Composed web services also stored in service repository.

3.7 Applicability

Use the Adaptive Design Pattern when:

- Web administrator will use this autonomic system for dynamic composition.
- An application or system can be simplified by being composed of multiple independently developed and dynamically configurable services; or

The management of multiple services can be simplified or optimized by configuring them using a single administrative unit.

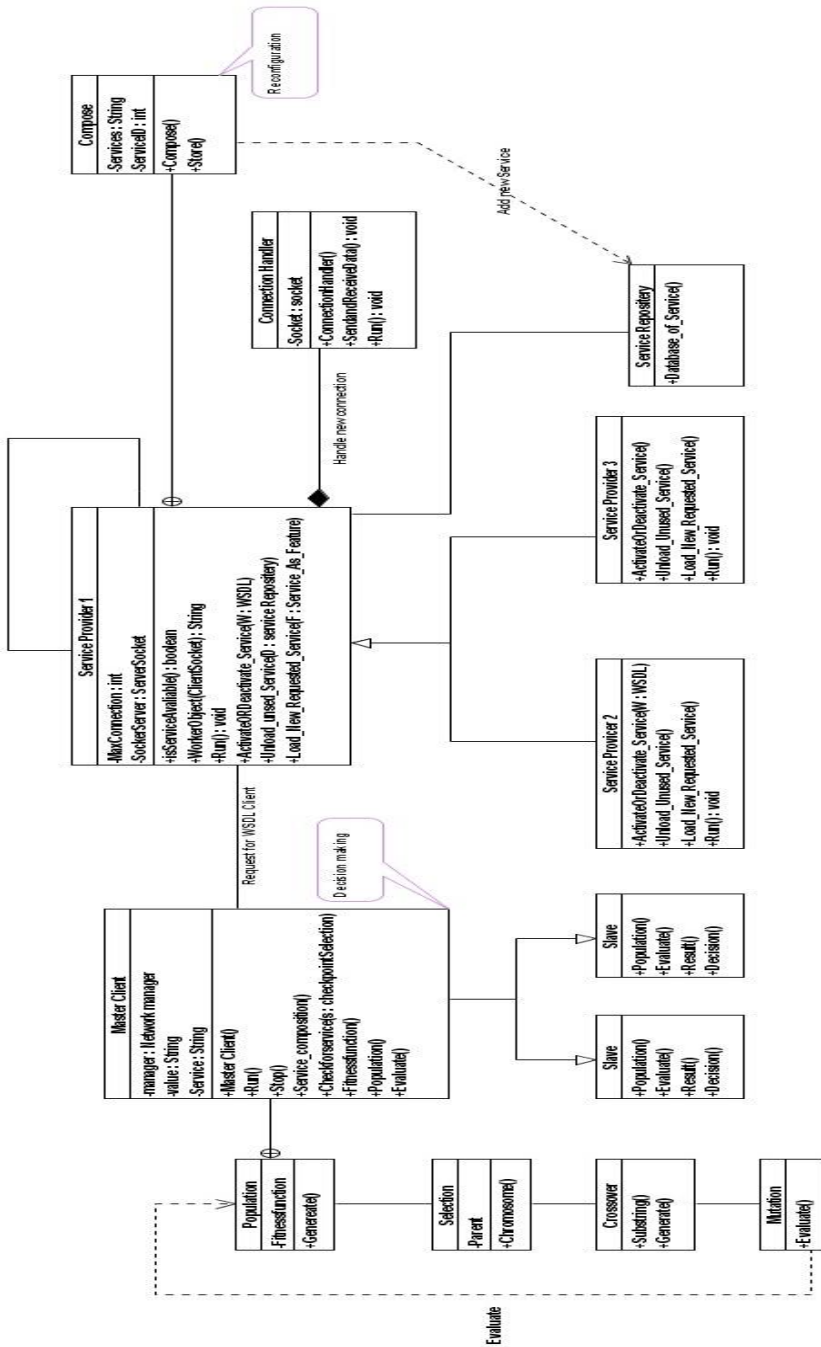


Fig. 2. Class Diagram for Adaptive Design Pattern

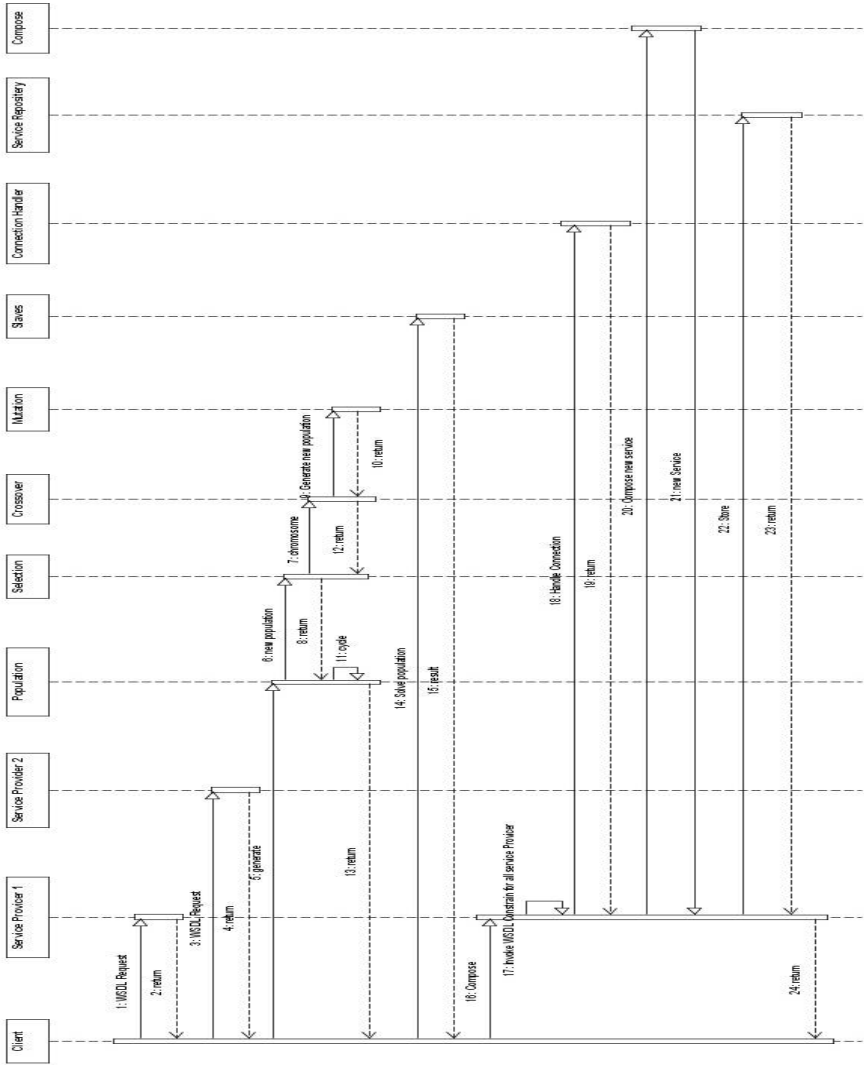


Fig. 3. Sequence Diagram for Adaptive Design Pattern

4 Case Study

In our experiments we evaluate the effectiveness of our proposed Adaptive Design Pattern. In order to make our proposal clear we have successfully developed some critical parts of our system i.e., we have developed the code for the Reconfiguration modules.

The simulation results for the reconfiguration module are collected with respect to:

- Used Heap memory
- Total Started Thread Count
- Process CPU Time
- Total Compilation Time

With the help of the comparison between the efficiency of a sample application (reconfiguration module) in our Adaptive Design Pattern architecture, which is initially developed in normal Object Oriented Programming code using java (which we refer it as a normal plain code without using any design patterns) and the same application which is developed using the design patterns and Object Oriented Programming techniques to provide the dynamic reconfiguration property with the help of Java Components.

When we have successfully executed the application with and without applying design patterns then we have observed the following results which are in the form of graphs as follows:

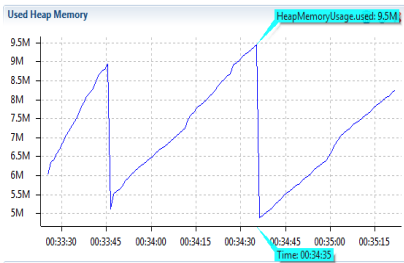


Fig. 4. Heap memory usage of Reconfiguration Module before applying pattern

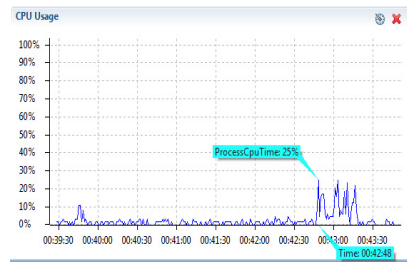


Fig. 6. CPU Usage of Reconfiguration Module before applying pattern

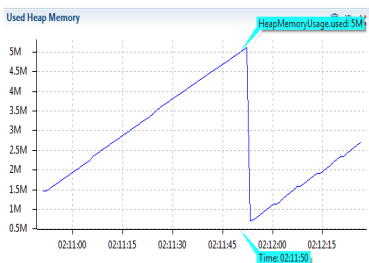


Fig. 5. Heap memory usage of Reconfiguration Module after applying pattern

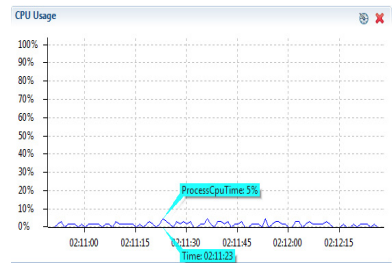


Fig. 7. CPU Usage of Reconfiguration Module after applying pattern

5 Discussion

From the Figures 4 and 5 we can evaluate that the amount of Heap Memory used by applying aspectual design pattern is 21,354 kbytes and where as for the amount of Heap Memory used without any design pattern is 28,634 kbytes.

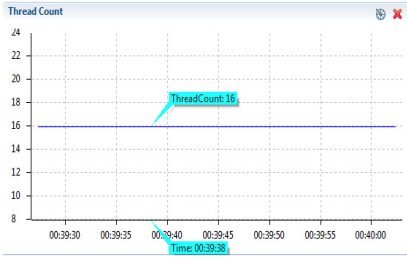


Fig. 8. Thread Count of Reconfiguration Module before applying pattern

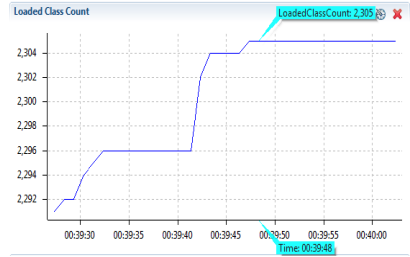


Fig. 10. Loaded Class Count of Reconfiguration Module before applying pattern

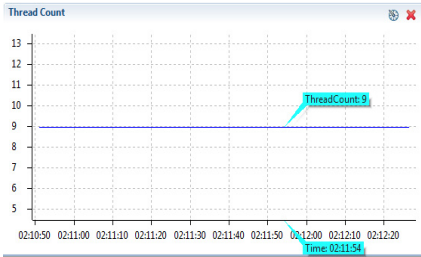


Fig. 9. Thread Count of Reconfiguration Module after applying pattern

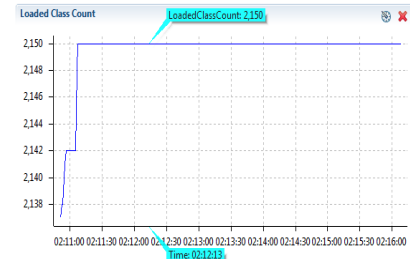


Fig. 11. Loaded Class Count of Reconfiguration Module after applying pattern

From the Figures 6 and 7 we can evaluate that the amount of CPU Time used by applying aspectual design pattern is 8.126sec and where as for the amount of CPU Time used without any design pattern is 11.435 sec. Figures 8 and 9 we can evaluate that the Total Started Thread Count by applying design pattern is 14 and where as for Total Started Thread Count without any design pattern is 15. Figure 10 and 11 shows loaded class count of self optimization module before and after applying patterns. The Total Compilation Time for the Self optimization Module with design pattern took 1034 sec and without design pattern is 1546 sec.

6 Conclusion

Paper describes parallel Genetic Algorithm based composition Design Pattern it will compose web service based client request. Composition of available Web services based on the requester's functional requirements is a challenging task. Web service is normally a collection of logically related operations and the requester normally requests for a single operation or multiple operations (complex service re-quest). Thus composition must focus on generating composition plan involving abstract operations of available Web services instead of just Web services. Proposed system will compose web service based on the user constrain at runtime by invoking Genetic Algorithm based composition rule. Our future goal is to implement this paper in aspect oriented programming satisfying autonomic characteristics of autonomic computing system.

References

1. Lecue, F., Mehandjiev, N.: Seeking Quality of Web Service Composition in a Semantic Dimension. *IEEE Transactions on Knowledge and Data Engineering* 23(6) (June 2011), doi:10.1109/TKDE.2010.237
2. D'Mello, D.A., Ananthanarayana, V.S., Salian, S.: *A Review of Dynamic Web Service Composition Techniques*. CCIS, pp. 85–97. Springer (2011)
3. Sheng, Q.Z.: Configurable Composition and Adaptive Provisioning of Web Services. *IEEE Transactions On Services Computing* 2(1), 34–49 (2009)
4. Ramirez, A.J., Betty, H.C.: Design patterns for developing dynamically adaptive Systems. In: *5th International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Cape Town, South Africa, pp. 29–67 (2010), doi:10.1145/1808984.1808990
5. Mannava, V., Ramesh, T.: A Novel Event Based Autonomic Design Pattern For Management of Webservices. CCIS, vol. 198, pp. 142–151 (2011)
6. Prasad Vasireddy, V.S., Mannava, V., Ramesh, T.: A Novel Autonomic Design Pattern for Invocation of Services. CCIS, vol. 196, pp. 545–551 (2011)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns Elements of Reusable Object-Oriented Software*, Hawthorne, New York (1997)
8. Mannava, V., Ramesh, T.: A Service Administration Design Pattern for Dynamically Configuring Communication Services in Autonomic Computing Systems. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) *ACIHDS 2012, Part I*. LNCS, vol. 7196, pp. 53–63. Springer, Heidelberg (2012)
9. Mannava, V., Ramesh, T.: An Aspectual Feature Module Based Adaptive Design Pattern for Autonomic Computing Systems. In: Pan, J.-S., Chen, S.-M., Nguyen, N.T. (eds.) *ACIHDS 2012, Part III*. LNCS, vol. 7198, pp. 130–140. Springer, Heidelberg (2012)

10. Mannava, V., Ramesh, T.: A novel adaptive re-configuration compliance design pattern for autonomic computing systems. *Procedia Engineering* 30, 1129–1137 (2012), doi:10.1016/j.proeng.2012.01.972, ISSN 1877-7058, <http://www.sciencedirect.com/science/article/pii/S1877705812009824>
11. Crane, S., Magee, J., Pryce, N.: Design Patterns for Binding in Distributed Systems. In: *The OOPSLA 1995 Workshop on Design Patterns for Concurrent, Parallel, and Distributed Object-Oriented Systems*, Austin, TX. ACM (1995)
12. Pree, W.: *Design Patterns for Object-Oriented Software Development*. Addison-Wesley, MA (1994)

Service-Oriented Ontology and Its Evolution

Weisen Pan, Shizhan Chen^{*}, and Zhiyong Feng

School of Computer Science and Technology, Tianjin University, Tianjin, China
{wspan, shizhan, zyfeng}@tju.edu.cn

Abstract. Web service was designed to solve the problem of the heterogeneous system integration and make heterogeneous systems interoperable. However, current Web service technologies are not sufficient to build distributed, heterogeneous Web service infrastructure, because they are provide by different service providers, use different conceptual model and design tools, which hinders Web service automatic discovery and composition. With diversification of user requirements, we need to reasonable abstraction and organization of the Web services through a new viewpoint. Thus, an lightweight Web services semantic description model, namely Service Ontology, is proposed in this paper, whose intention is to make comprehensive and multi-dimension semantic description for Web services. This paper also describes the evolution mechanism of service ontology. The mechanisms make use of the semantic tags to expand Service Ontology. Finally, a case study is presented in the paper that validates the Service Ontology can fulfill the multi-granularity requirement of the users. This paper has made the beneficial exploration in multi-dimension modeling and automation organization of Web services.

Keywords: Web Service, Service Ontology, Tag, Service Relation, Service Chain, Service Discovery.

1 Introduction

With incessant growing and improving of Web service, it has attracted intensive attention from the academia and industry. Web services are based on open standards such as HTTP (HyperText Transfer Protocol) and XML-based (Extensible Markup Language) protocols including SOAP (Simple Object Access Protocol) and WSDL (Web Services Description Language), their cost is low and the associated learning curve is smaller than that of many proprietary solutions. They are hardware, programming language, and operating system independent. This means that applications written in different programming languages and running on different platforms can seamlessly exchange data over intranets or the internet using Web services. So it is the best choice to achieve Service-Oriented Architecture (SOA) [1], Service-Oriented Computing (SOC) [2] and Software as a Service (SaaS) [3].

^{*} Corresponding author.

Web services can implement a SOA. The SOA has three roles: services provider, services registry and services requester. The service provider creates a Web service and possibly publishes its interface and access information to the service registry. The service registry is one of the fundamental pieces of SOA for achieving reuse. It refers to a place in which service providers can impart information about their offered services and potential clients can search for services. Various service registries already exist. The first major standard to appear was UDDI (Universal Description, Discovery and Integration), which was designed mostly with SOAP-based Web services in mind. The service requester locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its Web services.

In fact, it becomes much more difficult to find proper services from UDDI registry. Because UDDI lacks support semantic descriptions for Web services, which seriously affect the automatic service discovery, matching and composition. UDDI's search capability is syntax-based and relies solely on XML, which enables syntactic. Syntax-based matching lends itself to application-specific software development where reuse of Web services by other organizations is arduous. Semantic Web Technologies [4] are suitable to transform current Web services into services supporting such automation and reuse, it uses ontology reasoning to back up service automatic discovery, composition and interaction. In this paper, we will use the various semantic web technologies to achieve the semantic modeling for Web services through a new viewpoint.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 describes Service Ontology formal specification and its hierarchical structure in more detail. Section 4 presents the evolution mechanism of Service Ontology. Section 5 through case study shows the Service Ontology can fulfill the multi-granularity requirement of the users. Finally, we conclude in section 6.

2 Related Work

Semantic Web Service is enriching Web services with machine-processable semantics. With the rapid development of Semantic Web Service, ontology played a prominent role on it. Through the aid of semantic information by ontology can achieve automation Web service discovery and composition. How to add semantic to the Web services has been a question in academia for quite a long time. Thus far many Web services semantic description model and frame have been presented such as OWL-S, WSMO, SWSF and SAWSDL.

2.1 Works Related to Service Ontology

Ontology-based Web Language Service (OWL-S) [5] is an ontology of service concepts expressed in OWL-DL, a decidable description logic language. It have three main parts: the service profile for advertising and discovering services; the process model, which gives a detailed description of a services operation; and the grounding,

which provides details on how to interoperate with a service, via messages. OWL-S make possible for agents to discover, compose, invoke, and monitor services with a high degree of automation. It is the most influential research in semantic Web service.

Web Service Modeling Ontology (WSMO) [6] provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to facilitate the automation of discovering, combining and invoking electronic services over the Web. It is consist of four elements: ontologies, which provide the terminology used by other WSMO elements, Web service descriptions, which describe the functional and behavioral aspects of a Web service, goals that represent user desires, and mediators, which aim at automatically handling interoperability problems between different WSMO elements.

The Semantic Web Services Framework (SWSF) [7] consists of Semantic Web Services Language (SWSL) and the Semantic Web Services Ontology (SWSO). SWSL describes two variants: SWSL-FOL, a full first-order logic language, and SWSL-Rules, as rule-based language. SWSO presents a conceptual model by which Web services can be described, and an axiomatization, or formal characterization, of that model.

Semantic Annotations for WSDL and XML Schema (SAWSDL) [8] defines how to add semantic annotations to various parts of a WSDL document such as input and output message structures, interfaces and operations. To accomplish semantic annotation, SAWSDL defines extension attributes that can be applied both to WSDL elements and to XML Schema elements.

2.2 Comparative Analysis

The common denominator of these Service Ontology is the separation of aspects to describe a service in terms of inputs, outputs, and operations. To describe these aspects, Service Ontology rely on the existence of respective domain ontologies which can be referenced in actual service descriptions.

Although the fact that all OWL-S, WSMO, and SWSF were submitted to W3C, these comprehensive models have not become real standards. Because of the above-mentioned Service Ontology gave up the WSDL, which is traditional Web services description language. However, the current Web services on the Web were published in the form of WSDL, for example, there are more than thirty thousand this form Web services in Seekda [9].

Instead, the SAWSDL designs a solution to resolve the above-mentioned problem by add semantic annotations to various parts of WSDL. However, the SAWSDL are based on a hypothetical ontology, hide the truth that existing ontologies are not enough, especially the service-oriented ontology. Therefore, it is necessary to research of Service Ontology evolution mechanism. Also, the SAWSDL focus too much on service functionalities and not enough on the other profile of web service, and lack a comprehensive and deep description of the service relation and operation flow.

Aiming at the above-mentioned problems this paper put forward a lightweight Web services semantic description model, namely Service Ontology, whose intention is to make comprehensive and multi-dimension semantic description for Web services,

thus to meet the multi-dimension requirement of the users. This paper also describes the evolution mechanism of service ontology.

3 Service Ontology

The current Web services on Web have the characteristic of diversification because they are provide by different service providers, use different conceptual model and design tools. Thus, the constituent of Web Services Semantic Description Model have a greater difference. To sum up, a generic Web Services Semantic Description Model generally includes Service Functionality, Service Structure, Service Domain, Service Relation, Service Location, Service Time and QoS. Figure 1 show the Web services Semantic Description Model. This paper describes the different profile of Web services through domain ontology in order to make comprehensive and multi-dimension semantic description for Web services.

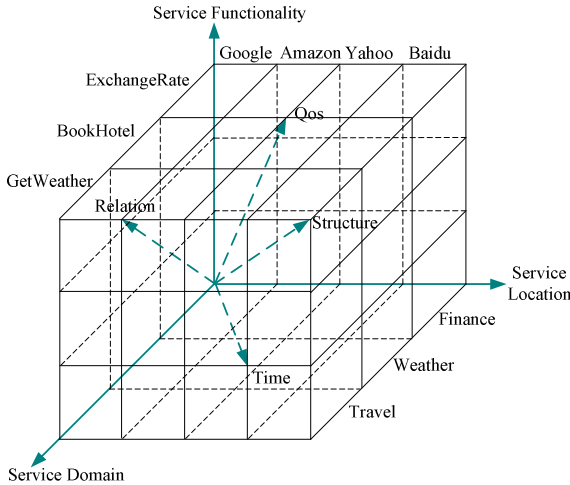


Fig. 1. Web services semantic description model

3.1 Formal Specification

Definition 1. The Service Ontology is defined as a 7-tuple:

$$SO = \langle SF, SS, SD, SR, SL, ST, QoS \rangle$$

Where the SF, SS, SD, SR, SL, ST, QoS are respectively Service Functionality, Service Structure, Service Domain, Service Relation, Service Location, Service Time and Quality of Service.

Service Functionality (SF) is the basic functionality of Web service. The operation of Web services can be consider an application which implementing some specific functionality.

Definition 2. Service Functionality is defined as:

$$SF = \langle Object, Action \rangle$$

The functionality of a Web service is represented by a pair of its action and the Object of the Action [10]. For example, there is a Web service – the book hotel system. It realizes the application of hotel reservation. That is to say the object of this service is “hotel”. Focusing on “hotel”, the service completes the operation – “book”, the functionality of a Web service which book hotel can be specified as $\langle Hotel, Book \rangle$.

Service Structure (SS) stores the Input, Output, Input type and Output type structure information of Web services. The functional semantics of their operations usually can not guarantee fulfill the requirement of the requester. Because two Web services with the same semantic functionality usually have different structure information, the service structure information that requester providers are different.

Definition 3. Service Structure is defined as:

$$SS = \langle Is, Os, IDTs, ODTs \rangle$$

Where

- Is (Input set) is the input parameters set of Web services.
- Os (Output set) store the Output parameters set of Web services.
- IDTs (Input Data Type set) store the input data type set of Web services.
- ODTs (Output Data Type set) is the out put data type set of services.

Service Domain (SD) is the domain information of Web service. With the help of domain experts and e-dictionary, Web services are categorized to specific domains. Domain is coarse-grained. Lots of Web services may not be put into a specific domain. In fact, there are intersections among domains, so one Web service may belong to several domains. Therefore, one Web service can be put into several classifications at the same time.

Service relation (SR) is the logic relationship among Web services. Our previous research divides the Service relation into competition and cooperation relations [11]. There are four competition relations: *exact*, which means the two services can do the same thing and replace each other, *plugin*, the two services (A, B) have plugin relation if the service A can be replaced by service B, *subsumes*, which is the inverse relation of plugin and also transitive, *intersection*, which means the two services have a similar degree greater than the predefined threshold but is not the types above. The competition relation can be seen as inspirational information for service discovery and is helpful to find substitutable service quickly. Cooperation relation presents in several modes such as precedence and subsequence. Two services (A, B) with precedence relation mean that the output of service A can satisfy, to some extent, the input of service B. subsequence definition in opposition to precedence. The cooperation relation is oriented to service composition, which can get the suitable candidate services through this relation.

Definition 4. Service Location description is defined as:

$$SL = \langle SP, PL \rangle$$

Service Location (SL) includes providers and physical location of Web service. The service provider is name of services offering. The Physical Location describes where the Web services located in. For example, there is a Web service – the get book name system, it is provider by Amazon and locate in Seattle.

Definition 5. Service Time description is defined as:

$$ST = \langle SPT, SL \rangle$$

ST (Service Time) includes the Web services Publish Time and Lifecycle. Service Publish Time (SPT) describes when a Web service is innovation. Service Lifecycle (SL) records all the change information of Web services functionality and structure from Web services innovation to extinction.

Quality of service (QoS) information is used for computing the quality degree of candidate Web services. The goal of QoS is to provide preferential delivery service for the requesters that need it by ensuring availability, accessibility and security, etc. Due to the dynamic and unpredictable characteristics of the Web services, it is not an easy task to provide the desired QoS for Web service users. Furthermore, different Web service applications with different QoS requirements will compete for network and system resources such as processing time and bandwidth. So, an QoS model for a Web service will bring competitive advantage for service provider.

Definition 6. QoS model is defined as:

$$QoS = \langle Availability, ResponseTime, Scalability, Reliability, Cost, Security \rangle$$

Where

- Availability is the percentage of time that a service is available during some time interval.
- Response Time describes the time interval between when a service is requested and when it is delivered.
- Scalability represents the capability of increasing the computing capacity of service provider's computer system and system's ability to process more users requests, operations or transactions in a given time interval [12].
- Reliability here represents the ability of a Web service to perform its required functions under stated conditions for a specified time interval.
- Service Cost is the price that a service customer has to pay for using the service.
- Security for Web services means providing authentication, authorization, confidentiality, traceability, data encryption, and non-repudiation.

3.2 Hierarchical Structure

In the previous section, we describe the Service Ontology, which is composed of 7-tuple. Each tuple can be described by ontology. So, the Service Ontology can be organized into hierarchical structure based on ontology methodology. The Service Ontology can basically be divided into four layers in Figure 2.

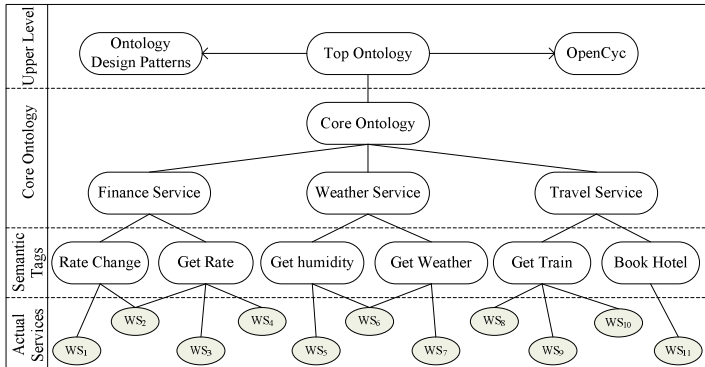


Fig. 2. The hierarchical structure of Service Ontology

First, the *upper level* contains the top ontology and ontology design patterns. An top ontology is an ontology which describes very general concepts that are the same across all knowledge domains. OpenCyc is a freely available subset of the top ontology. It includes a freely available, executable Knowledge Server that includes an inference engine and other tools for accessing, utilizing, and extending the content of the knowledge base [13]. OpenCyc also provides ontology design patterns as best practices for reoccurring modeling needs. So, it is the best choice for top ontology in Service Ontology.

Second, the *core ontology* consists of the domain ontology which is a knowledge representation of the domain as a set of concepts and relations. Domain ontology aims to capture relational domain knowledge, provides agreed understanding of domain knowledge, determines recognized vocabulary and defines vocabulary and their relations explicitly at different levels of conceptualization. All domain ontologies in this layer are aligned under the common roof of the top ontology. Different aspects of a service description (functionality, structure, etc.) can be description through the domain ontology and linked to the classes in the top ontology.

Third, *semantic tags layer* are composed of some semantic tags. These semantic tags results from users add metadata in the form of unstructured keywords to Web services. Because of user's participation, these tags with generality and semantics are better for WSDL document to express the Web service functionality and other description. Another advantage of tags is their ability to rapidly adapt to new changes in terminologies and domains.

Finally, *actual service layer* is consists of concrete Web services that are described by WSDL document on the Web. The actual service layer is the executable ones.

The Service Ontology is specified in OWL [14]. Each ontology basically coincides with an OWL file that imports other OWL files. The intended meaning of each class and relation is formally captured by axioms in the Service Ontology.

4 Evolution of Service Ontology

To fulfill the objective of automatic services discovery and composition, Service Ontology makes comprehensive and multi-dimension semantic description for Web services by domain ontology. However, at present the Web service on the web is growing at a rapid rate. How to accomplish automatic evolution of Service Ontology and able to meet incessant growing of Web services, it is a challenge. We first have to get the semantic information of Web services in order to achieve automatic evolution of Service Ontology. However, the current Web services on the Web are described through WSDL document, which definition and description information of Web service, like Web service name, operation, input and output. These WSDL documents use the optional `wsdl:document` element as a container for human readable documentation. The `wsdl:document` usually contains semantic description of Web services by providers such as services functionality, domain, location, etc. we usually need to extract semantic information through artificial way. It needs to spend a lot of effort and time, is a tedious and difficult task.

The emergence of social annotation gives an excellent resolution to above-mentioned problem. Social annotation provides a convenient way to annotate shared content by allowing users to use any tag or keyword. While free folksonomy is widely used in social software implementations and especially in Web services, it is generate a large number of the semantic tags data about Web services. These semantic tags are better for WSDL documents to express the Web services semantic information. For example, the most of Web services have semantic tags in Seekda.

The evolution mechanism of service ontology is set up the mapping relationship between tags and existing Service Ontology with the help of Semantic Dictionary. The mechanisms are illustrated in Figure 3. We use WordNet [15] to calculate semantic similarity between tags and concepts. By mapping tags into an ontology, the new addition Web services functionality and other profile can get further

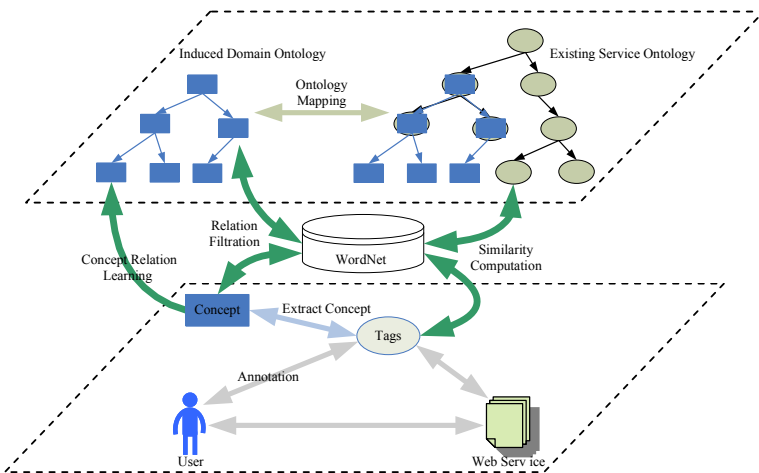


Fig. 3. The evolution mechanism of service ontology

interpretation. In further work, we want to learn concept relation from social annotation in order to further enrich the Induced Domain Ontology.

The core of Service Ontology evolution is establishing the mapping mechanism from concept to tag. To present the ideas of this mechanism in this paper, we define mapping rule from tag to concept.

Definition 7. $Mapping(Tag_i, Concept_j)$, where Tag is the new addition tag. Concept represents the concept name of existing Service Ontology. If the Mapping function return true, it represents they are synonymous and can be mapped, and vice versa.

$$Mapping(Tag_i, Concept_j) = \begin{cases} False & \text{if } Sim_{i,j} < n \\ True & \text{if } Sim_{i,j} \geq n \end{cases} \quad (1)$$

Where $Sim_{i,j}$ is the function of similarity computation between tag and concept. n represents threshold of similarity computation. Applying the package of JWordNetSim, $Sim_{i,j}$ gets the semantic similarity (between 0 and 1) between tag and concept.

Figure 4 show the mapping flow chart from tag to concept. First is the data selection and cleaning. This process is primarily to filter out noise data that is caused by the nonstandard user annotations. Noise data include stop word, abbreviations, numbers, and so on. Second, tags can be divided into simple tag and compound tag according to the different of tags performance features. The simple tag which is made up of single word, can be used as candidate tag by delete function words and restore stemming. The compound tag is mostly composed of verbs and nouns such as “GetWeather”, “BookHotel”. These compound tags are typical attribute-adjective

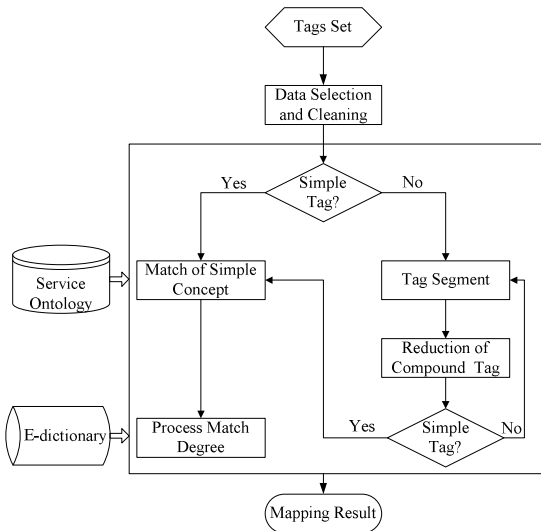


Fig. 4. The mapping flow chart from tag to concept

construction. The noun can be extracted from compound tag as candidate tag, and then delete verbs. Finally, we use WordNet to calculate semantic similarity between candidate tag and concept, and set up the mapping relationship from tag to concept.

5 Case Study

The construction of Service Ontology provides semantic support for Service Network. Service Network [16] is to use the idea of Semantic Web Technology and Social Network, constructed a novel service infrastructure on the basis of service relations. It is proposed to bridge the gap between service consumer and service provider. Look over the Service Network development, Service Ontology construction has become one of the main factors which restrict the development of SN. Service Ontology solves the existing problem of interoperability by introducing comprehensive and multi-dimension semantic description for Web services. A test version of Service Network is publicly available at <http://ikse.tju.edu.cn:8080/SN2.0>. It contains more than twenty thousand Web services. Requester can search Web services and check their details on this platform.

Aiming at Complexity and diversification of user requirements, Service Ontology exerts their functions through find services to meet user requirements. This paper divides the user requirements into two categories: functional requirements and non-functional requirements. Functional requirements are statements the Web service should provide such as service functionality. Non-functional requirements are constraints on the functions offered by the Web services and relate more to the quality aspects of the Web services. Non-functional requirements are often influenced by the nature of the Web service such as services response time, services reliability, etc.

Let us assume that a user want go to New York traveling. User hope to request a travel schedule services which can achieve flight and hotel reservation. He want to acquire the results of ticket and hotel reservation according to provide city name, date and credit card number. The consults result show there are some travel schedule services in Service Network, for example, “QueryFlight”, “GetHotel” and “CreditCardPay”, etc. The detailed information of Web services is shown in the Table 1.

Table 1. The detailed information of Web services

Num	Service Name	Input	Output	Location	Publish Time
S1	QueryFlight	CityNameSrc CityNameDst	FlightNum	China	2008
S2	GetHotel	CountryNum CityNum	Result	Europe	2004
S3	QueryHotel	CityName	HotelName	USA	2007
S4	BookFlightTicket	FlightNum OrderNum	OrderNum	China	2007
S5	CreditCardPay	CardID Password	AutoCode	USA	2006
S6	BookHotel	HotelName	Result	China	2005
S7	GetPlaneTicket	FlightNum	OrderNum	USA	2007
S8	ReserveHotel	CityName	OrderNum	China	2008

From the Table 1, we can see the user request cannot be satisfied, because there is no single service which matches the user's intention. The traditional method are manually composes services to satisfy the user's requirements. Service Ontology achieves the automation of this process. The service relation mining algorithm [12] runs in the background, it can mining the logical relation of Web services according to the match between input and output parameters. So, the service composition problem was transferred into find a service chain to meet the user's requirements from Service Ontology.

This paper divides the service discovery into two stages according to the user requirement.

The first stage is to meet the functional requirement according to user's functional description. From example, there is a user request – get flight ticket from Beijing to New York and book hotel in New York, user submits the request to Service Ontology and return the service chain (Shown in Fig.5) which can meet the functional requirement of requester.

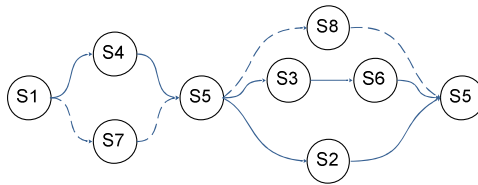


Fig. 5. The service chain

The second stage is to meet the non-functional requirement according to Web service input/output parameters, location, publish time and QoS etc. If the requester input parameters are “Beijing”, “New York”, “CardID”, “Password”, and requester stay in Beijing. From the Table 1, we can see that invoking “GetPlaneTicket” service spend much more time than invoking “BookFlightTicket” service, because the “BookFlightTicket” locate in china, it is more close to requester. The functional of “ReserveHotel” is the functional combination of “QueryHotel” and “BookHotel”. “ReserveHotel” service was published in 2011, it has more availability. After the analysis, “QueryFlight→GetPlaneTicket→CreditCardPay→ReserveHotel→CreditCardPayment” (The dotted path in Fig 5) is the most appropriate service chain for requester.

The case study shows that Service Ontology can not only meet the functional needs of requester, but also can satisfy the non-functional requirements such as location and QoS. Service Ontology accomplishes the optimum composition of Web services and satisfies the multi-granularity requirement of requester.

6 Conclusion and Future Work

As the incessant growing of Web services, it becomes much more difficult to find interesting services and meet the diversification requirement of requester. Service

Ontology is proposed to solve this problem. It makes the comprehensive and multi-dimension semantic description for Web services. The case study verifies that Service Ontology can effectively fulfill the diversification requirement of the user.

For future work, we will further extend and refine the Service Ontology in semantic level. To enrich the ontology, we hope to present an ontology relation learning mechanism based on social annotation. An automatic question and answer mechanism should be designed to acquire the accurate user's requirement.

Acknowledgment. This work was supported by the National Natural Science Foundation of China under Grant No. 61173155, the National High Technology Research and Development 863 Program of China under Grant No. 2007AA01Z130, and the Innovation Foundation of Tianjin University under Grant No. 2010XG-0009.

References

1. Erl, T.: *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, Upper Saddle River (2005)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F.: *Service-Oriented Computing: State of the Art and Research Challenges*. *Computer* 40, 38–45 (2007)
3. Turner, M., Budgen, D., Brereton, P.: *Turning Software into a Service*. *IEEE Computer* 10, 38–44 (2003)
4. Payne, T., Lassila, O.: *Semantic Web Services*. *IEEE Intelligent Systems* 19, 14–15 (2004)
5. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDemott, D., McIlraith, S., Narayana, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: *OWL-S: Semantic Markup for Web services*. W3C Member Submission (November 2004), <http://www.w3.org/Submission/OWL-S/>
6. Roman, D., Keller, U., Lausen, H., Bruijn, J.D., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: *Web Service Modeling Ontology*. *Applied Ontology* 1, 77–106 (2005)
7. Battle, S., Bernstein, A., Boley, H., Grosz, B., Gruninger, M., Hull, R., Kifer, M., Martin, D., McIlraith, S., McGuinness, D., Su, J.W., Tabet, S.: *Semantic Web Services Ontology*. W3C Member Submission (September 2005), <http://www.w3.org/Submission/SWSF-SWSO/>
8. Farrell, J., Lausen, H.: *Semantic Annotations for WSDL and XML Schema*. W3C Candidate Recommendation (January 2007), <http://www.w3.org/TR/sawSDL/>
9. Seekda, <http://webservices.seekda.com>
10. Dong-Hoon, S., Kyong-Ho, L., Tatsuya, S.: *Automated Generation of Composite Web Services based on Functional Semantics*. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web* 7, 332–343 (2009)
11. Chen, S.Z., Feng, Z.Y., Wang, H.: *Service Relations and Its Application in Services-Oriented Computing*. *Chinese Journal of Computers* 33, 2068–2083 (2010)
12. Ran, S.: *A Model for Web Services Discovery with QoS*. *SIGecom Exchanges* 4, 1–10 (2003)
13. Matuszek, C., Cabral, J., Witbrock, M., De Oliveira, J.: *An Introduction to the Syntax and Content of Cyc*. In: *Proc. AAAI 2006 Spring Symposium. Formalizing and Compiling Background Knowledge and Its Application to Knowledge Representation and Question Answering*, pp. 44–49. AAAI Press, Menlo Park (2006)

14. McGuinness, D.L., Harmelen, F.: OWL Web Ontology Language Overview. W3C Recommendation (February 2004), <http://www.w3.org/TR/owl-features/>
15. Miller, G., Beckwith, A., Fellbaum, R.C., Gross, D., Miller, K.: Introduction to WordNet: An on-line Lexical Database. *International Journal of Lexicography*, 235–244 (1993)
16. Wang, H., Feng, Z.Y., Yang, S., Chen, S.Z.: Service Network: An Infrastructure of Web service. In: 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, pp. 303–308 (2009)

Energy Efficient Activity Recognition Based on Low Resolution Accelerometer in Smart Phones

Yunji Liang, Xingshe Zhou, Zhiwen Yu, Bin Guo, and Yue Yang

School of Computer Science, Northwestern Polytechnical University,
Shaanxi, Xi'an, China, 710129

lemonliang@mail.nwpu.edu.cn,

{zhouxs, zhiwenyu, guob}@nwpu.edu.cn, nwpu.yangyue@gmail.com

Abstract. Smart phone is becoming an ideal platform for continuous and transparent sensing with lots of built-in sensors. Activity recognition on smart phones is still a challenge due to the constraints of resources, such as battery lifetime, computational workload. Keeping in view the demand of low energy activity recognition for mobile devices, we propose an energy-efficient method to recognize user activities based on a single low resolution tri-axial accelerometer in smart phones. This paper presents a hierarchical recognition scheme with variable step size, which reduces the cost of time consuming frequency domain features for low energy consumption and adjusts the size of sliding window to improve the recognition accuracy. Experimental results demonstrate the effectiveness of the proposed algorithm with more than 85% recognition accuracy for 11 activities and 3.2 hours extended battery life for mobile phones.

Keywords: energy efficient, hierarchical recognition, low resolution, activity recognition, tri-axial accelerometer.

1 Introduction

Activity is one of the most important contexts in pervasive computing. User activity has been used to evaluate the metabolic energy expenditure; to explore the activity patterns; and to enhance interactions in social groups [1-4]. To recognize user activity continuously, we need a nonintrusive, light weight, and real-time recognition scheme. Fortunately, the mobility, commercial built-in sensors, and nonintrusive detection make smart phones an ideal platform for monitoring user activities. However, activity recognition on smart phones is still a challenge due to constraints of low battery capacity and computational workload.

The sampling rate to assess daily physical activities should be no less than 20 Hz [5-7]. However, the long-term sensing with the full working load of sensors is energy-consuming. For example, the battery life of Samsung i909 reaches up to over 30 hours when all applications and sensors are turned off. But the battery life declines to 5.5 hours (50 Hz) and 8 hours (20 Hz) respectively, when the accelerometer is monitored.

Toward the energy efficiency of activity recognition based on the single accelerometer in the smart phone, it is feasible to reduce the energy consumption by adopting a lower sampling frequency. Lower sampling frequency means less work time for the heavy-duty sensors. However, low sampling frequency may result in the loss of important sampling data, reducing the recognition rate with low resolution sensory data [8]. In addition, many classification algorithms are heavy weight and time consuming for mobile devices. In general, the size of sliding window in most classification algorithms is constant. The fixed-step algorithm deteriorates the recognition rate to some extent, which not only reduces the ability to detect short-duration movements, but also occupies lots of resources with the consumption of battery power.

To overcome above issues, we consider two factors - sampling frequency and computational load - in the design of the activity recognition algorithm. Specifically, we propose an energy-efficient method to recognize user activities based on a single low resolution tri-axial accelerometer in the smart phone. The hierarchical recognition scheme reduces the cost of the time consuming frequency domain features for lower computational complexity and adjusts the size of sliding window according to similarity to enhance the recognition accuracy.

The rest of this paper is organized as follows: in Section 2, the related work about activity recognition based on accelerometer is summarized. Then Section 3 describes the process of data collection. Section 4 presents the details of our solution, including the framework of activity recognition, feature extraction, and the hierarchical recognition scheme. The evaluation is given in Section 5. Finally we conclude this paper in Section 6.

2 Related Work

2.1 Activity Recognition

Numerous studies have been conducted about the activity recognition based on accelerometers. The work toward the activity recognition based on the accelerometer is divided into three types roughly.

First, the activity recognition based on multi-accelerometer sensors is conducted [9-12]. Norbert et al. [9] implemented an activity recognition system by using a wristwatch-like device, named MotionBand. Three MotionBand devices are attached to the wrist, hip and ankle to collect the sensory data of user activities. Then all those sensory data is sent to a mobile phone by Bluetooth and is classified using the feed-forward back-propagation neural networks. Although lots of sensors are employed to benefit the recognition, sensors fixed on human body are barriers for users. On one hand, users are confined to the laboratory environment due to constraints of wearable sensors, which reduces the practicability of the prototype in daily life. On the other hand, users are distracted from their tasks. This is contradicted with the vision of pervasive computing for less attention taken from users.

Second, single accelerometer sensor is utilized to benefit the activity recognition [6, 13-15]. A. M. Khan et al. [6] carried out experiments to monitor physical activities based on a tri-axial accelerometer. A hierarchical recognition scheme was proposed to recognize 15 kinds of activities. Activity recognition based on a single accelerometer sensor relies on the design of specialized sensors. Those specialized sensors are not off-the-shelf items and just research-only devices confined to the laboratory. Meanwhile, those specialized sensors are power-consuming due to the wireless communication and the high sampling frequency.

Nowadays, with the advent of smart phones, the sensing abilities of smart phones are strengthened with lots of built-in sensors. Different from most previous work, the daily activity recognition on smart phones uses a commercial mass-marked device rather than a research-only device, and employs a single device conveniently kept in the user's pocket rather than multiple devices distributed across the body [7, 15]. In [7] J. R. Kwapisz et al. employed the accelerometer in the smart phone to recognize 6 categories of activities. However, the power consumption of recognition scheme is not considered in the previous work. Smart phones are resource-limited, the power consumption and the computational workload pose challenges to the activity recognition on smart phones. The classic recognition algorithms are time-consuming and heavyweight for the mobile phone [7].

2.2 Energy Conservation

Energy is a vital resource for mobile devices. The battery limitations pose a challenge to the success of the activity recognition on mobile devices. Y. Wang et al. [16] designed a scalable framework of energy efficient mobile sensing system (EEMSS) for automatic user state recognition. The core component of EEMSS is a sensor management scheme which defines user states and state transition rules by an XML configuration. The sensor management scheme allocates the minimum set of sensors and invokes new sensors when state transitions happen. P. Zappi et al. [17] selected the minimum set of sensors according to their contribution to classification accuracy during data training process and tested this solution by recognizing manipulative activities of assembly-line workers in a car production environment. X. Li et al. [18] applied machine learning technologies to infer the status of heavy-duty sensors for energy efficient context sensing. They tried to infer the status of high energy consuming sensors according to the outputs of light weight sensors. The existing solutions extend the battery life by the collaboration of multi-sensors and the reduction of sensor work time. Different from the above studies, we try to recognize user activity by a single accelerometer. The collaboration of multi sensors is infeasible in our solution.

Different from the previous work, we intend to address the energy consumption issue in accelerometer-based physical activity recognition. The sampling frequency has a dominating effect on the density of raw sampled data. To reduce the computational workload and the work time of sensors, the lower sampling frequency should be adopted to capture less raw data. On the other hand, features are important for the computational complexity as well. The frequency domain features, which need to transform the signal into frequency domain, is time consuming. Therefore, we try

to reduce the cost of the frequency-domain feature extraction. Additionally, Algorithms involved in the previous work are time-consuming. Those algorithms usually are performed on PC or workstation. The computational complexity is overwhelming for the resource-limited devices.

3 Data Acquisition

Human activities consist of some basic movements, such as walking, sitting, standing, running etc. The exploration of basic activities contributes to the far-reaching understanding of user activities with semantic information. We select the most common activities recognized in previous work as target activities. Target activities of our study are shown in Table 1.

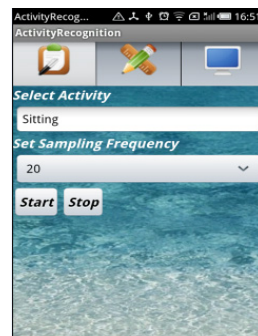
Table 1. Definition of target activities

Activity Type	Activity
Static	Standing, Sitting, Lying(prone), Lying (supine), Driving
Repetitive	Walking, Running, Ascending stairs, Descending stairs, Cycling, Jumping

A total of 24 subjects, 16 males and 8 females with age ranging from 22 to 35, were involved. All of them were recruited from the school of computer science, Northwestern Polytechnical University, China including students and staff in exchange for the use of a high-end smart phone for the duration of the experiment. Each subject was assigned with a smart phone, including HTC G11, Samsung i909. The range of the tri-axial accelerometer outputs is $\pm 2g$. The orientations of the tri-axial accelerometer in the smart phone (HTC) are presented in Fig. 1a. An android application was developed and pre-installed to record the real-time outputs of the accelerometer (See Fig. 1b).



(a) Orientations of accelerometer



(b) Interfaces on mobile phones

Fig. 1. Experimental interfaces on mobile phones

The subjects manually label their activities and set the sampling frequency in advance through the application as shown in Fig. 1b. The optional frequencies are: 0.5 Hz, 2 Hz, 10 Hz, and 20 Hz. All subjects are divided into four groups equally and each group utilizes the same sampling frequency. They launched the application when they began to perform the activities, selected the setups and put phones into their front pant pockets. When subjects started to perform an activity, a log file whose name contains information about the activity type and sampling frequency was produced with the contents of timestamps and accelerometer outputs on each axis. When activities were finished, they took out the phone and stopped the application. This process was repeated for daily activities. We monitored the user activities during two weeks. To rule out the dirty data of each log file, we cut off the data at the beginning and the end of the log files.

4 Activity Recognition

4.1 The Framework of Activity Recognition

As shown in Fig. 2, the framework of activity recognition consists of two parts: the offline data training and the online classification. The offline data training extracts features from the sampled data and constructs template for each activity respectively. The online classification extracts features of the sliding window, calculates the similarity between the target activity and templates, and selects a suitable class as the label of the sampled data in the sliding window.

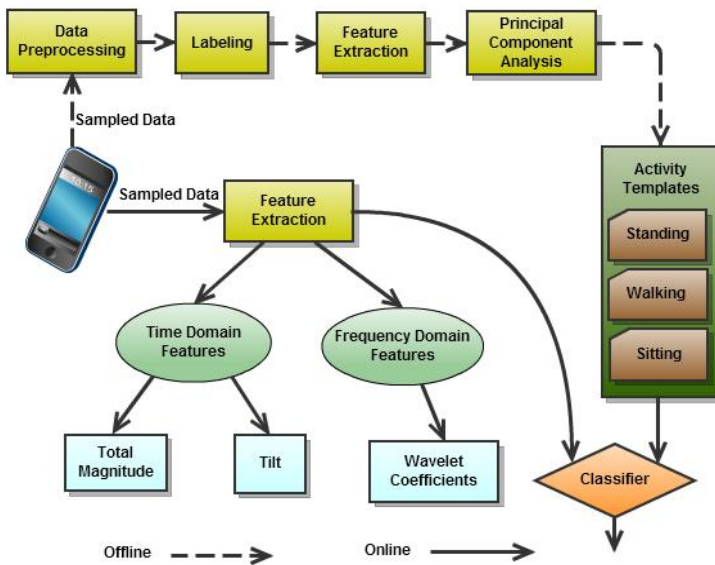


Fig. 2. The Framework of Activity Recognition

The offline data training consists of four steps. The data preprocessing takes charge of the data cleaning and the data representation. The labeling defines the class of each sampled data using all the target activities. The feature extraction captures characteristics of each activity. Principal Component Analysis (PCA) is introduced to select the most discriminative features. Finally, a template will be generated for each activity. To reduce the time consumption, the offline training is performed on the PC or workstation. Only those results are transplanted into the smart phone to serve as templates of user activities.

For the online process, we design a light weight, hierarchical recognition algorithm with variable steps. First, time-domain features are utilized to classify user activities based on the template-based classification. However, some activities are indistinguishable when only the time-domain features are taken into consideration. Then the frequency-domain features are introduced and the size of sliding window is segmented. For each such small section, the decision tree algorithm is performed based on the combined features.

4.2 Feature Extraction

Features play important roles for activity recognition. As mentioned above, the feature extraction is performed in two phases. The time-domain features are extracted from samples directly. Only when those time-domain features are unable to discriminate user activities, the frequency-domain features are introduced. The following presents the related features and their number.

- Mean of each axis (3): The acceleration signals of human activities on three axes are different as illustrated in Fig. 3.
- Deviation of each axis (3): The deviation indicates the fluctuation of signal magnitude on each axis.
- Mean of Total Magnitude (1): The intensity of user activity is a significantly important metric to discriminate activities. Based on the sampled data on each axis, Total Magnitude (TM) is calculated to according to Equation (1).

$$TM = \sqrt{x^2 + y^2 + z^2} \quad (1)$$

- Deviation of Total Magnitude (1): Like the deviation on each axis, the deviation of TM cues the fluctuations of TM.
- Tilt (1): Tilt is employed to calculate the angle between the gravity and the y-axis. The tilt gives a cue of body posture, e.g. forwardness or backwardness. The tilt is evaluated based on Equation (2).

$$\theta = \arccos \frac{y}{g} \quad (2)$$

- Linear regressive coefficients (4): To reveal the relationship among the TM and the magnitudes on three axes, we calculate the coefficients based on the linear

regression. Those coefficients enclose the contributions of each axis to the total magnitude. Linear regressive coefficients are calculated according to Equation (3), where S is the matrix of linear regressive coefficients, W represents the matrix of magnitudes on each axis and the Q is the matrix of total magnitudes.

$$S = (W^T W)^{-1} W^T Q \quad (3)$$

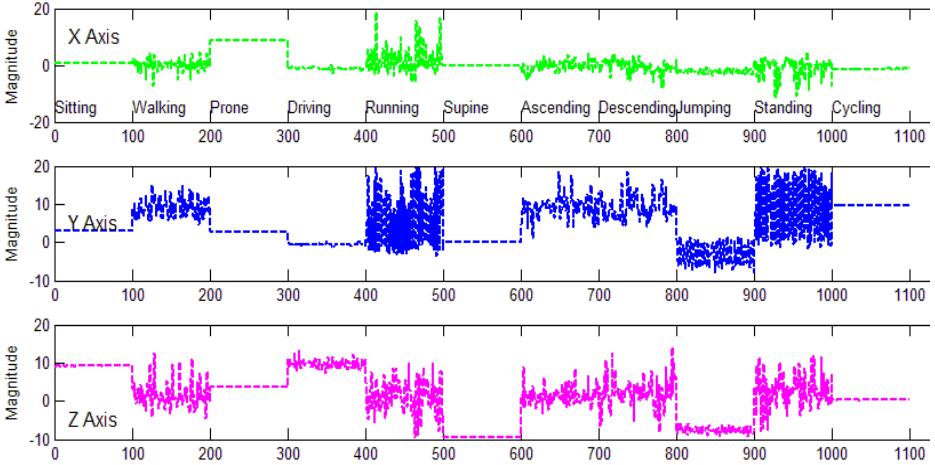


Fig. 3. Acceleration signals of target activities on three axes

- **Wavelet coefficients:** Different activities have discriminative frequency features, especially for repetitive activities. Meanwhile the frequency of human activities is low, thus we extract the low frequency features based on the wavelet analysis.

Different from the previous work, the feature extraction is performed in two steps. At first the time-domain features are extracted as the basic features. As the extraction of frequency domain features is time consuming, thus we try to reduce the opportunity of utilization of the frequency-domain features with the introduction of the two-step feature extraction.

4.3 Hierarchical Recognition Scheme

In general, the size of sliding window in classical classification algorithms such as Decision Tree (DT), Support Vector Machine (SVM) is constant (See Fig. 4a). The fixed-step algorithm deteriorates the recognition rate. The dynamics of activities enlighten the introduction of a hierarchical recognition scheme with variable step size, which is suitable for both static and repetitive activities.

The comparison of the classic classification algorithms with our proposed hierarchical recognition scheme is illustrated in Fig 4. As shown in Fig. 4b, differences of our proposed algorithm are in two aspects. Firstly, the feature extraction is completed in two steps, which reduces the opportunity of utilization of the time-consuming

frequency-domain features. Secondly, the size of sliding window is adjusted according to the similarity. When it is indiscriminative, the sliding window is split into small segments equally. Otherwise frequency-domain features are introduced to classify user activities based on decision tree. The hierarchical recognition scheme consists of the following three steps.

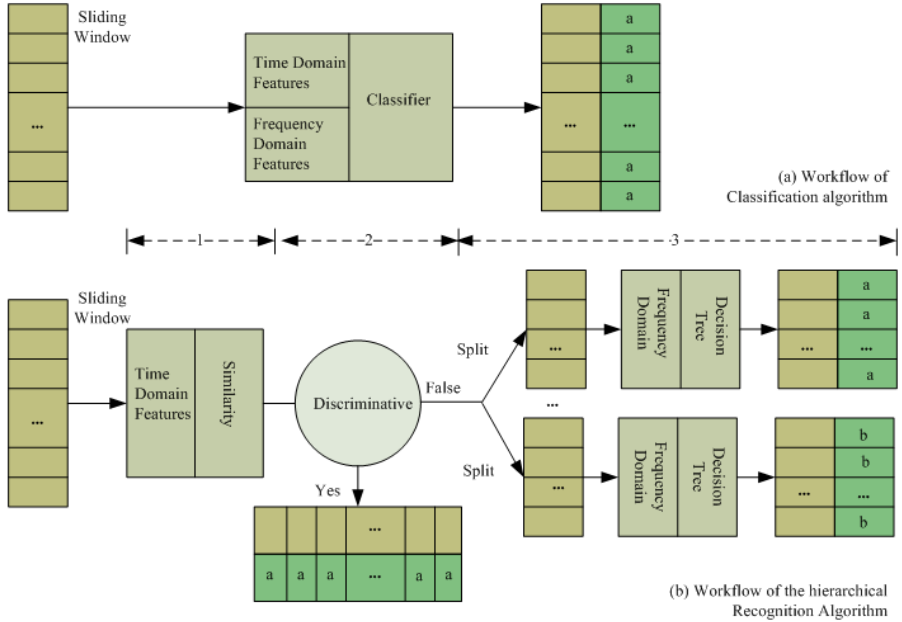


Fig. 4. Comparison of classical classification algorithms and the hierarchical algorithm

Similarity Measurement of Time-Domain Features: Similarity is utilized to demonstrate the likelihood of current inputs to the activity templates. For the sliding window, the time-domain features are calculated and represented with vector X . Then every activity template compares its characteristic parameter vector Y with the vector X according to Equation (4). Here, Y is the vector of time-domain features, which is obtained in the offline data training process. After the similarity measurement with each activity templates, a vector C is generated. The size of C is $1 \times M$, where M is the number of predefined activities and $c_i \in [0, 1]$, $1 \leq i \leq M$. Each element c_i in the vector C denotes the extent to which a feature vector belongs to a given class.

$$c_i = \cos(X, Y) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|} \quad (4)$$

Evaluation of Similarity Discrete Degree: Occasionally differences among those similarities are indiscriminative, e.g. the difference of c_i and c_j is tiny. Under such condition, it is not convincing to classify the user activity into the class with the highest similarity. Thus, we need to evaluate the discrete degree of the vector C .

A vector \mathbf{C} is indistinguishable, if it satisfies one of the following two constraints (See Equation (5) and (6)).

$$\sqrt{\frac{\sum (c_i - \bar{c})^2}{M}} \leq \delta \quad (5)$$

According to Equation (5), the standard deviation of vector \mathbf{C} is calculated. If the standard deviation of vector \mathbf{C} is smaller than a threshold δ , \mathbf{C} is indistinguishable, where δ is a constant and belongs to $[0.1, 0.2]$. In our experiments, δ equals to 0.15.

For the vector \mathbf{C} , elements are sorted in ascending order, represented with $c_{(1)}$, $c_{(2)}, \dots, c_{(M)}$ respectively. Then the differences are calculated according to $d_j = c_{(j+1)} - c_{(j)}$, $1 \leq j \leq M-1$. In Equation (6) letter E represents the expectation. According to (6), if the difference of $c_{(M)}$ and $c_{(M-1)}$ is smaller than the expectation, the vector \mathbf{C} is indiscriminative.

$$c_{(M)} - c_{(M-1)} \leq E(\{d_j\}) \quad 1 \leq j \leq M-1 \quad (6)$$

Based on Equation (5) and (6), we are able to judge whether the similarity vector \mathbf{C} is indistinguishable. If the similarity vector \mathbf{C} is differentiable, it means that those time-domain features are capable of explicitly differentiating those activities. Thus, we should classify the current activity into the corresponding activity with the largest similarity in the vector \mathbf{C} . On the contrary, if the vector \mathbf{C} is indiscriminate, the frequency domain features are introduced to classify user activities.

Table 2. Hierarchical recognition algorithm

Inputs: accelerometer signals

Outputs: classification results for accelerometer signals

1: Set the size of the sliding window with N .

2: **For each sliding window**

3: Extract time-domain features, including the mean of magnitude on each axis etc.

4: Calculate the similarity vector \mathbf{C} among the sliding window and the predefined activity templates based on (4).

5: Justify whether the vector \mathbf{C} is discriminative based on (5) and (6).

6: **IF** vector \mathbf{C} is discriminative

7: User Activity = $\max\{C_1, \dots, C_i, \dots, C_M\}$

8: **ELSE**

9: Divide the sliding window into K small sections equally, length of each section is $L = N/K$;

10: **For each small section**

11: Obtain low-frequency wavelet coefficients and time-domain features;

12: Based on those combined features, classify those data using the decision tree algorithm.

13: **End For**

14: **End IF**

15: **End For**

Hierarchical Recognition Algorithm: For the recognition algorithms with fixed step size, it is crucial for the recognition rate to select a reasonable step length. If the step length is long, it is prone to the ignorance of short-duration activities; if the step length is short, it leads to lots of redundant computational workload. Thus, we propose a fine-grain recognition algorithm with variable step size, which adjusts the size of sliding window according to similarities to enhance the recognition accuracy. The details of the proposed algorithm are presented in Table 2. For combined features decision tree (C4.5) algorithm is utilized as it provides a good balance between accuracy and computational complexity [8, 19].

As elaborated above, the hierarchical algorithm extracts fewer features and those features are calculated in different phases, which benefit the decline of the computational load. Furthermore, the size of sliding window is adjusted according to the similarities, which contributes to the recognition of short-duration activities with the increase of recognition rate.

5 Experiment and Evaluation

To evaluate the proposed algorithm on mobile devices, we perform person-independent experiment in terms of recognition accuracy, power consumption and computational load.

5.1 Activity Recognition Rate

Our hierarchical recognition algorithm includes two phases. We analyze the proportion of the two phases where inputs are classified and the recognition rate of the hierarchical scheme. The 5-folder cross-validation is used to evaluate the hierarchical recognition scheme with 2 Hz sampling frequency.

First, the average recognition rate reaches up to 89.1% (See Table 3). The recognition rate demonstrates the activity recognition based on the low resolution accelerometer with low sampling frequencies is feasible. Although activity recognition with low resolution sensory data is inconsistent with the previous work [5, 6], it is reasonable due to features of user activities including the repeatability, the symmetry and the normality. Regardless of static activities and repetitive activities, the features are repeated periodically.

Second, majority of target activities are recognized in the first phase based on time-domain features, especially for static activities. This demonstrates that those selected time-domain features are very useful to discriminate user activities. And this benefits the decrease of computational load. On one hand, opportunities of the time-consuming frequency-domain feature extraction and the heavyweight decision tree algorithm are minimized, which contributes to the reduction of computational workload. On the other hand, the introduction of classification based on combined features benefits the improvements of recognition rates, during which complex activities such as Ascending and Descending are discriminated based on frequency domain features.

Table 3. Activity Recognition Rates

Activity	Percentage of Records Correctly Recognized		
	Time Domain Features (%)	Frequency Domain Features (%)	Total (%)
Standing	98.98	1.02	98
Sitting	100	0	100
Lying (prone)	100	0	100
Lying (supine)	99.28	0.72	100
Walking	0	100	80
Jumping	0	100	82
Running	56.76	43.24	86
Ascending	0	100	88
Descending	0	100	82
Cycling	97.50	2.50	84
Driving	37.69	62.31	80
Average	53.66	46.34	89.1

5.2 Power Consumption

To test the battery life under different sampling frequencies, we measured the time spans and the recognition accuracy with changes of sampling frequencies when 90% of battery power is consumed (Fig. 5).

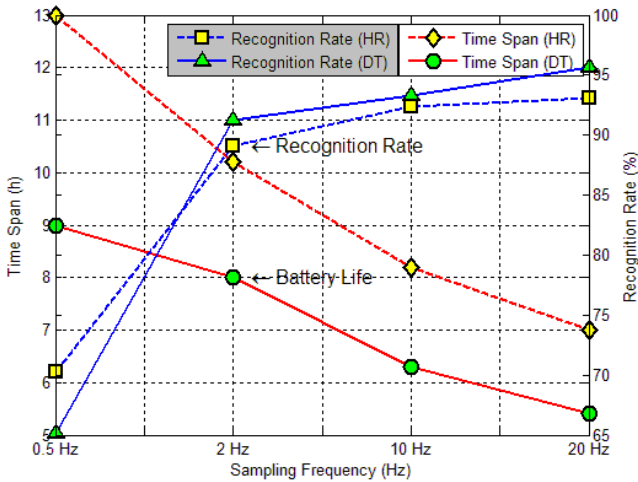


Fig. 5. Time Span and Recognition rate with changes of sampling frequencies

It is obvious that the battery life declines with the increase of sampling frequencies. For a resource-constraint device, the high sampling frequency leads to the rapid depletion of power. Also, it demonstrates that the recognition rates increase with the growth of the sampling frequencies. Although those sampled data indicate more features of user activities and benefits the increase of the recognition rate, it promotes the rapid increase of the power consumption. Additionally, the hierarchical recognition scheme (HR) and decision tree (DT) are compared in terms of time span and the recognition accuracy. As shown in Fig. 5, although the decision tree outperforms the proposed algorithm in recognition accuracy, the battery life is longer in our solution.

To reduce the power consumption and achieve better recognition accuracy, we adopt a reasonable sampling frequency. It is a tradeoff between the power consumption and the recognition rate. Compared with the battery life of 20Hz, the battery life of 2 Hz is lengthened by 3.2 hours and the average recognition rate of the proposed algorithm is over 85%. Thus, it is considered that the 2 Hz is a suitable frequency for the user activity recognition based on the tri-axial accelerometer.

5.3 Computational Load

We aim to provide a model to evaluate the computational workload of the proposed algorithm using the time complexity and compare our proposed recognition algorithm with decision tree in term of time complexity.

As elaborated in the previous section, the proposed algorithm consists of two steps: the recognition based on time-domain features and the recognition based on combined features, denoted with P_1 and P_2 respectively. As the time domain features extraction and the template-based similarity measurement are contained in the P_1 , the time consumption of P_1 , $T(P_1)$ is constant. By contrast, the time consumption of P_2 , $T(P_2)$ is variable due to the variability of the frequency domain features and the classification process.

To evaluate the time complexity of the recognition scheme, the probabilities of user activities are taken into consideration. The probability set ϕ of user activities is presented by $\phi = \{p_1, p_2, \dots, p_i, \dots, p_n\}, 0 \leq p_i \leq 1$, where n is the type of user activities, p_i is the probability of the i^{th} activity, and all the elements in ϕ satisfy $\sum_i^n p_i = 1$. Due to the repeatability of user daily activities, the probability of each activity is calculated by the statistical method.

Meanwhile, the percentages of the two recognition stages, which mean how many sampled data of a particular activity are correctly discriminated by each phase, are important factors to evaluate the time complexity as well. Here, the percentages of the two phases for a specified activity are denoted with u_i and v_i respectively. As shown in Table 3, the percentages of the two phases verify for different activities. The time complexity of the i^{th} activity is measured according to Equation (7), where $T_i(P_2)$ is the time consumption in the second step for the i^{th} activity.

$$\begin{aligned}
 t_i &= u_i \times T(P_1) + v_i \times (T(P_1) + T_i(P_2)) \\
 0 &\leq u_i \leq 1 \\
 0 &\leq v_i \leq 1 \\
 u_i + v_i &= 1
 \end{aligned}
 \tag{7}$$

Average Time Complexity (ATC) is presented in Equation (8), where n is the types of target activities. As p_i in Equation (8) is measured by analyzing the daily activities with statistical method, u_i, v_i are demonstrated in Table 3, and $T(P_1)$ is constant due to the fixed time consumptions of the time domain feature extraction and that of the similarity measurement in phase one. Thus, $T_i(P_2)$ is the only variable in Equation (8).

$$ATC = \sum_{i=1}^n p_i t_i = \sum_{i=1}^n p_i \times (T(P_1) + v_i \times T_i(P_2))
 \tag{8}$$

To evaluate the accuracy of Equation (8), we extract 500 sample data from each activity and construct a new test set to calculate the time consumption. To simplify the computation, here we use the mean of $T_i(P_2)$ as a substitution of $T_i(P_2)$. As the size of target activity set is 11, $n = 11$. For this test set, the p_i is same for each activity and equals to $1/11$. And the pair of $\langle u_i, v_i \rangle$ for every activity is presented in Table 3. Thus, the value of ATC is 4.81 ms.

We measured the execution time of the hierarchical recognition scheme. Compared with the ATC, the execution time is approximate to the theoretical value as shown in Fig. 6. Meanwhile, our experimental results demonstrate that our proposed

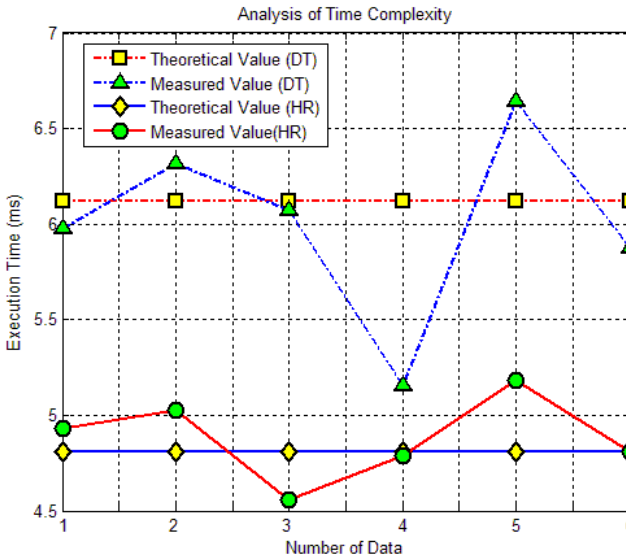


Fig. 6. Comparison of the theoretical value and the measured value

hierarchical recognition scheme outperforms the decision tree. As shown in Fig. 6, the average execution time of DT is longer than that of the HR, which confirms that HR benefits the decline of the time complexity and the computational complexity.

6 Conclusion

With the popularity of the smart phone, it is becoming an ideal platform for activity recognition based on the built-in accelerometer sensor. The constraints of mobile phones such as power consumption, computational load raise a challenge to the activity recognition. In this paper, we presented an approach for activity recognition with a hierarchical scheme for low resolution accelerometer data on the mobile phone. To achieve the goal of energy efficient activity recognition on the cell phone, we propose a hierarchical scheme with variable step size. To evaluate the validation of the method, total 24 healthy subjects are recruited to perform the 11 activities in their daily life. The average recognition rate of the proposed algorithm is over 85%, and the battery lifetime is extended by 3.2 hours. The experimental results demonstrate that the proposed hierarchical scheme not only reduces the power consumption with low resolution sensor data, but also classifies activities with good recognition rate.

Acknowledgments. This work was partially supported by the National Basic Research Program of China (No. 2012CB316400), the National Natural Science Foundation of China (No. 60903125, 61103063), the Program for New Century Excellent Talents in University (No. NCET-09-0079), Microsoft and the Doctorate Foundation of Northwestern Polytechnical University.

References

1. Kawahara, Y., Ryu, N., Asami, T.: Monitoring Daily Energy Expenditure Using a 3-Axis Accelerometer with a Low-Power Microprocessor. *International Journal on Human-Computer Interaction* 1(5), 145–154 (2009)
2. Kim, E., Helal, S., Cook, D.: Human Activity Recognition and Pattern Discovery. *IEEE Pervasive Computing* 9(1), 48–53 (2010)
3. Gu, T., Wang, L., Wu, Z., Tao, X., Lu, J.: A Pattern Mining Approach to Sensor-Based Human Activity Recognition. *IEEE Transactions on Knowledge and Data Engineering* 23(9), 1359–1372 (2011)
4. Nijholt, A., Zwiers, J., Peciva, J.: Mixed reality participants in smart meeting rooms and smart home environments. *Personal and Ubiquitous Computing* 13(1), 85–94 (2009)
5. Bouten, C., Koekkoek, K., Verduin, M., Kodde, R., Janssen, J.D.: A triaxial accelerometer and portable data processing unit for the assessment of daily physical activity. *IEEE Transactions on Biomedical Engineering* 44(3), 136–147 (1997)
6. Khan, A.M., Lee, Y., Lee, S.Y., Kim, T.: A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE Transactions on Information Technology in Biomedicine* 14(5), 1166–1172 (2010)
7. Kwapisz, J.R., Weiss, G.M., Moore, S.A.: Activity Recognition using Cell phone Accelerometers. *ACM SIGKDD Explorations* 12(2), 74–82 (2010)

8. Maurer, U., Smailagic, A., Siewiorek, D.P., Deisher, M.: Activity Recognition and Monitoring Using Multiple Sensors on Different Body Positions. In: Proc. of International Workshop on Wearable and Implantable Body Sensor Networks, pp. 113–116 (2006)
9. Györfi, N., Fábrián, Á., Hományi, G.: An Activity Recognition System for Mobile Phones. *Mobile Networks and Applications* 14(1), 82–91 (2009)
10. Mannini, A., Sabatini, A.M.: Machine Learning Methods for classifying Human physical activity from on-body accelerometers. *Sensor* 10(2), 1154–1175 (2010)
11. Krishnan, N.C., Juillard, C., Colbry, D.: Recognition of hand movements using wearable accelerometers. *Journal of Ambient Intelligence and Smart Environments* 1, 143–155 (2009)
12. Ruch, N., Rumo, M., Mader, U.: Recognition of activities in children by two uniaxial accelerometers in free-living conditions. *European Journal of Applied Physiology* 111(8), 1917–1927 (2011)
13. Lee, M., Khan, A.M., Kim, J., Cho, Y., Kim, T.: A Single Tri-axial Accelerometer-based Real-time Personal Life Log System Capable of Activity Classification and Exercise Information Generation. In: Proc. of 2010 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, pp. 1390–1393 (2010)
14. He, Z., Liu, Z., Jin, L., Zhen, L., Huang, J.: Light weightness Feature – A Novel Feature for single Tri-axial accelerometer based Activity Recognition. In: Proc. of 19th International Conference on Pattern Recognition, pp. 1–4 (2008)
15. Ravi, N., Dander, N., Mysore, P., Littman, M.L.: Activity Recognition from Accelerometer Data. In: Proc. of the 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference, pp. 1541–1546 (2005)
16. Wang, Y., Lin, J., Annavaram, M., Quinn, J.A., Jason, H., Bhaskar, K., Sadeh, N.: A Framework of Energy Efficient Mobile Sensing for Automatic User State Recognition. In: Proc. of the 7th ACM International Conference on Mobile Systems, Applications, and Services, pp. 179–192 (2009)
17. Zappi, P., Lombriser, C., Stiefmeier, T., Farella, E., Roggen, D., Benini, L., Tröster, G.: Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection. In: Verdone, R. (ed.) EWSN 2008. LNCS, vol. 4913, pp. 17–33. Springer, Heidelberg (2008)
18. Li, X., Cao, H., Chen, E., Tian, J.: Learning to Infer the Status of Heavy-Duty Sensors for Energy Efficient Context-Sensing. *ACM Transactions on Intelligent Systems and Technology* (unpublished)
19. Bao, L., Intille, S.S.: Activity Recognition from User-Annotated Acceleration Data. In: Ferscha, A., Mattern, F. (eds.) PERSASIVE 2004. LNCS, vol. 3001, pp. 1–17. Springer, Heidelberg (2004)

Energy Efficient Allocation of Virtual Machines in Cloud Computing Environments Based on Demand Forecast

Jian Cao, Yihua Wu, and Minglu Li

Shanghai Key Laboratory of Scalable Computing and Systems, Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
{cao-jian, li-ml}@cs.sjtu.edu.cn,
darwink@sjtu.edu.cn

Abstract. In cloud computing environments, demands from different users are often handled on virtual machines (VMs) which are deployed over plenty of hosts. Huge amount of electrical power is consumed by these hosts and auxiliary infrastructures that support them. However, demands are usually time-variant and of some seasonal pattern. It is possible to reduce power consumption by forecasting varying demands periodically and allocating VMs accordingly. In this paper, we propose a power-saving approach based on demand forecast for allocation of VMs. First of all, we forecast demands of next period with Holt-Winters' exponential smoothing method. Second, a modified knapsack algorithm is used to find the appropriate allocation between VMs and hosts. Third, a self-optimizing module updates the values of parameters in Holt-Winters' model and determines the reasonable forecast frequency. We carried out a set of experiments whose results indicate that our approach can reduce the frequency of switching on/off hosts. In comparison with other approaches, this method leads to considerable power saving for cloud computing environments.

Keywords: cloud computing, power consumption, demand forecast, allocation of virtual machines, modified knapsack algorithm, self-optimization.

1 Introduction

Cloud computing emerges as an efficient approach to meet the growing computational requirements of commercial projects and scientific research ones in a cost-effective way. IaaS (Infrastructure as a Service), the delivery of the computing infrastructure as a fully outsourced service, accelerates the development of cloud computing. In recent years, increasing demands for computational resources have led to significant growth in the number of cloud computing servers, along with an estimated doubling in the energy used by these servers and the power and cooling infrastructures that support them [1]. High power consumption gives rise to a large amount of operational cost which can accumulate more than the construction cost of servers and infrastructures in a short period of time. As a result, power management is important to cost control for cloud computing environments.

The adoption of virtualization brings about the problem of how to allocate VMs among hosts reasonably. On the one hand, more hosts are added to current environment when computing resources cannot meet with the requirements of all VMs. Service providers of cloud computing have to guarantee Quality of Service (QoS) according to Service Level Agreement (SLA), which results in switching on more hosts to deal with resource demands when needed. On the other hand, shutting down some hosts on which all deployed VMs are idle is a way to conserve power consumption. But energy is consumed while a host is being powered up or down and not performing any useful work. The frequent switching on/off hosts which usually needs two to three minutes also leads to the delay of task execution. One way to avoid this situation is to keep idle hosts in standby mode, which consumes much less power than switching on/off frequently. Therefore if we know that the amount of demand will increase in a very short period, we can keep some idle hosts waiting for the future demand rather than switching them off now and switching them on when needed.

In this paper, we propose an approach to make this decision with the help of demand forecast. Our approach can allocate VMs to hosts in accordance with time-variant demand of seasonal pattern. We define demand as the number of requests for VMs from each user. The result of our work is verified by experiments performed on the basis of CloudSim [2], a simulation framework for cloud computing environments.

The remainder of this paper is organized as follows. Section 2 presents the related works. Section 3 describes the system model behind our implementation and experiments. Section 4 introduces the design of algorithm in details. Experimental results are presented in section 5. Section 6 concludes the paper and discusses about the possible directions for future work.

2 Related Work

In recent years, extensive efforts have been put into the research of the VM allocation in cloud computing environments. Walsh et al. [3-4] proposed an approach based on utility function to dynamically manage data centers with different hosts. The utility function is designed for two levels. The service level is used to calculate resource demands of application and the resource level is used to allocate resources among applications. Walsh laid emphasis on the increase of resource utilization, while our focus is on the reduction of power consumption.

Beloglazov and Buyya [5] proposed an approach using Best Fit Decreasing algorithm to allocate VMs among hosts. VMs are sorted in descending order by current utilization and allocated to a host that provides the least increase of power consumption. The advantage of this approach is the stable utilization of resources while the disadvantage is that threshold policy may results in unnecessary migration of VMs and switching on/off hosts.

Bobroff et al. [6] put forward a dynamic management algorithm to reduce required active physical resources without violating the SLA agreement. The resource demand of VMs is predicted and sorted in descending order. Furthermore, the first-fit bin-packing heuristic is used to minimize the active hosts at each interval. However, this

algorithm only takes CPU into account, which is not enough for real application. This approach is similar to our proposed one, but it doesn't count the wasted energy due to the unnecessary switching on and off. More often than not, it is more energy efficient to keep hosts standby than switch them off and on in a very short period.

Khanna et al. [7] proposed a method to determine a VM configuration while minimizing the number of hosts needed. The placement of VMs is triggered to revise by events such as change of CPU utilization and memory availability. The strength of this method is that the authors take into account VM migration costs in terms of the additional CPU cycles and memory needed to stop a VM. But the placement algorithm in [7] does not consider power saving by switching off unneeded hosts.

Steinder et al. [8] presents a system that manages heterogeneous workloads to their performance goals by dynamic reallocation of VMs. The VMs are migrated as needed between host machines as new tasks arrive. While the placement approach in [8] consolidates workloads, it does not save power by switching off unneeded machines, and does not consider the cost of the control incurred during migration of the VMs.

The authors of [9] solved the problem of placing application instances on a given set of server machines to adjust the amount of resources available to applications in response to varying resource demands. Though this problem is similar to the allocation of VMs in cloud computing environments, they do not consider the physical machines in standby mode to reduce power consumption.

3 A System Model for VMs Allocation

In cloud computing environments, it is allowed to launch VMs with a variety of operating systems, load them with users' own custom application, manage network's access permissions and finish computing tasks as required. These diverse applications result in unpredictable demands. In general, service providers start, terminate and monitor hosts to present on-demand service, but the following issues cannot be avoided: firstly, some hosts have to be switched on or off frequently; secondly, unbalanced distribution of load exists among different hosts.

Without loss of generality, suppose there are m users $\{U_1, U_2, \dots, U_m\}$, corresponding to m types of VMs $\{V_1, V_2, \dots, V_m\}$. The demand from U_i can only be handled by V_i and V_i can be launched on various hosts depending on its resource requirements. An allocation algorithm should be applied to map VMs to hosts on condition that information of needed VMs to satisfy demands from users is gathered.

Figure 1 is the system architecture to implement our solution. As shown in Figure 1, the cloud computing environment is made up of many heterogeneous hosts. It is assumed that each user's demand corresponds to one VM which can be allocated to any host as long as this host can provide enough computing resources to run this VM.

The key components in this architecture are demand analyzer, scheduler and self-optimizing module. Demand analyzer is responsible for collecting, analyzing and predicting users' demands. It relies on a prediction model to forecast future load change. In order to make the prediction model work, its parameters' values should be estimated based on analysis of historical data. Demand analyzer provides scheduler with the forecasted resource demands as input for allocation process of VMs. In

addition to allocate VMs to some hosts, scheduler also determines which hosts to be shut down and which ones to be kept standby. The tasks of self-optimizing module are to estimate and update the values for the parameters in prediction and to select the reasonable forecast frequency which has a big influence on the results of our method.

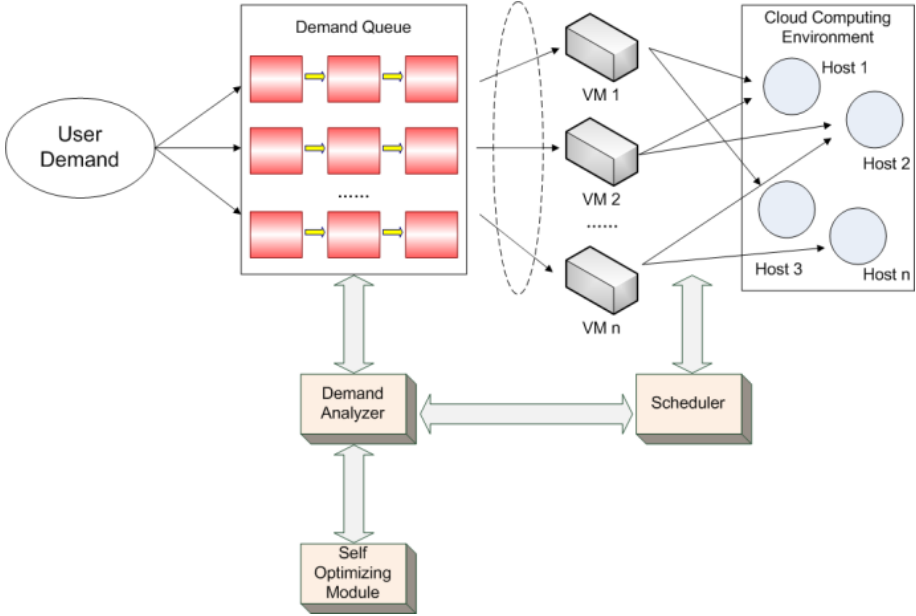


Fig. 1. System architecture for VM allocation

In this system, Holt-Winters’ additive model is selected for demand prediction. A modified knapsack algorithm is embedded in the scheduler. And self-optimizing module makes use of hill climbing method to determine optimized values for parameters and forecast frequency automatically.

4 Algorithms

4.1 Problem Statement

In this paper, we take two kinds of resources into consideration: computing cores and memory. The formal representation of this problem is as below:

For VMs:

$\{V_1, V_2, \dots, V_m\}$, m is total number of types.

$V_i: \{v_{i1}, v_{i2}, \dots, v_{ip_i}\}$, p_i is total number of VMs for type i .

For hosts:

$\{h_1, h_2, \dots, h_n\}$, n is total number of hosts.

$capacity_{core}(k) \forall k \in \{1, 2, \dots, n\}$ represents available cores of host k .

$capacity_{memory}(k) \forall k \in \{1, 2, \dots, n\}$ represents available memory of host k .

For each VM:

$$core_{ij}(k) = \begin{cases} c_{ij} & \text{if } v_{ij} \text{ is deployed on } h_k \\ 0 & \text{if } v_{ij} \text{ is not deployed on } h_k \end{cases}$$

$$memory_{ij}(k) = \begin{cases} m_{ij} & \text{if } v_{ij} \text{ is deployed on } h_k \\ 0 & \text{if } v_{ij} \text{ is not deployed on } h_k \end{cases}$$

Where c_{ij} represents the required number of cores for some VM, m_{ij} represents the required amount of memory for some VM.

Then the problem becomes:

For $\forall i \in \{1, 2, \dots, m\}$ $p_i = Forecast_i(t + \Delta t)$, we should keep: $\forall k \in \{1, 2, \dots, n\}$,

$$\sum_{i=1}^m \sum_{j=1}^n core_{ij}(k) \leq capacity_{core}(k)$$

$$\sum_{i=1}^m \sum_{j=1}^n memory_{ij}(k) \leq capacity_{memory}(k)$$

Where for each type of VM, the actual number of VMs to be deployed on hosts should be equal to the forecasted value $Forecast_i(t + \Delta t)$. On the other hand, $\sum_{i=1}^m \sum_{j=1}^n core_{ij}(k)$ or $\sum_{i=1}^m \sum_{j=1}^n memory_{ij}(k)$, total resources required for all the VMs deployed on a certain host should be less than or equal to the available resources on that host.

For example, $\{V_1, V_2, V_3\}$ (i.e. $m = 3$) stands for three different types of VMs. According to the prediction, the number of VMs for each type is 120, 42 and 12 respectively. $\{h_1, h_2, \dots, h_{18}\}$ (i.e. $n = 18$) stands for the set of hosts. Each host has limited resources, for instance, $capacity_{core}(10) = 24$ represents the number of available cores of the 10th host is 24. $capacity_{memory}(10) = 65536$ represents the amount of available memory of the 10th host is 65536MB. Then the problem becomes how to allocate 120 VMs of type 1, 42 VMs of type 2 and 12 VMs of type 3 to 18 hosts according to the resource limitations of each host and the resource requirements of each VM.

The details of demand forecast, self-optimizing module, etc. will be discussed in the following part.

4.2 Demand Forecast

In our proposed approach, demand actually refers to the number of requests for VMs. In order to let allocation of VMs to hosts keep pace with dynamic demands from different users, demand forecast is an effective way to deal with such issue. It can be observed from real applications that the following two assumptions are reasonable for the analysis of demands in cloud computing environments: First, demands from the same user are similar. Second, a certain type of demand follows some seasonal

pattern. For example, daily load curve, 10:00-11:00 and 14:00-15:00 are two peaks, while 02:00-03:00 is a valley.

On the basis of these assumptions, we make use of Holt-Winters' exponential smoothing method [10-12] which has ability to adapt to changes in trends and seasonal patterns. The equations of additive model for Holt-Winters are defined as [11]:

$$\text{level: } L_h = \alpha(y_h - S_{h-c}) + (1 - \alpha)(L_{h-1} + T_{h-1}) \quad (1)$$

$$\text{trend: } T_h = \beta(L_h - L_{h-1}) + (1 - \beta)T_{h-1} \quad (2)$$

$$\text{seasonal: } S_h = \gamma(y_h - L_h) + (1 - \gamma)S_{h-c} \quad (3)$$

$$\text{forecast: } F_{h+k} = L_h + kT_h + S_{h+k-c} \quad (4)$$

Where c is the length of the seasonal cycle, for $0 \leq \alpha \leq 1$, $0 \leq \beta \leq 1$, $0 \leq \gamma \leq 1$. We can choose the best values of α , β and γ through trial-and-error process with mean absolute percentage error (MAPE) [10-13] or mean absolute scaled error (MASE) [14] as evaluation criteria. On the other hand, initial values of the level, trend and seasonality are usually determined by two complete seasonal cycles. Equations are listed as below [11]:

$$L_c = \frac{1}{c} \sum_{i=1}^c y_i \quad (5)$$

$$T_c = \frac{1}{c} \left[\frac{y_{c+1} - y_1}{c} + \frac{y_{c+2} - y_2}{c} + \dots + \frac{y_{2c} - y_c}{c} \right] \quad (6)$$

$$S_i = y_i - L_c, i = 1, 2, \dots, c \quad (7)$$

Holt-Winters' exponential smoothing method assumes the time series is composed by a linear trend and a seasonal cycle. It constructs three statistically correlated series (smoothed, seasonal and trend) and projects forward the identified trend and seasonality. Obviously, the seasonal component is prerequisite for this method to model different demands. Furthermore, how to stop this method from being unduly influenced by demands that are unusually high or low (i.e. outliers) is another issue to deal with. In a word, Holt-Winters' exponential smoothing method can only be used to model time-varying demands or user behaviors of some seasonal pattern.

4.3 Modified Knapsack Algorithm for VMs Allocation

The knapsack problem is a problem in combinatorial optimization: Given a set of items, each with a cost and a value, determine the number of each item to include in a collection so that the total cost is less than or equal to a given limit and the total value is as large as possible [15-16]. The VM allocation is a problem for the admission of new requests for virtual machines provisioning and the placement of virtual machines on hosts. Its optimization goal is to produce more benefits with reduction of cost, for example, charge for running instances. Hence, these two problems are similar with

each other and we can use knowledge of knapsack problem to deal with VM allocation.

Our algorithm is made up of bounded and multi dimensional knapsack problems. In our algorithm, we regard each host as a knapsack and each VM as an item. The capacity of knapsack consists of available cores and memory for each host. As mentioned in the problem statement, each item has two different kinds of cost (i.e. cores and memory) and a value. We describe cost by using resource requirements of each VM. On the other hand, value is expressed in terms of Amazon's Elastic Cloud pricing [17]. It is assumed that there are three kinds of items corresponding to three types of VMs: small, large and extra large. Values for them are \$0.12, \$0.48 and \$0.96 per hour respectively. Our aim is to find how to allocate VMs among hosts with the largest value under the conditions of limited capacity.

Table 1. Modified Knapsack Algorithm Pseudo Code

Steps of Pseudo Code

1. Forecast demand;
 2. Get predicted amount of VMs for each type;
 3. for each host in hosts
 4. Get capacity of host;
 5. Initialize array of VM selection;
 6. Initialize array of state transition;
 7. iterate over each type of VM
 8. if both limit are observed
 9. Calculate new value by using state transition;
 10. if new value > current max value
 11. if bounded conditions are met
 12. Update current max value;
 13. Record results into VM selection;
 14. End if;
 15. End if;
 16. End if;
 17. End iterate;
 18. Update bounded conditions for next host;
 19. End for;
-

The equation of state transition is needed to solve knapsack problem with dynamic programming. Here we take 2-dimension knapsack problem as our basic model. The knapsack (i.e. host) will have to bear two kinds of cost when choosing any item (i.e. VM). The 1st kind of cost is the number of cores. The 2nd kind of cost is the amount of memory.

Then the equation of state transition is:

$$r[c(i)][u][w] = \max \left\{ \begin{array}{l} r[c(i-1)][u][w] \\ r[c(i-1)][u - k * x[i]][w - k * y[i]] + v[i] \end{array} \right\} \quad (8)$$

Where $0 \leq k * x[i] \leq U$ and $0 \leq k * y[i] \leq W$

For this equation:

i : The i^{th} category for item

u : Sum for 1st kind of cost

w : Sum for 2nd kind of cost

$x[i]$: The 1st kind of cost for item of i^{th} category

$y[i]$: The 2nd kind of cost for item of i^{th} category

$v[i]$: The value for item of i^{th} category

$c(i)$: The first i categories of items

$r[c(i)][u][w]$: The largest value by choosing the first i categories of items when the sum for 1st kind of cost is u and the sum for 2nd kind of cost is w .

U : The maximum for 1st kind of cost (i.e. capacity of knapsack)

W : The maximum for 2nd kind of cost (i.e. capacity of knapsack)

From the equation of state transition, we know that there are multiple choices (i.e. 0 pieces, 1 piece, 2 pieces...) for each item. No matter which choice is made, conditions $0 \leq k * x[i] \leq U$ and $0 \leq k * y[i] \leq W$ should be met to ensure the destination host has enough resources to deal with demands. For item i , we assign $s[i]$ to represent the range of k and it is:

$$0 \leq s[i] \leq \left\lfloor \min \left(\frac{U}{x[i]}, \frac{W}{y[i]} \right) \right\rfloor \quad (9)$$

As a result, the time complexity of our algorithm is $O(\max(U, W) \sum s[i])$. The better solution in dynamic programming for knapsack problem is to use monotone optimal policy, which is beyond the scope of this paper.

4.4 Self-optimizing Module

The self-optimizing module has two tasks: (1) updating values of parameters in forecasting model; (2) determining reasonable forecast period.

For the first task, initial values of parameters α , β and γ in forecasting model are chosen by MAPE or MASE as evaluation criteria. We take 0.1 as the minimum step to iterate from 0 to 1 for each parameter. The group of α , β and γ with the smallest forecast error is the initial values of parameters for future forecast. Along with the varying demands, MAPE or MASE for current parameters becomes larger and larger, which means the accuracy of forecast declines. Therefore, our self-optimizing approach is to set a threshold for forecast error. Given that forecast error exceeds the threshold, parameters are needed to update with the recent several complete cycles of data through trial-and-error process.

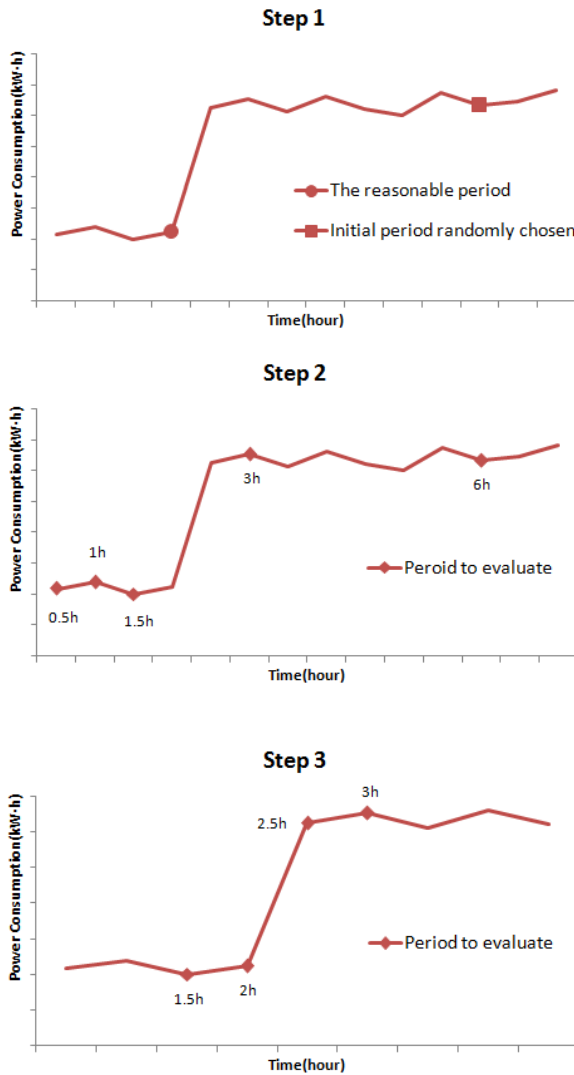


Fig. 2. Process of self-optimization for forecast period

For the second task, self-optimization of forecast period makes use of hill climbing method. Based on our experimental results, there is a sharp increase in the curve of power consumption for different periods and the most reasonable forecast period is close to the start point of this sharp increase. The first step is to select initial period randomly. The second step is to use neighborhood function to get a set of new periods in certain range and evaluate their effect on power consumption. These two steps will be repeated until a platform of low power consumption appears which means the start point

of the sharp increase is found. The third step is to evaluate periods on this platform with number of iterations for knapsack as criterion. Small forecast period leads to more iteration that is needed to work out allocation of VMs to hosts. For example, we make two assumptions: (1) the minimum step of period is 0.5 hour; (2) the most reasonable period is 2 hours. The initial random period in the first step is 6 hours. The set of new periods generated by the neighborhood function in second step is 6, 3, 1.5, 1 and 0.5 (the latter is half of the former). Based on the evaluation result, period of 1.5 hours becomes the start point of the sharp increase, while period of 3 hours is the end point. The set of periods to evaluate in the third step is 3, 2.5, 2 and 1.5. In the end, the forecast period with low power consumption and small number of iterations is found. The self-optimizing process of this example is illustrated in Figure 2.

5 Experiments

5.1 Experimental Setup

Our experiments are performed on CloudSim which is a new, generalized, and extensible simulation framework that allows seamless modeling, simulation, and experimentation of emerging cloud computing infrastructures and application services [2]. By using CloudSim, researchers and industry-based developers can test the performance of a newly developed application service in a controlled and easy to set-up environment [2].

To simulate our algorithm and collect the results, we need to set up the configuration of hosts and VMs.

Host Configuration. To simulate computing and storage abilities of hosts, we take IBM System X3850 X5 as our sample host. Configuration of X3850 is collected from IBM website: 4 CPUs (each with 6 cores), 32GB DDR3 1066Mhz memory [18]. Power parameters are listed in Table 2.

Table 2. Host Power Parameters [18]

Host	Power Rating	Peak Power	Standby Power	Start Duration	Idle Duration
IBM System X3850	1975W*2	2765W*2	98.75W*2	2min	20min

VM Configuration. Experiments are conducted with three types of VMs. Configurations of our VMs are determined with reference to standard instance types of Amazon EC2 which are well suited for most applications. The detailed configurations are listed in Table 3.

Table 3. Virtual Machine Types[19]

Virtual Machine Type	Core(s)	Memory	Storage
Small	1	1.7GB	160GB
Large	4	7.5GB	850GB
Extra Large	8	15GB	1690GB

5.2 Demand Collection

According to our assumption, three types of time-varying resource demands are collected and predicted. The first one is collected from statistics of visit requests published by Shanghai Education Website. The second one is provided by R2Group as example of their tool for website statistics. The third one is from sample.org. All of them include data of three continuous days (72 hours). A complete cycle is set to 24 hours, from 0:00AM-24:00PM, considering seasonal patterns of collected data. The first two complete cycles will help us calculate initial values of level, trend, seasonality and parameters α , β , γ in the equations. The third cycle is used to evaluate the accuracy of our forecasting model.

5.3 Performance of Forecast

Due to similar process of forecast for different types of demands, we just take data from Shanghai Education Website as an example for performance evaluation. Figure 3 illustrates the comparison of actual and forecasted demands. The parameters α , β and γ in forecasting model are set to real value between 0.1 and 1. The exact values of parameters are listed below in Table 4.

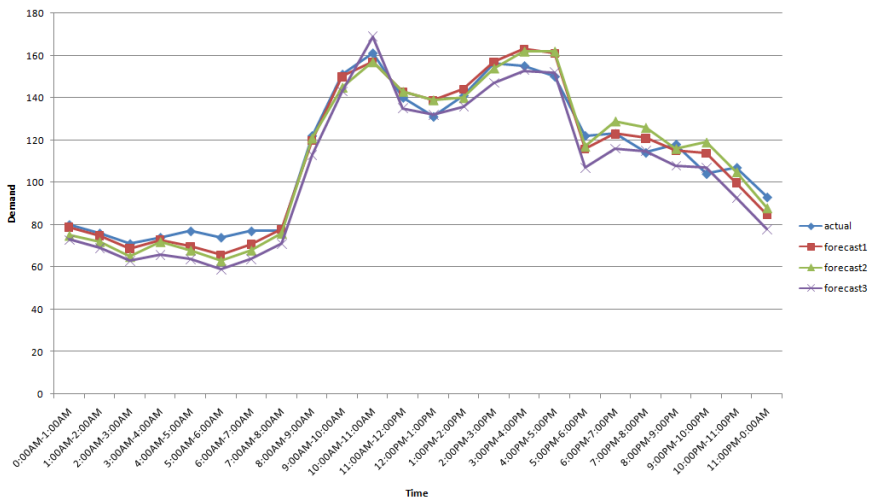
**Fig. 3.** Comparison of actual and forecasted demands

Table 4. Values of Model Parameters

Forecast	α	β	γ	MAPE
forecast1	0.1	0.2	0.3	0.0434
forecast2	0.9	0.2	0.1	0.0562
forecast3	0.5	0.1	0.4	0.0824

As illustrated in Figure 3, Holt-Winters’ exponential smoothing method with appropriate parameters has good accuracy of forecast. Here parameters’ values are determined by comparing the values of MAPE. The lower the value of MAPE, the better accuracy the forecast has. Based on current collected data, we set parameters’ values as $\alpha = 0.1$, $\beta = 0.2$ and $\gamma = 0.3$.

In our experiment, we make predictions for three types of demands which correspond to three types of VMs. These VMs are treated as items to go through the allocation process. The accuracy of forecast for different demands influences the performance of our algorithm (i.e. sum of power consumption) by changing allocation of VMs to hosts.

5.4 Performance of Algorithm

In this part, we evaluate the performance of our algorithm by comparing it with the others: one is without forecast, WF for short and the other is with constant number of hosts which are never shut down, CN for short. It is assumed that the arrival time of demand fits normal distribution of (0, 25) and the execution time of demand fits normal distribution of (30, 144).

The evaluation criteria of our experiment are the total power consumption during a certain period. Here we take 10 hosts as example for CN. As shown in Figure 4, our algorithm with forecast remains lower power consumption than WF for the complete cycle of 24 hours. The major features of WF are: (1) on-demand service; (2) shut

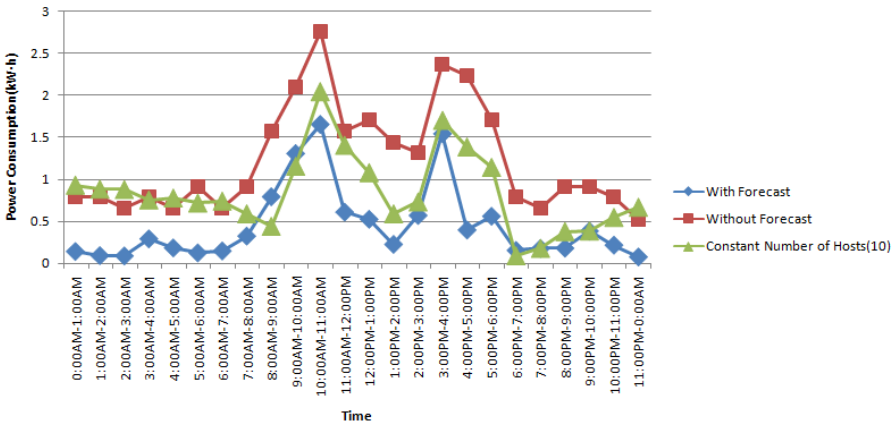


Fig. 4. Performance of different methods with interval of 1 hour

down hosts immediately. Due to no forecasted demand for reference, WF cannot keep hosts standby in advance and wait for future demands in a reasonable period of time. A lot of power is consumed by switching on/off hosts frequently. On the other hand, for every period in CN, a constant set of hosts are kept in active mode (full power state) or standby mode (low power state), which means they will never be shut down. Other hosts work in the same way as the ones in WF. According to our experimental results in Figure 4, WF consumes 29.625 kW·h in total, while our algorithm only consumes 10.734 kW·h, which saves energy by up to 60%.

Figure 5 shows the results with interval of 0.5, 1, 1.5 and 2 hours for comparison among our algorithm, WF and CN with 5, 10 and 15 hosts respectively. Based on actual demand from same users, our algorithm keeps its advantage over WF and CN in power consumption for different intervals. In addition, CN with different number of hosts has lower power consumption than WF for any interval, which indicates that it can save energy to keep reasonable set of hosts in standby mode for each period. This lays the experimental basis for our algorithm. On the other hand, the performance of all algorithms becomes worse with change of period from smaller ones to larger ones, for example, from 1 hour to 1.5 hours. This is because of the extension of period, which results in ignoring many changes during longer interval. Therefore, we design the self-optimizing module to find the reasonable forecast period which is important to the performance of our algorithm.

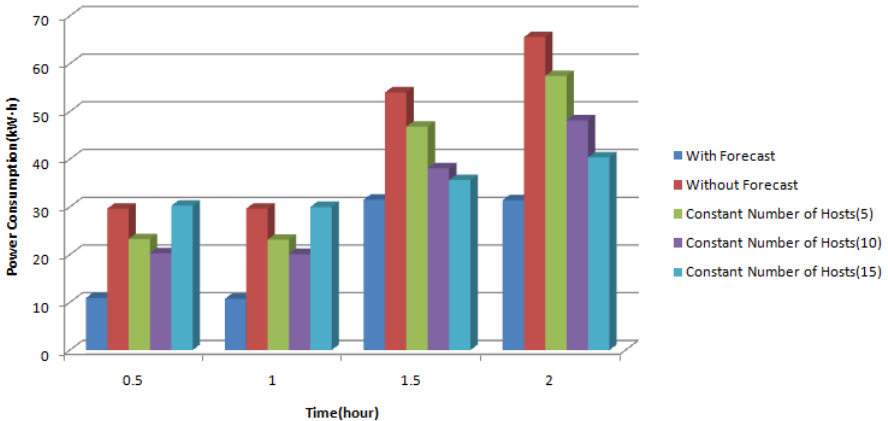


Fig. 5. Sum of power consumption with different intervals

We can conclude from Figure 5 that there is a sharp increase in the curve of power consumption for our algorithm. Self-optimizing module can help us find the most reasonable forecast period. The initial random period is 2 hours. The set of new periods generated by the neighborhood function is 2, 1 and 0.5. Based on our evaluation results of power consumption, the start point and end point of this sharp increase is 1 and 2 respectively. Then the set of periods to evaluate in the third step is 2, 1.5 and 1. Based on Figure 5 and Figure 6, we can see that the most reasonable forecast period with lower power consumption and less iteration is 1 hour.

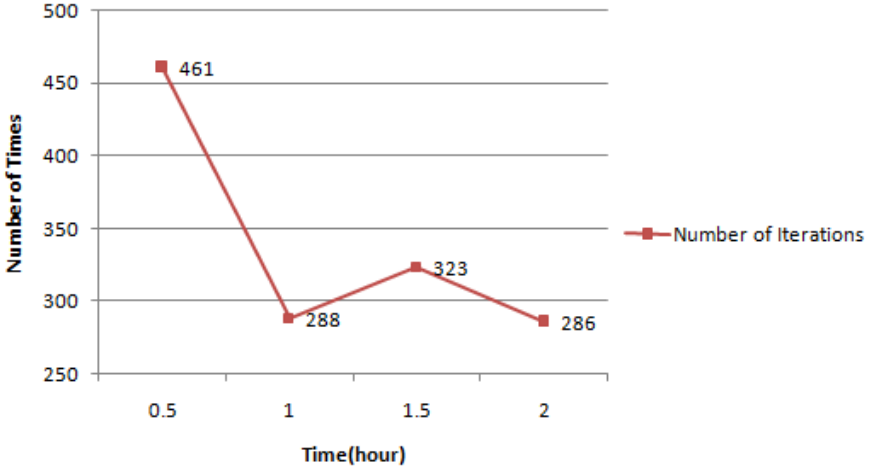


Fig. 6. Number of iterations with different intervals

6 Conclusions and Future Work

Along with the fast development of cloud computing, the number of servers grows rapidly. Apart from the construction cost of the equipment, how to control the operational cost becomes a major concern for every company. One possible way for operational cost control is to reduce power consumption of each host. This paper proposes an approach based on demand forecast to conserve energy. Our approach can predict users' demands of different seasonal patterns effectively and use modified knapsack algorithm to adjust physical locations of VMs. On the basis of forecasted resource demands and reasonable allocation of VMs, the frequency of switching on/off hosts can be reduced, which leads to a decrease in the total amount of energy and has practical value for cloud computing environments.

It is verified that our approach has good accuracy for demand forecast and an advantage in power consumption over other approaches. For the future work, we propose to take into account other kinds of resources in the allocation of VMs, such as disk storage and network bandwidth. The other research interest is to study the influence of VMs' live migration on power consumption, which may be another way to save energy.

Acknowledgements. This work is partially supported by China NSFC (Granted Number 61073021), Science and Technology Commission of Shanghai Municipality (Granted Number 10511501503, 10DZ1200200, 11511500102). The work described in this paper was also partially supported by Morgan Stanley. Morgan Stanley and Shanghai Jiao Tong University Innovation Center of Computing in Financial Services have entered into Collaboration Agreement No.CIP-A20110324-2.

References

1. U.S. Environmental Protection Agency: Report to Congress on Server and Data Center Energy Efficiency. Public Law, 109-431 (2007)
2. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience* 41(1), 23–50 (2011)
3. Walsh, W.E., Tesauro, G., Kephart, J.O., Das, R.: Utility Functions in Autonomic Systems. In: 1st IEEE International Conference on Autonomic Computing, pp. 70–77. IEEE Press, New York (2004)
4. Tesauro, G., Das, R., Walsh, W.E., Kephart, J.O.: Utility-Function-Driven Resource Allocation in Autonomic Systems. In: 2nd IEEE International Conference on Autonomic Computing, pp. 342–343. IEEE Press, New York (2005)
5. Beloglazov, A., Buyya, R.: Energy Efficient Allocation of Virtual Machines in Cloud Data Centers. In: 10th IEEE/ACM International Conference on Cluster, Cloud, and Grid Computing, pp. 577–578. IEEE Press, New York (2010)
6. Bobroff, N., Kochut, A., Beaty, K.: Dynamic Placement of Virtual Machines for Managing SLA Violations. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 119–128. IEEE Press, New York (2007)
7. Khanna, G., Beaty, K., Kar, G., Kochut, A.: Application performance management in virtualized server environments. In: Network Operations and Management Symposium, pp. 373–381. IEEE Press, New York (2006)
8. Steinder, M., Whalley, I., Carrera, D., Gaweda, I., Chess, D.: Server virtualization in autonomic management of heterogeneous workloads. In: 10th IFIP/IEEE International Symposium on Integrated Network Management, pp. 139–148. IEEE Press, New York (2007)
9. Karve, A., Kimbrel, T., Pacifici, G., Spreitzer, M., Steinder, M., Sviridenko, M., Tantawi, A.: Dynamic placement for clustered web applications. In: 15th International Conference on World Wide Web, pp. 593–604. ACM (2006)
10. Time series Forecasting using Holt-Winters Exponential Smoothing, http://www.it.iitb.ac.in/~praj/acads/.../04329008_ExponentialSmoothing.pdf
11. Holt-Winters' Exponential Smoothing with Seasonality, <http://www.cec.uchile.cl/~fbadilla/Helios/referencias/08HoltWintersSeason.pdf>
12. Goodwin, P.: The Holt-Winters Approach to Exponential Smoothing: 50 Years Old and Going Strong. *Foresight*, 30–33 (2010)
13. Mean Absolute Percentage Error, http://en.wikipedia.org/wiki/Mean_absolute_percentage_error
14. Mean Absolute Scaled Error, http://en.wikipedia.org/wiki/Mean_absolute_scaled_error
15. Knapsack Problem, http://en.wikipedia.org/wiki/Knapsack_problem
16. Chu, P.C., Beasley, J.E.: A Genetic Algorithm for the Multidimensional Knapsack Problem. *Journal of Heuristics*, 63–86 (1998)
17. Amazon EC2 Pricing, <http://aws.amazon.com/ec2/pricing/>
18. IBM System x3850 X5 Specifications, <http://www-03.ibm.com/systems/x/hardware/enterprise/x3850x5/specs.html>
19. Amazon EC2 Instance Types, <http://aws.amazon.com/ec2/instance-types/>

Energy Conservative Mobile Cloud Infrastructure

Ashok Chandrasekar¹, Karthik Chandrasekar¹, Harini Ramasatagopan¹,
and Rafica Abdul Rahim²

¹Department of Information Technology, Madras Institute of Technology, Anna University,
Chennai, India

²Department of Electronics and Communication, College of Engineering, Anna University,
Chennai, India

{ashoksekar07, karthidec1, hariniramasatagopan,
rafica.ar}@gmail.com

Abstract. Mobile cloud computing enables the users to improve productivity, share data and collaborate with the other mobile users. Mobile users are able to share resources and applications without a high level capital expenditure on hardware and software resources. Though this new phenomenon is welcomed, people are sceptical about its effectiveness in the real world. Mobile resources if utilised properly will be a boon as it may pave way for their ubiquitous access. Energy efficient infrastructure is very much needed in mobile cloud as energy is the main limitation for mobile users. In this paper an energy conservative mobile cloud infrastructure is proposed. Mobile nodes are classified on the helper factor algorithm, which is based on battery consumption, memory storage, network bandwidth and time delay. Mobile node's ability to contribute its resource to other mobile nodes in its circle is determined. The concept of peer data sharing among mobile devices to enable reusability of data in its own circle is also proposed. Local server cache has been implemented effectively to retrieve the recently used files quickly.

Keywords: Mobile cloud, File sharing, Cloud computing, Hadoop, Peer to Peer, Energy conservative computing.

1 Introduction

Smartphone users are in huge rise and most of the applications employed in smart phones are basically for single users. Many of the computational resources and data are largely underutilized in today's mobile applications. It is possible to use a networked collection of smart phones in a more efficient way. Each smart phone has some amount of storage, computation power, sensing abilities, multimedia data, and energy. Each of these capabilities is available only currently and is utilized by the smart phone user. These resources can be well utilized if these capabilities are somehow offered to the other users. It will be effective if we have a mobile-cloud infrastructure that enables the smart phone applications that are distributed both in terms of data and computation. In this paper, an energy conservative mobile cloud infrastructure is proposed. Multimedia

file sharing has become one of the indispensable tasks of smart phone users. A multimedia file cannot be transferred just like that, the file needs to be compressed, annotated, and then sent over the network, draining the battery in the process. These uploads and downloads are also a burden for wireless network [6] service providers who must handle these large uploads and also for other mobile users who experience the resulting network performance degradation.

2 Related Work

Hadoop based framework exists for mobile cloud computing and it is called hyrax[1]. A more scalable way to support multimedia sharing from mobile devices is to host files on phones and distribute queries and summarization tasks across many nodes. It eliminates the need for each user to upload large files and to retrieve files from remote services. Instead, large transfers could be performed directly within local networks. Search queries could include times and locations of interest and incorporate local sensor readings such as heading and movement to estimate video quality and relevance. Distributed data processing is provided via hadoop's MapReduce implementation, which divides the 'job' submitted by the user into independent 'tasks' and distributes these tasks to slave nodes, taking the physical location of input data into consideration.

In this method [11], a peer can use its idle bandwidth to help in the download of the other and can later get back a return help during its own download. This kind of collaborative download is a novel idea. Hence helper nodes based on collaborative mechanism is used in this system.

The Hummingbird [4] is a custom-developed mobile RF device designed to support awareness and collaboration between a set of users that are in the physical vicinity of each other. The Hummingbird is designed to supply awareness to users in any environment without relying on an underlying infrastructure and to augment forms of traditional, office communication media such as instant messaging and email.

3 Proposed Solution

3.1 Energy Conservative Method

As mentioned above, hundreds of smart phones are found in the same location. At times user mobile's battery percentage may go below the minimum threshold and if he needs to upload or download a relatively larger multimedia file, if this device starts the operation by itself, there is every possibility this task may terminate any time without completing the task. This is because, the device needs to contact remote server every time, which consumes considerable amount of battery power. It will be very useful if some other user in the close proximity helps this needy user (wanter). This is based on the concept that mobile device requires relatively less power to get or send data from a nearby device than the remote one. Suppose user A (wanter) needs to download some files from a remote server but his device has low battery, then it may drain its battery fully in the middle of some critical task. According to the proposed solution, a local server is present which has the complete information of all the smart phone devices present in that local circle. Local server will maintain a log

which has details about the entire download that has been performed by smart phones in this circle. Helper node is any mobile device in that circle which has adequate resources to help a needy user. Every mobile device should have a helper factor greater than helper threshold, only then node will be considered as a helper. Based on the helper factor, helper classifier selects all the mobile devices which satisfy this helper threshold condition. Amount of download to be performed by a single helper node depends on the total file size needed by the wanter node and the number of helper nodes. If the number of helpers in the circle is high, the job done by an individual helper node will be reduced. These helpers if they have agreed to mobile cloud will now perform their job of downloading pieces of data individually and these pieces will be joined together as a single required file with the help of modified version of hyrax, an adapted form of hadoop for mobile cloud. Pieces of data can be downloaded individually by different nodes from a remote server and then these files can be compiled to a single file [5]. This kind of file sharing concepts was pioneered by torrent and Gnutella.

3.2 Helper Classifier

Helper classifier is a component which is present in the local server, which calculates the helper factor for all the mobile devices in the circle, based on which the helpers are selected and classified.

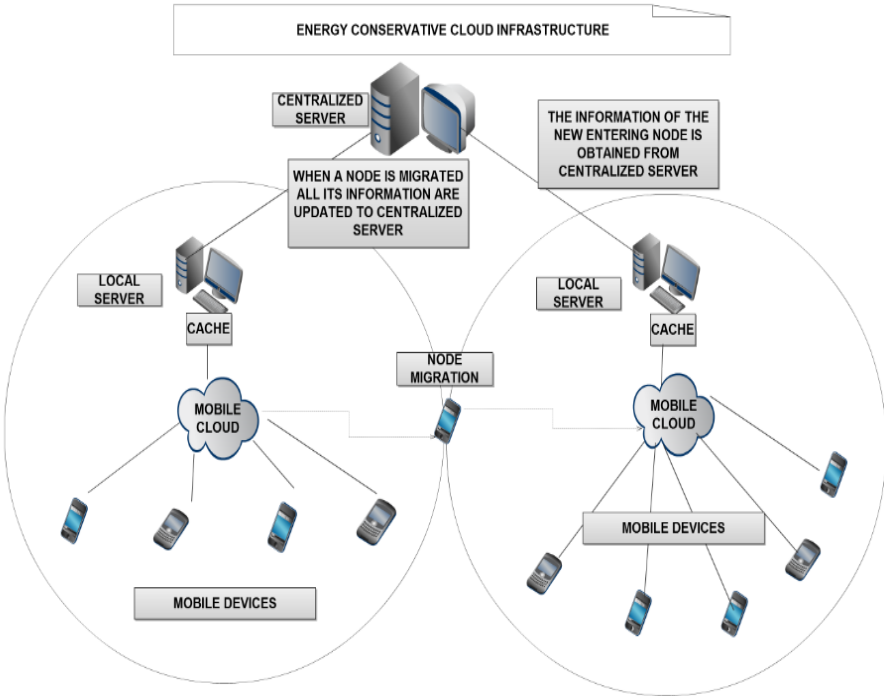


Fig. 1. Energy conservative Mobile cloud infrastructure

Helper Factor. Helper factor is based on many parameters such as battery life, network bandwidth, CPU utilisation and memory usage of individual mobile devices.

$$H_f = 4B - 2M - C + D \quad (1)$$

The above equation is used to calculate the Helper factor H_f of individual mobile nodes. This helper factor provides a steady and really efficient formula to select the appropriate mobile devices as helpers. B indicates the battery percentage remaining in the mobile devices; M indicates the percentage of memory used in the mobile device; C indicates percentage of CPU utilization of the mobile device; D indicates available network bandwidth. The values 4 and 2 in (1) represent the priorities assigned to various factors. These priorities can be changed. After several experiments, these values are found out to be optimal.

3.3 Helper Classification Method

The H_f is calculated for all mobile devices in a circle pertaining to single local server and the threshold values are fixed for various categories after a thorough analysis. For a mobile device to be in category 1, H_f value should be greater than 2 and B value should be greater than 70. For a mobile device to be in category 2, the H_f should fall within the range 1-2 and $B > 65$ and it belongs to the category 3 if its H_f value is less than 1. The H_f value is computed as mentioned and based on the above ranges it is classified. This classification is necessary to accurately determine the amount of resource it could spend on its peers. A device in the category 1 has the highest priority of being selected by the helper classifier as helper node. Category 2 devices have slightly lesser priority than category 1. Devices which fall under category 3 are not eligible for being selected as helper. Helper classifier will classify all the mobile devices in anyone of the category and this classification will happen dynamically after a particular time interval. Now that classification is done, the amount of work to be done by individual nodes should be fixed. The multimedia file to be downloaded by wanter node is identified and number of nodes in category 1 is counted. These threshold values are arbitrarily fixed for experimental purposes. The battery percentage for category 1 and category 2 are chosen as 70 and 65 after experimenting with several values. The most optimal and fair one for the helpers is chosen at the end. Three scenarios are possible. They are

- Category 1 nodes are sufficient.
- Category 1 nodes are insufficient.
- Category 2 nodes are insufficient.

3.4 Peer to Peer Transfer

As mentioned above, all the mobile devices in a circle is connected to a local server. This server has a component named history keeper which has the history of all the files uploaded or downloaded by mobile devices in that circle. So any mobile device

in the circle before downloading any file, checks the history keeper, if the same file has been downloaded by any other node in this circle. If it has been downloaded by any other node, instead of reaching remote server, it can download the file directly from its peer, provided the node that downloaded the file is currently present in the same circle. There is also a possibility that the node which downloaded the same file could have migrated to the other circle, as migration is very common among mobile devices. If the node is found to remain in the same circle and only if it falls under category 1 or category 2 then peer transfer will happen. If the node has migrated, peer sharing cannot be done.

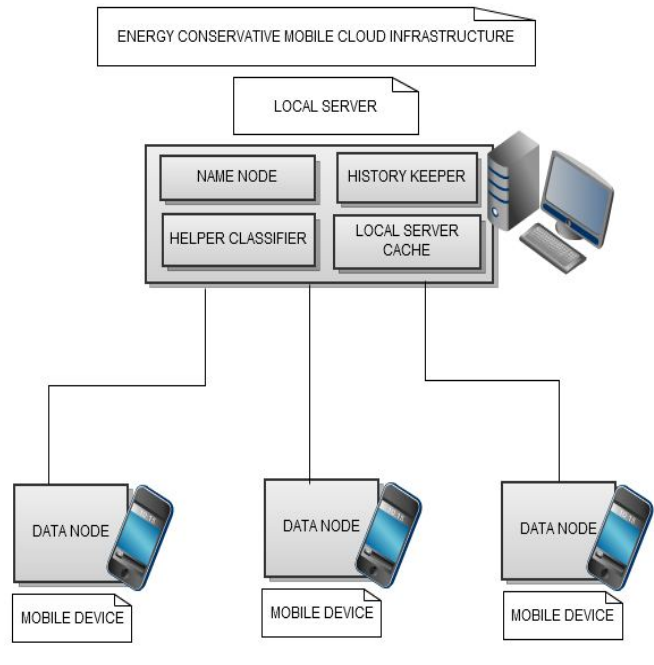


Fig. 2. Energy conservative Mobile cloud components

3.5 Limit Factor

The size of the file to be transferred is identified. Helper classifier dynamically classifies the helper nodes and number of devices falling in different categories is identified. Maximum Limit (ML) is fixed so that single mobile device resources are not used too much. After various results analysis Maximum Limit (ML) is assumed to be 20 Mb for every mobile device. This means a device will be considered for this job only when its limit factor (Lm) is less than Maximum Limit. Limit factor greater than Maximum Limit (ML) indicates that node has serviced enough for its peer group and will no more be considered by helper classifier for peer service.

3.6 Caching

The local server also has a cache attached to it, which contains the most recently accessed files in it. So if any mobile device's request matches any of the files in the cache, then files can be transferred directly from the cache without computing any helper factor for any mobile nodes.

3.7 Node Migration

Since we are dealing with mobile devices, migration of mobile nodes is usual happening and its adverse effects have to be handled properly. There are many possibilities for handling migration of nodes.

Less Than 30 Percent Transfer. When a job is assigned to a node and it started transfer and if less than 30 percent of the job is completed by that mobile device and now mobile device needs to migrate, even then local server can just transfer this job to some other mobile device to do it from start as this little amount of resource wastage might not create big impact.

More Than 30 Percent of Transfer. When more than 30 percent of job is completed by the mobile device and then if the device needs to migrate, then just ignoring the job performed by the node will decrease the performance, as the resources will be considerably wasted. So now when the node moves out of the circle it can submit the job completed so far to the local server. Local server now finds a new eligible helper node and hands over this partially completed job with all the necessary information. The new node is expected to complete the remaining job. Submitting the job to the local server and re-assigning the partially completed job to some other helper node involves overhead and hence it is done only when substantial amount of job is completed. After much analysis, the value is assumed as 30 %, which is found to be optimal in most of the cases. In few cases server cannot find capable helper node to transfer the partially completed job, and in that case, other possibilities should be taken into consideration. Node migration is an indispensable topic as we are dealing with mobile devices.

3.8 Centralized Server

A centralized server is maintained which has the control over all the local servers of various circles. This centralised server is necessary because, a device might have serviced till its maximum limit (ML) for its peer in one circle and it could have been migrated to the other circle. Now it has migrated to a new circle and there is possibility that this node might fall under category 1 once again and if there is no connection between local servers, the migrated user will again need to help its peer in the new circle. This may not be to the liking of a mobile node because it has serviced till its maximum limit and even then its resource is being used over and over. Therefore centralized server will maintain a log consisting of all the histories of serviced nodes obtained from all the local servers. So this log is checked by each local server before assigning job to any new helper node. Thus we can avoid a single node's resource getting exhausted.

4 Implementation and Evaluation

The proposed work seems to work very efficiently, but there are many real time complexities which makes it very hard for real time implementation.

We performed simple data transmission among mobile devices and between local server and mobile devices ignoring the concept of migration of mobile nodes and the presence of centralized server. Only when the complete work modification of hyrax is done, we will be able to implement the complete system. Because of the difficulties involved in bringing out the real world working model, we performed this small part of implementation whose results are analysed thoroughly and it is found that the results obtained bolsters our proposed work.

The network used consists of few android running smart phones connected to a local server. Since Android does not support peer-to-peer networking [2], the phones communicate with each other through WLAN using wireless router. The Name Node and Job Tracker processes run on a desktop machine that is connected behind this router via Ethernet. The Data Node and Task Tracker processes run on mobile nodes. All the mobile nodes have internet connection in them.

From the above setup, we were able to transfer data between mobile devices and between local server and mobile device. Remote data transfer is performed by connecting mobile devices to internet and by performing random downloads and uploads of files of various sizes. From the obtained values we were able to find the average data transfer time for set of files of various sizes. Using battery finder application we were able to find out the exact percentage of battery consumed for each and every transfer.

Fig 3 shows the battery consumption for data transfer by mobile devices (plotted in Y axis) for a set of files of various sizes (plotted in X axis). Any data transfer operation involves some amount of battery consumption. Since battery consumption

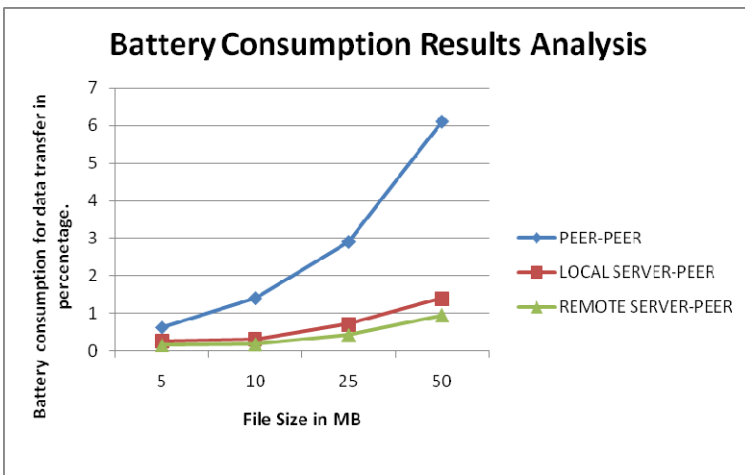


Fig. 3. Battery consumption results analysis

is one of the first and foremost factors to be considered in mobile cloud, amount of battery consumption for various file sizes are found out. It is very much clear that Remote server – Peer transfer is highly battery consuming process. Peer to Peer transfer consumes next higher amount of battery resource though it is very small when compared to remote server- peer transfer. Most optimal transfer is between local server and mobile device. It indicates if we have bigger cache in local server, more possibility of hitting data in local server and hence very less consumption of battery is required to get the files from cache. Similarly if the required file is found among peers, battery is consumed to at least a certain extent when compared to the remote server transfer.

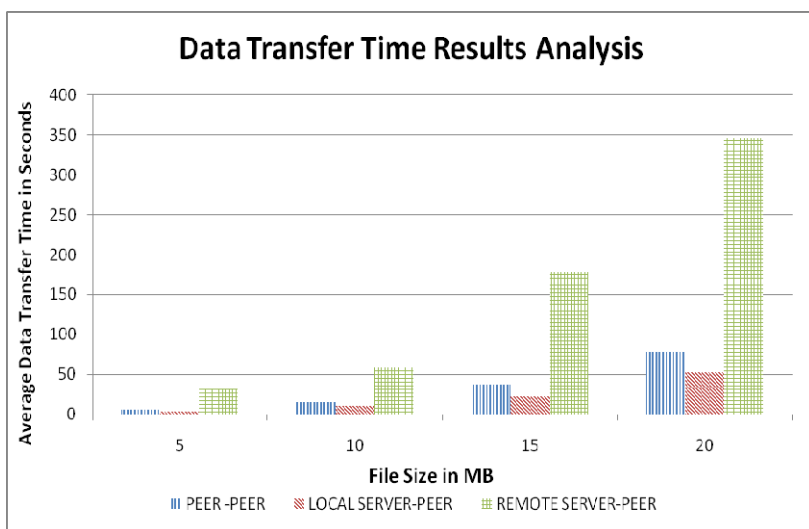


Fig. 4. Data transfer time results analysis

Fig 4 shows average data transfer time in seconds (plotted in Y axis) for set of files of various sizes (plotted in X axis). Data transfer time is very important factor to be considered as it directly affects the battery consumption of mobile devices. Time delay depends on various parameters such as bandwidth, network congestion, distance of the sender and the receiver.

From our experiment we found data transfer time was so high in the case of transfer between remote server and mobile device. Mobile device to mobile device transfer comes out as next higher time consuming data transfer although it is very less when compared to remote server transmission. So if the data transfer happens within particular circle, transfer rate will be very faster when compared to remote server involved transmissions.

4.1 Assumptions

- Mobile device users have firm control over their privacy. They can be sure that their protected data will not be accessed by any other external agents.
- Interoperability exists between various mobile devices and various local servers irrespective of their hardware specifications, provided, they have common software.
- The proposed technique works in same manner even after the introduction of very large number of mobile nodes.
- All mobile nodes are running android operating systems.
- Wireless network performance will be same as wired in the case of connection between local server and mobile nodes.

5 Conclusion

This paper makes the following unique contributions.

- Energy conservative method is proposed to complete a large task for a mobile device with low battery with the help of other mobile devices.
- A helper classification method is designed to find out the available mobile nodes which could help the needy peer.
- A dynamic mechanism to handle migration of mobile nodes is devised. This makes use of a centralized server and helps in avoiding monopolization of a single mobile node's resource.
- Simple evaluation is done through by which the efficiency of the proposed work is verified.

This method is mainly for usage in places where large crowd gathers. Since dedicated local server is used as master, scalability will not be a big problem as local server is capable of handling the requests of those relatively small numbers of nodes present in a single location. Similarly centralized server just performs the job of synchronising the logs of all local servers and hence it may not suffer from scalability issues. Full implementation of this work involves many real time complexities. The evaluation presented here is based on partial implementation of the system.

6 Future Work

Security is a must for a peer sharing system like this, since there is a higher possibility of attacks. Effective handling of monitoring data should be done as this monitoring information sent by mobile node to local server should not become a separate overhead. Hence our future works will be related to bringing all our above mentioned works closer to the real world implementation and making modifications for the same.

References

1. Marinelli, E.E.: Hyrax: Cloud Computing on Mobile Devices using Map Reduce. In: CMU-CS-09-16(September 2009)
2. Litke, A., Skoutas, D., Varvarigou, T.: Mobile Grid Computing: Changes and Challenges of Resource Management in a Mobile Grid Environment (2004)
3. Palmer, N., Kemp, R., Kielmann, T., Bal, H.: Ibis for Mobility: Solving Challenges of Mobile Computing Using Grid Techniques. In: HotMobile 2009, Santa Cruz, CA, USA, February 23-24 (2009)
4. Holmquist, L.E., Falk, J., Wigström, J.: Supporting Group Collaboration with Inter-Personal Awareness Devices. *Journal of Personal Technologies* 3(1-2) (1999)
5. Michalakakis, N., Kalofonos, D.N.: Designing an NFS-based Mobile Distributed File System for Ephemeral Sharing in Proximity Network (2004)
6. Borthakur, D.: The Hadoop Distributed File System: Architecture and Design. In: The Apache Software Foundation (2007)
7. Zheng, P., Ni, L.: Smart Phone and Next Generation Mobile Computing. Morgan Kaufmann (2006)
8. Alcock, B., Bester, J., Bresnahan, J., Chervenak, A.L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnal, D., Tuecke, S.: Data Management and Transfer in High Performance Computational Grid Environments. *Parallel Computing Journal* 28(5) (May 2002)
9. Graf, M.: Grid collaboration in the mobile environment. In: IBM. Speech at the Collaboration @ work Workshop (June 25, 2003)
10. Satyanarayanan, M.: Fundamental Challenges in Mobile Computing. In: Proceedings of the ACM Symposium on Principles of Distributed Computing (1996)
11. Kozuch, M., Satyanarayanan, M., Bressoud, T., Helfrich, C., Sinnamohideen, S.: Seamless Mobile Computing on Fixed Infrastructure. *IEEE Computer* 37(7) (July 2004)
12. Garbacki, P., Iosup, A., Epema, D., van Steen, M.: 2Fast: Collaborative downloads in P2P networks in P2P (2006)
13. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The Case for VM-based Cloudlets in Mobile Computing. In: Pervasive Computing. IEEE (2009)
14. Lo, C.-H., Peng, W.-C., Chen, C.-W., Lin, T.-Y., Lin, C.-S.: Carweb: A traffic data collection platform. In: Mobile Data Management (April 2008)
15. Lindemann, C., Waldhorst, O.P.: A distributed search service for peer-to-peer file sharing in mobile applications (2002)
16. Song, S., Yu, H.C.: The Scheduling method considering the Using Pattern of the Mobile device in Mobile Grid. *Journal of Korea Association of Computer Education* (2008)
17. Megdalena, B., Paul, C.: Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network. In: MOBISYS (2003)
18. Birje, M.N., Manvi, S.S.: Monitoring and Status Representation of Devices in Wireless Grids. In: Bellavista, P., Chang, R.-S., Chao, H.-C., Lin, S.-F., Sloat, P.M.A. (eds.) GPC 2010. LNCS, vol. 6104, pp. 341–352. Springer, Heidelberg (2010)
19. R-GMA: Relational Grid Monitoring Architecture (December 2003), <http://rgma.org/> (visit May 20, 2010)
20. Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and Grid computing 3600-degree compared. In: Grid Computing Environments Workshop (2008)
21. Comito, C., Falcone, D., Talia, D., Trunfio, P.: Energy Efficient Task Allocation over Mobile Networks. In: Proc. of the International Conference on Cloud and Green Computing (CGC 2011), Sydney, Australia, pp. 380–387. IEEE Computer Society Press (December 2011)

Power-Constrained Actuator Coordination for Agricultural Sensor Networks^{*}

Junghoon Lee¹, Gyung-Leen Park^{1,**}, Ho-Young Kwak², and Jikwang Han³

¹ Dept. of Computer Science and Statistics

² Dept. of Computer Engineering

Jeju National University, Republic of Korea

³ Jinwoo Soft Innovation

{jhlee, glpark, kwak}@jejunu.ac.kr, hmurdoc@jinwoosi.co.kr

Abstract. Upon the ubiquitous sensor network capable of deciding control actions by a sophisticated inference engine, this paper designs an actuator controller which coordinates the actuator operations over the multiple farms. Basically, not beginning tasks as soon as they get triggered, local schedulers determine the operation plan according to genetic algorithms, for the sake of reducing peak power consumption for the given scheduling window. For global scheduling, each local scheduler retrieves the current load information maintained in the coordinator, runs its own schedule, and finally sends to the coordinator. The fitness function gives penalty to the allocation which assigns much power to the heavily loaded time slots. The procedure reduces the peak load by up to 22.8 % for the given task set. Moreover, all schedules are not necessarily run with tight concurrency control. Our simulation shows that 40 % of schedulers can run in parallel just with negligible performance loss.

Keywords: wireless sensor network, actuator schedule, farm group, peak power reduction, demand response.

1 Introduction

Modern automation systems in agriculture are desirably empowered by sensor and actuator technologies, which provide fundamental building blocks for ubiquitous sensor networks, or USN in short [1]. The sensors capture the event over the farm area and report to the sensor stream processor, while the actuators manipulate the target device according to the control logic. For the correct operation of USN, the better accuracy of sensors is essential. Next comes a reliable communication mechanism as well as a data analysis engine with sufficient computing capacity. Necessarily, USN can handle a great volume of sensor data,

^{*} This research was supported by the MKE (The Ministry of Knowledge Economy), through the project of Region technical renovation, Republic of Korea.

^{**} Corresponding author.

detect events, and compute the real-time control action [2]. This infrastructure can host a lot of control strategies defined by a given system goal, exploiting from simple on-off logic to sophisticated artificial intelligence mechanisms [3]. Such distributed intelligence over the sensor network is indispensable for environmental monitoring, farming, irrigation, and the like.

The sensor network eventually decides the control actions, making a set of actuators run [4]. Some control actions are regularly executed, for example, in hourly or daily basis, while others on-demand basis. The actuator devices include heaters and air conditioners for greenhouses, sprinklers for crop area, ventilators for cattle sheds, and so on. Moreover, an irrigation system deals with heavy water gates, requiring much electric power [5]. Accordingly, the operation of an agricultural device inevitably imposes much power consumption. If many operations, possibly from multiple farms, concentrate on a specific time interval, the power load can put significant pressure on the power system. As a result, the integration of power management to the control action decision is of great necessity for agricultural USN. It must coordinate the operation of multiple actuators by creating an efficient schedule which can reduce the peak power load for a farm group, not just a single farm.

In this regard, this paper first presents an actuator node capable of reliably fulfilling the control action decided in the computing domain. Then, a power management scheme is designed for a farm group to integrate smart electricity consumption to control action scheduling. We are mainly targeting at a small scale farm group consisting of less than 20 farms. Local schedulers can have its own scheduling mechanism, while the genetic algorithm is one of the most promising schemes, as it cannot only directly control the scheduling time, but also obtain a schedule of acceptable quality, even if not optimal. The global coordinator synchronizes each local schedule according to the given system goal. In power consumption, concentrated power load can jeopardize the safe operation of the power distribution network and possibly force to purchase expensive energy. Hence, our scheduling scheme aims at reducing or reshaping the peak load. The coordinator and local schedulers interacts via two-way communication primitives over the appropriate network.

This paper is organized as follows: After issuing the problem in Section 1, Section 2 introduces background of this paper. Section 3 designs an actuator coordination scheme for a farm group. Section 4 discusses the performance measurement result and Section 5 finally concludes this paper with a brief introduction of future work.

2 Background and Related Work

Through the research and technical project on *Development of convergence techniques for agriculture, fisheries, and livestock industries based on ubiquitous sensor networks*, our project team has developed an intelligent USN framework [6]. It provides an efficient and seamless runtime environment for a variety of monitor-and-control applications on agricultural sensor networks. It is important to systematically cope with the sensor data stream as well as provide interoperability

between the low-level sensor data and application clients. Here, integrative middleware promotes the interaction between the application layer and the underlying sensor network, making it possible to analyze a great volume of sensor data. Then, the real-time data distributor selectively attaches an agent-based event handler so as to forward a specific event to the application handler, which is registered in the directory service. After all, each stream is activated and relinquished by the stream manager.

Basically, according to the required reliability level, the USN design chooses appropriate sensors, required actuators, and communication protocols. In farms, it is necessary to continuously monitor the environmental change in temperature, humidity, lightness, CO₂ and NH₂ levels in addition to the biosensors which capture the livestock disease. Our system develops a sensor board capable of coping with the heterogeneity of different sensors as well as providing stable power. Sensors and actuators are installed in fixed locations [7]. A control unit is embedded in actuator boards to control the overall board operation autonomously, even in the place where neither network connection nor remote control is available. It improves reliability and uniformly carries out diverse control actions. After all, the actuator board consists of a connector part to the interface board, a control signal level converter, a signal isolator to separate lower power DC signal from high power AC signal, and an AC-DC power converter to provide power to the whole remote control module.

In the mean time, smart power consumption draws much attention in many areas including agricultural farms, as most equipments consumes quite much energy for their operations. The smart grid brings intelligence in power consumption management based on sophisticated information technologies and ubiquitous communication infrastructures [8]. The demand response technique can support device operation planning to achieve various system goals such as cost saving, peak load reduction, renewable energy integration, and the like. However, this scheduling requires tremendous computing time in each scheduling unit such as homes, buildings, and farms. Accordingly, genetic algorithms are practical solutions, as they can yield an acceptable quality schedule within a reasonable time bound [9]. Our previous work has proposed a genetic scheduler, which begins the evolution procedure with better initial population by an extremely lightweight heuristic [10]. In addition to the local peak reduction, group-level coordination can further reduce the peak load in a farm group.

3 Power Management for a Farm Group

3.1 System and Task Models

A group of farms, possibly 10 to 20, can jointly produce crops, buy seeds or fertilizers, and undertake marketing. It will be also possible for them to purchase the power from the power vendor as a single unit as shown in Figure 1. A power provider can offer a discounted rate plan for massive purchase. The contract generally limits the maximum power consumption at each time instant. If the total power consumption exceeds the contracted amount during a specific time

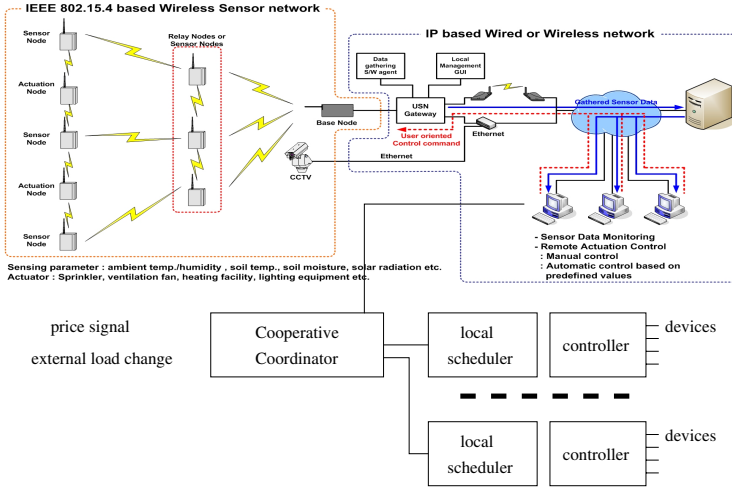


Fig. 1. Power management architecture

unit, expensive dispatchable energy must be additionally bought. Basically, a single farm is a scheduling unit for power scheduling. As each smart farm has its own local scheduling policy, it is necessary to globally coordinate the respective schedules. It is impossible to schedule all device operations from all local units in a single coordinator, as its computation time gets too much beyond the practical bound. Hence, the peak reduction in each local grid is the first step for global peak reduction, and then, it is necessary to reduce the effect of peak resonance which happens when the respective peaks concentrate on same time slots.

A series of control actions are decided after detecting specific events by the stream inference engine, creating an action set along with the regular device operations [7]. For their scheduling in a single scheduling unit, each control action is specified by a processing task model. Here, a task, T_i , is represented by $\langle A_i, U_i, D_i \rangle$, which denotes activation time, operation length, and deadline, respectively. Scheduling assumes that the power consumption behavior is known in priori by means of a consumption profile, which is aligned to a fixed size time slot. For more details on this task model, refer to [11]. A local schedule is generated by any computing device such as PC within a farm or a high-performance remote server residing in the Internet domain. After all, a power consumption schedule is generated in the form of $M \times N$ time table, where M is the number of time slots and N is the number of tasks. For a schedule, the power controller connects or disconnects the power line to each electric device [10]. The operation status can be changed only at each slot boundary.

3.2 Group-Level Coordination

For the local scheduling strategy, our design exploits the genetic algorithm, defining a fixed-length string of an integer-valued vector for chromosome

representation. Each value element indicates an allocation map for a task. For example, suppose that a feasible schedule for 3 tasks from T_0 to T_2 is (21, 32, 10) as shown in Figure 2. The vector element, or the allocation map, for T_0 is 21 and its binary equivalent is 10101. Additionally, let A_0 , U_0 , and D_0 be 1, 3, and 5, respectively. Then, the profile entry for T_1 is copied to the slots from 1 to 5, skipping zero entries marked in the map. It will make the first row as (3, 0, 2, 0, 3) from Slot 0. The dotted box is the valid scheduling range for T_0 . It starts from A_0 and ends at D_0 . For each complete allocation, the fitness function calculates per-slot power load and the largest will be the peak load of the schedule. It must be mentioned that the purpose of power scheduling is to reduce the power load of such a peaking slot. In this example, peak load takes place at Slot 1 and its value is 7.

With this encoding, the iteration consists of selection and reproduction. As for the initial population, we select feasible schedules randomly. The selection procedure picks parents by their fitness values. The Roulette Wheel selection gives more chances to chromosomes having better fitness values for mating. The fitness function simply calculates per-slot load and selects the maximum value. It can be easily modified to reflect other criteria and parameters. Reproduction, or crossover, randomly selects a pair of two crossover points and swaps the substrings from each parent. If a new chromosome is the same as the one included in the population, it will be replaced by another different one. Here, information on price signal change and external load status can be parameterized in the fitness function. Each local schedule reduces the local peak and the flattened load also reduces the global peak.

For a vector (21, 32, 10) = (10101, 11000, 1010)

	<A, U, D>	Profile	0	1	2	3	4	5	
T_0	<1, 3, 5>	3 2 3	0	3	0	2	0	3	
T_1	<0, 2, 4>	2 4	2	4	0	0	0	0	
T_2	<0, 2, 3>	2 3	2	0	3	0	0	0	
			4	7	3	2	0	3	Per-slot load

Fig. 2. Encoded allocation table

The global coordination begins with the assumption that the time slots in each scheduling unit are synchronized over the multiple farms. Nowadays, the GPS (Global Positioning System) technology allows such time synchronization quite easily [12]. A single global coordinator is responsible for managing a farm group. From the viewpoint of the global coordinator, it is necessary to avoid the peak resonance, where peaking slots of multiple local schedules meet at the same time slot or slots. For global reduction, the coordinator maintains the operation schedules for each grid unit as a long-term allocation time table. A local scheduler retrieves the current load for time slots belonging to a specific scheduling

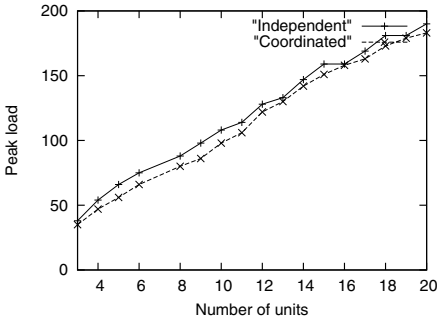
window before starting its local scheduling procedure. Then, the fitness function is modified, adding the current global load for each slot. As a result, we can give penalty to the allocation which assigns task operations to the heavily loaded slots during genetic iterations. After the completion of a unit schedule, the local scheduler reports its result back to the coordinator.

In this scheduling model, each scheduler retrieves and updates the global load information. Hence, a set of retrieval requests can arrive at the coordinator simultaneously, and thus access conflicts for the shared data can take place between them. We cannot expect any global coordination without a concurrency control mechanism. However, tight concurrency control can prolong the scheduling time, as each local scheduler must run one by one, waiting for its all predecessors to complete. This situation may be serious for the real-time control reaction. Actually, even if some schedulers decide their local schedules independently, making peak resonance among them, the last one or two schedulers can sufficiently avoid peaking slots up to the previous unit allocation. We assert that every local scheduler doesn't have to run sequentially and thus partial synchronization is promising for both scheduling overhead and accuracy.

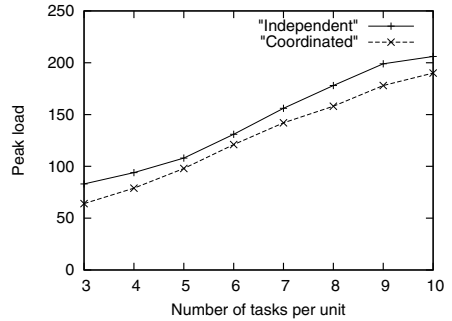
4 Performance Measurement

For the performance evaluation of the proposed actuation coordinator, we have implemented the power consumption scheduler using Microsoft Visual C++. For a task, the operation length and the slack exponentially distribute with the average of 3.0 and 2.0, respectively. For each time slot, the power amount ranges from 1 to 10. Our experiment assumes that the scheduling window size is 20. The slot length can be assumed to be 5 minutes. The power scale is not explicitly specified, as it depends on the electric device types included in the target scheduling unit. For the parameters regarding to genetic algorithms, the number of iterations is 1,000, the population size is 80, and the initial population is chosen randomly. We mainly measure the effect of global coordination by comparing the peak load for independent and coordinated scheduling. In independent scheduling, each scheduling unit just runs the genetic scheduler without considering the current group-wide power load. For coordinated scheduling, the shared load information is tightly synchronized.

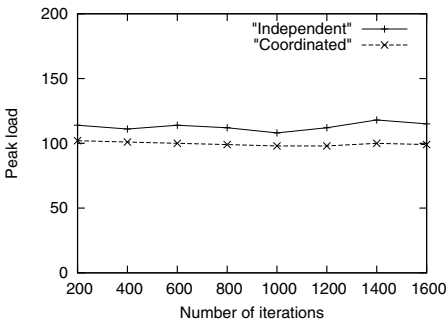
The first experiment measures the effect of the number of scheduling units to peak load. In this experiment, the number of tasks in each unit is set to 5. Figure 3(a) plots the peak load for independent and coordinated scheduling. According to the increase in the number of units, the number of tasks in the farm group also increases. Hence, the peak load gets larger for both curves. When there are just 3 units, the effect of global coordination is not so significant, showing just 7.1 % peak load reduction. The performance gap increases with more units, but remains stable when there are 13 or more units. Beyond this point, which has already sufficient number of total tasks, independent scheduling implicitly distributes the power load even without any manual regulation due to the randomness in slot assignment. The experiment also discovers that local power consumption



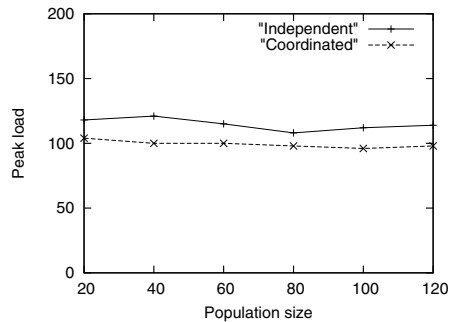
(a) Effect of the number of units



(b) Effect of the number of tasks



(c) Effect of iterations



(d) Effect of population size

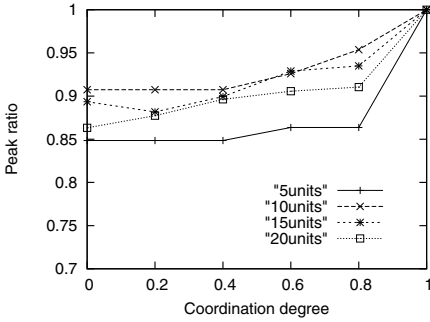
Fig. 3. Power load reduction by complete synchronization

schedule can reduce the global peak. Anyway, the maximum peak load reduction reaches 15.1 % in this experiment.

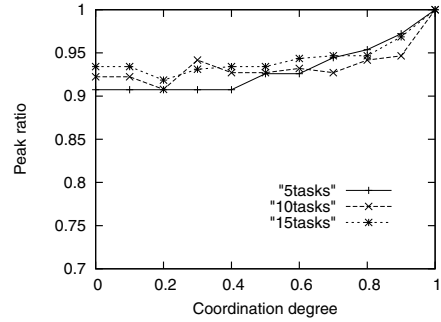
Next, Figure 3(b) plots the peak load according to the number of tasks in each scheduling unit. The experiment sets the number of units to 10, while changing the number of tasks from 3 to 10. Coordinated scheduling achieves the peak load reduction of 22.8 % for the 3 task case. When there are fewer tasks for a relatively large number of units, the peak resonance is highly likely to happen. At this point, we can better benefit from coordinated scheduling. For the remaining interval, the performance gap lies in the range of 7.7 % to 15.9 %. The peak reduction in each local scheduling unit also reduces the global peak load for the sufficient number of tasks as in the previous experiment. The number of tasks in a unit has more effect to the peak load than the number of units.

Figure 3(c) and Figure 3(d) measure the effect of genetic algorithm-specific metrics. For the number of iterations from 200 to 1,600 with the population size fixed to 80, coordinated scheduling stably shows smaller peak load by up to 15 %. Actually, the improvement by genetic iterations remains below 1.0 % for coordinated scheduling and 8.4 % for independent scheduling, after 800 iterations. The performance gap between two scheduling schemes largely increases

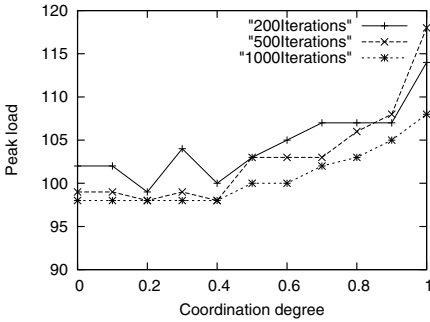
according to the increase in the number of iterations, but lies in the range from 9.0 % to 15.2 %. In addition, a larger population size generally leads to further reduction as shown in Figure 3(d). The population size significantly affects the execution time of genetic iterations which include a number of sorting steps. Even though both scheduling schemes reduce the peak load with larger population size by up to 7.6 % and 10.7 %, the peak load gap between them is not so significantly influenced. For the population size of 40, coordinated scheduling outperforms independent scheduling by 17.3 %.



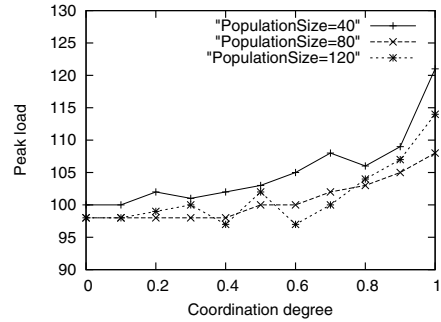
(a) Effect of the number of units



(b) Effect of the number of tasks



(a) Effect of iterations



(b) Effect of population size

Fig. 4. Power load reduction by partial synchronization

The second experiment set focuses on the partial synchronization where some local allocations decide their schedules using incorrect current load distribution. The experiment pessimistically assumes that all units simultaneously want to schedule their device operations and send retrieval requests for the current load distribution. We define the coordination degree to represent how many units go without synchronization. If the degree is 0, all local schedules are decided one by one through the regular tight concurrency control. This case corresponds to coordinated scheduling. On the contrary, if the degree is 1, all schedulers allocate task operations independently. This case coincides with independent scheduling.

For 10 tasks, the coordination degree of 0.8 makes 8 units schedule independently while only 2 units begin its scheduling procedure after 8 units are done.

To begin with, in Figure 4(a) and Figure 4(b), the y-axis denotes the peak ratio, which is the normalized peak to the maximum peak load. The maximum peak takes place when the coordination degree is 1.0. Figure 4(a) plots the effect of the coordination degree when the number of units is 5, 10, 15, and 20, respectively, with the number of tasks set to 5. In most cases, the peak ratio rapidly approaches to 1.0 after the degree is 0.8. Before 0.8, the maximum performance loss, or the peak load increase from coordinated scheduling, remains just below 1.5 % for the 5 task case. As contrast, for the case of 10 tasks, this is true just until 0.4. Next, Figure 4(b) plots the peak ratio for the case of 5, 10, and 15 tasks with the number of units fixed to 10. This figure shows the similar pattern to Figure 4(a), but the performance loss before 0.8 is 5 %. For other cases, the improvement is not so significant, but up to 40 % schedules can proceed in parallel just with less than 0.1 % performance loss.

In Figure 4(c), we have set the number of iterations to 200, 500, and 1,000. Here, as this experiment has the same number of units and the same number of tasks, the peak load is shown in the y-axis instead of peak ratio. There is a trade-off between execution time and accuracy. With more iterations, we can improve the accuracy, but the scheduling time will increase. It will be worse for coordinated scheduling. The case of smaller number of iterations is more affected by the coordination degree. However, below 0.4, the peak load increase is less than 1.7 % for all cases. The result is the same for the population size. As shown in Figure 4(d), small population size is more affected by the coordination degree. After all, even though the effect of coordination degree to peak load is dependent on the characteristics of each task set, 0.4 will be the safe bound. If further computation speed is required, we can extend the parallel execution with the coordination degree of 0.8, tolerating small performance loss.

5 Conclusions

Benefiting from the robust actuator management mechanism, we have designed a global coordination scheme to integrate power consumption scheduling in control action planning. Not just deciding how to respond to external events, our scheme generates an efficient actuator control schedule capable of reducing the peak load in a farm group. Each local scheduler calculates the action time table by a genetic algorithm implementation after retrieving current global power load and updating its fitness function to avoid peak resonance. The global coordinator interacts with local schedulers in smart farms belonging to a farm group via the Internet, managing the global power load information. The performance measurement by a prototype implementation shows that the proposed scheme reduces the peak load by up to 22.8 % for the given task set. Moreover, all schedules are not necessarily run with tight concurrency control. Judging from the experiment, 40 % of schedulers can run in parallel just with negligible performance loss.

As future work, we are planning to design a hierarchical power management scheme to extend the number of farm groups. It will include how to organize the farm group taking into account the characteristics of each scheduling unit.

References

1. Revenaz, A., Ruggeri, M., Martelli, M.: Wireless Communication Protocol for Agricultural Machines Synchronization and Fleet Management. In: International Symposium on Industrial Electronics, pp. 3498–3504 (2010)
2. Sigrimis, N., Antsaklis, P., Groumpos, P.: Advances in Control of Agriculture and the Environment. *IEEE Control Systems* 21, 8–12 (2001)
3. Esposito, F., Basile, T.M.A., Di Mauro, N., Ferilli, S.: A Relational Approach to Sensor Network Data Mining. In: Soro, A., Vargiu, E., Armano, G., Paddeu, G. (eds.) *Information Retrieval and Mining in Distributed Environments*. *SCI*, vol. 324, pp. 163–181. Springer, Heidelberg (2010)
4. Culler, D., Estrin, D., Srivastava, M.: Overview of Sensor Networks. *IEEE Computer* 37, 41–49 (2004)
5. Ruiz-Garcia, L., Lunadei, L., Barreiro, P., Robla, J.: A Review of Wireless Sensor Technologies and Applications in Agriculture and Food Industry: State of the Art and Current Trends. *Sensors* 9, 4728–4750 (2009)
6. Lee, J., Park, G., Kim, H., Kim, C., Kwak, H., Lee, S., Lee, S.: Intelligent Management Message Routing in Ubiquitous Sensor Networks. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) *ICCCI 2011, Part I*. *LNCS*, vol. 6922, pp. 537–545. Springer, Heidelberg (2011)
7. Lee, J., Kim, H.-J., Park, G.-L., Kwak, H.-Y., Kim, C.M.: Intelligent Ubiquitous Sensor Network for Agricultural and Livestock Farms. In: Xiang, Y., Cuzzocrea, A., Hobbs, M., Zhou, W. (eds.) *ICA3PP 2011, Part II*. *LNCS*, vol. 7017, pp. 196–204. Springer, Heidelberg (2011)
8. Ipakchi, A., Albuyeh, F.: Grid of the Future. *IEEE Power & Energy Magazine*, 52–62 (2009)
9. Katsigiannis, Y., Georgilakis, P., Karapidakis, E.: Multiobjective Genetic Algorithm Solution to the Optimum Economic and Environmental Performance Problem of Small Autonomous Hybrid Power Systems with Renewables. *IET Renewable Power Generation*, 404–419 (2010)
10. Lee, J., Kim, H., Park, G., Jeon, H.: Genetic Algorithm-based Charging Task Scheduler for Electric Vehicles in Smart Transportation. Accepted at Asian Conference on Intelligent Information and Database Systems (2012)
11. Derin, O., Ferrante, A.: Scheduling Energy Consumption with Local Renewable Micro-generation and Dynamic Electricity Prices. In: *First Workshop on Green and Smart Embedded System Technology: Infrastructures, Methods, and Tools* (2010)
12. Skog, I., Handel, P.: Time Synchronization Errors in Loosely Coupled GPS-Aided Inertial Navigation Systems. *IEEE Transactions on Intelligent Transportation Systems* 12, 1014–1023 (2011)

Design and Evaluation of Mobile Applications with Full and Partial Offloadings

Jennifer Kim

Department of Electrical Engineering and Computer Science,
University of California – Irvine,
Irvine, CA 92697, USA
Jenniyk2@uci.edu

Abstract. Mobile devices are widely accepted as a convergence machine providing both software functionality and cell phone capability. However, they have limited resources such as memory, processing power, and battery. Consequently, complex applications could not be deployed on the devices. An effective solution is to offload some functionality to more powerful servers and to run then on the servers, to yield improved performance and low resource consumption. In this paper, we propose a systematic process for designing mobile applications with full and partial offloading. And, we present schemes to quantitatively evaluate the resulting architecture. Using the proposed architecture design and evaluation methods, mobile applications with loading can be more systematically developed.

Keywords: Mobile App, Offloading Architecture, Quality Evaluation, Metrics.

1 Introduction

Mobile devices are emerging as a convenient client computing device for personal and enterprise computing. However, mobile devices have a limitation on their resources such as processing power, battery life, and memory mainly due to the small form-factor [1]. Consequently, complex applications consuming a large amount of resource could not be deployed on the devices.

To overcome the limitation, a scheme to *offload* some computation load to more powerful server is suggested [2][3]. Part of a functionality of an application is offloaded to a remote server in order for the mobile device to share the workload with a more powerful server. Mobile applications with offloading provide several benefits over stand-alone applications; reducing resource consumption on mobile devices, utilizing computing power on the server, and others [4][5].

To fully take advantage of offloading, the functionality of mobile applications has to be carefully analyzed and their architecture based on offloading principle should be well engineered [6]. Software architecture design can effectively remedy the resource limitation problem of mobile devices [7]. Current works are largely limited on these

aspects [8]. Moreover, there is a demand for evaluating the key quality attributes of offloaded architectures in formal manner. Two commonly known quality attributes for mobile applications architecture are response time and energy consumption. Moreover, practical methods to design mobile apps with partial offloading are yet to come. In this paper, we present a process to design mobile apps with offloading, and schemes to evaluate the resulting architecture with metrics.

2 Related Works

Kumar's work focuses on using computation offloading on mobile devices to save energy, thus keeping the computation cost low [9]. They consider two scenarios, one for running computation on the mobile device only, and the other for running the computation on the cloud server. The latter has every computation run on the cloud server. This work provides metrics for measuring data transfer time and processing time, not covering several other performance-related factors. Moreover, it covers full loading only. Yang's work proposes system architecture for offloading at runtime [10]. The architecture consists of a monitoring engine and an offloading decision engine. When receiving an offloading request, the system searches adequate services and compose them. A decision on the offloading is made at runtime, rather than addressing offloading-related design issues. Other related works such as [11][12][13] mainly focus on full offloading and provide limited evaluation schemes and process to design offloading architecture. Our work is to elaborate technical insights of both full and partial offloading, present a systematic process for applying offloading, and a metric-based evaluation scheme for mobile architectures with offloading.

3 Full and Partial Offloading

A standalone mobile application runs the whole functionality on a mobile device. Applications with high complexity or a large amount of dataset could not be deployed and run on the mobile device. Moreover, these applications consume a high amount of resource. Offloading can be an effective approach to remedying these problems.

With full offloading architecture, the whole application and its dataset are offloaded to a server as shown in Figure 1 (a). After offloading, the mobile device does not involve in computation, but waits results from the server.

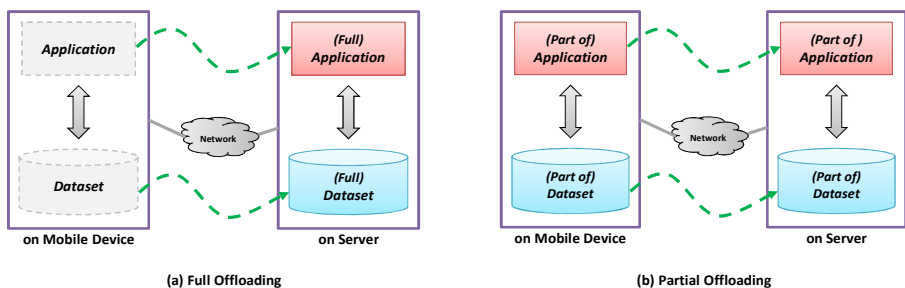


Fig. 1. Full Offloading vs. Partial Offloading

With partial offloading architecture, a part of the application and dataset manipulated by this part are offloaded as shown in Figure 1 (b). After offloading, the mobile device participates in some computation which involves interactions with users, while the server performs the rest of the computation which demands high processing power and resources. When properly designed, this architecture yields parallelism between two parties and hence a higher performance. An overhead with partial offloading is the state synchronization of replicated datasets between client and server.

4 Process to Design Mobile Applications with Offloading

4.1 Step 1. Identifying Remotable Functionality

This step is to identify which part of the whole functionality can be offloaded. To do this, we define a term *remotable function*;

A *remotable function* is a function which is originally intended to run on a mobile device but could run on a remote server to yield an enhanced quality of service such as increased performance, reduced energy consumption, and higher throughput.

We give general characteristics of *remotable function*, to guide the identification of remotable functionality;

- A function requiring intensive computation, i.e. needing a high processing power
- A function requiring high volumes of resources such as secondary storage
- A function manipulating a large amount of dataset which is located or acquirable on a server

And, general characteristics for non-remotable functions are;

- A function manipulating a large amount of dataset which is originally located on a mobile device. This type of function requires a large amount of network traffic due to the dataset. For example, an image processing function manipulating complex images stored on a mobile device is not remotable. In some cases, the gained benefit by offloading may not be justifiable due to the network cost.
- A function interacting with end users intensively. If this type of function is offloaded, it generates a large amount of network communication overhead since each user interaction has to be delivered to its server.

In addition to criteria, we define two factors to consider; *Inter-Function Dependency* and *Data Synchronization Overhead*.

- *Inter-Function Dependency*) As shown in Figure 2, F_i is a non-remotable function running on a mobile device. And, assume that F_j is determined to be remotable running on a remote server. In procedural programming, inter-function dependency occurs when a function calls another function. In object-oriented programming, inter-function dependency occurs when an object invokes a public method of another object. If F_i depends on F_j as shown in the figure, every invocation of F_j by F_i has to be made over the network resulting in network overhead. If the degree of this dependency is high, the amount of network overhead gets increased. Hence, we need to compare the cost of this network overhead to the cost benefit gained by offloading F_j when deciding the remotability of F_j .

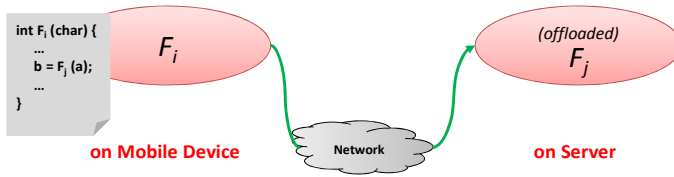


Fig. 2. Considering Inter-Function Dependency

- *Data Synchronization Overhead*) Whenever a client program on a mobile device or a server program on a server manipulates and modifies the state of an object in the data shared by both sides, that change has to be notified to and synchronized with the corresponding object on the other side. That is, the state consistency of objects in shared data has to be ensured. Figure 3 illustrates the need for maintaining consistency of the shared objects. F_i runs on a mobile device, F_j runs on a server, and each side maintains a dataset manipulated. Since both F_i and F_j collaborate to deliver the whole application functionality, there can be some objects which are manipulated by both functions. This replicated dataset is represented as $Dataset_R$ in the figure, whereas $Dataset_M$ and $Dataset_S$ are datasets manipulated exclusively by the mobile device and the server respectively. Whenever F_i manipulates objects in $Dataset_R$, it has to notify F_j and update the states of corresponding objects on the server side, and vice versa. This synchronization entails additional network overhead. If there is a heavy frequency in updating objects in $Dataset_R$, the synchronization cost would be high. This is not ideal since the synchronization cost could be greater than the cost benefit gained by partial offloading.

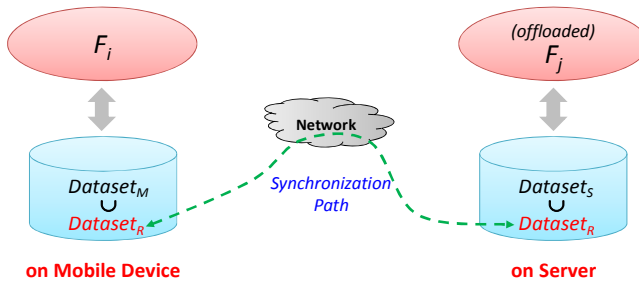


Fig. 3. Considering Dataset Synchronization Cost

4.2 Step 2. Partition Application and Datasets

This step is to partition the whole functionality and the whole dataset into mobile device part and server part using the results produced in step 1. We define terms used to illustrate the partitioned results;

- R_FnSet is the set of removable functions.
- NR_FnSet is the set of non-removable functions.

By observing what data objects are manipulated by each functional set, we can identify three datasets.

- $DataSet_M$ which is manipulated exclusively by NR_FnSet ,

- $DataSet_S$ which is manipulated exclusively by R_FnSet , and
- $DataSet_R$ which is manipulated by both sets.

Practical methods to identify manipulation relationships between datasets and functionality are found in conventional software modeling methods. The configuration with resulting functionality sets and datasets are shown in Figure 4

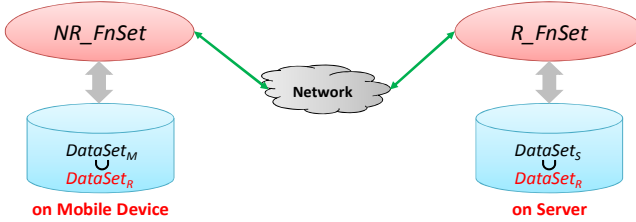


Fig. 4. Results of Partitioning Applications and Datasets

4.3 Step 3. Deploy Partitioned Applications and Datasets

This step is to implement the target application with the result of partitioning, and to deploy the functional units and dataset units. The actual units used for deployment can vary according to the programming language, deployment platform, or middleware used for the implementation.

4.4 Step 4. Run and Synchronize States

This step is to run the both client and server applications and synchronize the states of replicated dataset. At runtime, a client application will get executed first by end users, and server application will get invoked by the client application. When a state of an object in $DataSet_R$ is updated, its corresponding object will be synchronized.

5 Evaluating Response Time and Energy Consumption Rates

We first define variables and terms used for the evaluation.

- $SpeedM$ is the speed of the mobile device, i.e. processing power. It is largely determined by hardware components of the device including CPU clock speed, Cache, and Bus size.
- $SpeedS$ is the speed of a server. Both $SpeedM$ and $SpeedS$ are measured in instructions per second. Typically, $SpeedS$ is considerably faster than $SpeedM$.
- $WholeFnSet$ is the set of all functions in a mobile application. It can be measured in different ways; Lines of Code (LOC), Function Point (FP), or Number of Instructions to be carried out by processors. To be able to directly compute the time/effort that a processor takes to run mobile applications, we elect to use the unit, *Number of Instructions*. In practice, the number of instructions to run for a given program can be computed by analyzing the source program and using historical baseline data of ratios between high-level language constructs and the number of instructions.
- $DataSet$ is the dataset manipulated by $WholeFnSet$.

- BW is the bandwidth of the network between a mobile device and a server. The unit is the number of bytes transmittable in a second.
- EC_{com} is the amount of energy consumption made during running a program for one second on a mobile system, in watts. That is, this represents the rate of energy consumption on the mobile system.
- EC_{idle} is the amount of energy consumption made during one second of idle time on a mobile system, in watts.
- $EC_{transmit}$ is the amount of energy consumption made while transmitting or receiving data for one second, in watts.
- A generic function $size(x)$ returns the size of the given x . If x is a program, it returns the size (i.e. complexity) of the program, in number of instructions. If x is a dataset, it returns the size of the dataset, in number of bytes.

5.1 Evaluating Standalone Mobile Applications

For standalone mobile applications, there are $size(WholeFnSet)$ instructions, the mobile device can handle $SpeedM$ instructions per second, and it has an energy consumption rate of EC_{com} . Then, the response time, i.e. the time to run the application, is computed as;

$$\frac{size(WholeFnSet)}{SpeedM}$$

And, the amount of energy consumed to run the application is computed as;

$$\frac{size(WholeFnSet)}{SpeedM} \times EC_{com}$$

5.2 Evaluating Mobile Application with Full Offloading

The whole application has $size(WholeFnSet)$ instructions to be run, the server can handle $SpeedS$ instructions per second, and it has an energy consumption rate EC_{idle} for idle time and an $EC_{transmit}$ for transmission time. With this architecture, the whole functionality is offloaded, and the network bandwidth between two sides is BW . Then, the response time to run the application is the sum of the computation time on the server and the time to transfer the required dataset;

$$\frac{size(WholeFnSet)}{SpeedS} + \frac{size(DataSet)}{BW}$$

The first part in the metric represents the computation time on the server, and the second part represents the transfer time for the dataset. Now, the amount of energy consumed to run the application is computed as;

$$\frac{size(WholeFnSet)}{SpeedS} \times EC_{idle} + \frac{size(R_FnSet) + size(DataSet)}{BW} \times EC_{transmit}$$

The first part in the metric represents the energy consumed while the server performs the offloaded computation and mobile device is idle. And the second part represents the time to transfer the program and the dataset between two parties.

Applications which require minimal interaction with end users, intensive computation, and relatively small amount of dataset manipulated can benefit from this architecture.

5.3 Evaluating Mobile Application with Partial Offloading

When an application is partially offloaded, a set of non-remotable functions (i.e. NR_FnSet) runs on a mobile device and the set of remotable functions (i.e. R_FnSet) is offloaded to a server. That is, the whole functionality $WholeFnSet$ is partitioned into NR_FnSet and R_FnSet .

- $size(NR_FnSet)$ is the complexity of the application which runs on a mobile device when the whole functionality is partially offloaded to a server side.
- $size(R_FnSet)$ is the complexity of the application which runs on a server.

Considering datasets, $DataSet$ is partitioned into three parts. Hence, we have;

- $size(DataSet_M)$ is the size of the dataset manipulated only on the mobile device.
- $size(DataSet_S)$ is the size of the dataset manipulated only on the server side.
- $size(DataSet_R)$ is the size of replicated dataset.

Now, we consider the cost for synchronizing the dataset $DataSet_R$, and this synchronization will incur some network communication overhead. When computing the cost, we need to consider the frequencies of updating objects in the dataset. This is because the cost to update an object twice and the cost to update the same object for 200 times will be considerably different. Hence, we need to estimate the average frequency of updating the dataset.

Let $DataSet_R$ consist of data objects, $Obj_1, Obj_2, \dots, Obj_n$, and $Freq_i$ be the number of updates for i^{th} object Obj_i . Hence, the average number of updates for the entire $DataSet_R$ is computed as the following.

$$AvgNumUpdates = \frac{\sum_{i=1}^n (Freq_i)}{n}, \text{ where } n = \# \text{ of objects in Dataset}$$

If all the objects in the set are updated only once, then $AvgNumUpdates$ will be 1. If objects have multiple updates, then $AvgNumUpdates$ will be greater than 1. Hence, $AvgNumUpdates$ specifies how many times the entire set $DataSet_R$ is updated during a session. Now, we compute the time for the synchronization as;

$$SyncTime = \frac{size(DataSet_R) * AvgNumUpdates}{BW}$$

Now, we compute the response time for an application with partial offloading as the following;

$$Max \left(\frac{size(NR_FnSet)}{SpeedM}, \frac{size(R_FnSet)}{SpeedS} \right) + \frac{(size(R_FnSet) + size(DataSet_S) + size(DataSet_R))}{BW} + SyncTime$$

The first term in the *Max* function is the computation time for a client application running on a mobile device, and the second term is the computation time for a server program. Since these two programs can potentially run in parallel, we need to take a longer computation time of the two to compute the response time for the entire application. The second part of the equation is the time to transfer datasets which are offloaded to a server. The third part of the equation is the time to synchronize the replicated dataset. Now for computing the energy consumption, we need to consider two cases as illustrated in Figure 5.

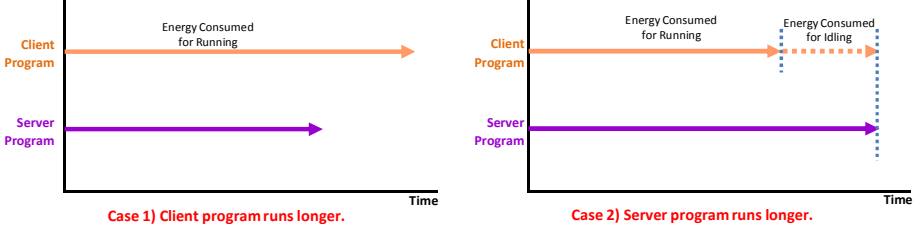


Fig. 5. Two Cases for Computing Energy Consumption

As in the figure, the energy consumption for running applications for case 1 is simply the consumption needed to run the client program;

$$EnergyForRunning = \frac{size(NR_FnSet)}{SpeedM} \times EC_{com}$$

The energy consumption for running applications for case 2 is the summation of two energy consumptions; the energy consumed for running the client program and the energy consumed for being idle between the time the client program finishes running and the time the server program finished running.

$$\begin{aligned} EnergyForRunning &= \frac{size(NR_FnSet)}{SpeedM} \times EC_{com} \\ &+ \left(\frac{size(R_FnSet)}{SpeedS} - \frac{size(NR_FnSet)}{SpeedM} \right) \times EC_{idle} \end{aligned}$$

The total amount of energy consumption will be the sum of three factors; (1) energy consumed for running programs, (2) energy consumed for offloading datasets, and (3) energy consumed for synchronization.

$$EnergyForRunning + \frac{size(R_FnSet) + size(DataSet_S) + size(DataSet_R)}{BW} \times EC_{transmit} + SyncTime \times EC_{transmit} .$$

6 Experiments and Evaluation

To understand how the proposed evaluation methods can be applied in practice, we run a number of experiments with different settings for characteristics of mobile

device, server, and mobile application. We developed and used a software tool which accepts parameters for each experiment in the set, initializes the experiment environment, initiates executions of client and server programs, monitors the measured values, and finally computes given metric values.

6.1 Experiment Settings and Results

To compare three different architectures for a given mobile application, we use fixed values for some parameters in order to minimize the experiment set. We choose parameters which have less impacts on architectural design; *SpeedM*, *SpeedS*, *DatasetRUpdated*, *UpdateFreq*, *BW*, *ECcom*, *ECidle*, and *ECtransmit*. We use variable values for the rest of the parameters which have higher impacts on architectural design; *AppSize*, *AppSizeM*, *AppSizeS*, *Dataset*, *DatasetM*, *DatasetS*, and *DatasetR*. For each parameter, we adopt to use two values to reduce the experiment set while not compromising the purpose of the experiments much. From the seven parameters, assigning with high or low values will result in the experiment set of which size is 27, i.e. 128 experiment cases. Some of these experiment cases are neither significant for architecture evaluation nor practical to be applied. Reducing such cases, we now have 15 experiment cases. Now, we assign values for each experiment case. The values for the parameters are derived by referring to specifications of various mobile devices, industry reports, and research papers. Using the experiment cases defined, we run the experiments and the results are summarized in Table 1.

Table 1. Results of Experiments

	Performance			Energy Consumption		
	Standalone	Full Off	Partial Off	Standalone	Full Off	Partial Off
Exp.01	75.00	30.47	14.69	67.50	39.14	52.05
Exp.02	75.00	30.47	9.89	67.50	39.14	45.81
Exp.03	75.00	30.47	24.43	67.50	39.14	64.72
Exp.04	75.00	30.47	23.19	67.50	39.14	63.11
Exp.05	75.00	12.47	7.33	67.50	15.74	42.49
Exp.06	75.00	30.47	14.64	67.50	39.14	30.93
Exp.07	75.00	30.47	10.21	67.50	39.14	25.17
Exp.08	75.00	30.47	24.64	67.50	39.14	43.93
Exp.09	75.00	30.47	23.41	67.50	39.14	42.33
Exp.10	75.00	12.47	7.54	67.50	15.74	21.70
Exp.11	15.00	30.09	14.35	13.50	39.03	23.14
Exp.12	15.00	30.09	9.91	13.50	39.03	17.37
Exp.13	15.00	30.09	24.35	13.50	39.03	36.14
Exp.14	15.00	30.09	23.11	13.50	39.03	34.53
Exp.15	15.00	12.09	7.25	13.50	15.63	7.16

6.2 Interpretations

Partial offloading results in the best performance as shown in Figure 6. The standalone architecture yields the shortest response time in cases 13, and 14, which has a small $size(Application)$, and a large $size(DataSet)$. This is because when $size(Application)$ is so small to begin with, there's no need to offload. Also, if the $size(DataSet)$ is too large, it would take a lot of overhead to transmit Dataset. Although full offloading is shown to perform better than the standalone architecture, it doesn't have as drastic of an improvement compared to the architecture using partial offloading.

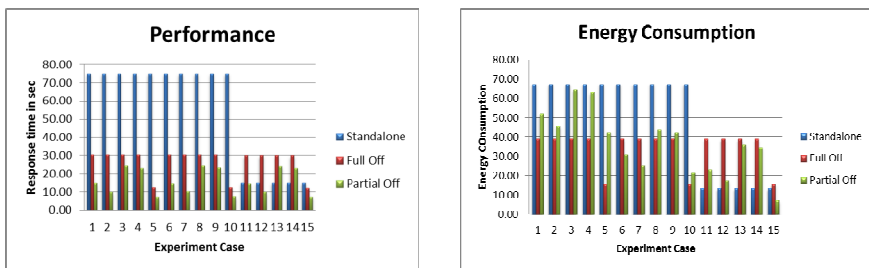


Fig. 6. Performance Analysis Results

For the energy consumption, we can see that the distribution is more diverse as shown in Figure 6. For the experiments with high $size(Application)$, full offloading had the lowest energy consumption majority of the time. When the application size is small, using the standalone architecture appeared to be the most efficient. When the application size is large, full offloading seems to be an energy saving choice for the most part. The architecture with partial offloading is shown to be energy conserving when $size(R_FnSet)$ is low, and $size(DataSet_M)$ is large.

7 Conclusion

Mobile applications with complex functionality and datasets can take advantage of architectural design, to remedy the resource constrains of mobile devices. Well-designed architecture for mobile applications will yield systems with high performance and low energy consumption. In this paper, we provided technical insights of *full offloading* and *partial offloading* architectures. And, we proposed a systematic process for applying offloading, and also defined a set of metrics to evaluate the resulting architecture. By utilizing the proposed process and metrics, mobile applications with high complexity and a requirement of high performance can be effectively developed and deployed on mobile devices.

References

- [1] König-Ries, B., Jena, F.: Challenges in Mobile Application Development. *IT-Information Technology* 52(2), 69–71 (2009)
- [2] Chen, G., Kang, B., Kandemir, M.: Studying Energy Trade Offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices. *Proceedings of IDDD Transaction on Distributed a Systems* 15(9), 795–809 (2004)
- [3] Chen, X., Lyu, M.: Performance and Effectiveness Analysis of Check pointing in Mobile Environments. In: *Proceeding of 22nd IEEE International Symposium on Reliable Distributed System* (2003)
- [4] Abukamil, A., Helal, A.: Energy Management for Mobile Devices through Computation Outsourcing within Pervasive Smart Spaces. *IEEE Transactions on Mobile Computing* (2007) (submitted)
- [5] Taylor, R.N., Medvidovic, N., Dashofy, E.M.: *Software Architecture: Foundations, Theory, and Practice*. Wiley (2010)
- [6] Chun, B., Maniatis, P.: Dynamically partitioning applications between weak devices and clouds. In: *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, San Francisco, CA, USA (2010)
- [7] Crk, I., Gniady, C.: Understanding Energy Consumption of Sensor Enabled Applications on Mobile Phones. In: *IEEE Annual International Conference of the IEEE*, Minneapolis, MN (2009)
- [8] Rao, K., Reddy, K., Rafi, S.K., Rao, T.: Effectiveness Analysis of Offloading Systems Operating in Mobile Wireless Environment. *International Journal of Engineering Science and Technology* 2(7), 3078–3086 (2010)
- [9] Kumar, K., Lu, Y.: Cloud Computing for Mobile Users: Can Offloading Computation Save Energy? *IEEE Computer* (March 2010)
- [10] Yang, K., Ou, S., Chen, H.H.: On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications. *IEEE Communications Magazine* 46(1) (2008)
- [11] Mahadevan, S.: Performance Analysis of offloading application-layer tasks to network processors. Master's Thesis, University of Massachusetts at Amherst (September 2007)
- [12] Huerta-Canepa, G., Lee, D.: An Adaptable Application Offloading Scheme Based on Application Behavior. In: *22nd International Conference on Advanced Information Networking and Applications*, Japan, Okinawa (March 2008)
- [13] Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., Bahl, P.: MAUI: Making smartphones last longer with code offloaded. In: *ACM MobiSys 2010*, San Francisco, CA, USA (June 2010)

A Cross-Layer Scheme to Improve TCP Performance in Wireless Multi-hop Networks

Fu-Quan Zhang^{1,2} and Inwhee Joe^{1,*}

¹ Division of Computer Science and Engineering, Hanyang University,
Seoul, 133-791 South Korea

² Division of Computer Software, Dong Seoul University, Gyeonggi-do,
Seongnam-si, 461-714 South Korea
iwjoe@hanyang.ac.kr

Abstract. TCP optimization problem in wireless multi-hop networks can be solved effectively by utilizing feedback from various layers. Since hop-count and round trip time are the critical factors that seriously affect TCP performance on end-to-end connection, we derive analytically the relation between these factors and TCP mechanism. The analytical result is facilitated to propose a cross-layer TCP congestion control scheme. The behavior of resulting scheme is analytically tractable. We show that our simple strategy significantly improves TCP performance in different topologies and flow patterns, in terms of throughput and delay.

Keywords: cross-layer, hop-count, round trip time, TCP, wireless multi-hop networks.

1 Introduction

During recent years wireless multi-hop networks, such as ad hoc and mesh networks, have attracted considerable research interest thanks to their easy deployment, maintenance and application variety. TCP seems to be the natural choice for users of these networks that want to communicate reliably with each other.

The suitability of TCP is coming under close scrutiny from the research community. Studies have shown that TCP performs poorly in wireless multi-hop networks [1]-[7].

This is because TCP over multi-hop wireless channel exhibits several different features [4] [8]. First, given a specific network topology and flow patterns, there exists an optimal window size, at which TCP achieves highest throughput. Further increasing the window size does not lead to further spatial channel reuse, but results in increased link layer contention and perceived packet losses. Second, the standard TCP protocol does not operate around optimal TCP window size, and typically grows its average window much larger than optimal value. The larger congestion window

* Corresponding author.

allows TCP to push more packets beyond the capacity of the path. In contrast to wireline links, dropping probability increases gradually in wireless MAC layer when the network is overloaded [9]-[11]. Consequently, TCP experiences throughput decrease due to reduced spatial channel reuse. That is, contention-induced packet loss exhibits a load-sensitive feature: as the offered TCP packets exceed the optimal TCP window size and increase further, link drop probability increases accordingly.

This phenomenon underscores the importance of setting appropriate congestion window to mitigate this problem. Some studies [2][6][7][12] have shown that TCP with a small congestion window tends to outperform TCP with a large congestion window in 802.11 multi-hop networks. These studies implies that there is a limit on the maximum sending rate of a TCP source, and in turn implies a limit on the maximum size of the congestion window.

However, how to properly increase congestion window and make TCP dynamically work around the optimal TCP window size remains an open problem in current research. As we will show later in the paper, this problem is a cross layer problem which needs to be addressed by considering factors of multiple layers.

In this paper, we analyze some critical factors from different layers, such as Hop-count of network layer and Round Trip Time of transport layer, which seriously affect the TCP performance on end-to-end connection. A relation between these factors and TCP mechanism is analytical derived firstly. Then a cross-layer solution is proposed based on the relation.

Extensive simulations were carried out under different topologies and flow patterns. The result shows that our scheme significantly improves TCP performance in the chain, grid topology and mobile scenario respectively, in terms of improving throughput with shorter delay.

The remainder of this paper is organized as follows. Section 2 presents the background of this paper and related previous works. Section 3 describes our scheme based on tractable analysis and the cross-layer solution. Section 3.1 discusses the impact of TCP window mechanism. Section 3.2 presents the assumptions and impact of loss event rate that we make for analysis. Section 3.3 proposes our scheme based on analysis result. Section 3.4 describes our cross-layer solution. Section 4 describes simulation study and simulation result. Section 5 makes the conclusion.

2 Background and Related Work

TCP and IEEE 802.11 MAC standard have been widely adopted in wireless networks. However, TCP was not primarily designed and optimized to work in wireless networks.

With the use of IEEE 802.11MAC protocol, when a node successfully obtains the channel and performs its transmission, any other node that is within the node's transmission range or the node's sensing range should not perform any transmission and must defer their transmissions for a later time. Although this conservative transmission policy can reduce the chance of packet collisions, it prevents concurrent transmissions within a neighborhood area. Therefore, the available bandwidth is underutilized. Moreover, contention allows an aggressive sender to capture the channel which reduces the chance of transmissions of the other senders in the vicinity.

Since 802.11 MAC cannot perfectly handle signal interference of general multi-hop topologies. Transmission interference of a TCP flow mainly comes from TCP data packets' self interference along the forward path. Moreover, a TCP window mechanism tends to drive wireless networks to be crowded with more packets, where a higher spatial density of packets in an area leads to a higher chance of signal interference and collision in a wireless medium. TCP pushes more packets to go beyond a certain limit, which drives excessive link-layer retransmission and eventually leads to more MAC contention loss [3][13].

Heavy contention is usually caused from using an inappropriately large congestion window. As we will show later in section 3, the reason for contention problems is that TCP does not operate its congestion window at an appropriate level.

TCP-Vegas [15] is the representative scheme that uses rate-based technique to control the congestion window size. Vegas could proactively avoid possible congestion by ensuring that the number of outstanding segments in the network is small. In this way, Vegas could prevent packet losses due to the aggressive growth of the congestion window. However, Vegas suffers from inaccuracies in the calculation of the estimated bandwidth after a route change event.

Recent studies have shown that the maximum congestion window size should be kept small and should be maintained at a level proportional to some fraction of the hop count on the path [4][6][14]

The representative scheme is Congestion Window Limit (CWL) [6]. It mitigates the congestion window overshoot problem by restricting the maximum congestion window size. CWL adjusts the maximum congestion window size dynamically according round-trip hop count so that the sending rate will not exceed the maximum spatial reuse of the channel. By this, the contention problems are reduced. However, CWL make hard constraint on the maximum window size. Moreover, it is unclear that the window limit will hold always.

Let's make a brief overview of unresolved problems of the TCP congestion control mechanisms now.

- Given a specific network topology and flow patterns, how to make TCP dynamically work around its optimal TCP window size.
- Before going to reach the optimal window size, at what kind of rate to grow TCP window size is good for different multi-hop topologies.
- How to decrease the fluctuation of the offered load on the networks which caused by the burst traffic of slow-start.
- How to reduce contention-induced packet loss which may lead to buffer overflow at intermediate routers.

3 Mechanism Analysis and TCP-CEV Scheme

3.1 Impact of TCP Window Mechanism

Below we firstly give an overview of TCP congestion control mechanisms required for the further analysis of TCP problems and development of our scheme. For more details of TCP, please see reference [16].

Slow-Start. It is used at the beginning of a transfer or a loss event occurs. The initial size of the congestion window (CWND) is set to one maximum segment size. For every arrival of non-duplicated acknowledgment, CWND increases by one. Slow-start is essentially a mechanism that estimates the available bandwidth by progressively probing for more bandwidth. During this phase, CWND increases exponentially at every round-trip time (RTT) until a packet loss event occurs or CWND reaches the slow-start threshold.

Congestion Avoidance. After CWND reaches threshold, TCP enters the congestion avoidance phase. During this phase, CWND linearly increases by one maximum segment size for each RTT until a packet loss event occurs.

During slow-start phase, if receiver acknowledges each received data packet and no congestion occurs, then congestion window size is 2^i at the end of the i^{th} RTTs. Here, we assume that CWND needs amount of m RTTs to reach threshold from 1.

The problem with slow-start is that CWND always doubles itself at the end of every next RTT. For example, at the end of the $m-2$ RTTs, CWND is one fourth of the threshold. At the end of the $m-1$ RTTs, congestion window size is equal to a half of threshold. It will reach the value of threshold at the next RTT.

If threshold is set to default or the available bandwidth is over-estimated, this acceleration at the end could cause congestion, and then some intermediate routers must queue the packets, and it's possible for that router to run out of space. Finally, the capacity of the network can be reached at some point, and then an intermediate router will start discarding packets.

The network overload caused by TCP congestion eventually leads to the hidden terminal problem and the MAC contention loss occurs after several retransmission attempts at the link layer. The packet loss at the link layer is perceived as a routing failure by routing protocol. Being confused with the routing failure, routing agent enters routing maintenance and rediscovery phase. The maintenance traffic becomes an extra network load at the very critical moment of network congestion. TCP connection is interrupted and then times out. Since there is no external packet entering the network during this period, the network overload is reduced and the routing and MAC functions are recovered. However, after the timeout, TCP restarts and soon leads to network overload again. Thus, the process repeats.

The well-known TCP-friendly equation [17] is a mathematical model to characterize the steady-state TCP behavior. It describes the average TCP transmission rate which captures the TCP throughput over a network path with certain loss rate and round-trip time. The average TCP transmission rate controlled by the TCP window over a long period of time is

$$T = \frac{s}{R \sqrt{\frac{2p}{3}} + t_{RTO} (3 \sqrt{\frac{3p}{8}}) p (1 + 32p^2)} \quad (1)$$

T is the transmission rate of TCP, s is the packet size, R is the RTT (round trip time), p is the packet loss rate in stable state. t_{RTO} is the retransmission timeout value.

When the p is small enough, equation (1) can be simplified as

$$T = \frac{s}{R \sqrt{\frac{2P}{3}}} \tag{2}$$

Equation (2) gives the upper bound of transmission rate.

3.2 Impact of Loss Event Rates

Fig.1 shows a multi-hop link topology, S_0 is the sender, R_0 is receiver, $S_i, D_i, 1, 2, \dots, n-1$ are the intermediate nodes, P_i is the packet loss rate of i^{th} hop, the loss rate P_0 of connection 0 is

$$P_0 = 1 - \prod_{i=1}^n (1 - P_i) \tag{3}$$

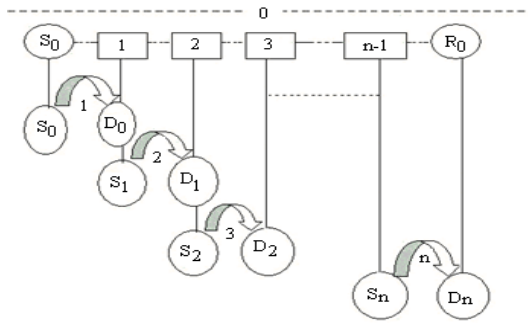


Fig. 1. Multi-hop link topology

If it is assumed that each hop has the same packet loss rate and the value of P is small enough, then

$$P_0 = 1 - (1 - p)^n \approx np \tag{4}$$

3.3 TCP CEV Scheme

To the multi-hop link, by the equation (2) and (4), the relative value between one-hop and the whole connection 0 is

$$T_0 = \frac{R_i}{R_0} * \frac{1}{\sqrt{n}} * T_i \tag{5}$$

Where R_i is RTT of the i^{th} hop, R_0 is RTT of connection 0, n is hop-count number. T_0 is transmission rate used by connection 0, T_i is the transmission rate of the i^{th} hop.

This means when hop-count and RTT change, the TCP transmission rate should change in accordance with the equation 5.

If it is assumed that each hop has same RTT, then $\frac{R_i}{R_0}$ is equal to $\frac{1}{n}$. Effect of $\frac{R_i}{R_0}$ was simulated in Paper [18]. Result shows that the assumption is reasonable to the fixed-distance placement nodes. For detailed information, please refer to [18]. It may be unreasonable to a random mobile scenario, however, TCP itself can provide the RTT ratio, so it is easy to implement in the TCP mechanism. The relative value between the i^{th} hop and connection 0 is

$$T_0 = \frac{1}{n} * \frac{1}{\sqrt{n}} * T_i \quad (6)$$

The existing TCP window mechanism is too aggressive because it tries to send packets to the network as much as possible without considering the effect of hop-count and RTT. For this reason, TCP performance is seriously influenced.

To fix this problem, we point out that TCP should Change Expected Value (TCP-CEV) and expect a reasonable throughput by giving an appropriate change scheme. This scheme dynamic controls the increase degree of congestion window when hop-count and RTT change.

For this purposes, we modified following TCP congestion control mechanism.

The slow-start algorithm progressively probes the network to determine its available capacity. During this phase, the exponential increase of congestion window for every useful received acknowledgment in standard TCP NewReno is

$$CWND = CWND + 1 \quad (7)$$

The additive increase of congestion window for every useful received acknowledgment in the Congestion-Avoidance phase of TCP NewReno is

$$CWND = CWND + \frac{1}{CWND} \quad (8)$$

It takes a full window to increment the window size by one.

We proposed CEV scheme (is short for TCP-CEV), which dynamically controls the CWND increment ratio by equation 6.

The exponential increase of congestion window for CEV is given as

$$CWND = CWND + \frac{1}{n} * \frac{1}{\sqrt{n}} * 1 \quad \text{Scheme (1-a)}$$

The initial CWND value is 1 MSS bytes, CWND doubles itself at the end of the first RTT. From the second RTT on, the window size still increases exponentially, but at a reduced rate. This exponential increase rate of CWND is $\frac{1}{n} * \frac{1}{\sqrt{n}}$ at the end of every RTT. During this period, this rate is dynamic response for different multi-hop topology. Congestion window size is $2^{(1 + \frac{1}{n} * \frac{1}{\sqrt{n}})^{i-1}}$ at the end of the i^{th} RTTs. The aggressiveness of the congestion window's growth during the slow-start period is reduced, thus reducing packet losses and dampening traffic burst. Paper [18] just modified the slow-start mechanism alone and got a good performance.

The additive increase of congestion window for CEV in the Congestion-Avoidance phase is given as

$$CWND = CWND + \frac{1}{n} * \frac{1}{\sqrt{n}} * \frac{1}{CWND} \quad \text{Scheme (1-b)}$$

Every arrival non-duplicated acknowledgment increases CWND by $\frac{1}{n} * \frac{1}{\sqrt{n}} * \frac{1}{CWND}$. If CWND is smaller than initial value, just let it be same as the initial value. During this phase, congestion window linearly increases at every round-trip time.

CEV uses the same multiplicative decrease mechanism as NewReno.

To one hop connection, our scheme is same as NewReno and appears to approach its optimal. This is because the MAC layer transmission self-interference problem does not exist in one hop. Therefore, a large congestion does not have the same negative effect on TCP performance as in longer chains. [6].

3.4 Cross-Layer Solution of CEV

Since cross-layer mechanism can effectively solve TCP optimization problem in wireless networks [19], the option of utilizing feedback from various layers have attracted significant attentions in recent years.

As we have shown in previous sections, improving the TCP performance is truly a cross-layer solution which needs to be addressed by considering factors from multiple layers. Our cross-layer solution only requires a feedback from network layer to transport layer. This feedback is the hop count value between source and destination.

4 Simulation and Comparisons

To evaluate the effectiveness of our proposal, we performed extensive simulations in NS. Each simulation is carried out under different topologies and traffic pattern to obtain the stable average value.

4.1 Simulation Parameters

The simulation parameters are listed in Table 1.

Table 1. simulation parameters

Parameters Name	Parameters Value
Examined Protocol	NewReno, CWL, Vegas, CEV
Routing protocol	AODV
MAC layer	IEEE 802.11
Propagation model	Two-ray Ground Reflection
Transmission range	250 m
Carrier sensing range	550m
Interference range	550m

4.2 Chain Topology

This section simulated our scheme in chain topologies with different hops as shown in Fig.2. Each node with 200 meters equidistance.

In investigating throughput, Figure 3 and 4 demonstrate the effect of hop count on all protocols. Increasing hop count deteriorates all of them due to the increasing possibility of MAC contention loss which explains the decrease of throughput as can be observed from both of the figures. As hop count increases, more packet loss at the link layer caused by MAC contention is perceived as a routing failure by routing protocol. Being confused with the routing failure, routing agent enters routing maintenance and rediscovery phase. While the route requests are propagating the network in search for a new route, buffers will get full and packets are dropped. As a result, TCP connection is interrupted and then times out.

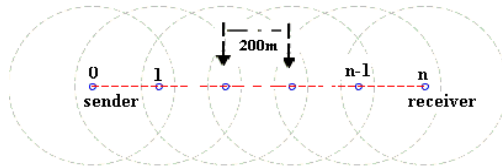


Fig. 2. n-hops chain topology

Moreover, heavy MAC contention is usually caused from using an inappropriately large congestion window. For this reason, the NewReno underperforms other protocols in most cases.

The performance of Vegas, CWL and CEV is broadly similar in terms of throughput in 1 flow, as shown in figure 3. Note that CWL make hard constraint on the maximum window size. It is difficult to hold always for different network topology and flow patterns. However, the hard constraint on the maximum window size helps to reduce contention problems. This makes CWL outperform the NewReno.

Because Vegas gauges expected throughput before increasing its congestion window size, Vegas can improve the TCP throughput. However, there are inaccuracies in the calculated expected throughput after a route change event. If bandwidth is over estimated , then network may be overloaded. On the other hand, it under utilize the available bandwidth. To some extent, the bandwidth calculation is affected by the multiplicity of flows. For this reason, Vegas performs well in 1 flow, but not well in 2 flows.

In the case of 2 flows throughput, our scheme almost outperforms all the others. The throughput of Vegas is comparable to that of CWL. In contrast to NewReno, CEV has revealed dramatic improvements in throughput (8-98%). CEV increases throughput up to 48% and 46% when compares to CWL and Vegas respectively.

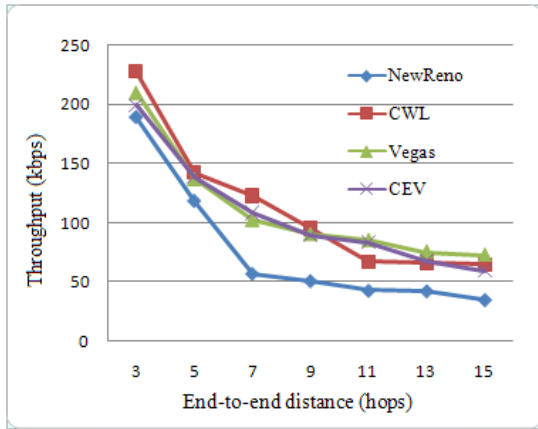


Fig. 3. Throughput in N-hops chain topology (1 flow)

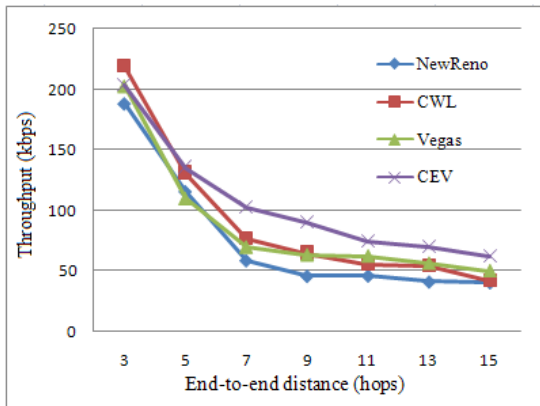


Fig. 4. Throughput in N-hops chain topology (2 flows)

Meanwhile, figure 5 and 6 show the effect of hop count on end-to-end delay. As indicated by both figures, while the path becomes more longer, the delay of all protocols undergoes increase.

This is because the time spent to receive, decode, and retransmit a packet and it is proportional to the number of hops between source and destination.

Moreover, with a smaller number of hops, each packet experiences fewer retransmissions at each intermediate hop. As the hop count increases and network is overload, there is more probability of contention and link failures. The number of nodes contending for channel access at same time becomes higher, thus the time spent by a packet waiting in a queue for transmission will be longer, which depends on the congestion level of wireless channels. When the network is highly loaded, this time is the dominant part of the end-to-end delay.

Since this delay is primarily caused by network overloaded and MAC contention. Moreover, MAC contention may increase with the increase of hop count. In contrast

to other protocols, CEV always take the hop count into consideration of its congestion window operation. This makes CEV keeps an appropriate level to forward data packets. As a result, CEV has shortest delay in most cases.

Since bandwidth calculation is more likely to approach available bandwidth in single flow. The delay of Vegas is comparable to that of CEV in 1 flow, and the delay of 2 flows in Vegas is longer than CEV.

The hard constraint of maximum window size helps CWL to reduce contention problems. However, as shown in figure 5 and 6, the hard constraint may apply only to a few instances. For example in short chains where the self-interference problem by TCP's data is less severe due to the small number of contending nodes[13].

In contrast to NewReno, our scheme has revealed dramatic improvements in reducing end-to-end delay (30–45% in 1flow and 26-46% in 2 flows). In the case of 2 flows, CEV also reduces end-to-end delay up to 42% and 35% when compared to CWL and Vegas respectively.

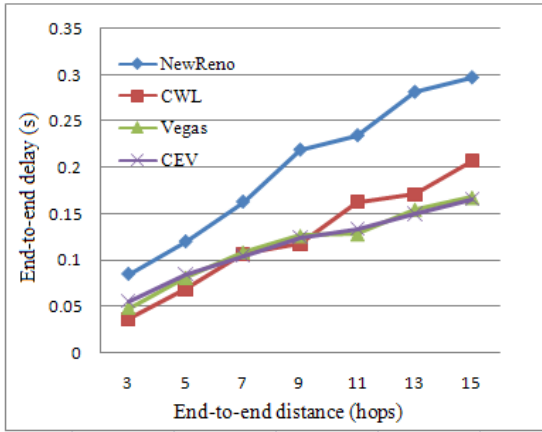


Fig. 5. End-to-end in N-hops chain topology (1 flow)

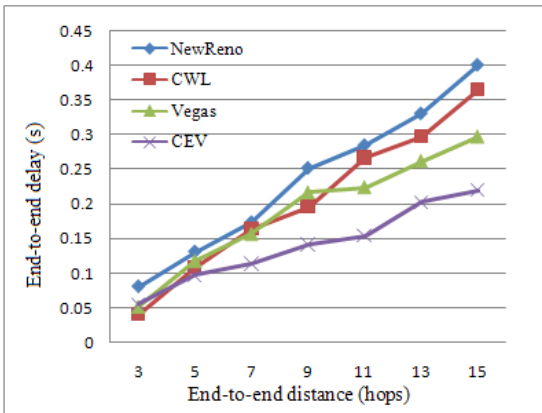


Fig. 6. End-to-end in N-hops chain topology (2 flows)

The next two figures, figure 7 and figure 8 show the congestion window behavior of all the protocols in 1 flow and 2 flows respectively, for the case of 7-hop chain topology. The standard NewReno is far from the ideal AIMD pattern of congestion control in all cases.

In contrast to NewReno, the hard constraint on the maximum window size makes CWL strictly operate its congestion under its limit. However, since the hard constraint on the maximum window size is set by considering a specific flow. It is inapplicable to a multiplicity of flows.

Vegas takes bandwidth estimation to enhance TCP performance. Since any route change will lead to incorrect bandwidth estimates. In contrast to single flow, route change is more frequently in multiplicity of flows. Thus, the congestion window of Vegas is unstable.

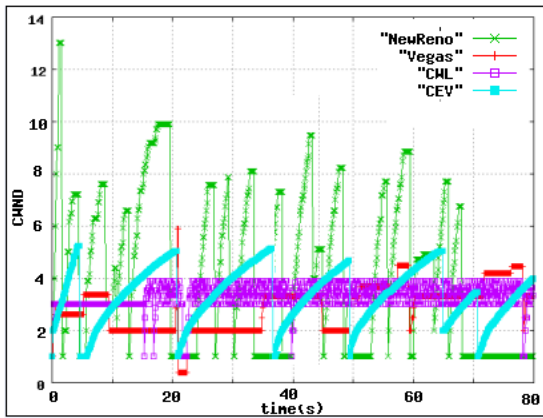


Fig. 7. Congestion window over time (1 flow)

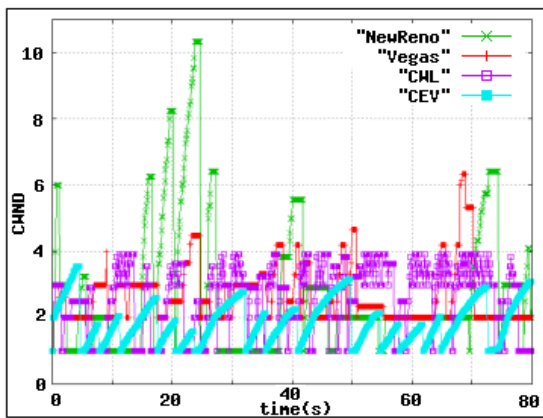


Fig. 8. Congestion window over time (2 flows)

The modified slow-start algorithm of CEV could avoid the typical sharp increase in congestion window during the slow-start phase by setting a reasonable increase rate. In this way, the contention caused by network overloaded and the route rediscovery due to congestion losses misinterpreted are reduced. As shown in figure 7 and 8, CEV can operate its congestion window with a more regular AIMD pattern. Moreover, according to the different flows, CEV can also maintain its congestion window at a proportional level, which was observed in [4][14].

4.3 More Complex Topology and Flow Patterns

In order to further verify the performance, we expand our study to scenarios of more complex topologies and TCP flow patterns, including grid topologies and mobile scenario.

Grid Topology

Grid topologies consist of nodes that are evenly deployed in a two-dimensional simulation area. This configuration has much more alternative path than the chain topology. We used a small scale (7x7 square) and a large scale (11x4 rectangle) grid topology, 200 meter distance between horizontal or vertical adjacent nodes. The sources are randomly selected from the left side and the destinations are selected randomly from the right side. We run 2, 4, 6 and 8 TCP flows respectively.

Figure 9 and 10 show 7x7 and 11x4 grid topology results respectively. Since NewReno always trend to inject packets overload the capacity of the path, this makes it underperform three other protocols in throughput.

As expected, CEV performs well. The main reason is that a reasonable increasing windows size avoids the sender injecting packets overload the capacity of the path. It reduces the wireless MAC layer’s dropping probability.

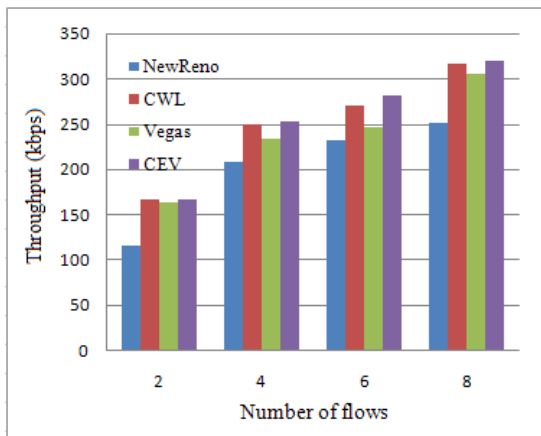


Fig. 9. 7x7 grid topology

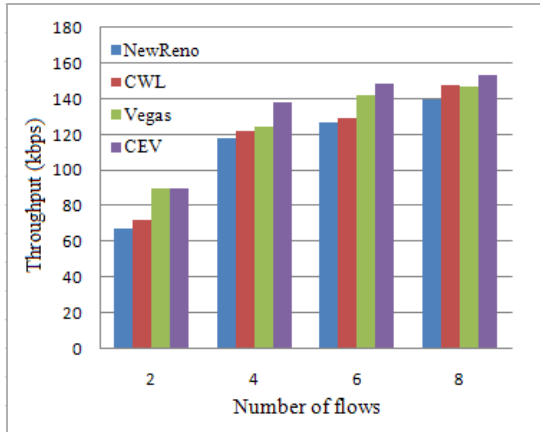


Fig. 10. 11x4 grid topology

Mobile Scenarios

We also run extensive simulations with mobile scenarios, in which 50 wireless mobile nodes are roaming in a 1500 x 300 meters flat space. The mobile model is the random way point. The minimum speed is 0m/s and the maximum speed is 10 m/s without pause time. We run 1, 2, 4, 8 and 12 TCP flows respectively. In each of these cases, flows are randomly created.

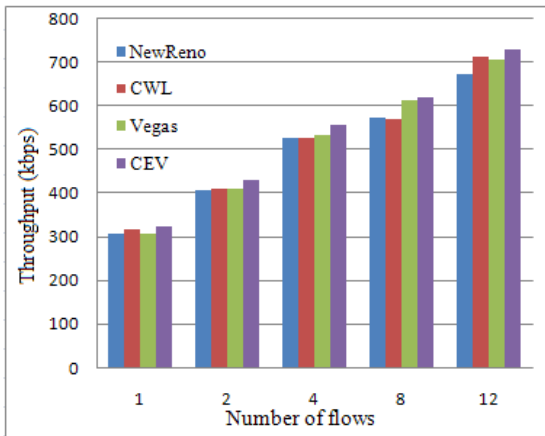


Fig. 11. Throughput in the mobile scenarios

As the flow increases, there is going to be more crowded in the network. However, CEV outperforms three other protocols and improve throughput, as shown in Fig. 11. This is because the link failure due to congestion over multiple hops is more frequent and persistent than those due to node mobility during the entire connection lifetime.

5 Conclusion

We have presented a simple but analytical traceable TCP Mechanism. It makes use of RTT and hop-count information to improve the throughput by changing the packet transmission rate.

There contributions of this paper are as follows:

- We analyze the factors impact on the TCP performance.
- Behavior of the CEV is analytically tractable.
- CEV significantly improves TCP performance in the chain, grid topologies and mobile scenarios, respectively,
- CEV makes TCP dynamically work around the optimal TCP window size in different network topology.

Acknowledgements. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government, Ministry of Education, Science and Technology (2011-0015681, 2011-0004974).

References

1. Gerla, M., Tang, K., Bagrodia, R.: TcP performance in wireless multihop networks. In: Proc. IEEE International Workshop on Mobile Computing Systems and Applications (WMCSA 1999), New Orleans, Louisiana, USA (February 1999)
2. Fu, Z., Meng, X., Lu, S.: How bad tcp can perform in mobile ad hoc networks. In: Proc. IEEE International Symposium on Computers and Communications (ISCC 2002), Taormina, Italy (July 2002)
3. Fu, Z., et al.: The Impact of Multihop Wireless Channel on TCP Performance. IEEE Transactions on Mobile Computing 4(2), 209–221 (2005)
4. Fu, Z., Zerfos, P., Luo, H., Lu, S., Zhang, L., Gerla, M.: The impact of multihop wireless channel on tcp throughput and loss. In: Proc. IEEE Infocom 2003, San Francisco, California, USA (April 2003)
5. Haitao, W., Lihua, S.: Performance of TCP in ad hoc network and its improvement polices. Journal of Northwest University 34(5), 442–445 (2004)
6. Chen, K., Xue, Y., Nahrstedt, K.: On setting TCP's congestion window limit in mobile ad hoc networks. In: Proc. IEEE ICC 2003, Anchorage, Alaska (May 2003)
7. Xu, K., Bae, S., Lee, S., Gerla, M.: TcP behavior across multihop wireless networks and the wired internet. In: Proc. ACM Workshop on Wireless Mobile Multimedia (WoWMoM 2002), Atlanta, Georgia, USA (September 2002)
8. Li, J., Blake, C., De Couto, D.S.J., Lee, H., Morris, R.: Capacity of ad hoc wireless networks. In: Proc. ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2001), Rome, Italy (July 2001)
9. Xu, S., Saadawi, T.: Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? IEEE Communications Magazine, 130–137 (June 2001)
10. Hamadani, E., Rakocevic, V.: TCP Contention Control: A Cross Layer Approach to Improve TCP Performance in Multihop Ad Hoc Networks. In: Boavida, F., Monteiro, E., Mascolo, S., Koucheryavy, Y. (eds.) WWIC 2007. LNCS, vol. 4517, pp. 1–16. Springer, Heidelberg (2007)

11. Ng, P.C., Liew, S.C.: Re-Routing Instability in IEEE 802.11 Multi-Hop Ad-Hoc Networks. In: 29th Annual IEEE International Conference on Local Computer Networks (2004)
12. Chen, K., Nahrstedt, K.: Limitations of equation-based congestion control in mobile ad hoc networks. In: Proc. IEEE WWAN, Tokyo, Japan (March 2003)
13. Chen, K., Xue, Y., Shah, S.: Understanding bandwidth-delay product in mobile ad hoc networks. Special issue on Protocol Engineering for Wired and Wireless Networks, Elsevier Computer Communications (2004)
14. Zhai, H., Chen, X., Fang, Y.: Alleviating Intra-flow and inter-flow contentions for reliable service in mobile ad hoc networks. In: IEEE MILCOM (2004)
15. Brakmo, L., O'Malley, S., Peterson, L.: TCP Vegas: New techniques for congestion detection and avoidance. In: Proceedings of the SIGCOMM 1994 Symposium, pp. 24–35 (1994)
16. <http://www.ietf.org/rfc/rfc2581.txt>
17. Padhye, J., Firoiu, V., Towsley, D., Krusoe, J.: Modeling TCP throughput: A simple model and its empirical validation. In: Proc. ACM SIGCOMM 1998, Vancouver, CA, pp. 303–314 (September 1998)
18. Quan, Z.F., Kai, M.L., Park, Y.-J.: Reasonable TCP's Congestion Window Change Rate to Improve the TCP Performance in 802.11 Wireless Networks. In: Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology, pp. 808–812 (2008)
19. Srivastava, V., Motani, M.: Cross-Layer Design: A Survey and the Road Ahead. IEEE Communications Magazine, 112–119 (December 2005)

A Fully Abstract View for Local Cause Semantics

Jianxin Xue and Xiaoju Dong

BASICS Lab, Department of Computer Science
MOE-MS Key Laboratory for Intelligent Computing and Intelligent Systems
Shanghai Jiao Tong University, Shanghai, China
Jason_xjx@sjtu.edu.cn, dong-xj@cs.sjtu.edu.cn

Abstract. Recent technological trends have pushed concurrency and mobile computing to the mainstream. Too many semantics have been proposed to picture the behavior of concurrent systems, and their relationship has been a hot topic in concurrency. We aim in this paper to bridge two important concurrent semantics, the true concurrency semantics and the interleaving semantics, based on the local cause semantics, in the framework of mobile computing. We enrich the polyadic π -calculus with the local cause semantics, and present a fully abstract, finiteness-respecting encoding from the local cause polyadic π -calculus to the polyadic π -calculus. Therefore, the local cause bisimulation is reduced to the observation bisimulation. Moreover, a new decidable approach is proposed for the local cause bisimulation on finite processes.

1 Introduction

Technological trends on grid computing and cloud computing have pushed concurrency to the mainstream. Some concurrency semantics has been applied to these fields. However, there are too many semantics in concurrency, for example, the interleaving semantics [Mil89, MPW92] and the true concurrency semantics [DP99, BCH⁺08]. The former can get the properties of composition and substitutivity easily, and the latter can describe the causality and distribution properly. But they can not occupy all these virtues. Investigating the relationship between various concurrency semantics can help making use of them adequately in concrete scenarios of grid computing and cloud computing. Therefore, our main goal is to bridge the gap of the interleaving semantics and the true concurrency semantics. And another one is to deduce from their relationship some new properties for the true concurrency semantics, for example, new decidable approaches on decidable fragments. As we know, it is impossible to develop a finite axiomatic system for any reasonable true concurrency semantics congruence on finite processes without auxiliary operators [Mo90].

This paper takes the local cause semantics [Kie94], a spatial-sensitive true concurrency semantics, as the testbed. The local cause semantics observes the dependency of actions by local causes and describes the local cause behavior equivalence on processes. And we follow the approach to encoding the local

cause mechanism into the polyadic π -calculus [Mil93], which is an interleaving framework and a classic mobile computing model. The reason for taking the polyadic π -calculus into account is that the rapid growing of the smart phones and tablets pushes the mobile calculus to be a hot topic.

This paper makes the following contributions:

- The polyadic π -calculus enriched with the local cause semantics, which is introduced for CCS [Kie94], is proposed. We investigate the relationship between the interleaving semantics and the local cause semantics based on the observation bisimulation and the local cause bisimulation in a unified framework, and general results is showed.
- A fully abstract encoding is proposed from the local cause polyadic π -calculus to the polyadic π -calculus on the basis of *finite wire processes* which models the local cause mechanism. The local cause bisimulation/congruence is reduced to the observation bisimulation/congruence. Moreover, this encoding respects process finiteness, that is, finite local cause processes are mapped onto finite standard processes.
- A new approach is presented to proving the local cause bisimulation on finite processes. In other words, the local cause bisimulation on finite local cause processes can be inferred from the proof system [Zhu09] for the observation bisimulation on finite processes.

The rest of the paper is organized as follows. Section 2 recalls the polyadic π -calculus. And the polyadic π -calculus is enriched with the local cause semantics in Section 3. Section 4 presents an encoding from the local cause polyadic π -calculus to the polyadic π -calculus, and proves its properties. Section 5 applies the result to decidable results of the local cause bisimulation. Section 6 discusses related work and concludes. For the limit of the space, detailed proofs are omitted and can be found in the full version [XD12].

2 Polyadic π -Calculus

The polyadic π -calculus adopted here is slightly different from the original one [Mil93] in the name dichotomy between names and name variables. For the discussion why the distinction is important can be consult [FL10, FZ11]. All processes of the polyadic π -calculus can be assumed *well-sorted* [Mil93], and the following conventions are adopted if without extra notations.

- The letters a, b, c, d, e, f, g range over the infinite set \mathcal{N} of *names*.
- The letters u, v, w, x, y, z range over the infinite set \mathcal{N}_v of *name variables*.
- The letters m, n, o range over $\mathcal{N} \cup \mathcal{N}_v$.

We often write \tilde{n} to stand for the tuple of names or name variables with length $|\tilde{n}|$. The set \mathcal{T} of the polyadic π -terms is inductively generated by the following grammar:

$$\begin{aligned}
T &:= \mathbf{0} \mid \pi.T \mid T+T \mid T|T \mid (c)T \mid [m=n]T \mid [m\neq n]T \mid !T \\
\pi &:= n(\tilde{x}) \mid \bar{n}(\tilde{m}).T \mid \tau
\end{aligned}$$

$$\begin{array}{c}
 \frac{a(\tilde{x}).T \xrightarrow{\alpha(\tilde{b})} T\{\tilde{b}/\tilde{x}\}}{S \xrightarrow{\lambda} S'} \quad \frac{\bar{a}(\tilde{b}).T \xrightarrow{\bar{\alpha}(\tilde{b})} T \quad \tau.T \xrightarrow{\tau} T}{S \xrightarrow{\alpha(\tilde{b})} S' \quad T \xrightarrow{(\tilde{b}')\bar{\alpha}(\tilde{b})} T'} \quad \frac{\tau.T \xrightarrow{\tau} T}{S | T \xrightarrow{\tau} (S' | T')} \\
 \frac{S | T \xrightarrow{\lambda} S' | T}{\ln(\lambda) \cap \text{gn}(T) = \emptyset} \quad \frac{S \xrightarrow{\alpha(\tilde{b})} S' \quad T \xrightarrow{(\tilde{b}')\bar{\alpha}(\tilde{b})} T'}{S | T \xrightarrow{\tau} (\tilde{b}')(S' | T')} \quad \frac{\tau.T \xrightarrow{\tau} T}{\{\tilde{b}'\} \cap \text{gn}(S) = \emptyset} \\
 \frac{T \xrightarrow{(\tilde{b}')\bar{\alpha}(\tilde{b})} T'}{(c)T \xrightarrow{(\tilde{b}')\bar{\alpha}(\tilde{b})} T'} \quad c \neq a, c \in \{\tilde{b}\} - \{\tilde{b}'\} \quad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'} \quad c \notin \text{ln}(\lambda) \\
 \frac{S \xrightarrow{\lambda} S'}{S + T \xrightarrow{\lambda} S'} \quad \frac{T \xrightarrow{\lambda} T'}{[a=a]T \xrightarrow{\lambda} T'} \quad \frac{T \xrightarrow{\lambda} T'}{[a \neq b]T \xrightarrow{\lambda} T'} \quad \frac{T | !T \xrightarrow{\lambda} T'}{!T \xrightarrow{\lambda} T'}
 \end{array}$$

Fig. 1. Semantics rules of the polyadic π -calculus

The nil process $\mathbf{0}$ can not do anything in any environment and is always omitted. The binder π of the prefix term $\pi.T$ is either an input prefix $n(\tilde{x})$, an output prefix $\bar{n}(\tilde{m})$ or a silent τ prefix. In $n(\tilde{x}).T$ the variables are bound. A name variable is free if it is not bound. The function $\text{fv}(\cdot)/\text{bv}(\cdot)$ returns free/bound name variables respectively. The terms $T+T$ and $T|T$ are respectively a choice and a concurrent composition. The conditional operator $[m=n]$ is a match and $[m \neq n]$ a mismatch. The replication term $!T$ introduces infinite behavior. The restriction term $(c)T$ is in localization form, where c is a local name. A name is global if it is not local. The function $\text{gn}(\cdot)/\text{ln}(\cdot)$ respectively returns global/local names. Both local names and bound variables are subject to α -conversion. A term is open if it contains free variables; it is closed otherwise. And a closed term is also called a process. The set \mathcal{P} of the polyadic π -processes is ranged over by O, P, Q . The set \mathcal{L} of observable actions is defined as $\{a(\tilde{b}), \bar{a}(\tilde{b}), (\tilde{b}')\bar{a}(\tilde{b}) \mid a, \tilde{b}, \tilde{b}' \in \mathcal{N}\}$, where $a(\tilde{b}), \bar{a}(\tilde{b}), (\tilde{b}')\bar{a}(\tilde{b})$ denotes respectively an input action, an output action and a bound output action, and ranged over by ℓ . The set of actions $\text{Act} \stackrel{\text{def}}{=} \mathcal{L} \cup \{\tau\}$ is ranged over by λ . The semantics rules are given in Fig. 1 and the symmetric ones are omitted.

We write $\xrightarrow{\hat{\lambda}}$ for \equiv if $\lambda = \tau$ and for $\xrightarrow{\lambda}$ otherwise. The notation \Longrightarrow denotes the reflexive and transitive closure of $\xrightarrow{\tau}$. The notation $\xRightarrow{\hat{\lambda}}$ stands for $\Longrightarrow \xrightarrow{\hat{\lambda}} \Longrightarrow$. Now we define Milner and Park’s bisimulation equality [Mil89].

Definition 1 (Weak bisimulation). A symmetry relation $\mathcal{R}: \mathcal{P} \times \mathcal{P}$ is a weak bisimulation if $PRQ \xrightarrow{\lambda} Q'$ then $P \xRightarrow{\hat{\lambda}} P'\mathcal{R}Q'$ for some P' and every $\lambda \in \text{Act}$. The weak bisimilarity \approx is the largest weak bisimulation.

Two techniques on the weak bisimulation, weak bisimulation up-to context and X-property (also called Bisimulation Lemma), should be recalled.

Definition 2 (Weak bisimulation up-to context, [San94]). A symmetric relation $\mathcal{R}: \mathcal{P} \times \mathcal{P}$ is a weak bisimulation up-to context if $PRQ \xrightarrow{\lambda} Q''$ then $Q'' = (\tilde{c})(R | Q')$, $P \xRightarrow{\hat{\lambda}} (\tilde{c})(R | P')$ and $P'\mathcal{R}Q'$ for some static context $(\tilde{c})(R | \cdot)$, processes P', Q' , and every $\lambda \in \text{Act}$.

Lemma 1 ([San94]). *If \mathcal{R} is a weak bisimulation up-to context, then $\mathcal{R} \sqsubseteq \approx$.*

Lemma 2 (X-property, [DNMV90]). *If $P \Longrightarrow \approx Q$ and $Q \Longrightarrow \approx P$, then $P \approx Q$.*

3 Local Cause Polyadic π -Calculus

We adapt for the polyadic π -calculus the local cause semantics introduced for CCS [Kie94], where the processes are extended with local causes capturing the spatial-dependency of actions. The set \mathcal{T}_{lc} of local cause terms is defined by the following grammar:

$$A := l:A \mid A|A \mid (c)A \mid T.$$

All notations, conventions and terminologies introduced on standard terms are extended to the enriched language and will not be repeated, except that the set \mathcal{P}_{lc} of processes of the local cause polyadic π -calculus is ranged over A, B . The mixed denotation of terms and processes does not matter because we focus on processes only. We comment more on the local cause prefix term $l:A$. The label l representing an executed action is an atomic local cause and taken from the countable set \mathcal{N}_{lc} , where $\mathcal{N}_{lc} \cap \mathcal{N} = \emptyset$ and $\mathcal{N}_{lc} \cap \mathcal{N}_v = \emptyset$. Local cause prefixes can not be executed but they can be observed with a visible transition. Local cause sequences, denoted by L, K , consist of ordered atomic local causes. The function $\text{cau}()$ returns atomic local causes of local cause sequences, terms and processes.

The semantic rules of local cause processes are defined by the enriched labeled transition system in Fig. 2. When every observable action is executed, a globally unique local cause (l) is generated dynamically to bind this action. The execution of this action depends on the executions of the actions observed at the atomic local causes of the set $(\text{cau}(L))$. The notation $A\{\varepsilon/l\}$ means substituting the local cause l of A by the empty cause which is not bound by any observable actions. Silent τ actions do not generate local causes. Just as CCS [Mil89] to the polyadic π -calculus, CCS enriched with local causes (CCS_{lc} for short) is a special case of the local cause polyadic π -calculus. Its detailed syntax and semantics can be found in [Kie94]. We use examples of CCS_{lc} instead of the local cause polyadic π -calculus for the convenience, and it is showed to be reasonable in Section 5. There are many local cause processes like $l:l:P$, which are legal syntactically and out of our foci. We are mainly interested in the reachable subset \mathcal{P}_r of the local cause processes, which can be obtained by evolving from standard processes. The local cause processes discussed in this paper are the ones of the reachable subset if without notations.

Definition 3 (Reachable set). *Let $\longrightarrow_{lc} \stackrel{\text{def}}{=} \bigcup_{\text{cau}(L);l} \xrightarrow{\ell} \cup \Longrightarrow$. \mathcal{P}_r is the least set of processes satisfying (1) $\mathcal{P} \subseteq \mathcal{P}_r$ and (2) if $P \in \mathcal{P}_r$ and $P \longrightarrow_{lc} P'$ then $P' \in \mathcal{P}_r$.*

We write $\xrightarrow[\text{cau}(L);l]{\ell}$ for $\Longrightarrow \xrightarrow[\text{cau}(L);l]{\ell} \Longrightarrow$. Based on the local cause transition system, the local cause bisimulation can be defined.

$$\begin{array}{c}
 \frac{}{a(\tilde{x}).T \xrightarrow[\emptyset;l]{a(\tilde{b})} l : T\{\tilde{b}/\tilde{x}\}} \quad \frac{}{\bar{a}(\tilde{b}).T \xrightarrow[\emptyset;l]{\bar{a}(\tilde{b})} l : T} \quad \frac{}{\tau.T \xrightarrow{\tau} T} \\
 \frac{T \xrightarrow[\emptyset;l]{\ell} T'}{[a=a]T \xrightarrow[\emptyset;l]{\ell} T'} \quad \frac{T \xrightarrow[\emptyset;l]{\ell} T'}{[a \neq b]T \xrightarrow[\emptyset;l]{\ell} T'} \quad \frac{T \mid !T \xrightarrow[\emptyset;l]{\ell} T'}{!T \xrightarrow[\emptyset;l]{\ell} T'} \quad \frac{T \mid !T \xrightarrow{\tau} T'}{!T \xrightarrow{\tau} T'} \\
 \frac{A \xrightarrow[\text{cau}(L);l]{(\tilde{b}')\bar{a}(\tilde{b})} A'}{c \neq a, c \in \{\tilde{b}\} - \{\tilde{b}'\}} \quad \frac{A \xrightarrow[\text{cau}(L);l]{\ell} A'}{c \notin \text{gn}(\ell)} \\
 \frac{(c)A \xrightarrow[\text{cau}(L);l]{(\tilde{b}')\bar{a}(\tilde{b})} A'}{(c)A \xrightarrow[\text{cau}(L);l]{\ell} (c)A'} \\
 \frac{A \xrightarrow[\text{cau}(L);l]{\ell} A'}{k : A \xrightarrow[\text{cau}(k;L);l]{\ell} k : A'} \quad \frac{A_1 \xrightarrow[\text{cau}(L);l]{\ell} A'_1}{A_1 \mid A_2 \xrightarrow[\text{cau}(L);l]{\ell} A'_1 \mid A_2} \quad \frac{}{\text{ln}(\ell) \cap \text{gn}(A_2) = \emptyset} \\
 \text{Com: } \frac{A_1 \xrightarrow[\text{cau}(L);l]{(\tilde{b}')\bar{a}(\tilde{b})} A'_1 \quad A_2 \xrightarrow[\text{cau}(K);k]{a(\tilde{b})} A'_2}{A_1 \mid A_2 \xrightarrow{\tau} (\tilde{b}')(A'_1\{\varepsilon/l\} \mid A'_2\{\varepsilon/k\})} \quad \frac{}{\{\tilde{b}'\} \cap \text{gn}(A_2) = \emptyset} \\
 \frac{A_1 \xrightarrow{\tau} A'_1}{A_1 \mid A_2 \xrightarrow{\tau} A'_1 \mid A_2} \quad \frac{A \xrightarrow{\tau} A'}{(c)A \xrightarrow{\tau} (c)A'} \quad \frac{A \xrightarrow{\tau} A'}{l : A \xrightarrow{\tau} l : A'}
 \end{array}$$

Fig. 2. Semantics rules of the local cause polyadic π -calculus

Definition 4 (Local cause bisimulation). A symmetric relation $\mathcal{R} : \mathcal{P}_{lc} \times \mathcal{P}_{lc}$ is a local cause bisimulation if the following statements are valid.

1. If $ARB \xrightarrow{\tau} B'$ then $A \Longrightarrow A'\mathcal{R}B'$ for some B' .
2. If $ARB \xrightarrow[\text{cau}(L);l]{\ell} B'$ then $A \xrightarrow[\text{cau}(L);l]{\ell} A'\mathcal{R}B'$ for some A' .

The local cause bisimilarity \approx_{lc} is the largest local cause bisimulation.

4 Encoding the Local Cause Polyadic π -Calculus into the Polyadic π -Calculus

An encoding is presented from the local cause polyadic π -calculus to the polyadic one, and its properties, such as finiteness-respecting, full abstraction, are proved.

4.1 Intuitive Example

The intuition underlying the encoding is as follows: local causes picturing the dependency of executed actions are encoded into a *finite wire process* of the form $l \triangleright L$, which means that the action binding l can not be executed until the actions binding atomic local causes in L have been executed. As an example, the local cause process $A = k:h:\bar{a}.c$ can perform the following transitions:

$$k:h:\bar{a}.c \xrightarrow[\{k,h\};l_1]{\bar{a}} k:h:l_1:c \xrightarrow[\{k,h,l_1\};l_2]{c} k:h:l_1:l_2:\mathbf{0}.$$

These transitions show that the action \bar{a} (respectively, b) binds the local cause l_1 (respectively, l_2) and it can execute only after the executions of the actions binding the local causes k and h (respectively, k, h and l_1).

Its encoding has the following transitions:

$$\begin{aligned} \llbracket k:h:\bar{a}.c \rrbracket &\xrightarrow{(b_1)\bar{a}\langle b_1 \rangle} \xrightarrow{(l_1)\bar{b}_1\langle l_1 \rangle} l_1.(\bar{k} \mid \bar{h}) \mid \llbracket k:h:l_1:c \rrbracket \\ &\xrightarrow{c(b_2)} \xrightarrow{(l_2)\bar{b}_2\langle l_2 \rangle} l_1.(\bar{k} \mid \bar{h}) \mid l_2.(\bar{k} \mid \bar{h} \mid \bar{l}_1) \mid \mathbf{0}. \end{aligned}$$

The actions $(b_1)\bar{a}\langle b_1 \rangle$ and $c(b_2)$ are respectively correspondent to the action \bar{a} and c of the pre-image. l_1 , which is extruded by the action $(l_1)\bar{b}_1\langle l_1 \rangle$, means that it is bound by the action \bar{a} in the pre-image; Similarly, l_2 is bound by the action c . $l_1.(\bar{k} \mid \bar{h})$ and $l_2.(\bar{k} \mid \bar{h} \mid \bar{l}_1)$ are both stances of the *finite wire process*. The former simulates that the action \bar{a} cannot be executed before the actions binding k, h are executed, and the latter simulates that the action c cannot be performed before the ones binding k, h, l_1 are performed.

The purpose of the extra actions $(l_1)\bar{b}_1\langle l_1 \rangle$ and $(l_2)\bar{b}_2\langle l_2 \rangle$ is to model properly the communications those who do not generate local causes, and an example is illustrated as follows:

$$\begin{aligned} \llbracket k_1:h_1:\bar{a}.c \rrbracket \mid \llbracket k_2:h_2:a.d \rrbracket &\xrightarrow{\tau} \\ (bl_1l_2)(\bar{b}l_1.l_1 \triangleright k_1:h_1 \mid \bar{b}l_2.l_2 \triangleright k_2:h_2 \mid \llbracket k_1:h_1:l_1:c \rrbracket \mid \llbracket k_2:h_2:l_2:d \rrbracket) &\stackrel{\text{def}}{=} A \end{aligned}$$

The prefixes guarding the *finite wire processes* ensure that the local causes l_1 and l_2 can not be extruded such that $A \sim \llbracket k_1:h_1:c \rrbracket \mid \llbracket k_2:h_2:d \rrbracket$.

4.2 Encoding

Two preliminary steps are needed before the encoding is formally presented. Firstly, we use directly the local causes as names, which are called local cause names. The local cause names are not included in the set of names. The set \mathcal{N} of names is extended to $\mathcal{N} \cup \mathcal{N}_{lc}$ abbreviated as \mathcal{N}^+ . And their syntax and semantics are the same as those of names. Secondly, a crucial *finite wire process*, which models the spatial-dependency of actions, is defined as follows

$$l \triangleright L \stackrel{\text{def}}{=} l(z). \prod_{k \in \text{cau}(L)} \bar{k}\langle z \rangle, \text{ where } l \in \mathcal{N}_{lc} - \text{cau}(L).$$

A *finite wire process* contains two information: the local cause (l) generated and bound dynamically while an action is executed, and the local causes ($\text{cau}(L)$) generated by the executions of the dependent actions of the current one. When L is empty, it denotes that the action observed at l does not depend on the executions of other actions. To model the communication actions which do not generate local causes, the *finite wire process* is adapted to be guarded by an extra prefix:

$$\bar{n}\langle l \rangle.l \triangleright L \stackrel{\text{def}}{=} \bar{n}\langle l \rangle.l(z). \prod_{k \in \text{cau}(L)} \bar{k}\langle z \rangle.$$

$$\begin{aligned}
\llbracket \mathbf{0} \rrbracket L &\stackrel{\text{def}}{=} \mathbf{0} & \llbracket T+T' \rrbracket L &\stackrel{\text{def}}{=} \llbracket T \rrbracket L + \llbracket T' \rrbracket L \\
\llbracket \tau.T \rrbracket L &\stackrel{\text{def}}{=} \tau. \llbracket T \rrbracket L & \llbracket [n=m]T \rrbracket L &\stackrel{\text{def}}{=} [n=m] \llbracket T \rrbracket L \\
\llbracket !T \rrbracket L &\stackrel{\text{def}}{=} ! \llbracket T \rrbracket L & \llbracket [n \neq m]T \rrbracket L &\stackrel{\text{def}}{=} [n \neq m] \llbracket T \rrbracket L \\
\llbracket n(\tilde{x}).T \rrbracket L &\stackrel{\text{def}}{=} n(\tilde{x}, y).(l)(\bar{y}\langle l \rangle.l \triangleright L \mid \llbracket T \rrbracket Ll), l \in \mathcal{N}_{ic} - \text{cau}(L), y \in \mathcal{N}_v - \text{fv}(T) \\
\llbracket \bar{n}\langle \tilde{m} \rangle.T \rrbracket L &\stackrel{\text{def}}{=} (b)\bar{n}\langle \tilde{m}, b \rangle.(l)(\bar{b}\langle l \rangle.l \triangleright L \mid \llbracket T \rrbracket Ll), l \in \mathcal{N}_{ic} - \text{cau}(L), b \in \mathcal{N} - \text{gn}(T) \\
\llbracket l : A \rrbracket L &\stackrel{\text{def}}{=} \llbracket A \rrbracket Ll, l \in \mathcal{N}_{ic} - \text{cau}(L), \text{cau}(A) \cap \text{cau}(L) = \emptyset \\
\llbracket A \mid B \rrbracket L &\stackrel{\text{def}}{=} \llbracket A \rrbracket L \mid \llbracket B \rrbracket L, \text{cau}(A) \cap \text{cau}(L) = \emptyset, \text{cau}(B) \cap \text{cau}(L) = \emptyset \\
\llbracket (c)A \rrbracket L &\stackrel{\text{def}}{=} (c) \llbracket A \rrbracket L, \text{cau}(A) \cap \text{cau}(L) = \emptyset
\end{aligned}$$

Fig. 3. Encoding from the local cause polyadic π to the polyadic π

So the encoding can perform communication actions without setting free new local cause names. The encoding $\llbracket A \rrbracket L$ from the local cause polyadic π -calculus to the polyadic π -calculus is presented in Fig. 3.

4.3 Property

The encoding has some important properties. The first one is about the process finiteness. Intuitively, there is no replication operator in the guarded finite wire processes, so that the encoding can respect process finiteness. In other words, finite local cause processes are mapped onto finite standard processes.

Lemma 3 (Finiteness-respecting). *If A is a finite process of the local cause polyadic π -calculus, then $\llbracket A \rrbracket L$ is a finite polyadic π -process.*

A new local cause is generated dynamically while an observable action is performed, moreover it is unique globally. The dynamic generative process is reflected in the encoding as the local cause substitution.

Lemma 4. *If $A \xrightarrow[\text{cau}(K);l_1]{\ell} A_1$, then there exists A_2, l_2 such that $A \xrightarrow[\text{cau}(K);l_2]{\ell} A_2$ and $\llbracket A_1 \rrbracket L \sim \llbracket A_2 \rrbracket L\{l_1/l_2\}$.*

The encoding is homomorphic but on input prefix and output prefix. An n -adic input prefix is encoded into an $n+1$ -adic input prefix, and the same to output prefixes. The n -adic input actions, n -adic output actions and τ actions are respectively correspondent to the $n+1$ -adic input actions, $n+1$ -adic output actions and τ actions on strong transitions. And the correspondent results can be extended to weak transitions routinely.

Lemma 5 (Operational Correspondence). *There is a precisely operational correspondence between the actions performed A and $\llbracket A \rrbracket L$.*

Now we can establish the full abstraction of the encoding. Unlike the involved proofs heavily depending on the cancelation lemmas in [San94] and [BS98], the soundness and completeness of the encoding can be proved more succinctly, because there are moderately finite local cause names in finite wire processes.

Theorem 1 (Soundness). *If $A \approx_{lc} B$ then $\llbracket A \rrbracket L \approx \llbracket B \rrbracket L$.*

Proof. It is sufficient to prove the relation $\mathcal{R} \stackrel{\text{def}}{=} \{(\llbracket A \rrbracket L, \llbracket B \rrbracket L) \mid A \approx_{lc} B\}$ is a weak bisimulation up-to context.

Theorem 2 (Completeness). *If $\llbracket A \rrbracket L \approx \llbracket B \rrbracket L$, then $A \approx_{lc} B$.*

Proof. We can prove that the relation $\mathcal{R} \stackrel{\text{def}}{=} \{(A, B) \mid \llbracket A \rrbracket L \approx \llbracket B \rrbracket L\}$ is a local cause bisimulation by combining Lemma 2, Lemma 4 with Lemma 5.

Corollary 1 (Full abstraction). *$A \approx_{lc} B$ iff $\llbracket A \rrbracket \approx \llbracket B \rrbracket$.*

Proof. Follows by Theorem 1 and Theorem 2.

We construct in this section a fully abstract encoding from the local cause polyadic π -calculus to the standard polyadic π -calculus. The local cause bisimulation between local cause processes is reduced to the original observation bisimulation between standard processes. And the finiteness-respecting property will play an important role in the next section.

5 Application

The full abstraction bridges the local cause bisimulation and the observation bisimulation. It has many potential applications [XD12]. For example, we can show the congruence properties of the local cause bisimulation like in [San94], and prove the local cause congruence in an axiomatic system for the observation congruence, which is not achieve in [San94, BS98]. Due to the space limitation, this section only applies it to the decidable result of the local cause bisimulation.

Algorithms for the decidable fragments is an important issue for process calculi. An important virtue of this encoding is that it respects process finiteness. The key effect of finiteness-respecting is that the local cause bisimulation on finite processes can be checked in some proof system of the observation bisimulation on finite processes. It is validated by the local cause bisimulation of CCS_{lc} and the observation bisimulation of the monadic π -calculus.

For CCS_{lc} , Kiehn developed a proof system for the restriction-free fragment of finite processes with the help of the left merge ($|'$) and the communication merge ($|^c$), which furnish CCS_{lc} with a expansion-like law [Kie93]. However, it is impossible to develop a finite axiomatic system or proof system for the local cause bisimulation without auxiliary operators, because there is no *expansion law* that is the fundamental for the axiomatization in the interleaving semantics [Mo190]. For the monadic π -calculus distinguishing names and name variables, a proof system, denoted by \mathcal{PS} , is proposed for the observation bisimulation on finite π -terms [Zhu09]. The observation bisimulation on finite π -processes can be proved in \mathcal{PS} with purely equational reasoning.

Proposition 1 ([Zhu09]). *Let P and Q be finite monadic π -processes. Then $P \approx Q$ iff $\mathcal{PS} \vdash \tau.P = \tau.Q$.*

Projecting Lemma 3 and Corollary 1 onto CCS_{lc} and the monadic π -calculus, the following properties can be achieved.

Corollary 2. *Let p be a CCS_{lc} -process, then $\llbracket p \rrbracket$ is a monadic π -process. Moreover, if p is finite then $\llbracket p \rrbracket$ is finite.*

Corollary 3. *Let p, q be CCS_{lc} -processes. Then $p \approx_{lc} q$ iff $\llbracket p \rrbracket \approx \llbracket q \rrbracket$.*

With Corollary 2 and Corollary 3, the local cause bisimulation on finite CCS_{lc} -processes is reduced to the observation bisimulation on finite monadic π -processes. Therefore, the local cause bisimulation on finite CCS_{lc} -processes can be proved in \mathcal{PS} with purely equational reasoning through the encoding.

Proposition 2. *$p \approx_{lc} q$ iff $\mathcal{PS} \vdash \tau.\llbracket p \rrbracket = \tau.\llbracket q \rrbracket$ for finite CCS_{lc} -processes p, q .*

We show in this section a new proof system for the local cause bisimulation on finite CCS_{lc} -processes. It is only a special case and can be strengthened for the local cause polyadic π -calculus. The crucial step is to develop a proof system for the observation bisimulation on finite processes of the polyadic π -calculus. Two ways can be taken into consideration. The first one is to encode the polyadic π -calculus into the monadic π -calculus. Milner has introduced the following abbreviations [Mil93]

$$\begin{aligned} a(x_1, \dots, x_n).T &\stackrel{\text{def}}{=} a(x).x(x_1) \dots x(x_n), \\ \bar{a}(x_1, \dots, x_n).T &\stackrel{\text{def}}{=} \bar{a}(c).\bar{c}(x_1) \dots \bar{c}(x_n). \end{aligned}$$

But the translation of the polyadic π -calculus to the monadic π -calculus is not sound with respect to the observation bisimulation. For instance $a(x_1, x_2)|b(y_1, y_2)$ is equivalent to $a(x_1, x_2).b(y_1, y_2) + b(y_1, y_2).a(x_1, x_2)$ in the polyadic π -calculus, but their translations in the monadic π -calculus are not equivalent. We also fail in attempts to propose a fully abstract encoding from the polyadic π -calculus to the monadic one. The second way is to present a proof system directly. And they are left as future work.

6 Conclusion

This paper bridges the true concurrency semantics and the interleaving semantics by analyzing the local cause semantics in the interleaving framework. The local cause bisimulation is reduced to the observation bisimulation of the polyadic π -calculus. A complete proof system, which does not depend on auxiliary operators, is proposed for the local cause bisimulation on finite CCS_{lc} -processes.

The approach to analyzing true concurrency semantics in the interleaving framework is originated by Sangiorgi [San94] to study the locality semantics [BCHK92], and also applied by Boreale and Sangiorgi [BS98] to investigate the causality semantics [DP92, Kie94]. Full abstractions are established from the locality semantics and the causality semantics to the interleaving semantics respectively. And the location bisimulation and the causality bisimulation are both

reduced to the observation bisimulation. However, the common drawback of their encoding schemes is that the replication operators are introduced in the *wire processes*, which model the locality mechanism and the causality mechanism. Finite processes are in general mapped onto processes which can do infinitely many actions. It prevents us from proving the location bisimulation and causality bisimulation on finite processes by some proof system for the observation bisimulation on finite processes.

The reasons why the replication operators are introduced in their encoding schemes are different. In [San94], the locality mechanism is modeled as a *wire process* of the form $!v.\bar{u}$. Location names u and v are respectively the images of an access path and an execution location. There are only two information in a wire process (an access path and an execution location), which are not sufficient because we must model the dependency of actions. Moreover, the bound on the locality-dependent actions is not decidable, because the constant definition performs actions with infinite access path. The infiniteness of modeling locality mechanism is caused by the preceding two reasons. And in [BS98], it is caused by another two reasons. Communication actions introduce causalities, and the bound of interaction is undecidable. Although the causality mechanism is modeled as a wire process of the form $!k. \prod_{k' \in K} !k'$ which takes sufficient causality information: the current marking k generated by the observable action, and all its causalities (k' s). To model interactions precisely, the replication operators are required.

By comparison, the local cause mechanism is closer to the causality mechanism than to the locality mechanism. The key difference between the former two is that interactions exchange causalities in the causality mechanism, but do not introduce local causes in the local cause mechanism. We can therefore model the local cause mechanism with the finite wire process.

There are some future directions in which the work could be extended.

By the full abstraction, the congruence properties of the local cause bisimulation will be investigated like in [San94]. And other applications remain to be discovered. An algorithm and a automated tool for supporting the fully abstract encoding scheme should be a continuation.

To strengthen the result, as stated in Section 5, it is a crucial step to develop a proof system or an axiomatic system for the observation bisimulation on the polyadic π -processes. We are inclined to that there is no fully abstract encoding from the polyadic π -calculus to the monadic one with respect to the observation bisimulation, and prefer to developing a proof system directly.

We have set about to extend the encoding scheme to improve the result of [San94], so that a similar proof system is proposed for the location bisimulation. And we also plan to extending this approach to other true concurrency semantics [DP99, BCH⁺08], and investigate its scalability and limitations.

We want to enrich the high order π -calculus [San92] with the local cause semantics to model the spartial-sensitive mobile cloud computing, a converging field of mobile computing and cloud computing. And whether or not the approach

to analyzing the local cause semantics is valid in the framework of the high order π -calculus still remains to be checked.

Acknowledgments. This work has been supported by NSFC (61033002, 61011140074, 61100053) and the Natural Science Foundation of Shanghai, China (10ZR1416800). The authors are grateful to Prof. Yuxi Fu, Dr. Huan Long and Dr. Xiaojuan Cai for their suggestions and discussions on this topic. They also would like to thank the anonymous referees for their comments and suggestions.

References

- [BCH⁺08] Boudol, G., Castellani, I., Hennessy, M., Nielsen, M., Winskel, G.: Twenty Years on: Reflections on the CEDISYS Project. Combining True Concurrency with Process Algebra. In: Degano, P., De Nicola, R., Meseguer, J. (eds.) Montanari Festschrift. LNCS, vol. 5065, pp. 757–777. Springer, Heidelberg (2008)
- [BCHK92] Boudol, G., Castellani, I., Hennessy, M., Kiehn, A.: A Theory of Processes with Localities. In: Cleaveland, W.R. (ed.) CONCUR 1992. LNCS, vol. 630, pp. 108–122. Springer, Heidelberg (1992)
- [BS98] Boreale, M., Sangiorgi, D.: A fully abstract semantics for causality in the π -calculus. *Acta Inf.* 35(5), 353–400 (1998)
- [DNMV90] De Nicola, R., Mantanari, U., Vaandrager, F.: Back and Forth Bisimulations. In: Baeten, J.C.M., Klop, J.W. (eds.) CONCUR 1990. LNCS, vol. 458, pp. 152–165. Springer, Heidelberg (1990)
- [DP92] Degano, P., Priami, C.: Proved Trees. In: Kuich, W. (ed.) ICALP 1992. LNCS, vol. 623, pp. 629–640. Springer, Heidelberg (1992)
- [DP99] Degano, P., Priami, C.: Non-interleaving semantics for mobile processes. *Theoretical Computer Science* 216, 237–270 (1999)
- [FL10] Fu, Y., Lu, H.: On the expressiveness of interaction. *Theoretical Computer Science* 411, 1387–1451 (2010)
- [FZ11] Fu, Y., Zhu, H.: The name-passing calculus (2011) (submitted)
- [Kie93] Kiehn, A.: Proof Systems for Cause Based Equivalences. In: Borzyszkowski, A.M., Sokolowski, S. (eds.) MFCS 1993. LNCS, vol. 711, pp. 547–556. Springer, Heidelberg (1993)
- [Kie94] Kiehn, A.: Comparing locality and causality based equivalences. *Acta Informatica* 31(8), 697–718 (1994)
- [Mil89] Milner, R.: *Communication and Concurrency*. Prentice Hall (1989)
- [Mil93] Milner, R.: The polyadic π -calculus: a tutorial. In: *Proceedings of the 1991 Marktobendorf Summer School on Logic and Algebra of Specification*. NATO ASI, Series F, Springer (1993)
- [Mol90] Moller, F.: The nonexistence of finite axiomatisations for ccs congruences. In: *LICS 1990*, pp. 142–153 (1990)
- [MPW92] Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes. *Information and Computation* 100, 1–40, (Part I), 41–77 (Part II) (1992)
- [San92] Sangiorgi, D.: *Expressing Mobility in Process Algebras: First Order and Higher Order Paradigm*. PhD thesis, Department of Computer Science, University of Edinburgh (1992)

- [San94] Sangiorgi, D.: Locality and True-Concurrency in Calculi for Mobile Processes. In: Hagiya, M., Mitchell, J.C. (eds.) TACS 1994. LNCS, vol. 789, pp. 405–424. Springer, Heidelberg (1994)
- [XD12] Xue, J., Dong, X.: A fully abstract view for local cause semantics (2012) full version, <http://basics.sjtu.edu.cn/~jianxin/>
- [Zhu09] Zhu, H.: Model Independent Theory of Mobile Calculi. PhD thesis, Shanghai Jiao Tong University (2009)

Efficiency Considerations in Policy Based Management in Resource Constrained Devices

Jignesh Kakkad and Nandan Parameswaran

School of Computer Science and Engineering, The University of New South Wales
Sydney, NSW 2052, Australia

{jmkka292, paramesh}@cse.unsw.edu.au

Abstract. Policies are being widely used in a variety of applications such as healthcare, disaster management and mobile networking. In this paper, we show how policies can be used to manage the resources effectively and in a user friendly way. Further, we advocate that while an agent is required to obey a given policy requirement, there are situations where the agent may consider the possibility of violating the policy (policy deviation) such as in an emergency or during a disaster. Our simulation results show that sometimes policy violations can be beneficial to the community of (application) agents and such violations must be managed *carefully*.

Keywords: Policy, Mobile Agent, Resource Management, Policy Violation.

1 Introduction

Management involves the process of controlling entities in an organization and it often involves stipulating policies and enforcing them. Policies can be represented in their simplest form by a set of rules which define the behavior of objects involved in a system situated in a given environment and they have been used effectively to achieve flexibility in complex distributed systems [2], [3], [9], [10], [11]. There exist a number of policy languages such as Rei, Ponder2 [4] and XACML in which policies can be written. In this paper, we investigate use of policies for resource management, and propose a measure of policy violation by relating it to the overall performance of the agent community.

The rest of the paper is organized as follows. In Section 2, we discuss policies with an example. Section 3 discusses policy based resource management in a mobile device. In Section 4, we present policy violation. In Section 5, we discuss measuring policy performance. Sections 6 and 7 present related work and conclusion, respectively.

2 Policy

Currently, mobile phone users are subjected to policies proposed by various agencies such as telecom operators (Service provider), phone manufacturers, service providers

(through their terms and conditions, for example, from Service provider) and government agencies. An example of polices is as follows:

Rule 1

*if (current offer = “pre-paid cap+” && recharge amount == \$30) then
current balance += current Data;
current limit += 400MB;*

Rule 2

*if (current offer = “Telstra long life” && recharge amount == \$20) then
current expiry += 60 days;*

Apart from mobile usage policies, there are terms and conditions (T&C) imposed by Service providers as well.

3 Policy Based Resource Management in a Mobile Device

Fig. 1 shows a simple architecture for policy based resource management consisting of a set of application agents and a set of resources. Access to resources are managed by a resource monitoring agent (MA). The monitoring agent maintains a set of policy rules and the history of the states of the resources. A request from an agent consists of an action *a* to be performed on behalf of the agent. When an application agent puts in a request, the monitoring agent evaluates the request using the states of available resources and policy rules and forwards the request to the resource operator. The resource operator executes the action *a* and the resulting new states of the involved resources are stored for future use. (More resources may be added if the action is modeled with more details of resources.)

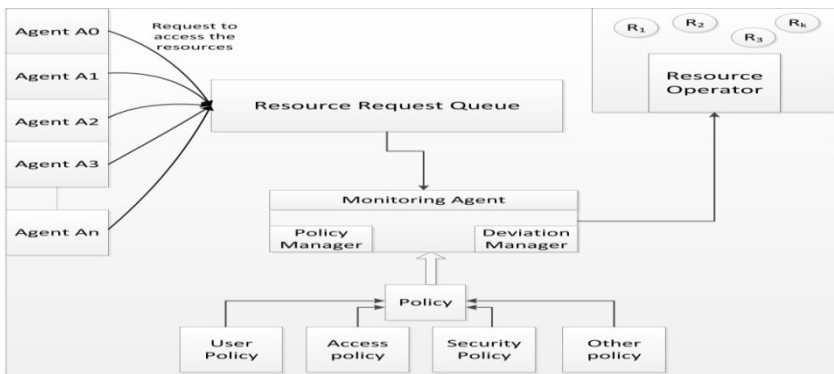


Fig. 1. An architecture for policy based resource management in a mobile phone

The monitoring agent has two sub parts: the Policy Manager which is responsible for the overall execution of the requested action; and the Deviation Manager which manages the agent request when the execution of these requests may result in the violation of policy conditions (discussed in the next section).

3.1 Policy Model

A policy in a multiagent system dictates how a set of resources must be operated by the agents in the world. As the agents perform operations on the resources, the resources change from one state s_i to s_j . (We often refer to the aggregate state of the resources as the world state.)

3.2 Option Graph

We model the world as a synchronous finite state machine (fsm) where the fsm goes from one state s_i to another s_j at the time when a clock occurs as defined by the transition function δ . Let S be the set of states, Γ be the set of actions an agent can perform, and Λ be the set of external events. Then, $\delta: S \times (\Gamma \cup \Lambda) \rightarrow S$.

We let η to denote a null action, which we will use to characterize a situation where an agent chooses not to do any action. Thus, $\eta \in \Gamma$, and $\delta(s_i, \eta) = s_i$. That is, a null action does not change the state of the world.

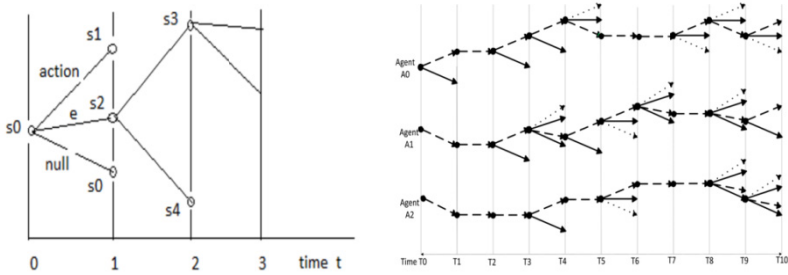


Fig. 2. Option graph and Option graph for three agents with policy. Options (dashed and dotted arrows) define the policy.

An option graph is a directed acyclic graph where the nodes of the graph represent the states of the world and edges denote transitions from one state to another which occur when actions are performed (See Fig. 2). Above is the graphical representation of the options available at each instant of time.

In the option graph above, we have shown how the (state of the) world changes as time progresses. Time moves discretely by one unit at a time, starting from $t=0$. At $t=0$, the world is at state s_0 , and there are three options:

- a) The first option has an action that the agent may execute intentionally. In this case, the agent is responsible for making the world transit from state s_i to state s_j .
- b) The second option has an external event that can spontaneously occur which will affect the state of the world making it transit from s_i to s_j .

- c) The third option is a null action which represents the situation where if the agent chooses to avoid executing any action time $t=0$, then the world does not change its state at time $t=1$.

Apart from the states enumerated in the option diagram, there exist other states that are known as error states which we have not shown. The world may enter into the error states when actions from Γ are not successfully completed, when agents perform actions that are not in Γ , or if the agent performs an action from Γ when an event from Λ has already started occurring. In this paper, we assume that the agent does not choose to perform an action $\beta \in \Gamma$ action if there occurs an event $\lambda \in \Lambda$ that can take the world to a next state valid state $s \in S$. Thus, in our model, an external event gets a higher priority to affect the world than the agent actions.

Option Graph for multiple agents

Fig. 2 also shows an option graph for three agents A_1 , A_2 , and A_3 where each agent A_i has its own option graph. The states in the option graph of A_1 , for example, are defined by the resources owned exclusively by A_1 and the resources it shares with the other agents A_2 , and A_3 . Thus, the three agents can all work concurrently as long as they do not operate on the shared resource. Operation on shared resources require specific policies for accessing and using them, but to keep our discussion simple, we will assume that the underlying action execution mechanism permits only one agent to operate on any resource at any time. A consequence of this assumption is that when an agent A_i changes the state of a shared resource, it may affect the *next* states of other agents that are currently operating on non-shared resources. (We will elaborate on this later in the next section.)

3.3 Policy Graph

A policy graph is a sub-graph of an option graph. The option graph in Fig. 2 defines a policy (let us call it) P . The policy defines the permitted options at any given state for an agent according to the policy. The options shown as dashed or dotted arrow at each state signifies the fact that these options are permitted by the policy. The options shown as a black arrow are not permitted by the policy and yet may be possible to execute by an agent. An agent can decide to choose any available option depending on what next state it wants. A policy obeying agent is the one that always chooses an option that is permitted by the policy (dashed line). If an agent chooses an option that is not permitted by the policy, then the agent is said to have violated the policy. (Dotted lines show options permitted by the policy, but not selected by the agent.)

Obeying policy and shared resources

When an agent operates on a shared resource, it can in general affect the next states of other agents. A standard solution to this problem is to permit only agent A_i at a time to “lock” the resource and start using it, while other agents wait for their turn. In a policy based model, we abstract out the details of how a shared resource is used,

and merely state that at any state the policy defines the future states that are known to all agents in the community, and when an agent selects an option, it also inherits the consequences of selecting that option. However, when an option not sanctioned by the policy is selected, it may affect the other agents' current state thus affecting their next options.

Fig. 2 shows the traces of three agents moving from time $t = 0$ towards their future goal states (goals states are not shown in the figure). When an agent executes an action, there are two cases that are considered. Let agent A_1 select and execute an action β_1 and agent A_2 select and execute an action β_2 . (The actions may or may not have been permitted by the policy.)

Case 1: The execution of the action β_1 by agent A_1 will not affect the execution of the action β_2 by the agent A_2 . In this scenario, both the agents can execute their selected actions without interfering with each other.

Case 2: The execution of the action β_1 by agent A_1 will affect the execution of the action β_2 by the agent A_2 . In this scenario (as we explained earlier), if the agents A_1 and A_2 both follow the policy (that is, chose the options marked dashed), then the execution of the actions β_1 and β_2 is said not have interfered with each other (according to the way the policy is defined). However, if the agents chose to violate the policy then noninterference may not be guaranteed.

3.4 Policy Semantics

We can formalize the notion of policy using the policy graph. As an example, consider Rule 1 above. Let us define the world state as a 4-tuple $\langle o, c, d, b \rangle$, where o = currentOffer, c = rechargeAmount, and d = currentData, and b = currentBalance. Let the current state be s_i where $s_i = \langle \text{"prepaidCap+"}, 0, 0, 0 \rangle$. Then upon executing action "pay \$30", the state s_i changes to s_{i+1} where $s_{i+1} = \langle \text{"prepaidCap+"}, \$30.00, 400\text{MB}, 400\text{MB} \rangle$. Thus, in the option diagram, we insert the following transition at s_i : $s_i \xrightarrow{\text{pay } \$30} s_{i+1}$, where s_i and s_{i+1} are defined as above.

Complex Policies

Policies can be simple such as the ones shown above, or more complex as shown below:

Rule

Let A be the application agent that manages the email account and let the policies it needs to follow are as shown below:

- a. Notify the service provider if the Email/SMS Bill email address changes;
- b. Contact the service provider if the Email/SMS Bill has not been received; and
- c. Keep the email/sms account secure to protect the privacy of your credit information contained in the Email/SMS Bill.

Translating such complex policies poses challenges in the sense that we need to model the underlying domain adequately. The policy rule (a) above can be depicted by a policy (sub) graph with the following characteristics:

Every node of the policy graph has a Λ type option in addition to any other options. The event on the Λ type option is “address changed”. If the previous transition was due to the “address changed” event, then there is a next Γ type option with the label “notify (new-address)”.

An agent that obeys the above policy should perform the following behavior: If the previous transition was due to the Λ type event “address changed”, then the agent chooses the Γ type option action “notify (new-address)”. The policy semantics for the rule (b) can similarly be given as follows: Each node in the policy graph will have at least one Γ type option included: “contact service provider”. Let the current node be s_i . If none of the last k transitions were triggered by the occurrence of the Λ type event “arrival of Email/SMS bill”, then the agent chooses the Γ type option “contact service provider”, and executes the action.

The policy graph for (c) requires that as soon as the agent receives an email/sms account, then the agent will choose the option of performing an operation on the confidential account details (such as encrypting them, etc.), and all the future states will contain the encrypted form of the confidential account details. Further, each future state may have a Γ or Λ type action that may for example decrypt the account details thus threatening the confidential nature of the account details.

4 Policy Violation

Policy violation occurs when agents choose options not sanctioned by policies. It may be permitted in a world that is inhabited by a single agent. However, in a world where multiple agents coexist, policy violations result in unpredictable future states of the world. In certain situations, however, it may become necessary for an agent to violate the policy in order to maximize its chances of achieving its goals. In such cases, policy violations (that is, deviation from policy based behavior) must be performed carefully and managed. Deviation Manager (DM) in Fig. 1 above monitors such violations and decides whether at any time policy violation may be permitted.

Dependent agents and Degree of violation

In a generalized scenario, each agent depends on other agents to achieve its goals, the degree of dependency varying according to agents and situations. While there are no simple ways to predict the consequences of a violation, an estimate of the consequences is still necessary to manage the deviations which we discuss in the following section.

5 Measuring Policy Performance

To measure the performance of a policy, we consider a community of agents and consider its option graph. The edges in the option graph have a numerical weight

which indicates the score an agent gets if it chooses that option in its next move. Thus, the total score up to the current moment t is the sum of all the weights w_i of the options that were chosen by the agent so far; that is,

$$score\ s = \sum_{i=0}^t w_i .$$

In the simulation below, we consider three agents, and the state consisting of three resources: $\langle r_1, r_2, r_3 \rangle$. A policy is implemented by choosing a DAG that is a sub-graph of the option graph restricting the number of options at any state in the option graph to not more than some maximum value (5 in our simulation). The maximum number of steps along the time axis is limited to t_{max} where $t_{max} = 50$.

Table 1 shows our initial measurement with the points scored by each agent for each policy P_1 , P_2 and P_3 . Typically, an agent will continue to choose the next highest score option until it reaches the state where the number of options is 0; that is, from this state the agent cannot proceed further as no more future states exist. As the agent chooses an option and executes the corresponding action, resources are consumed in the action execution, and thus the resources go to another state.

Table 1. Points scored by each agent

Policy	Agent's score	
	All resources are shared	No resources are shared
P_1	519	1151
P_2	779	1443
P_3	280	479

When no resources are shared, agents do not worry about the policies, and they choose any option that gives them the highest score. However, when resources are shared, agents need to follow the policy if they do not want to affect each other's behaviors. Thus, we see that in Table 1, the scores for individual agents are highest when they ignore other agents, but they are lower when the agents are required to follow the policies (in this example). While following a policy, each agent can only choose the option with the highest score from the options permitted by the policy. Incidentally, among the three policies we used, policy P_2 appears to be a better policy for this small community since it helps the community achieve the highest score.

Fig. 3 shows the agent community behavior that is, the average score of each agent over the period of time when each agent follows the policy P_1 .

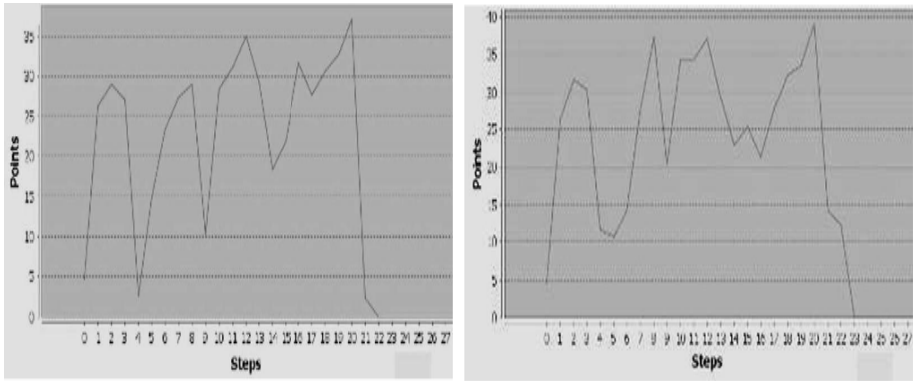


Fig. 3. (a) and (b): Community behavior when all agents follow policy(a) and violate policy(b)

Fig. 3 also shows the community behavior when agents violate the policy. From above figure, we see that when the agents violate the policy, the overall community score is different from the score where the agents do not violate the policy. Fig. 4 shows the difference in the behavior when agents follow and agents violate the policy.

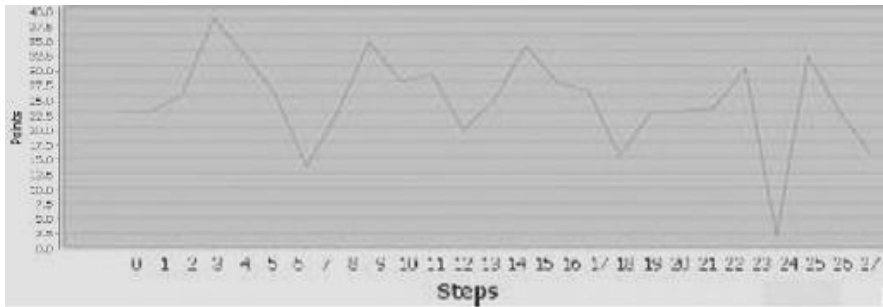
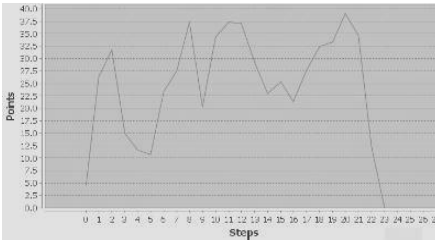


Fig. 4. Effect of Policy violation on agent community

The difference between these two graphs plays a vital role in deciding whether to permit a policy violation or not. Based on this, we propose a management policy for the monitoring agent (Fig. 1) since all requests from the application agents to use any available resource go through the monitoring agent. When a request from any agent to violate the policy comes to the MA, the MA computes the consequence (score difference) of the policy violation at the current state from Fig. 4. If the score difference is more than the specified threshold value, then MA rejects the request for policy violation, else the MA approves the violation.

The threshold value is an important parameter to decide in policy violations. This parameter is configurable and can be set by either the mobile phone user or service provider in their policy. If a value of the threshold is high, the MA is said to be

lenient. On the other hand, if the value of the threshold is too low, the MA is said to be strict. Following is an example of a policy violation management rule:



Rule: *if* (current request = “Policy violation”) **then**
if (get_actual_effect(current step) > Threshold) **then**
do (“reject a request for policy violation”); **else do** (“permit the request for policy violation”);

Fig. 5. Community behavior when threshold value is 15

Fig. 5 shows the behavior of the community when we consider the management policy rule above to manage policy violation and Table 2 shows the performance of the agent community for a different threshold value. From the table below, we observe that policy violation is not always a “bad thing” after all. Sometimes, the community seems to perform better when agents are permitted to violate policies. This indicates that the current policy is not a “good” policy.

Table 2. Points gained for different threshold values

Threshold value	Agent community points
0	529
2	535
5	541
10	561
15	583

6 Related Work

Policies can be expressed more formally in languages such as Ponder2 [4] and Rei. Twidle et al [7] have proposed a new approach to monitor the behavior of a dynamic system. The normal event-condition-action (ECA) rules are not able to monitor the behavior of the system. These rules direct a system to behave according to a given situation.

Al Sum et al [6] have proposed a framework for dynamic policy based management of the resources in a mobile device. Their proposed framework manages sensitive resources such as network resources which can cost money or system resource such as mobile phone battery which can affect phone performance. A policy based solution in an educational application has been proposed by D Goel et al in [1]. They propose a solution to access various resources such as class rooms using a set of policy rules. Information such as current location, time and role of the user

are used to grant access to use any class room. People use online social networking websites such as Facebook, Twitter, etc. to keep in touch with their family and friends. A number of solutions have been proposed to manage the information using policies [2] [3].

In pervasive computing scenarios, it is important for the administrator to monitor policy deviation. In [7] example of health care system, the authors have mentioned about deviation manager which keeps monitoring the request against policy. It grants the access to treat any patient based on policy rules and role of the user who is requesting the access. Ahmed AlSum et al [6] suggest a dynamic policy approach for monitoring all resources in a mobile device. Samir Al-Khayatt et al [5] propose a solution for detecting policy violation. As per their suggested approach, they use an automated tool to monitor the internet usage by all employees in the office which detects violations whenever they occur.

7 Conclusion

We have in this paper given option graph based semantics for policies, and sketched out a scheme for measuring the performance of policies in a multi agent framework where the application programs on a mobile device are viewed as resource hungry autonomous agents. Using this formulation, we have shown how two policies can be compared in terms of their performance metrics. We also have discussed the significance of policy violations, and shown how the consequences of violations can be quantified. Based on this violation metric, we argued that it is not always bad to violate policies, and we proposed how policy management violation rules can be written where a resource monitoring agent MA can decide at any time whether to allow an application program to violate a policy.

References

- [1] Goel, D., Kher, E., Joag, S., Mujumdar, V., Griss, M., Dey, A.K.: Context-Aware Authentication Framework. In: First International ICST Conference on Mobile Computing, Applications, and Services. Springer Pub. Co., San Diego (2010)
- [2] Shehab, M., Cheek, G., Touati, H., Squicciarini, A.C., Pau-Chen, C.: User Centric Policy Management in Online Social Networks. In: 2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), pp. 9–13 (2010)
- [3] Kodeswaran, P., Viegas, E.: A Policy Based Infrastructure for Social Data Access with Privacy Guarantees. In: 2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), pp. 14–17 (2010)
- [4] Damianou, N., Dulay, N., Lupu, E., Sloman, M.: The Ponder Policy Specification Language. In: Sloman, M., Lobo, J., Lupu, E.C. (eds.) POLICY 2001. LNCS, vol. 1995, pp. 18–38. Springer, Heidelberg (2001)
- [5] Al-Khayatt, S., Neale, R.: Automated detection of Internet usage policy violation. In: ACS/IEEE International Conference on Computer Systems and Applications, pp. 507–510 (2001)

- [6] AlSum, A., Abdel-Hamid, A., Abdel-Aziem, M.: Application-specific dynamic policy rules (ASDPR) for J2ME. In: IEEE/ACS International Conference on Computer Systems and Applications, AICCSA 2009, pp. 512–516 (2009)
- [7] Twidle, K., Marinovic, S., Dulay, N.: Teleo-Reactive Policies in Ponder2. In: 2010 IEEE International Symposium on Policies for Distributed Systems and Networks (POLICY), pp. 57–60 (2010)
- [8] Finnis, J., Saigal, N., Iamnitchi, A., Ligatti, J.: A location-based policy-specification language for mobile devices. *Pervasive and Mobile Computing* (2010)
- [9] Talaei-Khoei, A., Bleistein, S., Ray, P., Parameswaran, N.: P-CARD: Policy-Based Contextual Awareness Realization for Disasters. In: 2010 43rd Hawaii International Conference on System Sciences (HICSS), pp. 1–10 (2011)
- [10] Wang, F., Turner, K.J.: Towards personalised home care systems. In: Proceedings of the 1st International Conference on Pervasive Technologies Related to Assistive Environments, pp. 1–7. ACM, Athens (2008)
- [11] Talaei-Khoei, A., Bleistein, S., Ray, P., Parameswaran, N.: P-CARD: Policy-Based Contextual Awareness Realization for Disasters. In: 2010 43rd Hawaii International Conference on System Sciences (HICSS), p. 110 (2010)

Agent Based Quality Management Middleware for Context-Aware Pervasive Applications*

Di Zheng¹, Jun Wang², and Ke-rong Ben¹

¹ Department of Computer Science, Naval University of Engineering,
Wuhan, Hubei, China 430033

² Key Research Lab, Wuhan Air force Radar Institute,
Wuhan, Hubei, China 430019

Abstract. With the rapid development of the information technology, it is inevitable that the distributed mobile computing will evolve to the pervasive computing gradually whose final goal is fusing the information space composed of computers with the physical space in which the people are working and living in. To achieve this goal, one of the problems is how to continuously monitor/capture and interpret the environment related information efficiently to assure high context awareness. Many attentions have been paid to the research of the context-aware pervasive applications. However, most of them just use the raw context directly or take the Quality of Context (QoC) into account in just one or two aspect. Therefore, we propose a agent based quality management middleware to support QoC management through various layers. By the agents, we can configure different strategies to refinery raw context, discard duplicate and inconsistent context so as to protect and provide QoS-enriched context information of users to context-aware applications and services.

Keywords: QoC, middleware, Context-aware, Pervasive, Agent.

1 Introduction

With the technical evolution of wireless networks, mobile and sensor technology, the vision of pervasive computing is becoming a reality. The paradigm for pervasive computing aims at enabling people to contact anyone at anytime and anywhere in a convenient way. So, context-awareness has become one of the core technologies in pervasive computing environment gradually and been considered as the indispensable function for pervasive applications[1]. For recent years, many research efforts have been done for gathering, processing, providing, and using context information [2]. In contrast, existing pervasive systems rarely pay attention to the Quality of Context information (QoC) used for making context-aware decisions and for executing context-aware applications. Buchholz et al. [3] has been the first ones to define QoC “as any information describing the quality of information that is used as context”.

* This work was funded by National Natural Science Foundation of China (2011, No.61100041).

Furthermore, context information can be characterized by certain well-defined QoC aspects, such as accuracy, precision, completeness, security, and up-to-dateness [4].

Despite its importance few works [4~8] have proposed different QoC measuring methods. Moreover, these studies evaluate quality only on some aspects, i.e. they do not consider complex and comprehensive applications. Comparatively, pervasive environments have a wider range of applications such as performing collaborative work. Hence, complex data structures are used to gather data from sources ranging from the simple sensors to user interfaces and applications in mobile devices.

In our previous works, we have put forward a middleware for the context-aware component-based applications so as to make these applications can be adapted more easily than traditional applications by simply adding and deleting components [9,10,11].

Based on this middleware, we use the agents to support the configuration of different quality factors. Furthermore the strategies used by these agents can help us evaluate raw context, discard duplicate and inconsistent context so as to protect and provide QoS-enriched context information of users to context-aware applications and services.

2 Middleware Based QoS Management

2.1 Architecture of the Context-Aware Middleware

As our previous architecture depicted in figure 1 and figure 2 [9,10,11], the core provides the fundamental platform-independent services for the management of the component/service based applications such as component deployment, service discovery, service combination and so on.

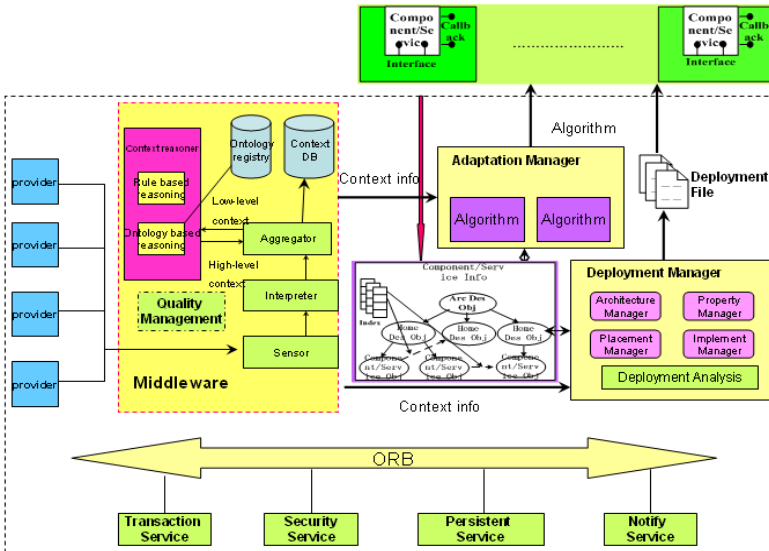


Fig. 1. Architecture of Context-aware Middleware

Context Manager is responsible for sensing and capturing context information and changes, providing access to context information (pull) and notifying context changes (push) to the Adaptation Manager. The Context Manager is also responsible for storing user needs and preferences on application services.

Adaptation Manager is responsible for reasoning on the impact of context changes on the application(s), and for planning and selecting the application variant or the device configuration that best fits the current context. As part of reasoning, the Adaptation Manager needs to assess the utility of these variants in the current context. The Adaptation Manager produces dynamically a model of the application variant that best fits the context.

Deployment manager is responsible for coordinating the initial instantiation of an application and the reconfiguration of an application or a device. When reconfiguring an application, the configurator proceeds according to the configuration template for the variant selected by the Adaptation Manager

2.2 Agent Based Quality Management of Context

As depicted in figure 2, we divide the entire context-aware process into five layers including sensor layer, retriever layer, deal layer, distribution layer and application layer. Different from existing methods, we pay attention to the quality management of context through all these layers.

Firstly, in the sensor layer, we set agents supporting different threshold to implement auto context discarding. The agents can be configured with one or more threshold, by this way we can reduce the number of the raw contexts.

Secondly, in the retriever layer, we use the context quality index to describe the quality of the contexts. All these factors are decided by user's demand and they may be different at all in various applications. Furthermore, we use the agents to complete the computation of these factors. Therefore, we can produce customize quality plan with different demands. The refined contexts from this layer will provide input for the higher layer.

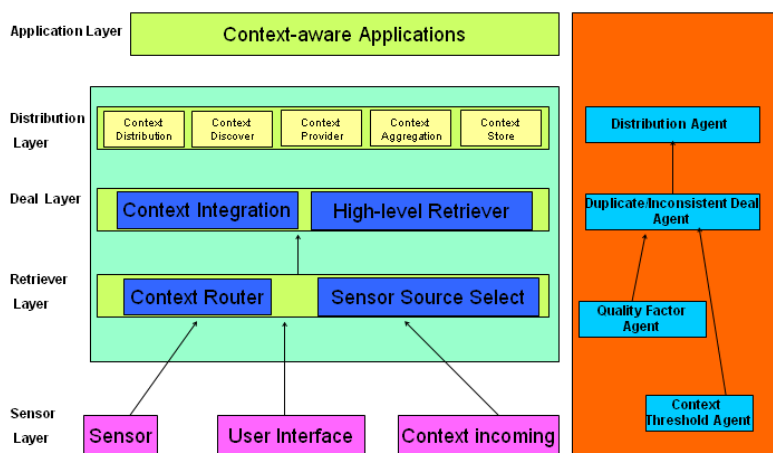


Fig. 2. Agent based Quality Management of Context

Then in the deal layer, we expand existing context dealing process with the duplicate context discarding and inconsistent context discarding to provide more accurate and more efficient contexts for the applications. All the algorithms are configured in the duplicate and inconsistent dealing agent.

At last, in the distribution layer and application layer, we expand traditional context-aware component/service adaptation/deployment algorithms with the help of the incoming contexts. This process is also helped by the distribution agent.

2.3 Ontology Based Quality Index

As depicted in figure3, we use quality threshold agent to complete the raw context discarding. For example, we can configure the agent as follows:

1. The sensor with lower than 50% accuracy will be ignored for several time.
2. The sensor with lower than 50% completeness will be ignored for several time.
3. The sensor with lower than 20% certainty will be ignored for several time.

These rules are used in the environments existing lots of sensors and they can help us discarding useless context information. The entire procession is adaptive and it depends on the information computed by the quality factor agent.

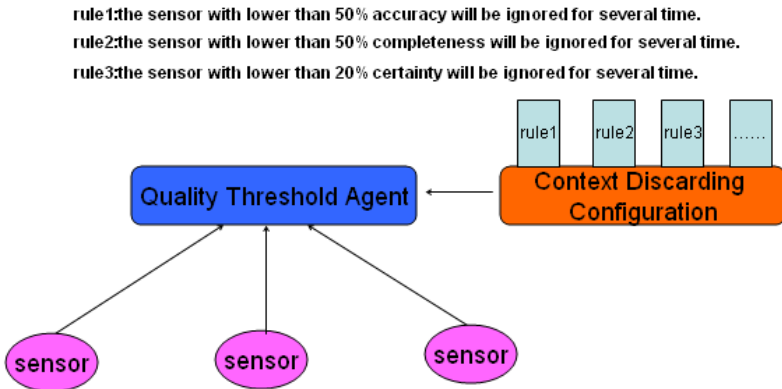


Fig. 3. Quality Threshold Agent

As depicted in figure 4, we use quality factor agent with different indicators to represent the quality of the context such as security, precision, resolution and so on. Users can produce different quality configuration plans with different indicators as they wish. Furthermore, they can define their own index by combing different indicators.

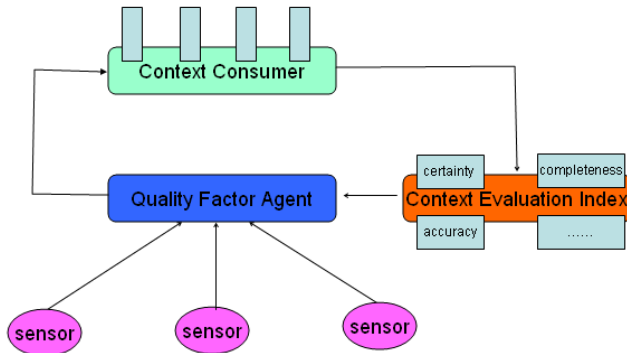


Fig. 4. Quality Factor Agent

We use different functions to define the indicators such as security, precision, resolution, freshness, certainty and completeness. For example, we use certainty as the probability of the accuracy of the context. As we all know, the context comes from different kinds of sources and some of them more sensitive and useful than the others. So when we have similar contexts from different sensors, we should choose the right context with the help of certainty. We define certainty as follows:

$$C(CxtObj) = \begin{cases} CO(CxtObj) \times \frac{NumberofAnsweredRequest + 1}{NumberofRequest + 1} \\ : \text{if } F(CxtObj) \neq 0 \text{ and } CxtObj \neq null \\ CO(CxtObj) \times \frac{NumberofAnsweredRequest}{NumberofRequest + 1} : \text{otherwise} \end{cases}$$

We use *NumberofRequest* to represent the number of the requests as well as the *NumberofAnsweredRequest* representing the number of the answer requests. The ratio of them will describe the communication of the context with the help of the indicator Completeness. If the ratio is much lower than 1, then it shows there are many unknown problems between the interaction of the sensors and the middleware. So we can say the certainty of the context source is not good.

2.4 Detection and Discarding of Duplicate Context

As depicted in figure 5, we configure the duplicate/Inconsistency deal agent with different algorithms. Firstly the agent gets the identifier of the newly arrived context and checks whether there is any context in the existing data representing the same entity or not. If we do not find any context having the same identifier, we will check the name/value pairs further. In our system, every context has context ID, context name/value pairs. So we define duplicate contexts as the contexts have the same identifier, or the same name/value pairs.

If there are some contexts having the same identifier or the same name/value pairs, we will check the sources of context. If they have different sources then some errors may occur and we should check the gathering of the contexts. If these two context objects are from the same source then we check the time when these context objects are generated. If they have the same timestamp then it means that they are the exact

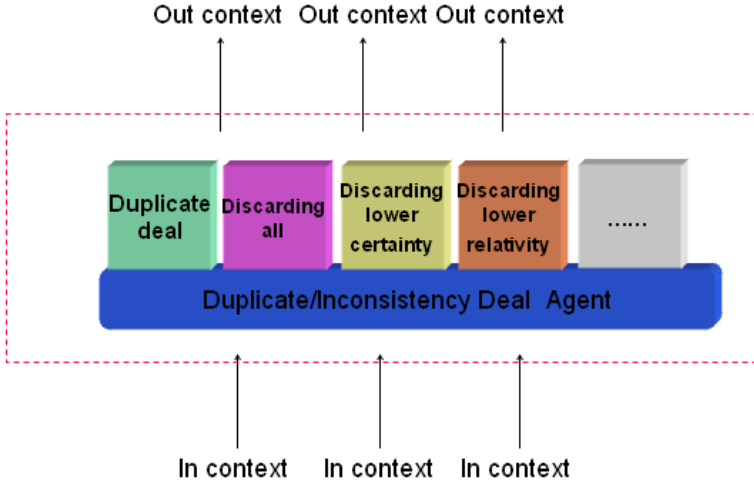


Fig. 5. Duplicate/Inconsistency deal agent

duplicate of each other and anyone of them can be discarded as well as keeping the other one. If they have the different timestamps it means that these are the duplicate contexts and will be discarded.

After duplicate context dealing we need inconsistent context dealing including matching of name/value pairs and quality-aware tuple. Consistency constraints on contexts can be generic (e.g., “nobody could be in two different rooms at the same time”) or application specific (e.g., “any goods in the warehouse should have a check-in record before its check-out record”).

After detecting inconsistent contexts we should use the algorithms as follows to discard the conflicted contexts.

Algorithm 1. Discarding all inconsistent context instances

INPUT: New arrived context

1. *get the new instance of context in the queue of matching patterns pat_que*
2. *To all the patterns pat₁, pat₂, ..., pat_n*
3. *In the pat’s trigger tgr*
4. *if exists ins₁ in pat_que₁, ins₂ in pat_que₂, ..., and ins_n in pat_que_n and tgr satisfy the constraint of ins₁, ins₂, ..., ins_n*
5. *then*
6. *if the constraint of tgr is satisfied*
7. *then*
8. *We get inconsistency*
9. ***delete all the inconsistent context instances***
10. *add the remaining instances to the repository*
11. *end if*
12. *end if*

Algorithm 1 is deleting all the inconsistent context instances. However, the occurrence of many conflicts is due to the entrance of the new incoming context instance. So we get algorithm 2, discarding the newest context instance. However, in some examples the newest context may be the right one, so we get algorithm 3 to discard the inconsistent context by the help of the field of *certainty*. In algorithm 3, we need compare the certainty of all the context instances and this may add the overall exhaustion of the algorithm. Therefore, to different kinds of contexts, we pay attention to the frequency of the contexts for the context having higher frequency may be right. So in algorithm 4, we compare the relativity of different contexts and discard the ones having the lower relativity.

2.5 QoC-Aware Component/Service Selection

As depicted in figure6, we can define context-aware criteria that link the two sides of context and service nonfunctional constraints in the distribution agent. Context-aware criteria consist of a number of criteria that are initialized from the meta data of the correct service category. For example, we can use one of the QoC criteria such as precision, freshness, certainty, completeness and so on. We can also use two or more criteria to complete the selection. All the selection is managed by the autonomic manager according to the configuration of users.

All the procession is controlled by the Autonomic Management Module. When deployed, every service will have several context configuration constraints which imply the services pay more attention to which contexts. We can also set the thresholds for the constraints. If a service's contexts are under thresholds, then the service cannot be selected.

Initially, the Autonomic manager of our context-aware middleware discovers the service providers that match the user's requirements by given service discover methods. Then the Autonomic manager compares QoC of different services by using the methods choose by users and selects one of the suitable providers and creates a binding to it. During service execution, when the Autonomic manager detects broken

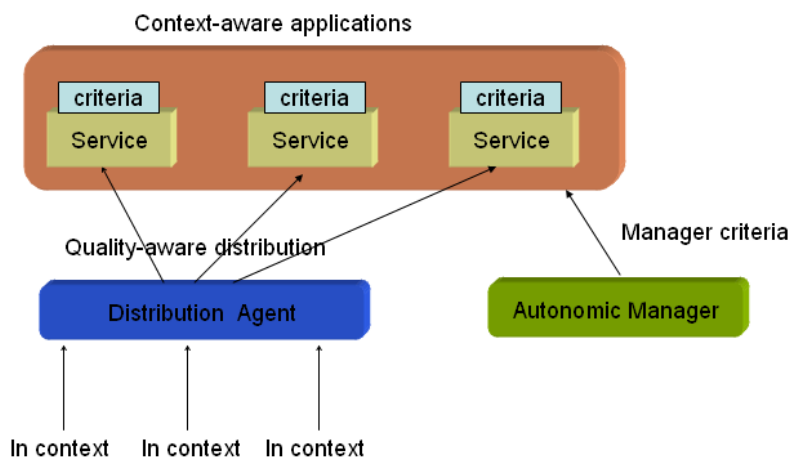


Fig. 6. Distribution agent

service bindings (e.g. the bound service provider becomes unavailable), it will repair them by discovering and binding to an alternative provider.

Besides the initial binding configuration and repair facilities, the Autonomic manager can be configured to continually optimize service selection during runtime. Furthermore, a service provider that was optimal in a certain context may be automatically replaced by a different service provider, which becomes the optimal choice in a new execution context. The context filter can be specified to trigger the dependency optimization each time a new service provider with the required specification becomes available. And we will take more complex replacement methods into account in the future researches.

3 Performance Results

We are deploying the proposed framework in a university building in order to provide context-aware services to the users, such as context-based access control and context-aware control of heating and lighting, among others. Afterwards, the gathered information was transmitted to a server running a CIS (Intel Core Duo 2.8GHz, 4 GB, Windows vista 32bits, SQL Server 2008). The sensing was carried out during 24 hours, with intervals of 5 seconds. To simplify the experiments we do not use the quality threshold agent. And we add a varying interference to the raw information to simulate the inconsistency. At the same time, we set several sensor pairs to produce duplicate context. The evaluation consisted of (i) a study of performance verifying the time overhead added by the quality support in the framework and (ii) the compare of the different discarding algorithms.

As depicted in figure 7, the curve with minimum time represents traditional context dealing. This time is composed of the time of sensing, transferring, reasoning and distribution. The second curve above the bottom represents the dealing with replicate context discarding and the process may exhaust more time. And the third curve above the bottom represents the dealing with replicate context discarding and inconsistent context dealing. We use all inconsistent contexts discarding algorithm and we can find it may exhaust more time. The fourth curve above the bottom represents the dealing with replicate context discarding, inconsistent context dealing and quality

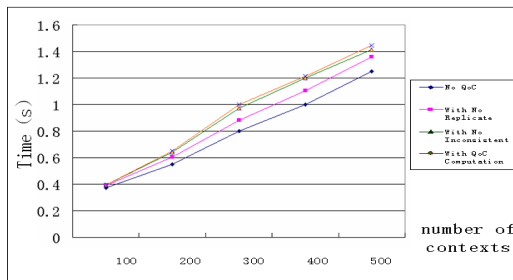


Fig. 7. The Overhead of the QoC-aware Context Dealing

factors computing. We can see the QoC dealing may exhaust more time and the time is lower than 30 percent of the normal dealing time of the context. In fact, comparing to the effect of the QoC, this degree of extra time may be accepted.

Furthermore we use different quality factors to help the inconsistent context discarding. We use up-to-dateness, relativity and certainty especially. Therefore, we use algorithm 1 representing no context discarding, algorithm 2 representing discarding all inconsistent context, algorithm 3 representing discarding newest context, algorithm 4 representing keeping most certainty context and algorithm 5 representing keeping most relative context.

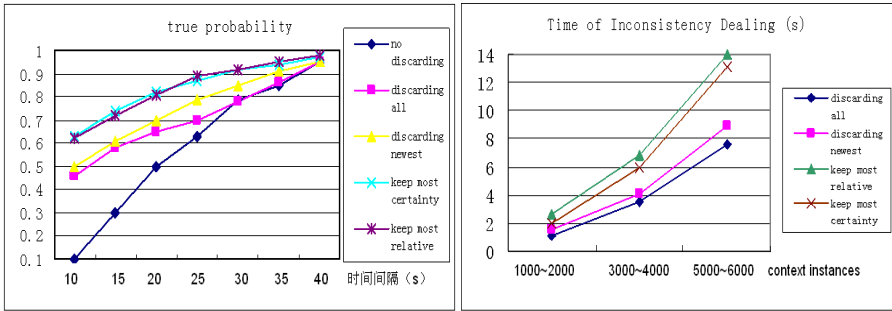


Fig. 8. (a) Analysis of the true probability

(b) Analysis of the dealing time

4 Conclusions

With the rapid development of the information technology, it is inevitable that the distributed mobile computing will evolve to the pervasive computing gradually whose final goal is fusing the information space composed of computers with the physical space in which the people are working and living in. To achieve this goal, one of the problems is how to continuously monitor/capture and interpret the environment related information efficiently. Sensing context information and making it available to the people, involved in coordinating a collaborative task, is a preliminary phase in making a system adaptable to the prevailing situation in pervasive environments.

Many attentions have been paid to the research of the context-aware pervasive applications. However, the diversity of the sources of context information, the characteristics of pervasive environments, and the nature of collaborative tasks pose a stern challenge to the efficient management of context information by sensing a lot of redundant and conflicting information. Most of existing research just use the raw context directly or take just some aspects of the Quality of Context (QoC) into account. In this paper, we have proposed an agent based context-aware framework that support QoC management. By using these agents we can evaluate raw context, discard duplicate and inconsistent context so as to protect and provide QoS-enriched context information of users to context-aware applications and services. In future work, we will complete more experiments to discuss more aspects of the framework.

References

1. Dey, K.: Understanding and using context. *Personal and Ubiquitous Computing* 5(1), 4–7 (2001)
2. Chen, G., Kotz, D.: A survey of context-aware mobile computing research. Hanover, NH, USA, Tech. Rep. (2000)
3. Buchholz, T., Küpper, A., Schiffers, M.: Quality of context: What it is and why we need it. In: *HPOVUA 2003*, Geneva (2003)
4. Kim, Y., Lee, K.: A quality measurement method of context information in ubiquitous environments. In: *CHIT 2006*, pp. 576–581. IEEE Computer Society, Washington, DC (2006)
5. Nixon, P., Razzaque, M.A., Dobson, S.: Categorization and modelling of quality in context information. In: *Proceedings of the IJCAI 2005* (2005)
6. Preuveneers, D., Berbers, Y.: Quality Extensions and Uncertainty Handling for Context Ontologies. In: Shvaiko, P., Euzenat, J., Léger, A., McGuinness, D.L., Wache, H. (eds.) *Proceedings of C&O 2006*, Riva del Garda, Italy, pp. 62–64 (August 2006), <http://www.cs.kuleuven.be/davy/publications/cando06.pdf>
7. Sheikh, K., Wegdam, M., van Sinderen, M.: Quality-of-context and its use for protecting privacy in context aware systems. *JSW* 3(3), 83–93 (2008)
8. Manzoor, A., Truong, H.-L., Dustdar, S.: On the Evaluation of Quality of Context. In: Roggen, D., Lombriser, C., Tröster, G., Kortuem, G., Havinga, P. (eds.) *EuroSSC 2008*. LNCS, vol. 5279, pp. 140–153. Springer, Heidelberg (2008)
9. Zheng, D., Jia, Y., Zhou, P., Han, W.-H.: Context-Aware Middleware Support for Component Based Applications in Pervasive Computing. In: Xu, M., Zhan, Y.-W., Cao, J., Liu, Y. (eds.) *APPT 2007*. LNCS, vol. 4847, pp. 161–171. Springer, Heidelberg (2007)
10. Zheng, D., Yan, H., Wang, J.: Research of the Middleware based Quality Management for Context-aware Pervasive Applications. In: *2011 International Conference on Computer and Management* (May 2011)
11. Zheng, D., Yan, H., Wang, J.: Research of the QoC-aware Service Selection for Middleware based Pervasive Applications. In: *The 2nd International Conference on Biomedical Engineering and Computer Science* (2011)
12. Zheng, D., Yan, H., Wang, J.: Research of the Middleware based Quality Management for Context-aware Pervasive Applications. In: *2011 International Conference on Computer and Management* (May 2011)
13. Zheng, D., Yan, H., Wang, J.: Research of the QoC-aware Service Selection for Middleware based Pervasive Applications. In: *The 2nd International Conference on Biomedical Engineering and Computer Science* (April 2011)

A Virtual File System for Streaming Loading of Virtual Software on Windows NT

Yabing Cui, Chunming Hu, Tianyu Wo, and Hanwen Wang

School of Computer Science and Engineering,
Beihang University, 100191, China
{cuiyb, hu cm, wot y, wanghw}@act.buaa.edu.cn

Abstract. With the cloud computing and virtualization technology popularizing and developing, the Software as a Service (SaaS), has become an innovative software delivery model. In this environment, if the virtual software runs in traditional way, that is to start up after completely downloaded, it will be time-consuming and greatly influence the users' experience. However, the streaming execution mode will enable the virtual software to start up while downloading. According to this conception, we design and implement a virtual file system for streaming delivery of software. The experimental results show that the first startup times of virtualized software have reduced by 20% to 60% and the users' experience has been improved effectively.

1 Introduction

In recent years, with the cloud computing and virtualization technology popularizing and developing, the Software as a Service (SaaS)[1], sometimes referred to as "on-demand software", largely enabled by the Internet, has become an innovative software delivery model. In contrast to traditional "on-premises" software that is deployed at the customer's premise, SaaS Software is run at a SaaS hosting provider and can be accessed over a network by the software user.

SaaS offers a set of advantages for software customers: SaaS simplifies the software access, reduces the Total Cost of Ownership (TCO) and opens up new payment models and revenue streams. Users will no longer need to build and maintain large data centers and hosting a big amount of middleware to run applications. Instead of licensing, maintenance and operational costs that occur in the traditional on-premise model a SaaS user pays only for the usage of the software for example in a pay-per-usage model. SaaS also provides a more stable and reliable computer experience for the end user. It is centralized, and an administrator installs, trains and maintains the applications over time.

The vSaaS system[2][3][4], illustrated in Figure 1, is a virtualization-based SaaS enabling architecture we implemented for cloud computing. In this system, virtualized applications are encapsulated and stored in vSaaS provider. When a user needs to run some software, the system will download the required data form vSaaS provider to local machine. The whole process is transparent to the user, just like the software has been installed.

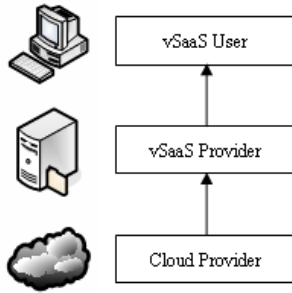


Fig. 1. User and Provider of vSaaS

Traditionally, an application can't be started and run without error unless it has been entirely stored on local disk. This means that the whole software must be downloaded before it can be run. So it is with virtualized software. Downloading the entire program delays its execution. In other words, application load time (the amount of time from when the application is selected to download to when the application can be executed) is longer than necessary.

To minimize the application load time, the software should be executable while downloading. Software streaming enables the overlapping of transmission (download) and execution of software. According to this conception, we design and implement a virtual file system for streaming delivery of virtual software. This system runs under the virtual execution layer of iVIC vSaaS, provides support for SaaS delivery model very well.

The rest of this paper is structured as follows. At first, we introduce key technologies related to our system in Section 2. Then, the design and implementation of the virtual file system is described in Section 3. In Section 4 we present some experiments to evaluate the function and performance of the system. Related work is detailed in Section 5. Finally, we conclude the paper and introduce the future work in Section 6.

2 Key Technologies

2.1 Application-Level On-Demand Streaming Loading

There are two prerequisites to achieve on-demand streaming loading on application level: encapsulation and interception.

Encapsulation is to encapsulate all data and files that are needed when an application runs inside one file, include executable files, configuration files, register data, and so on. Without encapsulation, we can implement streaming loading just on file-level, but can't application-level.

Virtual CD-ROM/hard disk image file is a well-designed encapsulation format, but it can't support streaming loading. Though we can enable it by create log files to maintain data blocks, it will lead to new problems on data synchronization and file

management. So, to design a new format which itself support streaming loading alongside encapsulation must be a more effective way.

Interception is to intercept file access, data loading and other related operations in order to get data or information that is not downloaded through network in a timely manner. By intercepting the request before it reaches its intended target, we can extend or replace functionality provided by the original target of the request.

In user mode, we can achieve interception by API hook, but it will hook a large number of APIs, and a global hook will hurt system performance and cause conflicts with other applications. So, we choose file system in kernel mode. As all file operations from upper layer are sent to the file system, we can receive all operations directly. At the same time, we can parse the encapsulation file by the file system, which can respond rapidly to file information queries and other operations.

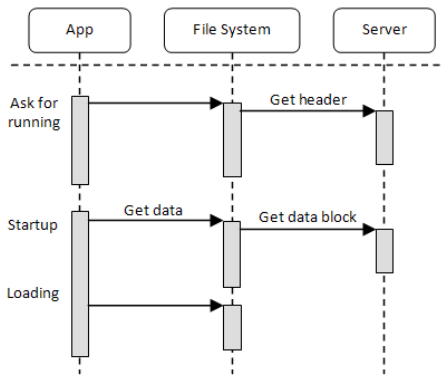


Fig. 2. Schematic Diagram of Streaming Loading

Virtual file system processes file operation requests and judges whether the required data has been downloaded to the local. If true, reads data directly, otherwise downloads data immediately, as illustrated in Figure 2.

2.2 Network File System Driver Development on Windows NT

As we all know, Windows operating system is closed source. This means that the development of its patch program - driver is more difficult than that in user mode. We should not only consider the underlying calls, but also be responsible for the needs of the upper. Because source is closed, and the development document is very brief, many data structures only can be clarified through reverse engineering.

File system driver runs based on CD-ROM/hard disk driver (include virtual CD-ROM/hard disk driver). It is responsible for all file operations, such as opening, closing, and locking a file, and calls the CD-ROM/hard disk driver to read or write data. Compared with other drivers, file system driver is particularly complex, for its dispatch functions are very complicated and difficult to develop.

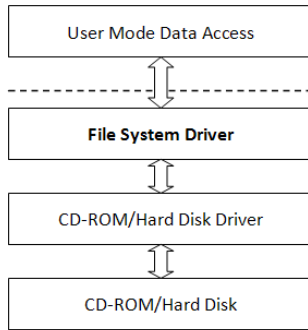


Fig. 3. Windows NT Driver Hierarchy Diagram

There are no well defined network functions can be called directly in kernel mode. We can only achieve network communications by calling TDI driver or even lower-level driver with self-constructed IRP (I/O Request Packet). Furthermore, in kernel mode there are only some underlying protocols (IP, TCP, and UDP), higher-level communication protocols must be completed by the developers themselves.

3 Design and Implementation

From the perspective of file system, it is an important part of operating system that is responsible for computer file data storage, retrieve, update and management on storage devices. File system consists of three parts: files to be managed, data structures for management and software to implement management.

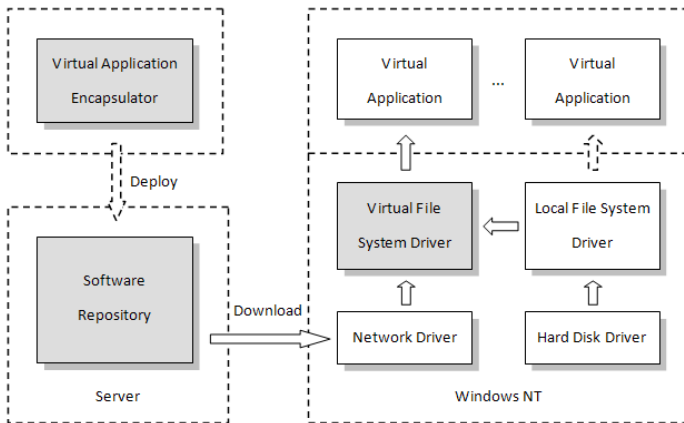


Fig. 4. Virtual File System Architecture Diagram

From the point view of software virtualization, if we want to implement streaming loading on application level, we should first encapsulate the application, and then deliver it in software repository. When client machine starts up some applications, virtual file system will receive requests. It checks the local cache, if the request data or information has been downloaded, returns directly, otherwise, gets related data through network, and makes the program run successfully.

Based on the above two viewpoints, we divide our virtual file system into three modules: virtual application encapsulator, virtual file system driver and software repository, as shown in Figure 4.

3.1 Virtual Application Encapsulator

Virtual application encapsulator is a key component which packages an application for virtualization and streaming. It should be run in a separate virtual machine or physical machine. Virtualized resources include executable files, specific DLL files, related data such as user profile information, and configuration repositories like registry hives and INI files. The process of encapsulation does not change the application itself.



Fig. 5. Encapsulation Format

Figure 5 shows the encapsulation file format designed for streaming. It contains four data structures: HEADER, BITMAP, DATA and DIRECTORY. Each section is started by the same Common Header, not only can be used for distinguishing the type of that section, but also have benefits to future extensions.

The members *bitmap_offset*, *file_offset* and *root_directory_recod* in HEADER section hold the offsets of other sections BITMAP, DATA and DIRECTORY respectively. The members *size* and *block_size* represent the total size of DATA section and the block size for streaming loading; the member *bitmap_len* represents the bit length of BITMAP section.

BITMAP data, which is one of the most important metadata to support streaming loading, is used for marking whether a block has been already downloaded to the local. Blocks that have been downloaded will be marked as COMPLETED.

DIRECTORY section is used for maintaining the directory structure of all files that are encapsulated. It contains lots of directory entries. If an entry represents a file, the member *size* simply indicates the size of the file; if an entry represents a folder, the *size* indicates the number of sub-entries. When traversing in this folder, if the name does not match, we can achieve rapid traverse by skipping sub-entries.

The actual data of encapsulated files is stored in DATA section. The offset and length of each file are stored in the corresponding directory entry in DIRECTORY section. All the file data is arranged in a line, and is logically divided into continuous blocks. A block is a basic unit for streaming loading, the size of which is the *block_size* in HEADER section.

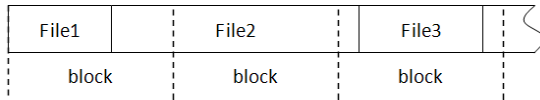


Fig. 6. Blocks of DATA Section

3.2 Virtual File System Driver

The virtual file system driver which is support virtual process directly is run in Windows NT client machine. Its main features are on-demand streaming loading and avoiding re-download by caching.

When a virtual application is to be started or loaded, virtual file system receives requests, and repeats the following steps:

- 1) Check whether the corresponding encapsulation file has been downloaded in local cache directory. If no, continue with step 2; if yes, jump to step 4.
- 2) Check whether there is corresponding encapsulation file in the software repository. If yes, continue with step 3; if no, return error message.
- 3) Create encapsulation file in cache directory, download file header through network, and save it in the cache file.
- 4) Check whether the required data has been downloaded and saved. If yes, read data, fill buffer, and return; if no, continue to step 5.

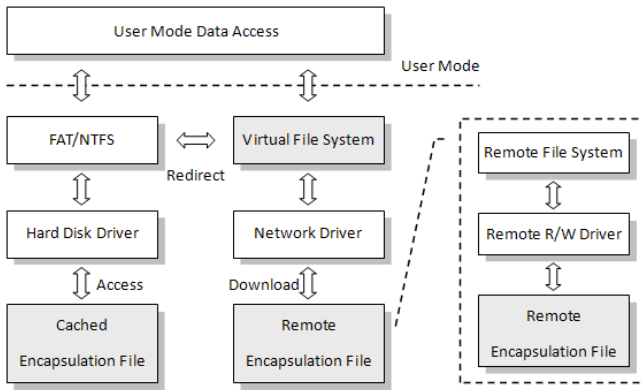


Fig. 7. Data Flow of Virtual File System Driver

- 5) Get required data through network, save it in the cache file, set corresponding BITMAP flag to COMPLETED, fill buffer correctly, and return.

According to different functions, the virtual file system driver is divided into three layers which are described as follows.

File System Interface Layer

This layer is responsible for parsing and forwarding IRPs, maintaining information of devices, and managing system resources.

All file operations, such as open, close, query, lock and map, are sent to interface layer by I/O manager. Interface layer checks parameters for validity, calls appropriate functions of parsing layer to process IRPs, and if some IRPs can't be completed successfully, fail it or forward it.

Encapsulation File Parsing Layer

The main function of this layer is to parse encapsulation file, and return required data or information in correct format according to the parameters passed by interface layer.

There are two important data structures in this layer: FCB (File Control Block) and CCB (Context Control Block). Because this layer is the critical layer related to the overhead caused by file system virtualization, the optimization by the two structures is essential for improving the speed of whole system.

FCB is stored in the form of circular linked list, and used for caching information related to virtual files, such as resource, time, size, offset, and sharing control flag. The main purpose is to provide the system with control data, manage file sharing access, and speed up information delivery speed.

When a file needs to be opened, system will check FCB list first. If there is no corresponding FCB, then the system tries to create a new one; otherwise the system will check the sharing control information and determine whether it can open the file.

If the file is opened successfully, the reference count of FCB will be increased. Then, when some information is needed, it will be read from FCB directly. When the file is being closed, the reference count of FCB must be decreased. If the reference count equals 0, which means the file has not been used, so the system should remove the corresponding FCB from linked list and free up space.

CCB is used for caching context-sensitive information, such as file handle, data offset, and search pattern. Sometimes, some operations can't be completed in one time, for example, enumerating folders, which requires that certain information is stored in the CCB. These operations are context-sensitive; the corresponding CCB is created when the file handle is opened, stored in the file's object, and released when the file handle is closed.

Encapsulation File Download Layer

This layer is the actual implementation of streaming loading that get required data from software repository in streams according to parsing layer's demand.

This layer is optimized for speed by SCB (Stream Control Block). The parameters sent from parsing layer are in the form of $\langle File, Offset, Length \rangle$ which represents reading Length bytes of data from Offset in File. Download layer check first whether there is a SCB corresponding to File in the SCB list. If none, the system tries to create a new one; if finding or creating FCB successfully, the system will get BITMAP by SCB, and check the corresponding flag bits in BITMAP. If all the bits are COMPLETED, reads data and returns; otherwise downloads data immediately through network and modify the BITMAP.

3.3 Software Repository

Software repository is essentially software delivery server, and in principle can be built by any protocols supporting streaming. Here we adopt the HTTP protocol, using IIS or Apache to set up software repository. Speed optimization can be considered by using different protocols in the future.

4 Experiments

To evaluate the functionality and performance of the virtual file system, we conduct series of experiments. Our experiments are based on two PC machines, each with 1 Intel Core2 2.83GHZ CPU, 2GB RAM and 1Gb/s network card. We set up software repository by Microsoft IIS on server machine and install Windows XP SP2 operating system on client machine.

4.1 Function Evaluation

We first prepared 12 programs to test dozens of file operation APIs, such as CreateFile, CloseFile, SetFilePointer, FindFirstFile and FindNextFile. After verifying

```

Directory of F:\eclipse

1601-01-01  08:02                16,536  ep1-v10.html
1601-01-01  08:02                 357    eclipse.ini
1601-01-01  08:02                28,672  eclipsec.exe
1601-01-01  08:02                <DIR>    dropins
1601-01-01  08:02                <DIR>    p2
1601-01-01  08:02                57,344  eclipse.exe
1601-01-01  08:02                71,085  artifacts.xml
1601-01-01  08:02                <DIR>    configuration
1601-01-01  08:02                 6,506  notice.html
1601-01-01  08:02                <DIR>    plugins
1601-01-01  08:02                <DIR>    features
1601-01-01  08:02                <DIR>    readme
1601-01-01  08:02                 59    .eclipseproduct
              7 File(s)              180,559 bytes

```

Fig. 8. Directory in Virtual Drive

the returned results are all correct, we launch software running experiments in DOS Command Line and File Explorer.

Figure 8 shows the directory structure of the folder "Eclipse" in the virtual drive "F:". As soon as the command "eclipse.exe" is inputted, startup screen appears quickly, which indicates that the underlying driver has downloaded related data timely while the data is being required. Compared with starting up after total downloaded mode, the users' experience has been improved effectively.

4.2 Function Evaluation

To evaluate the performance of the virtual file system, we select 4 typical software as sample programs whose size, function and type are all different: Msn Messenger (24MB), Photoshop (43MB), Eclipse (94MB), Adobe Reader (143MB).

At first, we assess the impact of different block sizes on streaming starting. In the Figure 9, taking Adobe Reader as an example, horizontal axis represents the changes in the block size, left vertical axis represents the software start-up time, and right vertical axis represents initial launch bytes percentage of total software.

We can see that the software start-up time declines rapidly at first, with the block size changes from small to large, and then rises slowly. The smaller the block size is set, the more frequently access to network will be taken, resulting in the cost of time; but if the block size is set too large, much data unrelated to startup will be downloaded. We can imagine that if the block size is equal to the total size of the whole software, the streaming loading mode will degenerate into full download mode, that is to say the software can't be started until all the data has been downloaded, which is intolerable for large-scale software.

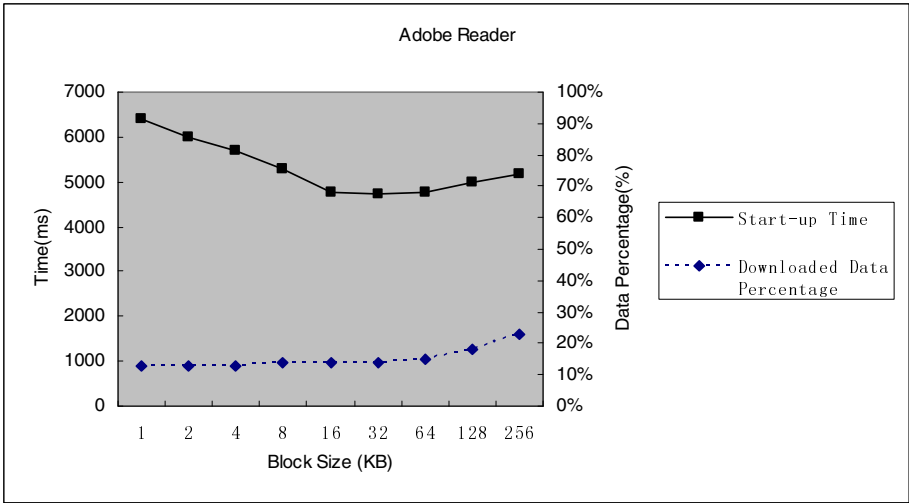


Fig. 9. Software Start-up Times at Different Block Size

The following experiments are carried out with the block size of 16KB. We measured the startup time of the four software referred above in streaming loading mode, and compared with full download mode. As shown in Figure 10, t_s represents streaming startup time, t_a represents full download startup time, the startup time of Adobe Reader is significantly reduced by 67.6%, but Photoshop is only reduced by 20.4%, that is because Photoshop itself starts up a little slower, the change in the data loading time affects the entire startup time relatively small.

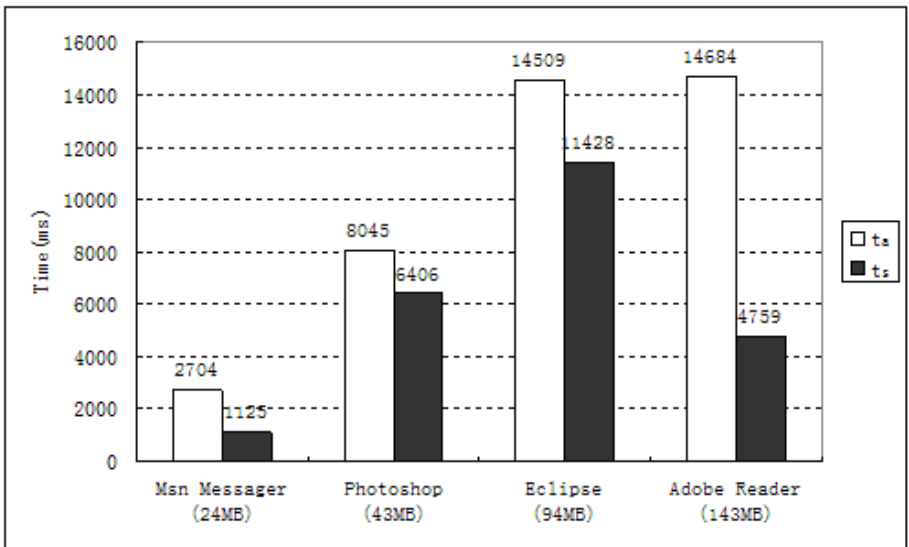


Fig. 10. Start-up Times in Streaming Loading Mode

The introduction of virtualization will bring performance overhead and increase execution latency. As shown in Figure 11, t_d represents original launch time and t_c represents virtualized launch time, we can see that the delta of t_d and t_c is at 0.1s level, which is acceptable in the real user scenarios.

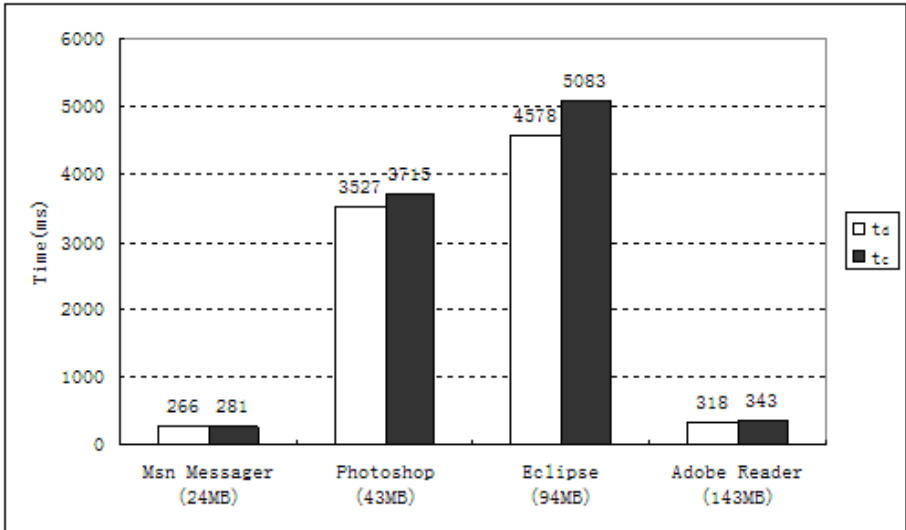


Fig. 11. Performance Cost of Virtualization

Through the above analysis, the virtual file system for streaming loading reduces the startup time of virtual software effectively and makes virtual process run more smoothly. The experiment also shows that network latency is still a bottleneck restricting programs from starting up quickly. In the future, further optimizations for download speed can be conducted.

5 Related Work

App-V[6] (Microsoft Application Virtualization, formerly Microsoft SoftGrid) is an application virtualization and application streaming solution from Microsoft. App-V allows applications to be deployed in real-time to any client from a virtual application server. It removes the need for local installation of the applications. Instead, only the App-v client needs to be installed on the client machines. All application data is permanently stored on the virtual application server. Whichever software is needed is either streamed or locally cached from the application server on demand and run locally. But the sequencer of App-V has not been further optimized for packaging operation, and because of the reliance on many Windows components, App-V is difficult to be deployed and is not suitable for migration of personal computing environment.

Symantec Workspace Streaming[7] (SWS, formerly AppStream) is an application streaming solution that enables on-demand application provisioning, offline cache, license recovery and instant application upgrades. Workspace Streaming increases end user productivity with controlled, guaranteed access to any Windows based applications from any location at any time, including remote and mobile users. But Workspace Streaming only achieves streaming loading on file-level; it doesn't need to package files, let alone virtual file system; data loading process is coupled tightly with software execution environment, difficult to be designed, maintained and optimized.

VMware ThinApp[9] (formerly Thininstall[8]) is an application virtualization and portable application creator suite by VMware that can package conventional applications so that they become portable applications. ThinApp is able to execute applications without them being installed in the traditional sense by virtualizing resources such as environment variables, files and Windows Registry keys. The virtual environment presented to the client is a merged view of the underlying physical and virtual resources, thereby allowing the virtualization layer to fool the application into thinking that it is running as if it were fully installed. Because every executable file packaged by ThinApp contains a lightweight virtual machine and many useless files and data, lots of storage space and computing resource are wasted. Moreover, ThinApp is a compact solution and doesn't provide any support for software delivery and streaming loading.

6 Conclusion and Future Work

In this paper, we present a virtual file system for streaming loading of virtual software on Windows NT, and describe the design and implementation of the system. We have deployed the system in iVIC vSaaS environment and performed a set of experiments. The results show that the system is well designed to achieve the goal that enables virtual software to start up while downloading, avoids full download before virtual software can be run, accelerates the launch process, smoothes the running process, and improves the users' experience significantly.

As the network latency is still the bottleneck causing the system time-consuming, our on-going work is focusing on adding prefetch function in the driver or attaching push function to software repository, accelerating data loading and delivering, increasing the throughput of the system, and further improving the users' experience.

Acknowledgments. This work is partially supported by grants from China 973 Fundamental R&D Program (No. 2011CB302602) and Natural Science Foundation of China (No. 91018004, 90818028). We would like to thank the anonymous reviewers for their thoughtful comments and suggestions.

References

1. Troni, F., Silver, M.A.: Use Processes and Tools to Reduce TCO for PCs, Update, Gartner Group (2005-2006)
2. Ma, D.: The Business Model of "Software As A Service". In: 2007 IEEE International Conference on Services Computing (SCC 2007), pp. 701–70 (July 2007)

3. Huai, J., Li, Q., Hu, C.: CIVIC: A Hypervisor Based Virtual Computing Environment. In: Proceedings of the 2007 International Conference on Parallel Processing Workshops (September 2007)
4. Zhong, L., Wo, T., Li, J., Li, B.: A Virtualization-based SaaS Enabling Architecture for Cloud Computing. In: ICAS 2010 (March 2010)
5. Zhong, L., Wo, T., Li, J., Li, B.: vSaaS: A Virtual Software as a Service Architecture for Cloud Computing Environment. e-Science (December 2009)
6. App-V (November 3, 2010), <http://www.microsoft.com/windows/enterprise/products/mdop/app-v.aspx>
7. Symantec Workspace Streaming, <http://www.symantec.com/en/hk/business/workspace-streaming>
8. Thinstall, Application virtualization: A technical overview of the thinstall application virtualization platform, http://www.creekpointe.com/helpdesk/pdf/Thinstall_ApplicVirtualization.pdf
9. VMware ThinApp for Application Virtualization, <http://www.vmware.com/products/thinapp/overview.html>

TBF: A High-Efficient Query Mechanism in De-duplication Backup System

Bin Zhou^{1,2}, Hai Jin¹, Xia Xie¹, and PingPeng Yuan¹

¹ Services Computing Technology and System Lab
Cluster and Grid Computing Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

² School of Computer Science and Technology
South-Central University for Nationalities, Wuhan, China

Abstract. For the big data, the fingerprints of the data chunks are very huge and cannot be stored in the memory completely. Accordingly, a new query mechanism namely Two-stage Bloom Filter mechanism is proposed. First, each bit of the second grade bloom filter represents the chunks having the identical fingerprints which reducing the rate of false positives. Second, a two-dimensional list is created corresponding to the two grade bloom filter to gather the absolute addresses of the data chunks with the identical fingerprints. Finally, we suggest a new hash function class with the strong global random characteristic. Two-stage Bloom Filter decreases the number of accessing disks, improves the speed of detecting the redundant data chunks, and reduces the rate of false positive. Our experiments indicate that Two-stage Bloom Filter reduces about 30~40% storage accessing of false positive with the same length of the first grade Bloom Filter.

Keywords: Two-stage Bloom Filter, Standard Bloom Filter, De-duplication, False positive, Fingerprint.

1 Introduction

In the information explosion era, a great deal of information and data will be generated every day. According to the *International Data Corporation* (IDC) statistics, the amount of data of the whole world was just 180EB in 2006, and this figure increased to 1800EB in 2011. Recent IDC reports predict that this figure will arrive to 8000EB (almost 8ZB) in 2015. It sounds good to store all the data. The fact is that IT budget becomes seriously critical. The growing rate of the annual input is just 3 percent [1]. A wide gap comes into being between the demand of the information storage and the affordability.

How should we respond to and make use of the massive information? Recently, data de-duplication, a hot emerging technology, has received a broad attention from both academia and industry. Its basic thought is to divide the file data into different

chunks and the same chunks in the saving system only save one copy, while the others, referenced to by a pointer are pointing to the array position of the chunks. To the storage system for big data, the number of the chunks is very huge especially in the case of fine-grain. It is a key to improve the performance of the system by judging whether a new data chunk can be stored in the storage system as quickly as possible.

The information retrieval comes in many forms, such as dynamic array, database, RB/B/B+/B* tree and hash table. The hash retrieval is famous for its $O(1)$ performance. It seems to be a good solution to adopt hash table to save the meta-data index information [2-3], because the overhead for querying the hash table is a constant theoretically. In fact, when the data scale becomes big, it is unable to save the index information completely in the memory and we need to carry on an IO operation to access the disk in the query, which is a consuming process. The key is how to avoid the IO operation to accelerate the query speed of data fingerprint.

In this paper, the contributions of our work are as follows: First, we introduce the bloom filter into our de-duplication storage system. Because of the shortcoming of the false positive rate in the original bloom filter, we present a new mechanism named *Two-stage Bloom Filter* (TBF). Second, a two-dimensional list is created corresponding to the two grade bloom filter. Finally, since the fingerprints made by the MD5 algorithm have the characteristics of being random, a new hash function class with the strong global random character comes out. Comparing with the existing algorithms of detecting the redundant data chunks, TBF decreases the number of accessing disks, improves the speed of detecting the redundant data chunks, and reduces the rate of false positives. All these measures improve the overall performance of system.

The following paper reads as follows. Section 2 introduces related works. Section 3 describes the organization of de-duplication storage system and section 4 studies two-stage bloom filter mechanism in detail. In section 5, various experiments are performed to measure the improvement of the rate of false positives based on the TBF. At last, section 6 draws the conclusion.

2 Related Works

It is a serious concern to find out more redundancy data in the data de-duplication system. For the data quantitative is huge, we can not compare the chunks directly, but make use of hash algorithm (MD5 or SHA-1) to compute a hash value (data fingerprint) of each chunk and deposit this fingerprint to a data structure. We compute its data fingerprint first when the new chunk coming, and then we compare it with the existing ones. If this fingerprint has been in the set, an index is stored; otherwise, the new chunk and fingerprint all should be stored in the system.

It is great to meet the system performance by using the fingerprints that produced by the MD5 algorithms when the data quantity is not big. However, it will generate many other factors to affect the performance of the system while the data quantity becomes huge.

The most important factor affecting the system performance is the lack of the memory. It should compare all the saved fingerprints to make sure that a chunk has been existed, and an I/O operation to query the fingerprints is already inevitable when

the contents of hash table exceed far beyond the memory. This operation consumes so much time that it will affect the overall performance of system greatly.

To tackle these problems, Lillibridge et al. [4] adopted different methods, such as sampling, sparse index and chunk locality. Thewl et al. [5] introduced the B+ tree into the data de-duplication system. Bhagwat et al. [6] then proposed to divide the chunk indices into two levels.

In some intuitive sense, the shorter the chunk, the higher the opportunity of discovering the redundancy data, as while as the whole metadata is also bigger. Bobbarjung et al. [3] tried to minimize the length of the chunk as 1KB or so, as while as making a set of chunks as a unit to deal with. Kruus et al. [7] proposed a kind of two-stage chunking algorithm that re-chunked transitional and non-duplicated big CDC chunks into small CDC chunks. It could discover more redundancy chunks comparing with the basic CDC algorithm and obviously enlarged the amount of metadata. However, a new kind of algorithm was proposed [8] according to the chunks appearing frequency to decide whether it further divided the CDC chunks, and it could decline the amount of metadata to a certain degree and improved the performance of de-duplication.

Zhu et al. [2] introduced the bloom filter into document backup system, which also made use of the bloom filter to create a summary vector in the memory while backing up the data that could query the memory vector directly and knew whether the data chunks had already been in, while searching the data fingerprints.

However, some literature [9-11] also introduced the bloom filter, the efficient memory data structure into the redundancy data detection and reduced the number of the IO operations to judge whether the chunk had been in the storage system, but all of them had seldom deeply studied the problem of the false positive.

For keeping a low rate of false positive, the availability of RAM space of the machine decides the length of the bloom filter. For the big datasets, the RAM space is not sufficient. Debnath et al. [12] advocated the flash memory to serve as suitable medium for storing bloom filters and Bender et al. [13] suggested the Cascade Filter and Guo et al.[14] proposed dynamic Bloom filters to overcome this shortcoming, respectively.

In our system, we present the solution of the two-stage bloom filter mechanism making use of the second grade bloom filter on the base of the algorithm of the standard bloom filter. It reduces the number of IO operation directly to improve the throughput by decreasing the false positive rate as far as possible.

3 Data Fingerprint Query Based on the TBF

In our system, the large-scale data is the object. Under the assumption that we need to store the data about 2^n TB, each chunk is about 2^3 KB and it will have about $2^{n-3} \cdot 10^9$ unique chunks. If the index of each chunk is 16 Bytes, it needs about 2^{n+1} GB space to store them. The storage space increases exponentially when n increases. Not all these indices can be stored in the memory and the frequent disk IO operations are unavoidable. Supposed that the average time of one IO accessing is about 4ms, it can

retrieve 250 chunks per second, which means that the throughput of the system is about 2MB/s. This is not acceptable.

To avoid accessing the disks and keep the query time within the limit, it is the prerequisite to simplify the index data and keep them in the memory fully.

3.1 Standard Bloom Filter (SBF) Algorithm

The original bloom filter composed of a large bit vector BV and k hash functions is an efficient and simplified memory data structure, which could realize the efficient query. Compared with the traditional tree query algorithm and hash query algorithm, the space that the bloom filter needs shows no correlation with the size of the element needed by query, but it only shows a correlation with the numbers of the functions mapping the element to the bit vector, which will economize storage space greatly.

In the initial state, the bloom filter is a vector whose length is m , with the initial value of 0. The set S includes n elements (data fingerprint) $\{x_1, x_2, \dots, x_n\}$, there are k independent hash functions $h_i(x)$ in the Bloom Filter, which map the each fingerprint to $(0, 1, \dots, m-1)$, respectively. The value of $h_i(x_j)$ ($1 \leq i \leq k$) will be calculated if a data fingerprint x_j is inserted into the set S . If the hash value is s , the counterpoint array position should be set to 1.

By applying k hash functions on the y data fingerprint respectively, it can be concluded that the fingerprint is not in the set if there is one or more 0s in the corresponding array positions. However, it can not be concluded that the data fingerprint belongs to the set if the bits at all these k array positions are set to 1, as false positive may appear at this time [15], which can be stated as the formula:

$$FP = \left(1 - \left(1 - \frac{1}{m} \right)^{kn} \right)^k \approx (1 - e^{-kn/m})^k \quad (1)$$

In the formula, m is the length of the vector BV , namely the length of bloom filter; while n is the number of the set elements, that is the number of chunks and k is the number of hash functions.

It is known from the above analysis that for the given n and m , they need more hash functions, that is to say, to increase the value of k , to ensure the enough low probability of the false positive, which will directly lead to the low performance of adding and querying operations for the data fingerprints.

3.2 Principle of TBF Algorithm

The rate of false positive can not be decreased by the increasing value of k continuously. For k , the FP has a minimum value as $FP_{min} = 0.6185^{m/n}$ [15]. In order to minimize the value of FP , we enlarge the value of m/n at this time. It sets up larger bloom filter for the given element amount n , at the price of enlarging the memory occupation.

Inspired by the multidimensional bloom filter suggested by Guo et al. [16], we consider adopting TBF mechanism by two bloom filters in series to reduce the rate of false positive.

Because each hash function of the SBF is independent and has no contact with each other the values of the hash function for the different elements produce collision easily. However, SBF takes no measure to deal with the above-mentioned situation. For example, when the same value of the different hash function for the other element generated after the array position had been set, it just did nothing but simply queried.

Fig.1 shows us the principle of the TBF mechanism. All values of the hash functions for per element are incorporated into one value, namely the whole characteristic of the element. Moreover, the second grade bloom filter will show the characteristic of the value. Therefore, the integrated information leads to the falling rate of the collision produced by the single hash function value reducing the rate of the false positive for the overall system.

Given the set of the data fingerprints is $S=\{x_1, x_2 \dots, x_n\}$, the TBF is composed of two bloom filters in series. The length of the first filter is m , which represents the values of the fingerprints generated by the set of the k hash functions:

$$element_i_addr_j = h1_j(x_i) \{0, 1, \dots, m-1\} (1 \leq i \leq n, 1 \leq j \leq k)$$

$element_i_addr_j$ represents the j^{th} hash value for the i^{th} element.

The length of the second filter is n' , which represents the integrated information of k hash functions for each element. The hash function is shown as below:

$$h2_i(x) = element_i_addr_1 \oplus element_i_addr_2 \oplus element_i_addr_3 \oplus \dots \oplus element_i_addr_k; (1 \leq i \leq n') \quad (2)$$

The second bloom filter address comprises of the value of XOR operation, which represents the integrity of the hash values of each element of the first bloom filter.

The initial values of the two grade Bloom Filters are set to 0.

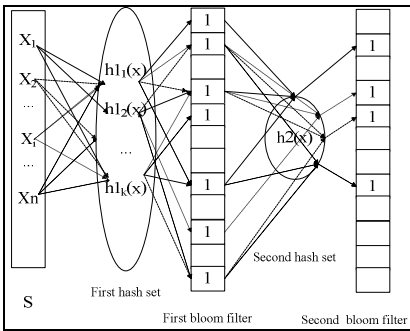


Fig. 1. Principle of the TBF

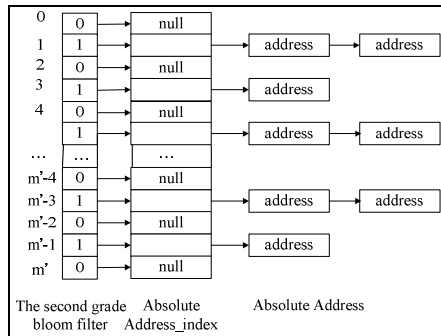


Fig. 2. Absolute addresses list for initial data chunks

When a new partitioned data chunk inserted, its fingerprint is hashed k times, and the corresponding places in the first grade Bloom Filter are set to 1. Then the value of the XOR according to the formula (2) is got, and the corresponding place in the second Bloom Filter is set to 1. This place of the second grade Bloom Filter has two meanings. The first is that it represents the values of the k hash functions that are independent in the first grade Bloom Filter as an entirety. The second is that this place

corresponds to the data chunks, which have the same fingerprint. It can link the absolute addresses of the data chunks having the same fingerprint together. Once the hash collision occurs, the content of the data chunk is got in time of need directly, resolving the defects that it cannot get the initial content from the map value of the hash function.

It needs to judge whether the chunk has been in the storage system when the sequent partitioned chunks come. If the false positive occurs through the query of the first grade Bloom Filter, we go directly to the query of the second grade Bloom Filter. If the correspond place of the second Bloom Filter is set to 0, it can tell that the data chunk has not been stored and there is no need to make a comparison with the original chunk content bit by bit. Thus, it decreases the false positive rate, reduces the times of IO operations and improves the throughput and performance of the system.

We build a two-dimension link list to save absolute address of the data chunks shown in Fig.2. The length of the list is the same as that of the second grade bloom filter, and the initial values are null. Each array position of the second grade bloom filter corresponds to a series of absolute addresses. The system will record the absolute address of a new data chunk when the data chunk is inserted as well as the value of the corresponding array position of the second grade bloom filter is set to 1.

3.3 Performance Analysis for TBF

Theorem. The rate of false positive for TBF algorithm is less than that of the SBF algorithm.

In this paper, we prove the following: Let the rate of false positive for standard bloom filter algorithm be FP_1 , the length of bloom filter is m , the number of elements is n , and the number of hash functions is k . The rate of false positive for the second grade is FP_2 , and the length of the second bloom filter is m' , and the number of elements is n' , and the number of hash functions is k' . The rate of false positive for the TBF algorithm is FP .

$$FP_1 = (1 - e^{-kn/m})^k \tag{3}$$

For just one hash function is set in the second grade bloom filter, namely $k'=1$, the rate of the false positive for the second grade is as follows:

$$FP_2 = (1 - e^{-k'n'/m'})^{k'} = (1 - e^{-n'/m'}) \tag{4}$$

The two bloom filters are in series, so the overall rate of false positive is:

$$FP = FP_1 * FP_2 = (1 - e^{-kn/m})^k * (1 - e^{-n'/m'}) \tag{5}$$

Because $n' < m'$, then $(1 - e^{-n'/m'}) < 1$, so $FP < FP_1$.

4 Experiments

A prototype system is designed. We test the improvement of performance by adopting the TBF mechanism that helps to avoid the disk bottleneck, and to analyze the impacts of different number of hash functions and different lengths of the bloom filters.

4.1 Experiment Setup

Our experiment is performed on two nodes configured with 2-way quad-core Xeon E5405 2GHz CPUs and 8GB DDR RAM. The cache capacity for each core is 6144KB. The data files (chunks) are stored on RAID 0 with two disks (Seagate Barracuda 7200RPM, 1TB each). Each node has an Intel 80003ES2LAN gigabit network interface card (NIC) and is connected via switched gigabit Ethernet. One node is the server and the other is the mirror.

4.2 Hash Functions Class

The most crucial factor is to decrease the collisions of hash functions in TBF, and then the hash functions should have the outstanding global distributing characteristics.

In the hash functions of H_3 defined by Carter et al. [17-18], each H_3 hash function is corresponding to a 0, 1 function matrix. It has the unusual global distribution, but consuming excessive CPU resource.

Considering the rate of collision for the data fingerprints is less by MD5 algorithm, new hash functions should be created on the base of fingerprints.

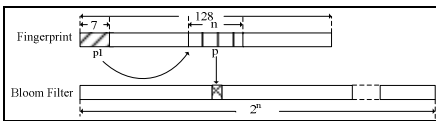


Fig. 3. The first grade index hash

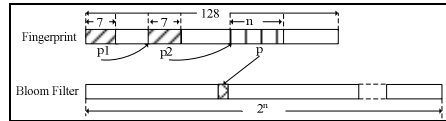


Fig. 4. The second grade index hash

The First Grade Index Hash. The length of fingerprint created by MD5 is 128 bits, and let length of the first grade bloom filter be 2^n bits, shown in Fig.3. We get the first 7 bits of the fingerprint to gain $p1$, whose value range is from 0 to 127. Then $p1$ points to the array position of the fingerprint, and n bits should be got. This value of n bits is the hash value, who will point to the absolute address of the first bloom filter.

The Second Grade Index Hash. The second grade index hash is an improvement upon the first grade index hash. It gets the content of the first 7 bits $p1$ as a pointer to point to the array position of the fingerprint, and then gets the content of the next 7 bits $p2$ as the second grade pointer pointing to the next array position. At last, the n bits should be got as the pointer p . In addition, p is the value of the hash function and it will point to the array position of the first grade bloom filter, shown in Fig.4.

Furthermore, both the first and the second grade index hash have a shortcoming that when it gets the pointer $p1$ or $p2$, namely the content of the 7 bits of the fingerprint, the state will follow: $128-p1 < n$ or $128-p2 < n$. It should link the head and tail of the fingerprints to form a loop.

4.3 Improvement of Performance for TBF

In these experiments, we mainly study the influence of the selection of the various number of hash functions to the throughput and the number of storage accessing of the false positive. In the experiments, the data is about 1TB, and the number of the data chunks, namely n , is 114,845,208.

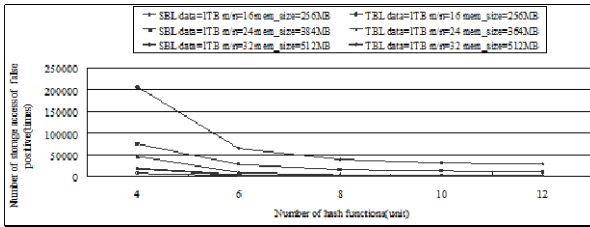


Fig. 5. Number of storage accessing of the false positive

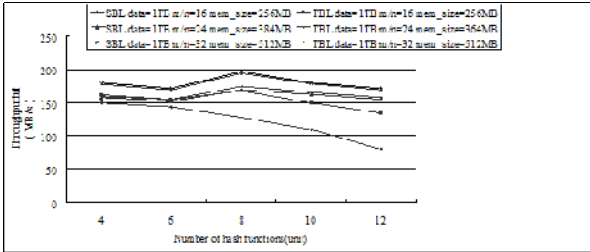


Fig. 6. Throughput

For 1TB data, the total number of false positive storage accessing is no more than 250,000 after the filtering by the first grade Bloom Filter. While the memory space occupied by the second grade Bloom Filter is negligible. Fig.5 illustrates that TBF reduces about 30~40% storage accessing of false positive with the same length of the first grade Bloom Filter. In addition, it indicates that the performance-to-price is the optimal while the k is 8 and the length of the first Bloom Filter is 384MB. TBF has a limited capability to decrease the rate of the accessing of the false positive when increasing the numbers of the function or the length of the first grade Bloom Filter.

Fig.6 shows that the throughput achieves 195MB/s when k is 8 and the length of the first grade Bloom Filter is 384MB by adopting TBL. The throughput will decrease if k is increased. However when the length of first grade Bloom Filter increases, it is not obvious to observe the increase of the throughput.

When $k \approx (m/n) * \ln 2$, the performance-to-price is optimized from Fig. 5 and Fig. 6 which is in coincidence with the result of [18].

5 Conclusions

It is a key problem in the de-duplication backup system to judge whether a new partitioned data segment has been in the storage system as quickly as possible. In this paper, a new mechanism TBF is proposed.

The false positive existed in the comparison algorithm of the traditional data fingerprints. The result was decided by the accessing of the disk through IO operations in most cases. However, introducing the TBF mechanism is not simply to add a new bloom filter on the SBF. The second grade bloom filter represents the entirety of the first bloom filter that each bit of the second grade bloom filter representing the chunks that have the identical fingerprints. It also introduces a two-dimensional list, which gathers the absolute addresses of the data chunks having the same fingerprints in a list. By this characteristic, the fingerprints need not to be stored completely. It can judge whether the new input data chunk has been stored by traversing the list when meeting the identical fingerprints. A new hash function class based on the MD5 finds its way to solve the problem of collisions for the fingerprints.

The experiments reveal that if the system stores the fingerprints directly. Owing to the large number of the fingerprints, they cannot be stored in the memory and then frequent accessing of the back storage system is unavoidable in the query process. By adopting the TBF mechanism, it saves nearly 65% of the memory space when the m/n is 24 although in the price of a second bloom filter space and an absolute address list space. Moreover, the back storage system is avoided being accessed and greatly reduces the operation expenses.

Furthermore, comparing with the initial bloom filter, the TBF algorithm represents the multiple separate hash functions of the data fingerprints as an entirety to achieve smaller rate of false positive. Its mechanism has the following merit standing out obviously: FP is about 30~40% of FP_1 , which is about 60~70% of disk IO accessing operations being reduced when the value of m'/n' is 2.

The above measures reduce the query time and an additional space cost has little effect on the system.

References

1. The International Data Corporation website, <http://www.idc.com>
2. Zhu, B., Li, K., Patterson, H.: Avoiding the Disk Bottleneck in the Data Domain Deduplication File System. In: Proceedings of 6th USENIX Conference on File and Storage Technologies, pp. 1–14. USENIX Association, San Jose (2008)
3. Bobbarjung, D.R., Jaqannathan, S., Dubnicki, C.: Improving Duplicate Elimination in Storage Systems. ACM Transactions on Storage 2, 424–448 (2006)
4. Lillibridge, M.: Sparse Indexing, Large Scale, Inline Deduplication Using Sampling and Locality. In: Proceedings of 7th USENIX Conference on File and Storage Technologies, pp. 111–123. USENIX Association, San Francisco (2009)

5. Thewl, T.T., Thein, N.L.: An Efficient Indexing Mechanism for Data Deduplication. In: Proceedings of 2009 International Conference on the Current Trends in Information Technology, pp. 1–5. IEEE Press, Dubai (2009)
6. Bhagwat, D., Eshghi, K., Long, D.D.E., Lillibridge, M.: Extreme Binning: Scalable, Parallel Deduplication for Chunk-based File Backup. In: Proceedings of 17th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 1–9. IEEE Press, London (2009)
7. Kruus, E., Ungureanu, C., Dubnicki, C.: Bimodal Content Defined Chunking for Backup Streams. In: Proceedings of 8th USENIX Conference on File and Storage Technologies, pp. 239–252. USENIX Association, Berkeley (2010)
8. Lu, G.L., Jin, Y., Du, D.H.C.: Frequency Based Chunking for Data De-Duplication. In: Proceedings of 18th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 287–296. IEEE Press, Miami (2010)
9. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data. In: Proceedings of 7th USENIX Symposium on Operating Systems Design and Implementation, pp. 205–218. USENIX Association, Berkeley (2006)
10. Jain, N., Dahlin, M., Tewari, R.: TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization. In: Proceedings of 4th USENIX Conference on File And Storage Technologies, pp. 281–294. USENIX Association, Berkeley (2005)
11. Bhattacharjee, S., Naranq, A., Garq, V.K.: High Throughput Data Redundancy Removal Algorithm with Scalable Performance. In: Proceedings of 6th International Conference on High Performance and Embedded Architectures and Compilers, pp. 87–96. ACM, New York (2011)
12. Debnath, B., Sengupta, S., Li, J., Lilja, D.J., Du, D.: BloomFlash: Bloom Filter on Flash-Based Storage. In: Proceedings of 31th International Conference on Distributed Computing Systems, pp. 635–644. IEEE Computer Society, Washington (2011)
13. Bender, M.A., Farach-Colton, M., Johnson, R., Kuszmaul, B.C., Medjedovic, D., Montes, P., Shetty, P., Spillane, R.P., Zadok, E.: Don't Thrash: How to Cache Your Hash on Flash. In: Proceedings of 3rd USENIX Conference on Hot Topics in Storage and File Systems, p. 1. USENIX Association, Berkeley (2011)
14. Guo, D., Wu, J., Chen, H.H., Yuan, Y., Luo, X.S.: The Dynamic Bloom Filters. *IEEE Transactions on Knowledge and Data Engineering* 22, 120–133 (2010)
15. Song, H.Y., Dharmapurikar, S., Turner, J., Lockwood, J.: Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing. In: Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 181–192. ACM, New York (2005)
16. Guo, D., Chen, H.H., Luo, X.S.: Theory and Network Application of Dynamic Bloom Filters. In: Proceedings of 25th Annual Joint Conference of the IEEE Computer and Communications Societies, pp. 1–12. IEEE Press, Spain (2006)
17. Ahmadi, M., Wong, S.: Modified Collision Packet Classification Using Counting Bloom Filter in Tuple Space. In: Proceedings of 25th IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 70–76. ACTA Press, Anaheim (2007)
18. Ahmadi, M., Wong, S.: A Memory-optimized Bloom Filter Using an Additional Hashing Function. In: Proceedings of Global Telecommunications Conference, pp. 1–5. IEEE Press, New Orleans (2008)

Estimating Deadline-Miss Probabilities of Tasks in Large Distributed Systems

Dongping Wang¹, Bin Gong¹, and Guoling Zhao²

¹ Department of Computer Science and Technology, ShanDong University, Jinan, China
wdp2006@gmail.com, gb@sdu.edu.cn

² Shandong College of Electronic Technology, Jinan, China
zhaogl@sdcet.cn

Abstract. In the past decade, large distributed systems with unreliable hosts including P2P systems and volunteer computing systems have become common. The volatility nature of resources makes it a challenge to schedule tasks with soft deadline in such systems. In this paper we examine one of the critical problems, estimating deadline-miss probabilities of tasks running on unreliable hosts. Through analysis of trace data gathered from an actual volunteer computing system, we get a general property about host's period available fraction, based on which we propose an efficient method of estimating deadline-miss probability. To evaluate the accuracy of this method, we conduct trace-driven simulations whose results show that average absolute difference between estimated probability and real ratio is smaller than 2%. To compare our method with two other methods, we simulate a scheduler which distributes task based on estimated probability. Results show that our method performs better.

Keywords: availability prediction, resource failures, stochastic scheduling, volunteer computing.

1 Introduction

In the past ten years, the advance of internet and computing technology has led to rapid development of large distributed computing systems composed of huge amount unreliable hosts. As an important part of such systems, volunteer computing systems have attracted more and more attention. Actually, volunteer computing projects such as SETI@home [1] and Folding@home [2] have sustained a computing ability of about tens or hundreds of TeraFLOPS for several years [3].

Although volunteer computing projects have pooled powerful computing ability, their applications are currently restricted to coarse grained, embarrassingly parallel applications of which the main performance metric is throughput. So, one of the main research goals in this area is to support other types of applications such as application with deadline, application with synchronization requirement, etc. [4, 5, 13]

We focus on scheduling applications with soft deadline in volunteer computing environment and deal with deadline ranging from several hours to a day. In this paper, we examine the problem of estimating deadline-miss probability of tasks running on

unreliable hosts. The accuracy of estimated probability substantially affects resource utilization efficiency.

We make a simplifying assumption that a task will be checkpointed when a resource failure occurs during its execution so that it can recover at the start of next available interval without loss of computation, which is used in [13]. Then, the period available fraction of a host determines whether a task assigned to it will miss deadline or not. Since pull-style is used in task assignment of volunteer computing system, a host only requests task at available state. So, the problem of estimating deadline-miss probability of task can be transformed into that of estimating available fraction of period starting with available state. For simplicity, we denote period starting with available state as PAS and denote period starting with unavailable state as PUS.

Through analysis of trace data from SETI@home, we get a general property about host's period available fraction and propose a method of estimating task's deadline-miss probability. In particular, our main contributions are as follows:

- We get a general property about host's available fraction during PAS: when the available fraction of a host during a particular PAS is sorted in increasing order along with available fractions of the host during randomly-chosen previous PASs using an unstable sorting algorithm, it will be at each position of the result with approximately equal probability.
- We propose an efficient method of estimating task's deadline-miss probability. The accuracy of this method is testified in trace-driven simulations. To the best of our knowledge, an estimating method with such accuracy does not exist.
- We simulate a scheduler for the application of low latency batches [13] and compare our method with other two methods in the scheduler. As a result, our method performs better than the other two.

The remainder of this paper is organized as follows. Related works about availability prediction are reviewed in section 2. The analysis of trace data is described in section 3. In section 4, we propose our method and evaluate its accuracy with simulations. In section 5, we simulate a scheduler for the application of low latency batches and compare our method with the other two methods. Conclusions are given in the last section.

2 Related Works

There have been many works about analysis of availability trace data and modeling availability [6-9, 16]. These works modeled either lengths of available durations or available fraction with probability distributions. Different from these works, this work cares about period available fraction during PAS. John Brevik et al. [10] examine the problem of determining lower bounds at a desired level of confidence for quantiles for the population of availability duration.

Availability prediction in [11, 12, 17] share a common feature: The availability history of a worker is represented as a binary string in which each binary number corresponds to the availability state of an interval such as one hour, and availability prediction is transformed into predicting binary numbers from a binary string. In [12], availability predictor for each host is implemented as a Naïve Bayes classifier and

other features are also used to strengthen the classifier. As having been noted in [11], such methods do not work well for those hosts with less regular availability patterns.

In [13], the authors propose algorithms for computing batches of medium grained tasks with deadlines in pull-style volunteer computing environments. In their algorithm, a method for predicting task deadline-meet probability is mentioned. Their method is based on statistical property about host's period available fraction. One difference between their method and our method is that their method does not distinguish PAS and PUS. Moreover, this method is partly based on regular usage pattern (mainly daily usage pattern).

3 Analysis of Trace Data

The trace data used in this section and following sections is SETI@home trace data [6] provided in FTA [14]. It is collected using BOINC [15] server for SETI@home from April 1, 2007 to January 1, 2009. It embraces CPU availability trace data from about 230,000 hosts over Internet. Each trace event corresponds to an interval, having the CPU availability state, start time and end time.

To know the relationship between a host's period available fraction during a particular PAS and those period available fractions during randomly-chosen previous PASs, we conduct simulations with trace data as follows. Each simulation repeats the following experiment 10000 times. Randomly choose a host with a point chosen at available state during its lifetime, and randomly choose n PASs before the point. The host's period available fraction during the period starting with the chosen point is denoted as f and those period available fractions of the host during chosen PASs are denoted as f_1, f_2, \dots, f_n . Sort these $n+1$ values in increasing order with an unstable sorting algorithm and record the position of f in the result. After all experiments of a simulation are completed, ratios of each position are computed.

Some details about the experiment should be noted. First, the host should be ensured to have enough availability history at the chosen point. We have tried 7 days and 10 days in simulations. Second, randomly-chosen PASs before the point may overlap. Last, the lengths of all periods in a simulation are equal and should be an acceptable length of task deadline. We have tried four different lengths in simulations, 4 hours, 6 hours, 8 hours and 12 hours.

Fig. 1 shows results of simulations in which a host at least has 7 days availability history at the chosen point. In this figure, rows show results of simulations with different n (the number of randomly-chosen PASs before the point); columns represent results of simulations with different period length. As can be seen from this figure, ratio of each position (ranging from 1 to 20) approximates to 0.05 when n is 19, ratio of each position (ranging from 1 to 50) approximates to 0.02 when n is 49 and ratio of each position (ranging from 1 to 100) approximates to 0.01 when n is 99. For those simulations in which a host at least has 10 days availability history at the chosen point, similar results are observed. So we can conclude a property for period availability fraction during PAS.

When it is sorted in increasing order along with period available fractions during randomly-chosen previous PASs using an unstable sorting algorithm, the period available fraction of a host during a particular PAS will be at each position of the result with approximately equal probability.

According to this property, if there are n randomly-chosen previous PASs, the sorting result will have $(n+1)$ positions and the approximate probability will be $1/(n+1)$.

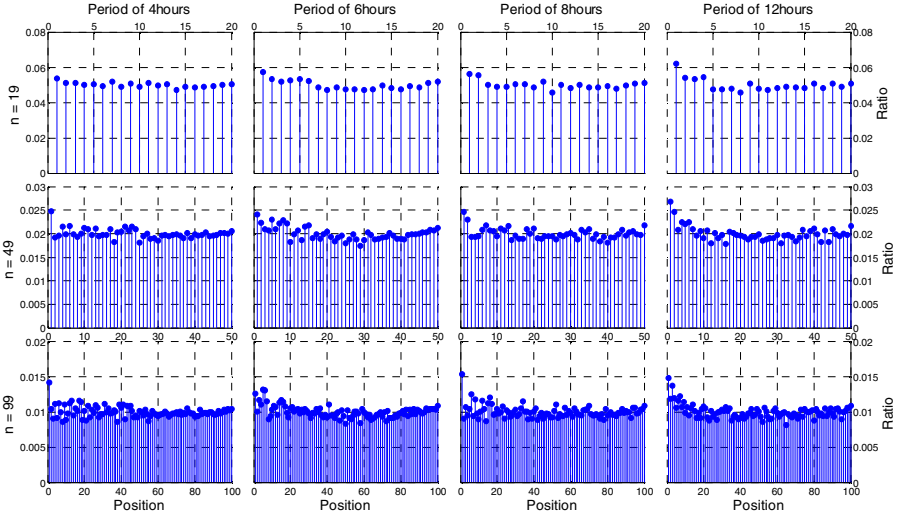


Fig. 1. Ratio of each position (at least 7 days availability history)

4 Estimating Deadline-Miss Probability

In this section, we deal with the problem of estimating deadline-miss probability of a task on a given host. Particularly, we propose an efficient method (called *Random History Method*) based on the general property which is shown in section 3.

The problem can be phrased as follows. Suppose a host receives a task at time R . The deadline of the task is time $R+D$. The time needed to finish the task on the host is C . During the execution of the task, it will be stopped at the end of an available duration and be continued at the start of the next available duration. The task recovering cost is omitted. What is the probability that the task will miss its deadline on the host?

4.1 Random History Method

Lemma 1: For given length of period D , let x be the period available fraction of the host during a particular PAS, (x_1, x_2, \dots, x_n) be period available fractions of the host during randomly-chosen previous PASs, and $(x_{(1)}, x_{(2)}, \dots, x_{(n)})$ be the permutation of (x_1, x_2, \dots, x_n) in increasing order. Then, for $1 \leq s \leq n$, the probability that x is less than $x_{(s)}$ is not more than $s/(n+1)$, i.e., $\Pr(x < x_{(s)}) \leq s/(n+1)$.

Proof. Assume α be the result of the sorting in increasing order for x_1, x_2, \dots, x_n, x with unstable sorting algorithm. Let Y be the random variable denoting the position of x in α . If x is in the i_{th} position of α , $Y=i$.

From the assumptions and the general property of period available fraction during PAS shown in section 3, we get

$$\Pr(Y=i) \approx 1/(n+1), 1 \leq i \leq n+1.$$

If $x < x_{(s)}$, then $Y < s+1$. So we get

$$\Pr(x < x_{(s)}) \leq \Pr(Y < s+1) \approx s/(n+1). \quad \square$$

According to *lemma 1*, we propose an efficient estimating method, which we term Random History Method. When n is 100, the method can be described as follows:

- *Step 1:* Randomly choose 100 PASs in the past of the host. Length of each period is D . Compute available fractions of these periods, denoted as x_1, x_2, \dots, x_{100} .
- *Step 2:* Compare the minimum necessary period available fraction needed to finish the task, C/D , with those available fractions of randomly-chosen PASs (i.e., x_1, x_2, \dots, x_{100}), and obtain three values, k , $maxl$ and $minr$. k denotes the number of available fractions which are less than C/D , $maxl$ denotes the maximum value of available fractions which are less than C/D , and $minr$ denotes the minimum value of available fractions which are not less than C/D . If k is zero, we set $maxl$ to zero. If k is 100, we set $minr$ to infinite. Then we give deadline-miss probability estimation with the following equation:

$$P_{est} = \frac{k}{101} + \frac{C / D - maxl}{101 * (minr - maxl)} \tag{1}$$

The value of n affects the accuracy of estimations in this method. Bigger n leads to more accurate estimations. However, bigger n causes bigger computation cost. So, a reasonable value of n should take both factors into consideration. 100 is one of the good choices for n .

To evaluate the accuracy of this method, we conduct simulations with trace data introduced in section 3. Each simulation has one million predictions. Each prediction involves randomly selecting a host with a time selected within its lifetime as the predicting point. If the host is in unavailable state at the predicting point, the predicting point is adjusted to the start of next available duration. We record estimated deadline-miss probability and real situation (miss deadline or meet deadline) every time a prediction is finished. After all predictions of a simulation are finished, we calculate real deadline-miss ratio for each value of estimated probability. To ensure as many prediction times as possible in calculating real ratio, we only consider discrete probability value such as 0.01, 0.02, ..., 0.99, and treat others as their smaller neighbor. For example, 0.023, 0.027 are all treated as 0.02.

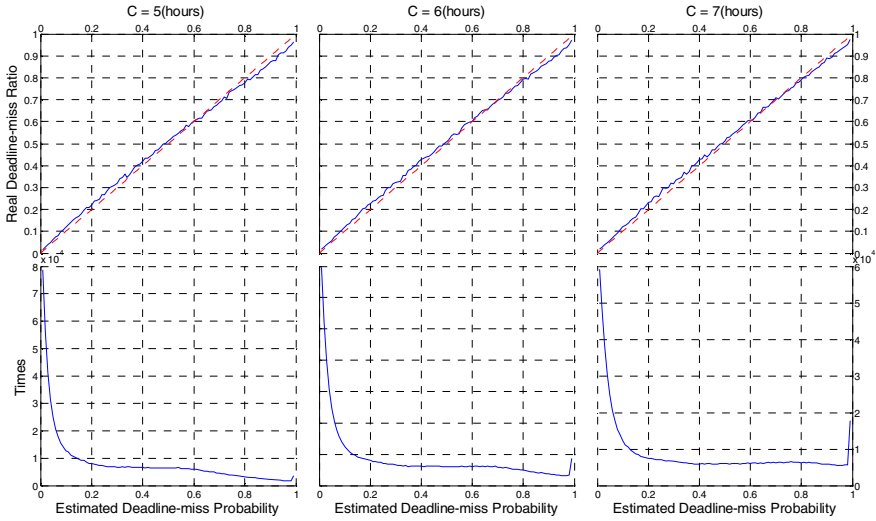


Fig. 2. Accuracy of prediction simulations for $D = 8$ hours

Fig. 2 shows the results for prediction simulations of $D=8$ hours. The first row compares estimated probability and real ratio, and the second row shows the times that specific estimated probability occurs in results, based on which real ratio is calculated. For each simulation, we calculate an average absolute difference between estimated probability and real ratio as follows: divide the sum of absolute difference between estimated probability and real ratio at picked 99 points by 99. For all simulations, average absolute difference is smaller than 2%.

4.2 Other Methods

We also examine two other methods. The first one uses Naïve Bayes Classifier [18] as its predictive model to predict period available fraction of a host, which we term the *Classifier Method*. The second one is the method of estimating deadline-meet probability proposed in [13], which we term the *Diurnal Pattern Method*.

Our implementation of the Classifier Method is partly based on the experience shown in [12] and uses similar features. To compute values of *hist* features [12], training data should be transformed into a binary (01) string, each binary of which corresponds to one hour. To determine the threshold used in transforming training data into binary string and the form of discrete classes, we design two groups of simulations. Details about the first group of simulations are listed as follows. Period length is 4 hours. Five threshold values are examined, 0.6, 0.7, 0.8, 0.9, 1.0. Two kinds of form of classes are examined. The first form uses four binary digits and each digit corresponds to one hour. So this form has 16 classes, such as 0000, 0001, 0010, etc. The second form has 4 classes (denoted as 1, 2, 3, 4), which divide the space of [0, 1] into four parts, [0, 0.25], (0.25, 0.50], (0.50, 0.75], (0.75, 1.0]. We term this form 4-level form. Similarly, we define 7-level form, 10-level form, etc. Each simulation computes 100,000 predictions. Fig. 3a shows success ratios of these

simulations. First, the success ratios are almost equal for different thresholds. If not specifically mentioned, the threshold used in subsequent simulations is 0.8. Second, 4-level form results in higher success ratio than the form of four binary digits. Because these two forms have approximately equal capability in expressing the fraction, we prefer 4-level form out of these two forms. We conduct another group of simulations to compare the following four forms, 4-level form, 7-level form, 10-level form and 20-level form. Four period lengths are tested, 2 hours, 4 hours, 6 hours and 8 hours. Results are shown in Fig. 3b. As can be seen in the figure, form with more levels leads to lower success ratio of predictions. However, the classes of a form with more levels are more accurate.

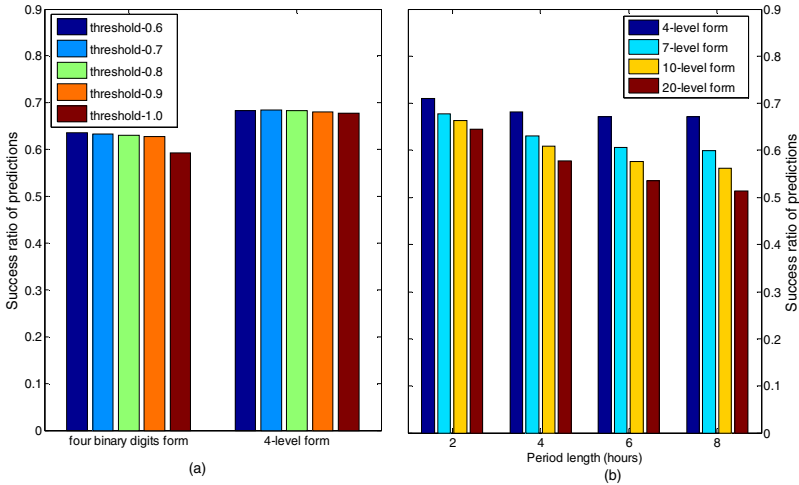


Fig. 3. Results of simulations for determining threshold and classes form

5 Simulations of Scheduling Low Latency Batches

In this section, we simulate a scheduler for low latency batches which is based on the scheduling algorithm of [13]. Each batch has many tasks with same deadline and same computation amount. When it reaches one batch's deadline, another batch is submitted until all batches are completed. At any time, there is at most one batch being scheduled. When it receives a task request from a host, the scheduler estimates the probability that a task of current batch will miss its deadline on the host and determine whether assign a task to the host based on the probability. When the scheduler decides to assign a task to a host, a task with highest cumulate deadline-miss probability is chosen and the cumulate deadline-miss probability of the task is updated accordingly. Initial cumulate deadline-miss probability of each task is 1.0. We record the number of tasks missing deadline for each batch. Parameters for simulations are listed in Table 1. For each combination of D and C , we simulate three cases with different first batch submission time and compute the ratio of tasks meeting deadline for each batch. From the trace data, we randomly choose 42,365

hosts for these simulations. In these simulations, an average of two task requests will be received for each task before batch distribution deadlines. To control these hosts for providing stable speed of task requests, the scheduler uses the Poisson method proposed in [13]. After a host gets a task, it will execute the task until the completion of the task or the end of the host's lifetime.

Table 1. Parameters for scheduling simulations

Batches number	256
Tasks per batch	4096
Batch deadline (D)	$D=4$ hours, $C=1$ hour;
Computation time per task (C)	$D=4$ hours, $C=2$ hours;
	$D=6$ hours, $C=3$ hours
First batch submission time (S_0)	Dec 1, 2007; Mar 1, 2008; Jun 1, 2008
Other batch submission time (S_i)	$S_{i+1}=S_i+D$

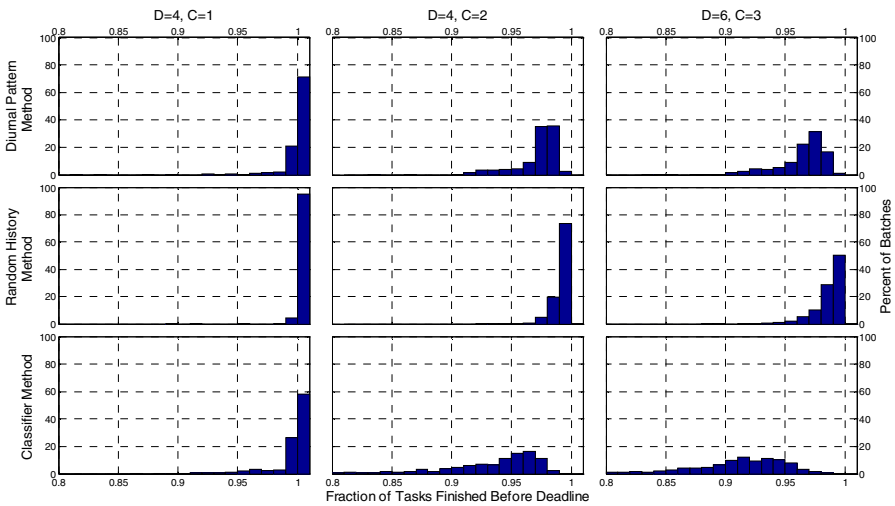


Fig. 4. Simulation results of scheduling low latency batches

The three methods introduced in last section are used in the scheduler for estimating deadline-miss probability. Details are noted here. For the Random History Method, a task request will be rejected when the estimated deadline-miss probability is bigger than 0.5. The Classifier Method uses classes of 10-level form. When a task request comes, this method predicts the available fraction class for the host and gives deadline-meet probability with $A/(A+B)$, where A denotes the predicted class of the host and B denotes the least class necessary for finishing the task before deadline. The task request is rejected when $A < B$. For the Diurnal Pattern Method, the availability

history used for predicting is ten days. In other words, we only consider ten previous tasks for each prediction. Other details about this method can be obtained from [13].

Results of these simulations are shown in Fig. 4, in which rows correspond to different estimating methods and columns correspond to different combinations of D and C . As can be seen from the figure, the Random History Method is the most efficient one among these three methods.

6 Conclusion

Finishing tasks in time is critical important for computing applications with soft deadline in large-scale systems with unreliable hosts. Duplicating tasks based on task deadline-miss probability estimations is an approach to control the ratio of tasks missing deadline. In this paper, we analyze the property of period available fraction from trace data and propose a method of estimating deadline-miss probability, called the Random History Method. Results of trace driven simulations testify the accuracy of this method. Moreover, we simulate a scheduler for low latency batches to compare the performance of our method with that of the other two methods.

The assumption used in this paper, the recovering cost of a task after a resource failure is negligible, may not be proper in some cases. So, one of our future work is to take the recovering cost into consideration in estimating method. Another plan for future work is to enhance the performance of our method with deterministic pattern of host's period available fraction.

References

1. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: an experiment in public-resource computing. *Commun. ACM* 45(11), 56–61 (2002)
2. Larson, S.M., Snow, C.D., Shirts, M.R., Pande, V.S.: Folding@home and Genome@home: Using distributed computing to tackle previously intractable problems in computational biology. In: *Modern Methods in Computational Biology*, Horizon, Marseille (2003)
3. Anderson, D.P., Fedak, G.: The Computational and Storage Potential of Volunteer Computing. In: *CCGRID 2006: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pp. 73–80. IEEE Computer Society, Washington, DC (2006)
4. Yi, S., Jeannot, E., Kondo, D., Anderson, D.P.: Towards Real-Time, Volunteer Distributed Computing. In: *CCGRID 2011: Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 154–163 (2011)
5. Bouguerra, M.-S., Kondo, D., Trystram, D.: On the Scheduling of Checkpoints in Desktop Grids. In: *CCGRID 2011: Proceedings of the 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pp. 305–313 (2011)
6. Javadi, B., Kondo, D., Vincent, J.-M., Anderson, D.P.: Mining for Statistical Availability Models in Large-Scale Distributed Systems: An Empirical Study of SETI@home. In: *MASCOTS 2009: Proceedings of the 17th Annual Meeting of the IEEE/ACM International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 1–10 (2009)

7. Nurmi, D., Brevik, J., Wolski, R.: Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments. In: Cunha, J.C., Medeiros, P.D. (eds.) Euro-Par 2005. LNCS, vol. 3648, pp. 432–441. Springer, Heidelberg (2005)
8. Wolski, R., Nurmi, D., Brevik, J.: An Analysis of Availability Distributions in Condor. In: IPDPS 2007: Proceedings of the 21th International Parallel and Distributed Processing Symposium, pp. 1–6. IEEE (2007)
9. Kondo, D., Andrzejak, A., Anderson, D.P.: On correlated availability in Internet-distributed systems. In: GRID 2008: Proceedings of the 9th IEEE/ACM International Conference on Grid Computing, pp. 276–283 (2008)
10. Brevik, J., Nurmi, D., Wolski, R.: Automatic Methods for Predicting Machine Availability in Desktop Grid and Peer-to-peer Systems. In: CCGrid 2004: Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 190–199. IEEE Computer Society (2004)
11. Mickens, J.W., Noble, B.D.: Exploiting Availability Prediction in Distributed Systems. In: NSDI 2006: Proceedings of the 3rd Symposium on Networked Systems Design and Implementation, pp. 73–86. USENIX (2006)
12. Andrzejak, A., Kondo, D., Anderson, D.P.: Ensuring Collective Availability in Volatile Resource Pools Via Forecasting. In: De Turck, F., Kellerer, W., Kormentzas, G. (eds.) DSOM 2008. LNCS, vol. 5273, pp. 149–161. Springer, Heidelberg (2008)
13. Heien, E.M., Anderson, D.P., Hagihara, K.: Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments. *J. Grid Comput.* 7(4), 501–518 (2009)
14. Kondo, D., Javadi, B., Iosup, A., Epema, D.H.J.: The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In: CCGRID 2010: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 398–407. IEEE (2010)
15. Anderson, D.P.: BOINC: a system for public-resource computing and storage. In: GRID 2004: Proceedings of the fifth IEEE/ACM International Workshop on Grid Computing, pp. 4–10 (2004)
16. Douceur, J.R.: Is remote host availability governed by a universal law? *SIGMETRICS Performance Evaluation Review* 31(3), 25–29 (2003)
17. Lázaro, D., Kondo, D., Marquès, J.M.: Long-term availability prediction for groups of volunteer resources. *J. Parallel Distrib. Comput.* (2011), doi:10.1016/j.jpdc.2011.10.007
18. John, G., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: Proc. 11th Conference on Uncertainty in Artificial Intelligence, pp. 338–345. Morgan Kaufmann (1995)

Global Pricing in Large Scale Computational Markets

Lilia Chourou¹, Ahmed Elleuch², and Mohamed Jemni¹

¹LaTICE Laboratory, Higher School of Sciences and Techniques of Tunis,
University of Tunis, Tunis, Tunisia

`lilia.gherir@planet.tn`, `Mohamed.jemni@fst.rnu.tn`

²CRISTAL Laboratory, National School of Computer Sciences,

University of Manouba, Manouba, Tunisia

`Ahmed.Elleuch@ensi.rnu.tn`

Abstract. Scale up the number of computing resources is a challenging issue when building a global computational system. For this purpose, we present an approach that adopts the commodity market model as an economic incentive model and ensures the balance between supply and demand. We show how this model may be adapted and applied to a large scale computational infrastructure. To achieve a competitive equilibrium, prices are adjusted according to a tâtonnement like process. However, this process, like several other pricing algorithms proposed in the literature, does not fulfill the scalability requirement: all prices of all commodities are computed by only one auctioneer. In the present work, a fully distributed pricing algorithm is proposed based on an existing partially distributed version. While in this last version, the price of each commodity is computed by only one auctioneer, in our algorithm, a variable number of auctioneers is used. To each auctioneer is associated a limited number of consumers and suppliers with low communication delay. Our algorithm is then scalable with respect to the number of suppliers and consumers. To evaluate our algorithm, we have performed a simulation study. For different number of auctioneers per commodity, the experimental results show that our algorithm converges as well as the partially distributed version. Moreover, by splitting the search space among auctioneers, our algorithm accelerates the convergence.

Keywords: pricing, competitive equilibrium, scalability, global computational system.

1 Introduction

One major problem facing high-throughput applications is the need for large-scale computational resources. Global computing systems are a cost-effective alternative to parallel computers. They are able to provide a great number of idle resources donated by volunteers and accessible via the Internet. However, the number of available resources still needs to be expanded. For this purpose, we have to apply an economic incentive model that would allow increasing the number of suppliers and consumers by embedding a strategy for welfare. The economic model would also be used to resolve

the load balancing problem. In this paper, we have adopted the commodity market model where transactions are governed by markets able to find a general equilibrium between the global supply and demand. These quantities are balanced through a dynamic adjustment of prices. However, existing pricing algorithms are not suited for large scale computational markets. To overcome this drawback, we propose a scalable pricing algorithm. Our algorithm uses a partially distributed solution proposed by Cheng and Wellman [6] which is founded on the principle of the centralized tâtonnement process of L. Walras [12]. In [6], the price of each commodity is computed by only one auctioneer, there is one-to-one correspondence between markets and commodities. However, this distributed solution is not suited when there is a massive number of suppliers and consumers. Our solution uses a variable number of auctioneers per commodity. To each auctioneer is associated a limited number of consumers and suppliers with low communication delay. The auctioneers of the same commodity collaborate together to find its equilibrium price. The scalability of the pricing algorithm is a fundamental property ensured by our algorithm.

To evaluate the effectiveness of our algorithm, two modelling and simulation case studies have been carried out. In the first one, the behaviour of the actors is modelled by utility functions commonly used in micro-economics and in several works implementing market mechanisms. These functions fulfill the convexity of the preferences and the gross substitutability assumptions needed to ensure the convergence of the tâtonnement process. For this first case study, the simulation results show that our distributed algorithm converges. Furthermore, it accelerates remarkably the convergence to a general equilibrium. In the second case study, the convexity of preferences is relaxed and the behaviour of the actors is modelled by more realistic utility functions. As the preferences are not convex, the demand functions are discontinuous. Consequently, if the equilibrium price does not exist, our pricing algorithm is able only to determine the two nearest prices leading to a change of the sign of the excess demand. The simulations results show that these prices will have very close values. Consequently we may consider these prices as equilibrium prices. Like in the first case study, the simulation results show that our distributed algorithm converges as well.

The remainder of this paper is organized as follows. Section 2 introduces the adopted economic model. The hypotheses to assure the existence and uniqueness of the equilibrium price are also introduced in this section. Section 3 describes the most representative pricing algorithms and shows their limits in a large-scale system. Section 4 outlines the proposed computational market model that supports our scalable pricing algorithm. This algorithm is then presented in section 5. Section 6 provides an empirical evaluation of the proposed algorithm. Finally, section 7 presents the conclusions and future work.

2 Concepts, Hypotheses and Choice of an Economic Model

Buyya et al. proposed an architecture called GRACE (Grid Architecture for Computational Economy) supporting different economic models for the management of resources in a grid [4] [5]. Among these models, we have the commodity market model, the auction model and the bargaining model. All these economic models are price oriented. We are particularly interested in the commodity market model.

In contrast with the other models, this model takes into consideration global supply and demand. It continuously looks for bringing them back into equilibrium. The equilibrium price satisfies both the suppliers and the consumers. This model represents an incentive model to build large scale computational markets.

To find equilibrium between supply and demand according to such a model, the economy must fulfill some hypotheses typically not satisfied by conventional markets [8]. For computational markets, these hypotheses should be established in an artificial way. More precisely, the following conditions of pure and perfect competition have to be met:

- The atomicity of the market: the market must include a big number of suppliers and of consumers. A supplier or a consumer is then unable alone to have an influence on the market. For the target large-scale computational systems, this condition is well satisfied.
- The homogeneity of goods: all goods having the same characteristics are considered identical. Hence, software or hardware resources with different quality of service must be differentiated as not equivalent goods.
- Independent decision making: providers and consumers are free from any restrictions with regard to their transactions. Besides, there is no collusion between providers or consumers. For a large-scale computational market, the basic resources are commodity systems owned and used by a great number of independent actors.
- No barriers to entry or departure to and from the market: suppliers and consumers are not constrained to technical, financial, regulatory or legal barrier which can hinder their entrance or exit from the market. It is also possible to pass from a production to another one. This assumption can be enforced through the use of the Internet as a marketplace. Indeed, the Internet is a worldwide network that allows a pervasive access as a utility. Besides, the production of software as a service may be shifted easily according to the demand.
- Perfect knowledge: every actor must be able to have global and complete information to take the best decisions that maximize his welfare. These requirements must be guaranteed by the information service in a computational market.
- Perfect fluidity of production factors: this term means the complete mobility of suppliers. Thanks to the ubiquity of the Internet, computing resources can be deployed in any place.
- The absence of transport costs: no costs are incurred in making transaction. In a computational market, goods are dematerialized and remotely accessible with no transport cost. All goods are computational services.

Different commodities correspond to different markets. These goods can be substituted one for the other. The price of a good has then an influence on the prices of other goods. A partial equilibrium is a state in which the equilibrium is established with respect to a specific market, and so a specific type of goods, assuming that the prices of all other goods remain unchanged. A general equilibrium is reached when all markets are cleared simultaneously. To prove the existence of Walras's equilibrium, K. Arrow and G. Debreu have showed that the hypotheses of pure and perfect

competition are not sufficient. They added other conditions about consumer's preferences and his endowment [3]:

- every consumer has a non-null initial endowment,
- his preferences must be convex (so the demand function is continuous).

Under these conditions, K. Arrow and G. Debreu showed also that any general equilibrium in pure and perfect competition leads to a Pareto efficient allocation. Besides, K. Arrow et al. [2] provide a proof of general equilibrium stability when all goods are gross substitutes. This property is verified if an increase in the price of a good increases the demand for another good.

The basic pricing approach is Walras's theory of the general equilibrium [12]. A market maker, also called central auctioneer or coordinator, gathers information to find the clearing prices. L. Walras was the first who proposed such price adjustment process called the *tâtonnement* process. For each good, the auctioneer announces, at first, an arbitrary price. If this price does not clear the difference between demand and supply, i.e. the excess demand, the price is decreased (respectively, increased) when the offer is greater (respectively, lower) than demand to encourage the demand (respectively, the offer). Due to the interdependence between markets, the new computed clearing price of a good is obtained according to the current estimated clearing price of the other goods. So, when a clearing price of one market changes, the clearing prices of the other markets may also change. By successive adjustments, the equilibrium would be progressively determined for all markets. When the excess demand is cleared for all goods at the same time, all prices are fixed and allocation of goods can be performed. However, this price adjustment process has two major drawbacks. First, even if the consumer preferences are convex and the gross substitutability assumption is fulfilled, the *tâtonnement* process can exhibit a slow convergence to equilibrium [15]. Second, the pricing algorithm is not scalable. Indeed it uses only a central auctioneer that iteratively, and for each good, submits a new price, gathers consequent supply and demand quantities and adjusts the price. If we consider a large number of suppliers and consumers, in a large-scale computing system, the communication between all suppliers and consumers on one hand, and the auctioneer on the other hand, will be a bottleneck.

3 Related Work

The WALRAS algorithm proposed by Cheng and Wellman [6] is based on the *tâtonnement* process of Walras. However, they associate to each market one auctioneer. Another important feature of the WALRAS algorithm is that the announcement of offers and demands is asynchronous. Suppliers and consumers are not necessarily negotiating the same goods at the same time using the same price information. Cheng and Wellman show under the hypothesis of convex preferences and gross substitutability between goods, that the pricing process generated by their algorithm converges to a unique equilibrium. However, when each good is considered separately, the price adjustment process remains centralized and is performed by a dedicated auctioneer. The auctioneer is then overloaded if the number of suppliers and consumers is massive. In [13], C. Weng et al. proposed to associate one auctioneer to

a group of resources which their prices are strongly correlated. They use the tâtonnement process to reach equilibrium. The authors show experimentally that their group pricing algorithm converges faster than the WALRAS algorithm. Besides, the authors propose to organize the resources of the grid into autonomous administrative domains. Each domain is governed by an agent responsible for gathering information associated to its domain and submitting the excess demand information to the auctioneer. The organisation into domains addresses the scalability problem but it resolves only the aggregation of the excess demand, the price adjustment process is still centralized.

Wolski et al. proposed the system "First bank of G" [14]. They use an adaptation of Smale's method [10] that aims to determine a trajectory for prices to find the equilibrium prices. To address the scalability problem due to the periodical interrogation of the suppliers and consumers, Wolski et al. propose to approximate each excess demand function by a polynomial used when the prices get closer to equilibrium. Wolski et al. consider complementary resources (like CPUs and disk storage). So the gross substitute property is not verified and then the uniqueness of equilibrium is not guaranteed. Besides, this system is still based on a centralised approach which is not scalable. In [16], a macroeconomic model is built for adjusting the unreasonable prices of dependent resources. However, this model supposes that the dependencies between resources, the aggregated demand and offer are known in advance.

Stuer et al. propose a pricing algorithm of substitutable resources [11]. They limit themselves to two categories of substitutable resources: slow and high speed CPUs. A consumer determines for each CPU category a factor of preference according to its speedup, its price and a weight given by him to this category. This factor is then used to fix demand. The Smale method is also used and adapted to substitutable goods context. The simulations show that the proposed algorithm converges. But the scalability problem was not addressed and the pricing algorithm remains centralized.

In [17], X. Zhao and al. propose a distributed resource pricing mechanism. It allows each provider to adjust his resource prices according to a future demand prediction. The chains of Markov are used to predict future demands. The strategy of providers is to balance the resource load and to maximize its profits. The proposed pricing model is scalable, but this approach provides incentives for providers more than the consumers. The convergence of the price-adjusting mechanism toward a general equilibrium is not guaranteed. A hierarchical approach is used in the distributed pricing algorithm called COTREE (COMBINATORIC TREE) [1]. The different actors in the system, called agents, are organized according to a logical tree. These agents are classified into two categories: the suppliers and the consumers considered as leafs of the tree, and the auctioneers. Each auctioneer is responsible for the aggregation of requests of their children (suppliers, consumers and possibly auctioneers). The simulation results show that COTREE requires significantly less messages to exchange. Besides, the pricing algorithm is scalable, but a failure of tree root is fatal.

Our algorithm uses the work of Cheng and Wellman [6] which is based on the basic concepts of general equilibrium theory. The proposed asynchronous bidding protocol is retained. To resolve the scalability problem, we consider a variable number of auctioneers for each market. The demand is aggregated, like in the COTREE system, according to a distributed model rather than a hierarchical model.

Each auctioneer builds then a demand function which is used to determine the equilibrium. To avoid a frequent polling that would slow down the pricing process, we approximate, like in [14], each demand function by a polynomial. Besides, to accelerate the price convergence, the search space is divided among the auctioneers.

4 The Distributed Computational Market Model

As we have described, in a commodity market model, pricing is achieved by a central auctioneer. To resolve the scalability problem, for each commodity, we use a variable number of auctioneers called super-peers. A proximity relationship between a super-peer and its peers must exist to reduce the communication delays. The set of super-peers that belong to the same region l is denoted SP_l ($1 \leq l \leq R$). The total number of good types is denoted K , and R is the total number of grid regions. The set of super-peers related to the same market, and so to the same type of good g , is denoted SP_g . These super-peers belong to the same pricing system and agree on sharing their pricing information. sp_g^l is the super-peer in the region l associated with the good g .

To a super-peer is associated a limited number of suppliers and consumers that we call peers. The set of peers associated to the super-peer sp_g^l is denoted PE_g^l . A partial equilibrium is reached when a super-peer sp_g^l finds a clearing price of a good g , then it announces this price to all the other super-peers that belong to SP_g . To speed up the price adjustment, the searching space is divided into contiguous subspaces explored separately and in parallel by the super-peers of SP_g . Each super-peer collects and aggregates the preference information, for an announced price, from its peers and also from super-peers associated to the same good and located in other regions. The general equilibrium is reached when prices no longer change, so each super-peer needs also to continuously receive, from the other super-peers of his region, the current clearing prices of the other goods. Using the same approach as in the WALRAS system, asynchronous communication is used between peers and super-peers. At any point of time, a peer uses the last received prices and a super-peer uses the last received preference information. The detailed algorithm of the price adjustment is described in section 5. To guarantee that pricing algorithm converges to a unique equilibrium price vector, we need to fulfill the hypothesis of gross substitutes. Complementary commodities, like CPUs and disk storage used in [14], are then not suitable. In [11], the retained approach is to build markets with substitutable commodities by clustering computing nodes into a finite set of resource categories such as high or low speed nodes. Instead of trading computing nodes, we propose to build substitutable commodities at higher level of abstraction. These commodities are services as defined in service oriented architecture. The determination of all the services is an important issue in a real global computing system. However, in this paper, we are mainly concerned by the elaboration and the evaluation of the pricing algorithm. Nevertheless, to illustrate the approach of substitutable services and to evaluate the pricing algorithm for such computational commodities, we have developed a limited case study where goods are services

remotely accessible and useful for matrix computation. A commodity is defined by the type of matrix operation and the matrix size. The substitutability between these goods is described in section 6.

5 The Pricing

Our algorithm uses the work of Cheng and Wellman [6] where the competitive equilibrium is computed via a distributed tâtonnement process (the WALRAS algorithm). While in this process, only one auctioneer is used per type of good g , in our algorithm a set of super-peers SP_g is deployed. To ensure the convergence in the same way as the proof given in [6], the behavior of each set of super-peers SP_g must be equivalent to a single super-peer per type of good g . Note that when the size of SP_g is equal to one for all types of goods, our algorithm corresponds to the WALRAS algorithm.

5.1 Super-Peer Algorithm

A super-peer is directly involved in the pricing of only a single good. The pricing algorithm applies the principle of the tâtonnement process of Walras. Whereas, according to this process and for each adjusted price, the auctioneer requests from consumers (respectively, producers) to provide their demand (respectively, supply). In our solution, a super-peer submits different potential prices and then receives the corresponding proposed quantities, i.e. preference information. It aggregates this information, received from all peers of its region, and builds an approximated local excess demand¹ function. This approximation is used to avoid a frequent polling that would slow down the pricing process. A super-peer receives also from the other super-peers, associated to the same good, their approximated local excess demand function and then builds the global excess demand function which is used to search the equilibrium price. A global demand $d_global_g^l(p_g^l(t))$ calculated by a super-peer $sp_g^l \in SP_g$ for a price $p_g^l(t)$ at an instant t , is the sum of the local accumulated demand $d_local_g^l(p_g^l(t))$ of his associated peers and the not local accumulated demands $\tilde{d_local}_g^{l'}(p_g^l(t))$ for every super-peer $sp_g^{l'}$ ($sp_g^{l'} \in SP_g \setminus \{sp_g^l\}$). The expressions (1) and (2) represent the global excess demand:

$$d_global_g^l(p_g^l(t)) = \sum_{pe_i \in PE_g^l} d_local_g^l(p_g^l(t)) + \sum_{\substack{l' \in SP_g \\ l' \neq l}} \tilde{d_local}_g^{l'}(p_g^l(t)) \tag{1}$$

or

$$d_global_g^l(p_g^l(t)) = \sum_{pe_i \in PE_g^l} d_{i,g}^l(p_g^l(t), p_{-g}^*(t - s_{-g}^i(t))) + \sum_{\substack{l' \in SP_g \\ l' \neq l}} \tilde{d_local}_g^{l'}(p_g^l(t)) \tag{2}$$

¹ The term demand will be used indifferently to refer to the supply or the demand.

where $d_{i,g}^l$ is the excess demand announced by a peer pe_i to the super-peer sp_g^l . Note that $d_{i,g}^l$ depends on the price $p_g^l(t)$ of the good g and also on the partial equilibrium prices $p_{-g}^*(t - s_{-g}^i(t))$ of the other goods, noted by $(-g)$. Each of them is updated since a delay period $s_{-g}^i(t)$ before t . If the price p_g^l computed by a super-peer sp_g^l ($sp_g^l \in SP_g$) is a clearing price, it will be announced to all peers $pe_i \in PE_g^l$ and all super-peers in SP_g . Consequently, they stop their price search process and adopt the price p_g^l . A partial equilibrium for the good g is then reached. This price p_g^l is also announced to the super-peers, of the other goods, in SP_i . A general equilibrium is reached when prices of all goods do not change anymore. So before reaching a general equilibrium, several partial equilibrium prices are computed for each good. Indeed, while a super-peer adjusts the price of a good, the prices of the other goods may change. The demand of a peer changes during time since it depends on prices of all goods and these prices also change from a partial equilibrium to another one. Like in the WALRAS system and without compromising the convergence of the pricing algorithm [6], a peer does not need to have the most recent price of every good; instead, it uses the last received value. Similarly, a super-peer uses the last preference information received from each peer. Variable communication delays are tolerated. Peers and super-peers exchange data asynchronously.

To speed up the search of a partial equilibrium price of a good g , the search space is divided into intervals. These intervals are explored in parallel by separate super-peers. The maximum price is limited according to the endowment of the consumers. Every super-peer sp_g^l limits its search to an interval I_l . A price is calculated according to the previous price and to the global excess demand which is multiplied by a step-size. A super-peer sp_g^l calculates its price p_g^l as following:

$$p_g^l = p_g^l + \lambda_g^l \cdot d_global_g^l \tag{3}$$

As in [9], the determination of the step-size λ_g^l is like a binary search. λ_g^l is divided by 2 if the global excess demand changes sign and its absolute value does not decrease significantly (it means that the equilibrium price was exceeded). Otherwise and if λ_g^l is not greater than 1/2 then it is multiplied by 2. Figure 1 describes the pricing algorithm applied by a super-peer sp_g^l .

5.2 Peer Algorithm

The peers keep a price vector of dimension K where each element of this vector is the last received equilibrium price of a specific good. When all prices of this vector are clearing prices then the exchanges can take place.

- 1) Allocate to super-peer sp_g^l an interval of search I_l
- 2) $Part_found = False$
- 3) Start a thread that repeatedly receives from local peers $\in PE_g^l$ their preferences and sends new local excess demand to super-peers $\in SP_g \setminus \{sp_g^l\}$
- 4) Start a thread that repeatedly receives from the super-peers $\in SP_g \setminus \{sp_g^l\}$ their local excess demand
- 5) Start a thread that repeatedly receives from super-peers $\in SP_g \setminus \{sp_g^l\}$ a partial equilibrium price of g , assigns it to $Part_price$ and True to $Part_found$
- 6) Start a thread that repeatedly receives from super-peers $\in SP^l \setminus \{sp_g^l\}$ partial equilib. prices of other goods
- 7) Set λ_g^l to a positive number
- 8) Repeat
- 9) Set p_g^l to a randomly value in I_l
- 10) Submit the vector of potential prices of g
- 11) $Investigate_interval = False$
- 12) while Not($Investigate_interval$ or $Part_found$)
- 13) $Binary_search(\lambda_g^l)$
- 14) if $(p_g^l + \lambda_g^l \cdot d_global_g^l(p_g^l)) \in I_l$
- 15) $p_g^l = p_g^l + \lambda_g^l \cdot d_global_g^l(p_g^l)$
- 16) if $d_global_g^l(p_g^l) < \epsilon$
- 17) $Investigate_Interval = True$
- 18) $Part_found = True$
- 19) $Part_price = P_g^l$
- 20) Send $Part_price$ to all super-peers $\in SP_g \setminus \{sp_g^l\}$
- 21) Else $Investigate_interval = True$
- 22) if Not $Part_found$
- 23) Wait the reception of partial equilibrium price p_g^l from any $sp_g^l \in SP_g \setminus \{sp_g^l\}$
- 24) Send $Part_price$ to all super-peers $\in SP^l \setminus \{sp_g^l\}$
- 25) Send $Part_price$ to all peers $\in PE_g^l$
- 26) $Part_found = False$
- 27) Until equilib. price vector does not change any more
- 28) Send to local peers $\in PE_g^l$: «*general equilibrium reached*»

Fig. 1. Pricing algorithm for a super peer $l \in SP_g$ of the region r

Besides and for a good g , each peer receives periodically a potential prices vector from a super-peer $l \in SP_g$ of his region. Consequently, a peer pe_i calculates for each price $p_g^l(t)$ (in the potential prices vector) a new demand $d_{i,g}^l$ defined by:

$$d_{i,g}^l(p_g^l(t), p_{-g}^*(t - s_{-g}^i(t))) = x_{i,g}^l(p_g^l(t), p_{-g}^*(t - s_{-g}^i(t))) - e_{i,g} \quad (4)$$

if pe_i is a consumer else,

$$d_{i,g}^l(p_g^l(t), p_{-g}^*(t - s_{-g}^i(t))) = -y_{i,g}^l(p_g^l(t), p_{-g}^*(t - s_{-g}^i(t))) \quad (5)$$

if pe_i is a supplier. $x_{i,g}^l$ (respectively $y_{i,g}^l$) is the optimal demand for the good g requested (respectively produced) by the peer pe_i given the potential price $p_g^l(t)$. The value of $x_{i,g}^l$ (respectively $y_{i,g}^l$) is determined by a maximization program resolution as described in the section 6. The demand depends also on the price of the other goods -g. For the peer pe_i , $e_{i,g}$ is the initial endowment of the good g .

Figure 2 describes algorithm of a peer i located in the region l .

- 1) Start a thread which repeatedly receives (for any good g) a potential price vector from any super-peer $\in SP^g$
- 2) Start a thread which repeatedly receives from super-peers $\in SP^l$ the partial equilibrium prices
- 3) While the message «general equilibrium» is not received
- 4) Updates for each good g the demands $d_{i,g}^l$ for all prices in the potential price vector of g Announce the demands $d_{i,g}^l$ to the super peer SP_g
- 5) Consume or produce

Fig. 2. Algorithm of a peer i in the region r

6 Simulations and Results

In order to evaluate our pricing algorithm, we have carried out two modelling and simulation case studies. In the first one, consumer preferences are represented by commonly used utility functions that fulfill the hypotheses of convexity and gross substitutability needed to ensure convergence. In the second case study, the convexity of preferences is not preserved. Nevertheless, the results of the simulations show that this last condition may be relaxed and the pricing algorithm still converges. In the two cases, a limited computational market is modeled where commodities correspond to a representative set of computing services needed for matrix computation. As we will

show later, these services are substitutable commodities and the degrees of substitutability can be used to express the utility functions.

6.1 Substitutable Commodities for Matrix Computation

In this case study, the considered commodities are a limited number of representative services for matrix computing. We consider only square matrices, their sizes are: $N \times N$ and $2N \times 2N$. The possible matrix operations are addition and multiplication. We assume that all commodities work at the same speed V . As simulations are very time consuming, we consider only four commodities denoted P1, P2, P3 and P4. Each of them is defined by the type of operation and the matrix size as described in table 1. We consider only one computation speed V but our model can easily be extended to more than one speed resulting in more commodities. The features of a commodity can be defined according to an SLA (Service Level Agreement) specification. As we are looking to evaluate the pricing algorithm, the problem of fixing the set of commodities is not an issue for the present work but it is an important work once the commodity market model is deployed in a real global computing system.

Table 1. The considered commodities

<i>Commodity</i>	<i>Operation</i>	Matrix Size
P1	Addition	$N \times N$
P2	Multiplication	$N \times N$
P3	Addition	$2N \times 2N$
P4	Multiplication	$2N \times 2N$

Table 2. The normalized memory size storage and the allocation time

<i>Commodity</i>	<i>Memory size</i>	<i>Allocation time</i>	<i>Mem. Size*Allocation time</i>
P1	3	1	3
P2	3	$2N-1 \approx 2N$	6N
P3	12	4	48
P4	12	$16N-4 \approx 16N$	192N

Table 3. Substitution matrix in terms of memory space

i\j	P1	P2	P3	P4
P1	1	1/2N	1/16	1/64N
P2	2N	1	N/8	1/32
P3	16	8/N	1	1/4N
P4	64N	32	4N	1

Recall that one of the hypotheses of convergence for the adopted tâtonnement process is the gross substitution between commodities. According to the matrix size, table 2 gives the memory size (normalized to N^2) and the allocation time (normalized

to N^2/V) needed for each good. It is then easy to deduce the matrix of substitution, described by table 3, where each element $m_{i,j}$ (with $1 \leq i, j \leq 4$) represents the quantity of the good j needed to substitute the good i in term of the memory space. In this matrix, we have assumed that 1 is negligible compared to N .

6.2 The First Case Study: Convex Preferences

The preferences of a consumer pe_i are represented by a CES (Constant Elasticity of Substitution) utility function. This function is commonly used in several works implementing market mechanisms [6] [7] [16]. The form of this function is given by the following expression:

$$utility_i(t) = \left(\sum_{g=1}^K \alpha_g \cdot (x_{i,g}(t))^\rho \right)^{1/\rho} \tag{6}$$

where α_g is a weight associated with the commodity g and ρ is a generic parameter. ρ represents the elasticity of substitution which measures the ease to substitute one good for another. In CES function, the elasticity of substitution is constant. To ensure the convexity of preferences, the range of ρ must be restricted to the interval $]-\infty, 1[$. As in WALRAS algorithm [6], we set ρ to 0.5. We fix each α_g coefficient to the substitutability's degree of the commodity g (as described in table 3).

A peer pe_i has an initial endowment, in particular, the currency is considered as being a good and every consumer has an initial monetary budget which will be periodically refreshed. The initial endowment of the peer pe_i for the good g is denoted $e_{i,g}$. This quantity can be sold or exchanged on the market (so a consumer may also be a supplier). In this exchange economy, the value of the initial endowment of a peer is its wealth. The wealth of a peer pe_i at the instant t is defined by:

$$p(t) * e_i = \sum_{g=1}^K p_g(t) * e_{i,g} \tag{7}$$

Each consumer aims to determine an optimum demand which maximizes, under a budgetary constraint, the quantity of computing to be performed. So a consumer pe_i must resolve the following maximization problem:

$$\max utility_i(t) \text{ under constraint } \sum_{g=1}^K p_g(t) * x_{i,g}(t) \leq \sum_{g=1}^K p_g(t) * e_{i,g} \tag{8}$$

where $x_{i,g}$ ($1 \leq g \leq K$) is the demand announced by the consumer pe_i for the commodity g . We have implemented a discrete event simulator that simulates the computational market model and the pricing algorithm proposed in the previous sections. Fifty computational economies and fifty trials have been generated each with four commodities, sixteen consumers and sixteen suppliers. The suppliers

(respectively, consumers) were given randomly-generated memory size that limits the feasible production (respectively, currency endowments, so each consumer have only currency as good). These economies are used to average each point in the following comparison charts. Vertical lines at the top of a bar indicate the minimum and the maximum values for the fifty trials. To simulate the asynchronous behavior of peers, we assume that the number of new demands, some bids for a subset of commodities, follows a random draw². When a peer does not submit a new demand for a commodity, the super-peer considers its demand from the last iteration. Each super-peer, upon receiving the demands, computes the clearing price. Then it notifies its peers (the bidders) the new price. Peers do not possess the same state of price information on which they compute their next demands. Our developed simulator follows iteratively this process on each super-peer until the equilibrium prices are reached. For each iteration (called a cycle), a random number of super-peers compute new clearing prices.

The simulations results show that our distributed algorithm converges to a unique equilibrium for each generated economy. Moreover, we have varied the number of super-peers per commodity and have measured the number of cycles and the number of explored prices to achieve the general equilibrium (i.e. prices computed as described by the line 20 of figure 1). Since the super-peers work in parallel, at each cycle the number of considered prices is taken as the maximal value among all super-peers. Figure 3 gives the average number of cycles and the average number of computed prices to reach the general equilibrium, for different numbers of super-peers associated with the same commodity. Recall that the WALRAS algorithm corresponds to the case where only one super-peer is used per commodity. It should be noted that increasing the number of super-peers does not result in a worse number of cycles. Moreover, a significant improvement of the average number and the maximal number of cycles is observed. Hence, the asynchrony of the bidding process – implying that new partial equilibrium prices are computed according to not fully updated information – is not emphasized by using more than one super-peer per commodity. Besides, both average and maximal number of computed prices decreases significantly when the number of super-peers is increased. Indeed, a parallel search is performed and the search space assigned to each super-peer is reduced by increasing the number of super-peers per commodity.

6.3 The Second Case Study: Linear Preferences

Let us consider the simple case where preferences of consumers are represented, at the time t , by a linear utility function as described by:

$$utility_i(t) = \left(\sum_{g=1}^K \alpha_g \cdot x_{i,g}(t) \right) \quad (9)$$

² Like in WALRAS, our simulator chooses with equal probability from the set $\{0, 1, 2\}$. So each peer submits an average of one demand per cycle.

Like in the first case, each consumer aims to determine an optimum demand under the budgetary constraint as described by the expression (8). As the preferences are not convex, the demand functions are discontinuous. Consequently, if the equilibrium price does not exist, our pricing algorithm is able only to determine the two nearest prices leading to a change of the sign of the excess demand. If these two prices have close enough values, we may consider these prices as equilibrium prices. A general equilibrium is reached when prices of goods do not change anymore or oscillate between two close values. For all simulations, we calculate the difference between oscillating prices and we found that these gaps are negligible compared to the corresponding prices. Like in the first case study, the simulation results show that our distributed algorithm converges well, as it is shown in figure 4. These results suggest that for linear preferences the pricing algorithm converges well to equilibrium and behaves as in the case of convex preferences.

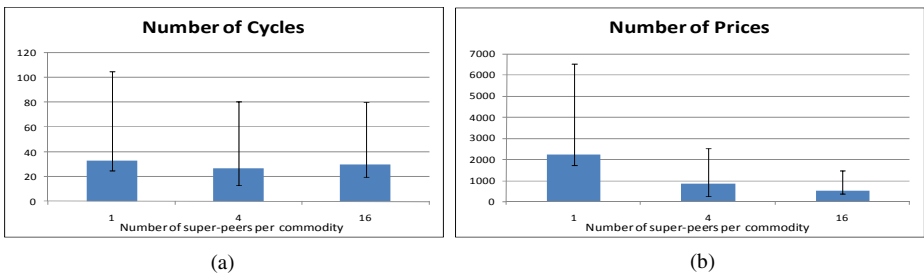


Fig. 3. Convex preferences case: (a) Average number of cycles. (b) Number of estimated prices.

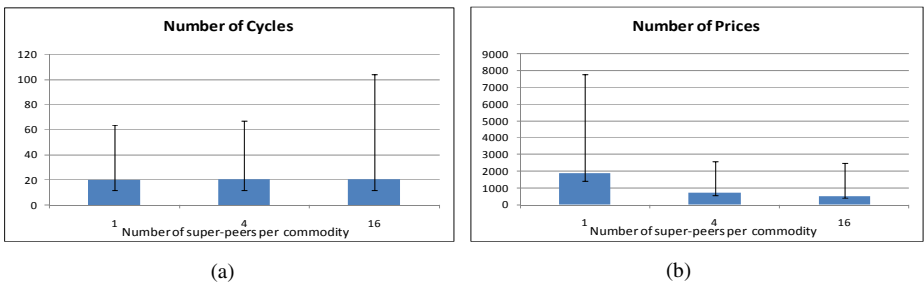


Fig. 4. Linear preferences case: (a) Average number of cycles. (b) Maximal number of estimated prices according to the number of super peers by good.

7 Conclusion and Perspectives

In this paper, we have shown how the commodity market model may be applied to build large scale computational systems. A distributed pricing algorithm is proposed for achieving a competitive equilibrium. Prices are adjusted according to a tâtonnement like process. The search space is divided into intervals, explored by separate coordinated auctioneers, to speed up the convergence of the pricing algorithm. Although the convergence guarantee depends on several conditions, in

particular the preference convexity assumption, we have found empirically that our algorithm converges even if the preferences are linear. As a future work, we intend to include the trust granted to each announced offer or demand in the pricing process. A reputation mechanism will be used to evaluate the trust level of each bidding peer.

References

1. Andersson, A., Ygge, F.: Managing large scale computational markets, Research Report 4/98, Department of Computer Science and Business Administration, University of Karlskrona/Ronneby, Sweden
2. Arrow, K., Block, H.D., Hurwicz, L.: On the stability of competitive equilibrium 2. *Econometrica* 27, 82–109 (1959)
3. Arrow, K.J., Debreu, G.: Existence of an equilibrium for a competitive economy. *Econometrica* 22, 265–290 (1954)
4. Buyya, R., Giddy, J., Abramson, D.: A Case for Economy Grid Architecture for Service-oriented Grid Computing. In: IPDPS 2001: Proceedings of the 10th Heterogeneous Computing Workshop (HCW 2001), San Francisco, California, USA, pp. 83–88 (April 2001)
5. Buyya, R., Stockinger, H., Giddy, J., Abramson, D.: Economic models for management of resources in peer-to-peer and grid computing. In: SPIE International Symposium on the Convergence of Information Technologies and Communications (ITCom 2001), August 20–24, Denver, Colorado (2001)
6. Cheng, J.Q., Wellman, M.P.: The WALRAS algorithm: a convergent distributed implementation of general equilibrium outcomes. Kluwer Academic Publishers (1998)
7. Gomes, E.R., Kowalczyk, R.: Learning in market-based resource allocation. In: 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007), pp. 475–482 (2007)
8. Knight, F.H.: Risk, uncertainty, and profit. Hart, Schaffner, and Marx Prize Essays, vol. 31. Houghton Mifflin, Boston (1921)
9. Sandholm, T., Ygge, F.: Constructing speculative demand functions in equilibrium markets. WUCS-99-26 (1999)
10. Smale, S.: Price adjustment and global Newton methods. *Frontiers of Quantitative Economics*. IIIA, 191–205 (1975)
11. Stuer, G., Vanmechelen, K., Broeckhove, J.: A commodity market algorithm for pricing substitutable grid resources. *Future Generation Computer Systems* 23(5), 688–701 (2007)
12. Walras, L.: Elements of pure economics. Allen and Unwin Originally published (1874) English translation by William Jaffé (1954)
13. Weng, C., Lu, X., Deng, Q.: A Distributed Approach for Resource Pricing in Grid Environments. In: Li, M., Sun, X.-H., Deng, Q.-n., Ni, J. (eds.) GCC 2003. LNCS, vol. 3033, pp. 620–627. Springer, Heidelberg (2004)
14. Wolski, R., Plank, J.S., Brevik, J., Bryan, T.: Analyzing market-based resource allocation strategies for the computational grid. *International Journal of High Performance Computing Applications* 15(3), 258–281 (2001)
15. Ygge, F.: Market-oriented programming and its application to power load management, Ph.D. thesis, Department of Computer Science, Lund University (1998)
16. Yuan, L., Zeng, G., Wang, W.: A Grid resource price-adjusting strategy based on price influence model. In: Proceedings of Fifth International Conference on Grid and Cooperative Computing (GCC 2006), pp. 311–318 (2006)
17. Zhao, X., Xu, L., Wang, B.: A Dynamic price model with demand prediction and task classification in grid. In: Proceedings of the Sixth International Conference on Grid and Cooperative Computing Table of Contents, pp. 775–782 (2007)

A New RBAC Based Access Control Model for Cloud Computing

Zhuo Tang¹, Juan Wei¹, Ahmed Sallam¹, Kenli Li¹, and Ruixuan Li²

¹ College of Information Science and Engineering, Hunan University,
Changsha 410082, China
hust_tz@126.com

² School of Computer Science and Technology, Huazhong University of Science and
Technology, Wuhan 430074, Hubei, China

Abstract. Access Control is an important component of Cloud Computing; specially, User access control management; however, Access Control in Cloud environment is different from traditional access environment and using general access control model can't cover all entities within Cloud Computing, noting that Cloud environment includes different entities such as data owner, end user, and service provider. In this paper, we propose a new access control based on Role-based access control (RBAC) model. This model includes two kind of roles, user role (UR) and owner role (OR); such that, Users get credential from owners to communicate with service provider and to get access permissions of resources. We also discuss the aspects of user access control management, such as authentication, privilege management, and deprovisioning. Moreover, we use administrative scope to update hierarchy when there is a role added or revoked to simplify the user access control management. By applying the model in Cloud environment the results shows that it can reduce the security problems to two classes in the RT [\leftarrow , \cap] role-based trust-management language with a test-paper system.

Keywords: Cloud computing, access control, user access control management, security analysis.

1 Introduction

Over the passed few decades, many access control models have been proposed to specify the different access control policies. These models primarily include DAC, MAC and RBAC. In DAC [1] the creator controls all the objects and the object owner has rights to give the access permissions to others; however, it is difficult to control the permission hierarchy. MAC [2] on the other side cannot implement distributed management. A better alternative is the RBAC [3] that is a widely used access control mechanism, which associates permissions and roles; such that, users get the corresponding permissions by playing roles in RBAC.

In the RBAC family which was proposed by Sandhu in 1996 [7], users' privileges are attached to their roles, where the users can acquire when needed. A role is a

permission set for a special work station. When the users' privileges need to be changed, we have to revoke the user role or re-distributing the roles.

James B.D. Joshi et al. [4, 5] extended RBAC theory. In many practical scenarios, users may be restricted to assumed roles only at predefined time periods. GTRBAC allow expressing the role hierarchies and separating of duty (SoD) constraints [6] for specifying fine-grained temporal semantics.

Li et al. [8] proposed the notion of security analysis and studied security analysis in the context of $RT[\leftarrow, \cap]$, a role-based trust-management language. They showed that a security analysis instance in $RT[\leftarrow, \cap]$ involving only semi-static queries can be solved efficiently. The work by Koch [9] analyzes the safety in RBAC with the RBAC state and state-change rules posed as graph formalism.

Li and Tripunitara [10] performed security analysis on two restricted versions of administrative RBAC. These are known as AATU and AAR. They proposed two reduction algorithms and studied complexity results for various analysis problems such as safety, availability and containment.

Jason Crampton et al. [11] and Koch et Al. [12] have introduced the administrative scope in a role hierarchy and develop a family of models for role hierarchy administration. The administrative role could update the role hierarchy dynamically when some roles added or deleted. Youngmin Jung Et al. [13] have proposed an adaptive security model for Cloud Computing environment. The model based on the improved RBAC model and adapts a role switching model. Weichao Wang et al [14] have discussed the outsourced data security. But this paper only considered the data update, and they did not consider the entities update.

2 CARBAC: The RBAC for Cloud Computing Environment

2.1 Models and Definitions

RBAC is a very useful access control model, despite its benefits it is fundamentally limited by its subject-centric nature. The CARBAC model proposed in this section addresses these situations by incorporating support for environmental state and object state in access control policies. Because the resources stored in Cloud Computing servers are very huge, we have to divide resources to data blocks. Our model focuses on the access control in the SaaS delivery model [15] where applications are delivered as a service to end users. In the following we list a set of important definitions.

User Role (UR): A user role in our model is similar to RBAC role. The user is an individual within an organization gets his/her access permission through the role assigned to him.

Owner Role (OR): It is the set of permissions that data owner has to give credential to users and update Cloud resources. Data owner processing users' request through OR.

Request: User's access request is denoted as $Req=(UR, OR, request\ index, data\ block, op)$, where "request index" is to label user's access request., "data block" is to label the data block the user requests. And "op" is the operation that the user will apply on resources.

Role hierarchies are almost inevitably included whenever roles are discussed in the literature [16, 17]. In the following, we define role hierarchies and describe their semantics.

We define a partial order set $\langle R, \leq \rangle$ and R is the set of roles. We denote x covers y as $y \leq x$, and if there is an edge between the role x and the role y , x is called the parent role of y . And the role x can inherit the permission of role y .

We use the administrative scope definition introduced in Jason Crampton et al. [12]. When a role is added to or deleted from the hierarchy, the administrative role will dynamically change the administrative scope. The model can be seen from figure1. UR and OR can update the role hierarchy by themselves. User role hierarchy and owner role hierarchy has its administrator. The administrator of owner role hierarchy administrates the relationship between OR and UR while OR has the responsibility to manage users' information and the credential of user. Roles assigned to user with the following constraints:

- 1) Only one role can be assigned to one user in one session, and one user cannot be signed to two roles at the same time.
- 2) Every user has a tag to note the role he/she assigned to.

2.2 User Access Control Management

This paper discusses the user access control management in universal cloud computing environment which includes authentication, privilege management, and deprovisioning. The authentication control is among end users, data owner, and service provider. The privilege management is for some special users.

Authorization and Authentication

In an access session S , as Figure 2 shows, the process can be divided into two parts: $\langle UR, OR, Req \rangle$ and $\langle UR, SP, P \rangle$, in which UR is the set of user roles, OR is the set of data owner roles, SP is the service provider, P is the set of permissions, and Req represents the set of user's access requests where $Req = (UR, OR, request\ index, data\ block, op)$.

The authorization procedure works as follows.

1. UR sends a data access request to OR . $UR \rightarrow OR: \{UR, OR, Req\}$. The request includes the index number of data block that the user wants to access.
2. When data owner receives the request; it will check the integrity and lateness of the message, Then OR needs to check whether UR has the right to access the data blocks which it request to access. If UR passes the check, the OR will send a reply to UR . $OR \rightarrow UR: \{OR, UR, Req, Cred\}$. Where the $Cred$ is a Credential to the service.
3. UR will send the packet $\{UR, SP, request\ index, Cred\}$ to the service provider., then the service provider firstly authenticate $Cred$, and that the $Cred$ is generated by OR , because the $Cred$ is encrypted by a secret key which is only known by OR and SP ; moreover, The $Cred$ has the information of data user and date owner. Next, S will send the permission to UR . So UR gets the access rights to the data blocks.

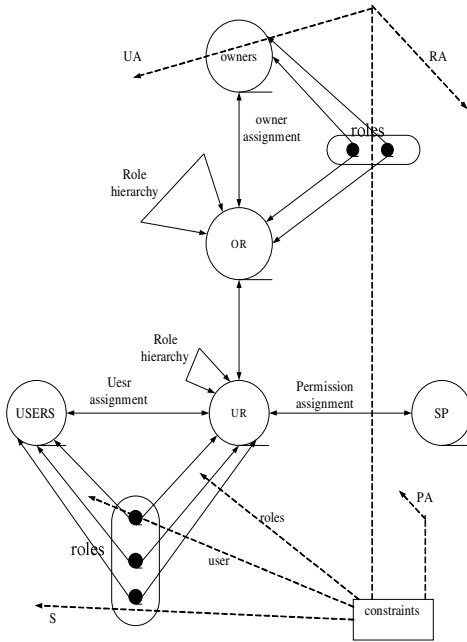


Fig. 1. An access control based RBAC for cloud computing

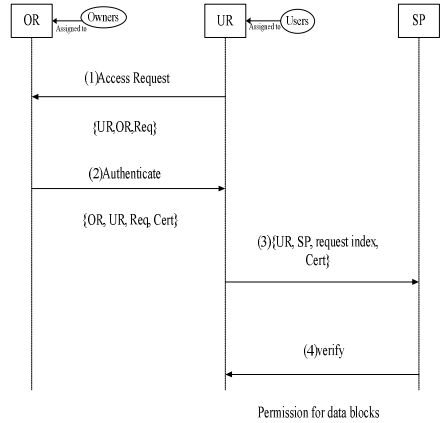


Fig. 2. Authorize Process

The request sent by the end user who has the privilege to send access requests to the data owner is noted as ReqP, and it has the following form.
 $ReqP = (UR, OR, ID, request\ index, data\ block, op)$.

Deprovisioning and Insertion

When a new role node is added to the system or a role is deleted from the role hierarchy, we use the administrative scope model to update the role hierarchy; this procedure includes operations on roles and edges. These operations can be denoted as $AddRole(a, r, \Delta r, \nabla r)$, $DeleteRole(a, r)$, $AddEdge(a, c, p)$, $DeleteEdge(a, c, p)$ respectively., where a is an administrative role, r is the new role, Δr is the immediate child role of role r , ∇r is the immediate preceding role of role r , c is the child role, and p is the parent role. Mentioning that, OR give credential to UR according to the first come first served with high priority first principles.

Suppose a new role is inserted to the hierarchy. When it wants to access data blocks, first it needs to get access authorization, so it gains access rights in the same way as described in section 3.1. And the service provider and the data owner do not change to adapt to the update. On the other side, if a user role is removed from the hierarchy, besides the UR hierarchy update, the OR also needs to update its Credential management. It labels the data block which the UR was accessing to note that the UR does not have further access to that data block. If a user needs to access the data block again, it is necessary to gain new access rights from the data owner and service

provider and if a user needs to change the accessed data block, it sends access request to the data owner, the data owner delete its access rights of the last block and sends a new Credential to the UR. Also note that, the data block which the user does not access is also labeled.

2.3 Instances

In a typical Cloud Computing environment, suppose there are three users: Alice, Bob and Carol. Alice and Bob are assigned to the same role E., see UR and OR hierarchies in Figure 3. Alice requests to access the data block DB1, Bob requests to access the data block DB2 and Carol request to access the data block DB3. Following the proposed model.

(1) Alice sends access request to the owner role OR1 through role E, consider that Alice has higher priority than Bob and Bob cannot get edit permission from E. this request can formally be written as following:

$$\text{Alice} \rightarrow \text{OR1: } \{E, \text{OR1, Req}\} \tag{1}$$

Req = { UR=E,OR=OR1 , request index=1, data block=DB1,permission=edit}

Alice is the first time for Alice to send request, the request index is “1”. And permission =edit means that the access rights Alice needs to gain is to edit the data block.

(2) The role OR1 authenticates the message and replies with the Credential. At the same time, it label the data block in service.

$$\text{OR1} \rightarrow \text{E: } \{\text{OR1, E, Req, Cred}\} \tag{2}$$

Req={UR=E,OR=OR1 , request index=1, data block=DB1,permission=edit} and Cred is a Credential for Alice.

(3)When E receives the message, it first checks the freshness of the message and then sends it to the service provider, and waits to receive the access permission.

$$\text{E} \rightarrow \text{SP: } \{E, \text{SP, request index, Cred}\} \tag{3}$$

The SP receives the request, and checks whether the Cred matches role E. then it sends the Edit permission to role E; consequently, user Alice gets the edit permission to DB1.

When the role PE is deleted, its administrator will send a notification to the data owner OR. The notification contains the ID of the role PE and data block. Next, OR labels the data block to show that the role’s rights have been revoked and the notification of the information is as following:

$$\text{M} \rightarrow \text{OR: } \{\text{PE, OR, ID, data index}\}.$$

When the user wants to access the same data block again, it will need to send an access request to OR.

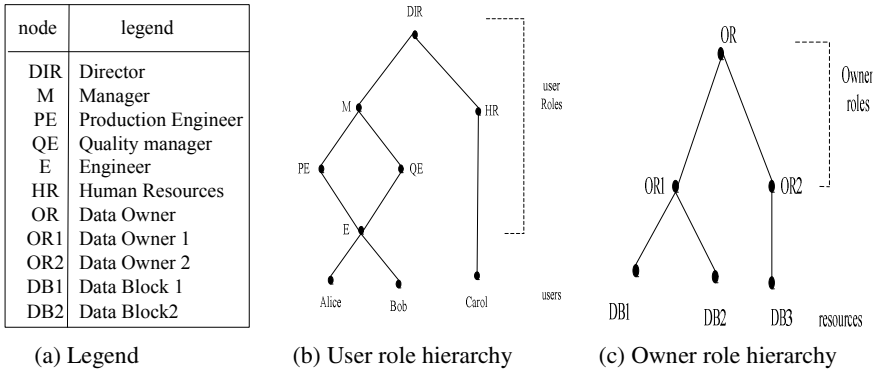


Fig. 3. Application of access control model for Cloud Computing

3 The Security Analysis

In a multi-users environment, access control is usually used to control and protect resources. When an access control policy is made, some security problems should be considered, for example, in the given authorization status and policy description, whether a subject can access the object safely? Security analysis techniques answer two questions: whether an undesirable state is reachable and whether every reachable state satisfies some safety or availability properties.

In Li et al. [8], a version of security analysis is defined in the context of trust management, and Li et al [10], introduced solutions to two classes of security analysis problems, which are assignment and trusted users (AATU) and assignment and revocation (AAR).

3.1 Reduction for AATU

Given an AATU security instance in Cloud Computing $\langle \gamma = \langle UA, PA, RA \rangle, q = u_1 \sqsubseteq u_2, \varphi = \langle can_assign, T \rangle, \Pi \in \{\exists, \forall\} \rangle$. The can_assign relation determines who can assign users to roles and which preconditions the users should satisfy, where $can_assign = \{ \langle DA, GT, \{Expertss\} \rangle, \langle SA, true, \{GU, DA\} \rangle \}$. DA means the department administrator, GT is a general teacher, SA standards the school administrator, and GU is general users. DA, GT, SA, and GU are all end users in the Cloud. Let q reduction results take the form of $\langle \gamma^T, q^T, \varphi^T \rangle$. q^T is $Sys.Experts \sqsubseteq \{Alice\}$.

We use the principal Sys to represent Cloud Computing RBAC system. The RT $[\leftarrow, \cap]$ role Sys.ur represents the user role r in Cloud Computing, and the RT $[\leftarrow, \cap]$ role Sys.p represents the cloud computing RBAC system permission p. Each $(u, ur) \in UA$ is translated into the RT $[\leftarrow, \cap]$ statement $Sys.ur \leftarrow u$, and Each $r_1 \leq r_2$ means that a member of r_1 is also a member of r_2 .

According to Figure 4, the following RT statements in γ^T result from UA:

- Sys.Students \leftarrow Alice Sys.GT \leftarrow Bob Sys.DA \leftarrow Carol

the following statements in γ^T result from PA:

- Sys.View \leftarrow Sys.DA Sys.Edit \leftarrow Sys.GT

and the following statements in γ^T result from RA:

- Sys.GU \leftarrow Sys.Students Sys.GU \leftarrow Sys.GT Sys.GT \leftarrow Sys.Experts
- Sys.DA \leftarrow Sys.SA Sys.Access \leftarrow Sys.Students

In Figure 3, GT stand for General Teachers, GU stand for General Users, DA represents Department Administrator, and SA represents the School Administrator. Roles are shown in solid boxes, permissions in dashed boxes, and users in ovals. A line segment represents a role–role relationship, the assignment of permission to a role, or the assignment of a user to a role. The following statements in γ^T result from can_assign; such that, the first two statements reflect the ability of a member of Department Administrator to assign users to Students and General Teachers with no prediction, then the remaining statements will reflect the ability of a member of School Administrator to assign users to Experts provided that they are already members of Department Administrator and General Teachers.

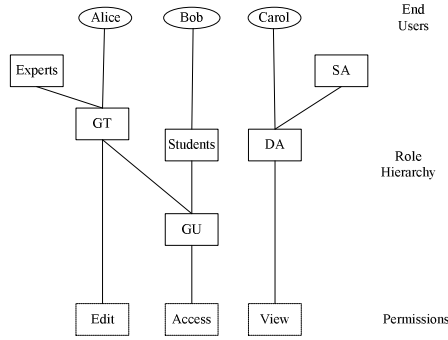
- Sys.Students \leftarrow Sys.DA.Sdtudents Sys.GT \leftarrow Sys.DA.GT
- Sys.NewRole1 \leftarrow Sys.DA \cap Sys.GT Sys.NewRole2 \leftarrow Sys.Experts
- Sys.Experts \leftarrow Sys.NewRole1 \cap NewRole2

$\gamma^T = \langle G, S \rangle$, where G is the [10] set of growth-restricted roles and S is the [10] set of shrink-restricted roles. Statements defining roles in G can't be added, and statements defining roles in S cannot be removed. Now, it is easy to see that the security analysis instance $\langle \gamma^T, q^T, \phi^T, \exists \rangle$ is false, as Alice is not a member of the Experts.

3.2 Reduction for AAR

In Cloud Systems, we use five special principles: Sys, RSys, ASys, HSys, BSys. The Sys roles simulate RBAC permissions, the roles of RSys contain all the initial roles memberships in UA, ASys roles represent every user that exercises the user-role assignment operation, Hsys.r maintains the history of the user role r. And BSys is similar to ASys, but it is used to construct the HSys roles.

An example of a state-change rule in AAR is $\phi = \langle can_assign, can_revoke \rangle$, where can_assign is the same as in AATU and $can_revoke \subseteq R \times 2^R$ determines who can remove users from roles. can_revoke consists of two tuples $\langle DA, \{GT, Experts\} \rangle$ and $\langle SA, \{GU, DA\} \rangle$.



$$\begin{aligned}
 RA &= \{(Experts, GT), (GT, GU), (students, GU), (SA, DA)\} \\
 PA &= \{(Edit, GT), (Access, GU), (View, DA)\} \\
 UA &= \{(Alice, GT), (Bob, Students), (Carol, DA)\}
 \end{aligned}$$

Fig. 4. An example RBAC state for cloud computing

Let \mathcal{Y} be the same as in AATU and q is $Experts \underline{=} Alice$. The output of AAR is $\langle \mathcal{Y}^T, q^T, \phi^T \rangle$. q^T is $Sys.Expert \underline{=} \{Alice\}$.

The following RT statements in \mathcal{Y}^T result from UA:

- $HSys.Students \leftarrow Alice$ $RSys.Students \leftarrow Alice$ $HSys.GT \leftarrow Bob$
- $RSys.GT \leftarrow Bob$ $HSys.DA \leftarrow Carol$ $RSys.DA \leftarrow Carol$
- $Sys.Students \leftarrow RSys.Students$ $Sys.GT \leftarrow RSys.GT$ $Sys.DA \leftarrow RSys.DA$,

the following statements in \mathcal{Y}^T result from RA:

- $Sys.GT \leftarrow Sys.Experts$ $HSys.GT \leftarrow HSys.Experts$ $Sys.GU \leftarrow Sys.Students$
- $HSys.GU \leftarrow HSys.Student$ $Sys.GU \leftarrow Sys.GT$ $HSys.GU \leftarrow HSys.GT$
- $Sys.DA \leftarrow Sys.SA$ $HSys.DA \leftarrow HSys.SA$,

the following statements in \mathcal{Y}^T result from PA:

$Sys.Edit \leftarrow Sys.GU$ $Sys.Access \leftarrow Sys.Students$ $Sys.View \leftarrow Sys.Administrator$,

and the following statements in \mathcal{Y}^T result from can_assign:

- $HSys.GU \leftarrow BSys.GU$ $Sys.GU \leftarrow ASys.GU$ $HSys.DA \leftarrow BSys.DA$
- $Sys.DA \leftarrow ASys.DA$ $Sys.NewRole1 \leftarrow HSys.GU \cap HSys.DA$
- $HSys.Experts \leftarrow BSys.Experts \cap Sys.NewRole1$
- $Sys.Experts \leftarrow ASys.Experts \cap Sys.NewRole1$

$\phi^T = \langle G, S \rangle$, where G is the set of growth-restricted roles and S is the set of shrink-restricted roles. Moreover, there exists a state \mathcal{Y}_1^T that is reachable from \mathcal{Y}^T and has the following statements in addition to the ones in \mathcal{Y}^T .

- $BSys.D \leftarrow Alice$ $ASys.Experts \leftarrow Alice$

We can now infer that in γ_1^T , $\text{HSys.DA} \leftarrow \text{Alice}$ and, therefore, $\text{HSys.NewRole1} \leftarrow \text{Alice}$, and $\text{Sys.Experts} \leftarrow \text{Alice}$. Thus the security analysis instance $\langle \gamma_1^T, q^T, \phi^T, \exists \rangle$ is true.

If we consider instead, the query q_1^T , which is $\text{Sys.Experts} \underline{\underline{=}} \text{Alice}$, then as RSystem.GT is a shrink-unrestricted role, there exists a state γ_2^T that is reachable from γ^T in which the statement $\text{RSystem.GT} \leftarrow \text{Alice}$ is absent. Therefore, we would conclude that Sys.Experts does not include Alice. So the analysis instance $\langle \phi^T, q_1^T, \gamma^T, \forall \rangle$ is false.

We need to consider if only GT has edit right to a test paper and if David has right to access the Testpaper.

Given a state γ , state-change rule

$$\phi = \{ \langle \text{Students, Access, Test paper} \rangle, \langle \text{GT, Edit and Access, Test paper} \rangle \}$$

is authorized to user assigned to GT has right to access and edit the Testpaper. And users assigned to SA has right to assign Students, GT, Experts, DA, or SA to users. For the query $q = \{ \text{David} \} \underline{\underline{=}} \text{Access}$ and $\gamma \rightarrow_{\phi} \gamma_1$, the instance $\langle \gamma, q, \phi, \exists \rangle$ asks whether there exists a reachable state γ_1 such that end user David can access the Testpaper. It is clear that the instance is true. For the query $q = \text{Edit} \underline{\underline{=}} \{ \text{David} \}$, and $\gamma \rightarrow_{\phi} \gamma_1$, the instance $\langle \gamma, q, \phi, \exists \rangle$ asks whether there exists a reachable state γ_1 such that end user David can edit the Testpaper. It is clear that the instance is false.

4 Conclusion

In this paper we introduce a new access control model for Cloud Computing based on RBAC Where both the user and the data owner have a corresponding role hierarchy. Each role hierarchy has administrative roles, so that it can be updated dynamically if a new role is inserted or deleted from the role hierarchy; moreover, the role hierarchy can update roles relationship by administrator scope. We discuss the authentication, privilege management, and deprovisioning of the user access control.

We also analyze the security of the access control system in a test-paper Cloud Computing system and reduced the security problem to two classes of AAU and AAR using the $\text{RT}[\leftarrow, \cap]$ role-based trust-management language. The results indicate that our proposed model can satisfy the security needs in Cloud Systems.

Acknowledgments. This work is supported by the National Postdoctor Science Foundation of China (20100480936).

References

1. Osborn, S., Sandhu, R., Munawer, Q.: Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security* 3(2), 85–106 (2000)
2. Jiang, Y., Lin, C., Yin, H., Tan, Z.: Security Analysis of Mandatory Access Control Model, *Systems, Man and Cybernetics* 6, 5013–5018 (2004)
3. Ferraiolo, D., Kuhn, R.: Role-based access controls. In: 15th NIST-NISC National Computer Security Conference, October 13–16, pp. 554–563. Baltimore, MD (1992)
4. Joshi, J.B.D., Bertino, E., Ghafoor, A.: A Generalized Temporal Role-Based Access Control Model. reference IEEECS (accepted December 9, 2003. Published online November 18, 2004)
5. Joshi, J.B.D., Bertino, E., Ghafoor, A.: Temporal hierarchies and inheritance semantics for gtrbac. In: SACMAT 2002: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, pp. 74–83. ACM, New York (2002)
6. Li, N., Tripunitara, M.V., Bizri, Z.: On mutually exclusive roles and separation of duty. *ACM Transactions on Information and System Security* 10(2) (May 2007)
7. Sandhu, R., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role Based Access Control Models. *Computer* 29(2) (February 1996)
8. Li, N., Mitchell, J.C., Winsborough, W.H.: Beyond proof-of-compliance: Security analysis in trust management. *Journal of the ACM* 52(3), 474–514 (2005)
9. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Decidability of Safety in Graph-Based Models for Access Control. In: Gollmann, D., Karjoth, G., Waidner, M. (eds.) *ESORICS 2002*. LNCS, vol. 2502, pp. 229–243. Springer, Heidelberg (2002)
10. Li, N., Tripunitara, M.V.: Security Analysis in Role-Based Access Control. *ACM Transactions on Information and System Security* 9(4), 391–420 (2006)
11. Crampton, J., Loizou, G.: Administrative Scope: A Foundation for Role-Based Administrative Models. *ACM Transactions on Information and System Security* 6(2), 201–231 (2003)
12. Koch, M., Mancini, L.V., Parisi-Presicce, F.: Administrative scope in the graph-based framework. In: Proceedings of the Ninth ACM Symposium on Access Control Models and Technologies (SACMAT 2004), pp. 97–104 (2004)
13. Jung, Y., Chung, M.: Adaptive Security Management Model in the Cloud Computing Environment. In: 2010 the 12th International Conference on Advanced Communication Technology (ICACT), vol. 2, pp. 1664–1669 (2010)
14. Wang, W., Li, Z., Owens, R., Bhargava, B.: Secure and Efficient Access to Outsourced Data. In: *CCSW 2009*, Chicago, Illinois, USA, November 13 (2009)
15. Mather, T., Kumaraswamy, S., Latif, S.: *Cloud Security and Privacy*, pp. 18–19. O'Reilly Media, Inc. (2009)
16. Ferraiolo, D., Kuhn, R.: Role-based access controls. In: 15th NISTNCS National Computer Security Conference, Baltimore, MD, October 13–16, pp. 554–563 (1992)
17. Nyanchama, M., Osborn, S.: Access rights administration in role-based security systems. In: Biskup, J., Morgernstern, M., Landwehr, C. (eds.) *Database Security VIII: Status and Prospects*. North-Holland (1995)

QoS Monitoring and Dynamic Trust Establishment in the Cloud

Ashok Chandrasekar, Karthik Chandrasekar, Malairaja Mahadevan,
and P. Varalakshmi

Department of Information Technology, Madras Institute of Technology, Anna University,
Chennai, India

{ashoksekar07, karthidecl, raja90mit, varanip}@gmail.com

Abstract. In the current cloud computing scenario, the need for establishing an SLA is essential. There is an even stronger necessity for monitoring whether the QoS mentioned in the SLA is met by the service provider. The next big issue is the trust on the service provider. Even though the providers pledge to meet the agreed upon QoS, there is a desideratum for a trust model which will give a quantitative measure of the trust to the requester before choosing a service provider. In this paper, we portray a novel approach which will monitor the QoS, negating the drawbacks associated with the existing techniques. The amount of monitoring data sent over the network is reduced by employing a derived and state monitoring approach. Based on the monitoring results, trust is established dynamically by making use of the Markov Chain model. The proposed approach is implemented using a web based system which will elucidate its use in real time. We believe that our dynamic trust establishment technique will be munificent in supporting the way trust management will be viewed in future.

Keywords: Dynamic Trust, QoS Monitoring, Cloud monitoring, Markov Chain.

1 Introduction

Trust plays a key role in the emerging cloud computing paradigm. It is one of the less trodden areas of cloud computing, making it a bit vulnerable in that aspect. Service providers and requesters in cloud formulate a Service Level Agreement(SLA) for every service provided and a QoS is established, to which the provider must conform throughout without any lapses as shown in Fig. 1. Now the real issue arises. How a requester can trust a provider? Based on what aspect can the trust be established? Is there a quantitative measure on how much can a provider be trusted? A major part of these questions still remain unanswered. Our paper concentrates mainly on addressing all the above issues in the best possible manner. For a provider to be trusted, there should be enough feedback on the services offered by it before and how it has managed to meet the QoS requirements. This shows that for trust to be established there is a need for QoS monitoring. QoS monitoring involves extracting the QoS

parameters from the SLA agreed upon by the service provider and the requester. Monitoring information about the parameters needs to be retrieved and checked for conformity with the QoS. This monitoring is done discretely at regular intervals or continuous monitoring is employed. When a violation on any of the QoS parameters is detected, it has to be properly recorded and informed to the requester. The work presented here throws enough insight into how the two aspects involved in trust establishment can be effectively done. Words service requester and user are used interchangeably in the upcoming sections of the paper.

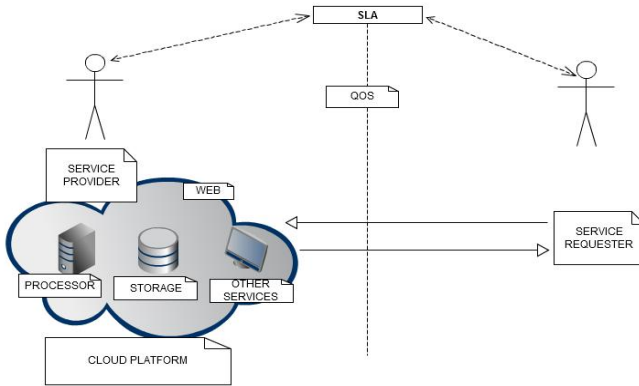


Fig. 1. Conceptual view of the interaction between service provider and requester

The rest of the paper is organized as follows: Section 2 overviews the issues and challenges involved in QoS monitoring and trust establishment. It also overviews other related initiatives. Our proposed solution for QoS monitoring and dynamic trust establishment is presented in Section 3. Design details, implementation and results are showcased in Section 4. Works related to the proposed system are analysed in Section 5. Finally some conclusions are arrived at and put together in Section 6.

2 Issues in Trust Establishment

Qos Monitoring Issues. The first issue involved in trust establishment is monitoring the QoS in a regular basis, to verify whether it is in compliance with the SLA. The real problem here is that monitoring the QoS involves monitoring all the parameters at the provider side and sending this information over the network to the QoS Monitor. Sending this monitoring data consumes some bandwidth. As QoS needs to be monitored in a continual basis, it involves sending quite a lot of monitoring data continuously over the network making it a bandwidth intensive task.

Trusting the Third Party. The Third Party which performs this task of QoS monitoring should be a trusted entity. It should not be in any way inclined towards

either the service provider or the requester. At present, the QoS Monitor is just assumed to be a Trusted Third Party [2].

Establishing Trust over a Provider. Current trust models rely mainly on the feedback from the provider's earlier services and tends to be static in most occasions. Trust once established is difficult to modify even if the provider has not been good of late. This provides an opportunity for the provider to maintain enough trust even without living up to its reputation. This problem has to be accepted.

3 Proposed Solution

The first part of the solution involves monitoring the QoS by making use of an effective monitoring technique which reduces the amount of monitoring data sent over the network. The second part deals with developing a dynamic trust model by employing the concept of Markov chains. Outline of the proposed architecture is shown in Fig. 2.

3.1 QoS Monitor

QoS Monitor is the component which monitors the QoS parameters continuously. There will be a Monitoring Agent deployed at the provider's end. This agent is responsible for getting the monitoring information. Monitoring information involves all the QoS parameters mentioned in the SLA. These parameters are extracted from the SLA and maintained separately along with their threshold values (values which should be met by the provider). These parameter details are alone extracted by the Monitoring Agent and sent to the QoS Monitor. This process consumes a considerable bandwidth as the information are sent very frequently. So we propose the concept of state monitoring and derived monitoring.

State Monitoring. The main constraint in monitoring QoS is sending the monitoring data over the network. So our agenda is to reduce the information sent over the network, at the same time ensuring an accurate and efficient monitoring. State monitoring gives exactly that kind of efficiency and accuracy. It involves sending only the state data of each parameter. State data is a single bit value representing whether the parameter meets the criterion or not. We use a 1 for criterion met and a 0 for failure to meet the specified QoS value. This reduces the transfer of monitoring data to sending a few bits. Only one bit is sent for every parameter, which will definitely lead to a much lesser data being transferred. It is the duty of the Monitoring Agent to check whether the value of the parameter under study is lesser or greater than the threshold and sending a 0 or 1 based on the result. But this alone cannot be used to monitor all the parameters, because on most of the parameters we need the exact value, so that we can have a realistic notion about the extent to which the QoS is satisfied or missed.

Derived Monitoring. There will be a few parameters which can be derived from the value of other parameters. Say for example, the amount of memory remaining can be

derived from the amount of used memory. It is not necessary to send both these parameters, as one can be derived from another. The next way of deriving involves sending only the difference in value with the previous value of the same parameter sent. This reduces the data sent because a lot of values remain same over a short time interval leading to a zero difference.

These two techniques can be combined based on our needs to build an effective and efficient monitoring scheme.

3.2 Dynamic Trust Model Based on Markov Chain

Trust is calculated dynamically based on the way the provider services its requests. After the monitoring information reaches the QoS Monitor from the Monitoring Agent, it is compared with the QoS parameters and trust is established based on its deviation from the actual values. This is done as follows.

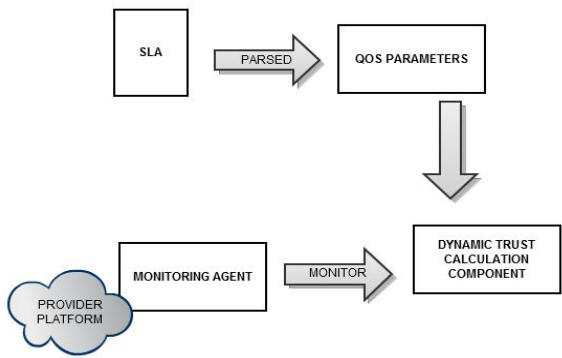


Fig. 2. Overall Architecture of the System

Cost Calculation. Each QoS parameter is unique and they are not equally significant. In the sense that failure to meet one parameter may not have the same effect as the failure to meet some other parameter. So we cannot have them affect the trust in the same way. We adhere to this difference in significance by assigning a different weight w_i for each parameter based on its importance and cost c is calculated based on that.

$$c = \sum_{i=1}^n w_i d_i$$

where w_i is the weight of the i^{th} QoS parameter, d_i is the difference between the actual monitored value and the expected value of the i^{th} QoS parameter and n is the total number of QoS parameters.

Markov Chain Based on Cost. Cost c is calculated as per the aforementioned formula for the cost function. This cost is calculated periodically and based on this cost, a Markov Chain is formulated with the following states: *Steady state*, *Unsteady state* and *Failure state*. Transition between the states can be visualised in the Fig. 3

which shows the Markov Chain state diagram, Based on cost c , it can be categorized into any one of the states mentioned above. The demarcation is fixed based on the level of compliance to the QoS. When in steady state, cost function will be such that the QoS is met without any problem. Unsteady state means the QoS parameters are partially unmet. Failure state means the failure to afford the service as agreed. It is the Denial of Service as a whole.

Trust Calculation. A base trust of 50% is fixed at the beginning for any untrusted provider. This gives the new providers a chance to attract customers. If we start from zero trust, then the probability of them getting a service request is less. Now with the Markov state chain and base trust setup, we can go for calculating trust. The first step in trust computation involves formulating the transition matrix which shows the transition probability from one state to another.

$$P = \begin{matrix} & \begin{matrix} Steady & Unsteady & Failure \end{matrix} \\ \begin{matrix} Steady \\ Unsteady \\ Failure \end{matrix} & \begin{pmatrix} p_{ss} & p_{su} & p_{sf} \\ p_{us} & p_{uu} & p_{uf} \\ p_{fs} & p_{fu} & p_{ff} \end{pmatrix} \end{matrix}$$

The above matrix is the transition matrix which shows the probability of transition from one state to another. Steady, unsteady and failure are the various states into which the cost value can fall. p_{ij} is the probability that the chain will from state i to state j .

This model can be extended into an Absorption Markov Chain with a new state introduced called Ideal State, which will be the absorption state. An absorption state is a state in the Markov chain which when reached, it is not possible to move to any other state from it. There is no ‘out transition’ from an absorption state. A Markov chain with such a state is known as Absorption Markov chain [6]. We have introduced a new absorption state(Ideal) into the chain to denote the maximum trust. This means that, a provider which has reached an ideal state will never falter. Even though it is almost impossible for any provider to reach the Ideal state, it is employed here for having a comparison with a maximum quantity. After the inclusion of the new absorption state, the state diagram becomes the one as mentioned in Fig. 3.

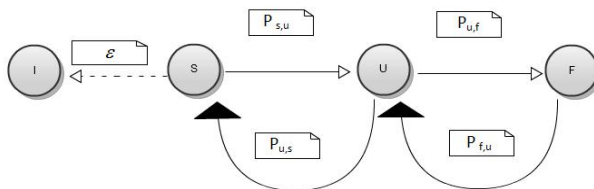


Fig. 3. State diagram showing the states and transition

The transition matrix becomes as follows:

$$P = \begin{matrix} & \begin{matrix} steady & unsteady & failure & ideal \end{matrix} \\ \begin{matrix} steady \\ unsteady \\ failure \\ ideal \end{matrix} & \begin{pmatrix} p_{ss} & p_{su} & p_{sf} & p_{si} \\ p_{us} & p_{uu} & p_{uf} & p_{ui} \\ p_{fs} & p_{fu} & p_{ff} & p_{fi} \\ p_{is} & p_{iu} & p_{if} & p_{ii} \end{pmatrix} \end{matrix}$$

The base probability of state transition from one state to another is formulated based on the Markov Chain. The transition matrix formulated is as follows:

$$P^0 = \begin{matrix} & \begin{matrix} steady & unsteady & failure & ideal \end{matrix} \\ \begin{matrix} steady \\ unsteady \\ failure \\ ideal \end{matrix} & \begin{pmatrix} 1/2 & 1/2-\xi & 0 & \xi \\ 1/3 & 1/3 & 1/3 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}$$

The above matrix shows the base probability of transition among states. ξ is used to represent an infinitesimally small probability of the chain moving from steady state to ideal state. It is almost impossible in real world for the chain to move to ideal state, which explains the use of it. From this base probability, the probability with which a state will move to another state after two time intervals can be calculated just like the dot product of two vectors. In general, for a Markov chain with r states it is,

$$p_{ij}^2 = \sum_{k=1}^r p_{ik} p_{kj}$$

For example, the probability of a chain in steady state moving to an unsteady state after two time intervals is computed as follows:

$$p_{su}^2 = p_{ss} p_{su} + p_{su} p_{uu} + p_{sf} p_{fu} + p_{si} p_{iu}$$

This can be used to calculate the expected probability after any time interval which are denoted by P^i (expected probability at i^{th} time interval). This is used to calculate the expected probability transition matrix at n time intervals. From this expected transition matrix, we can predict the state in which the chain will be after n time intervals.

$$u^{(n)} = u P^n$$

where u is the probability vector which contains the probability that the chain will start with that particular state. This state is important unless it is a regular Markov Chain where the state at which the chain starts does not affect the prediction or the present state. In the proposed system we take this u vector as (1, 0, 0, 0) since the chain will start with the steady state on all occasions. This can be used in the calculation for predicting the state at n^{th} time interval.

$$u^{(n)} = (1 \quad 0 \quad 0 \quad 0) \begin{pmatrix} p_{ss}^n & p_{su}^n & p_{sf}^n & p_{si}^n \\ p_{us}^n & p_{uu}^n & p_{uf}^n & p_{ui}^n \\ p_{fs}^n & p_{fu}^n & p_{ff}^n & p_{fi}^n \\ p_{is}^n & p_{iu}^n & p_{if}^n & p_{ii}^n \end{pmatrix}$$

$$u^{(n)} = (p_{ss}^n \quad p_{su}^n \quad p_{sf}^n \quad p_{si}^n)$$

The above matrix shows that at n^{th} time interval, the chain has p_{ss}^n probability of being in steady state, p_{su}^n probability of being in unsteady state, p_{sf}^n probability of being in failure state and p_{si}^n probability of being in ideal state.

Our trust computation uses this prediction and compares it with the state it is currently in and it is calculated repeatedly at each time interval. There are two cases possible.

*Case 1 – The actual state at the n^{th} time interval is the predicted state(the state with the highest probability in the prediction matrix):*In this case, if the actual state is steady state, then trust value is calculated as follows:

$$T_{current} = T_{previous} + \left((1 - P_{predicted}) \times 10 \right)$$

where $T_{previous}$ is the value of trust calculated during last time interval. At beginning when there is no trust established $T_{previous}$ takes the base value of 50. $T_{current}$ is the current calculated trust value of the provider. $P_{predicted}$ is the probability with which the actual state is predicted. We have subtracted the predicted probability from 1, because 1 is the maximum possible probability. After subtraction, it gives a value less than 1. To show a considerable change in the trust value, we have multiplied it with 10.

If the actual state is unsteady state or failure state, then trust value is calculated using,

$$T_{current} = T_{previous} - \left((1 - P_{predicted}) \times 10 \right)$$

Here the state is unsteady or failure, which means that the provider has not met the QoS. Obviously this calls for a decrease in trust, so we subtract from the previous calculated trust.

Case 2 – The actual state at the n^{th} time interval is not the predicted state(actual state is some other state with lower probability): If the actual state is steady state,

$$T_{current} = T_{previous} + \left((1 - P_{predicted}) \times 10 \right)$$

In this case we have a surprise. We don't expect the provider to meet the QoS or in other words be in steady state, yet she delivers. This calls for a better increase in trust. But we have not made any change to the formula used in *Case 1* steady state. This is because, unpredicted state will have a lower $P_{\text{predicted}}$ value. So when subtracted from 1, it increases the trust to a greater extent than before.

If the actual state is unsteady state,

$$T_{\text{current}} = T_{\text{previous}} - \left((1 - P_{\text{predicted}}) \times 10 \right)$$

If the actual state is failure state,

$$T_{\text{current}} = T_{\text{previous}} - \left(2(1 - P_{\text{predicted}}) \times 10 \right)$$

Here we have multiplied with the factor 2 to impose a higher penalty for failing unexpectedly.

By the aforementioned formulae, it is clear that the dynamic trust calculation takes into account, the predicted conformity and failure and also the unpredicted conformity and failure of the provider. Appropriate importance has been given to each of those happenings. For instance, an unpredicted failure is always more difficult to digest than a predicted one. Trust value thus calculated over a consistent time period can never exceed 100. It is the upper bound. When there is a positive trust after reaching 100, trust value is not altered, it is maintained at 100. The number of time intervals the provider stayed at the trust value of 100 is maintained separately and it is taken into account. This is considered to be the surplus trust provided to the provider, which makes sure that the provider is not punished for a single QoS violation after servicing consistently. When there is a failure calling for a reduction in trust, it will not be directly subtracted from the current trust of 100. In turn it is done by decrementing the same from the surplus. This surplus also called as tolerance level gets incremented by one, every time the provider hits or stays at 100. Thus the proposed approach makes possible a dynamic, poised, consistent, reliable and efficient trust establishment scheme.

3.3 Establishing Trust over the Third Party

The third party which handles the entire operation of monitoring the QoS and trust establishment, should be a trusted entity. If not, the whole purpose of the trust establishment process will be meaningless. In the present scenario, the third party which monitors the QoS is assumed to be a trusted one. This assumption is a rather presumptuous one. To establish trust over the third party, [2] hints at the concept of anonymity to anonymize the identity of both the provider and the user using an intermediate Anonymization system. But this intermediary system could itself be compromised. So we provide another solution to this problem, which ensures that the third party can be trusted based on it and also the third party can trust the providers and users that they are indeed the actual persons they claim to be. To achieve this, we propose mutual authentication using X.509 certificates [18] which is the most widely used to prevent masquerading. The third party has to authenticate itself with both the

user and the provider and the vice versa. The provider and user can also confirm their identity to the third party. This overcomes the issue of masquerading.

The second security threat involved is deploying the Monitoring Agent at the provider end. There is a possibility of the provider altering the code of the Monitoring Agent and make it send some other values to the QoS monitor instead of the actual monitored data. This can be overcome by using safe code interpretation, a feature which makes Java a much safer language. But there is a possibility of valid byte code modification, which can be solved by going for other mobile agent security techniques [21].

4 Implementation and Evaluation

The proposed system is implemented by using Xen [9] as the hypervisor with Eucalyptus [8] on top of it on systems which run CentOS 5.6. This constitutes our cloud platform. Hardware specifications of the nodes in the cloud: Intel Core2 Duo T6400 2.00GHz, 4GB RAM, 320GB storage and a LAN which services at 64Mbps. The VM runs Ubuntu 64bit OS. The QoS monitoring and Trust computation project encompasses several modules of which the monitoring agent which runs at the provider end is implemented using Java. Monitoring data is collected using XenMon [11], which can be used to gain insight into the VM instance's resource usage and requirements, that is provided to the user. Monitoring tools like Ganglia [20] or MonALISA [21] can also be used in place of XenMon to provide interoperability. A simple SLA format is used which is created when the user's service or instance gets launched. It is then parsed to get the parameter values, which are read from the files into a Java program which compares it with the actual monitoring data sent by the monitoring agent and calculates trust periodically using the aforementioned algorithm. A web interface is provided to both the provider and the user, which uses an Apache web server for deploying the web system, MySQL as the backend database to store the monitoring data, trust values and other intermediate results for further analysis and the web application is developed with JSP and Servlets. Implementation includes the following modules:

4.1 Web Interface

An interface is provided to both the user and the service provider by the Trusted Third Party.

4.2 QoS Monitor

QoS Monitor uses a Monitoring Agent to capture the monitoring data from each VM and send it over the network by employing state and derived monitoring to all the parameters that apply. Not all parameters can be monitored using them, since there may be a loss of accuracy in information leading to an inaccurate trust calculation. Fig. 4 shows the difference in the amount of monitoring data sent with the default

scheme and the proposed monitoring scheme. The monitoring data includes the QoS offered by the provider, which is collected from over 400 different VM instances provided to 400 different users in the cloud and sent over the network. First, the default monitoring approach is employed and data is collected over a period of 8 hours. The same instances are monitored with our proposed QoS Monitor over the same period of time and the amount of data sent are recorded, which clearly shows the decrease in amount of data sent from 1632 MB using the default approach to 1254 MB. This is about 23.16% lesser than the default scheme. In real time, where there will be hundreds of thousands of users, amount of data will be humungous. In such a scenario, the proposed system will be indispensable.

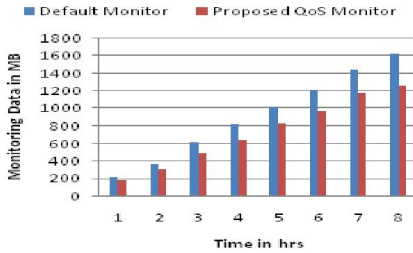


Fig. 4. Monitoring data Analysis

4.3 Trust Calculator

Trust calculator is implemented using Java. It gets the monitoring information sent by the Monitoring Agent. Threshold values of each parameter got from the SLA is compared with the information from the Monitoring Agent and cost is calculated.

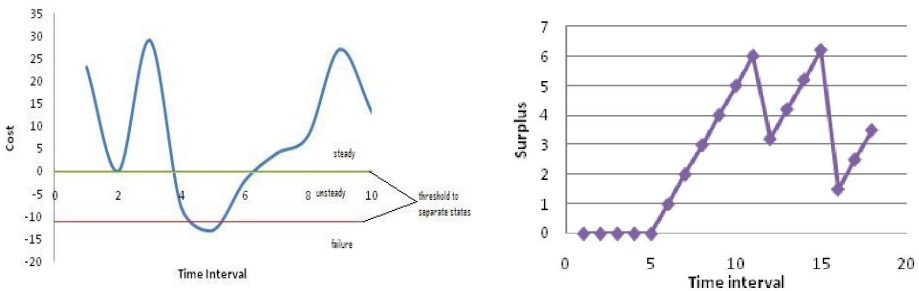


Fig. 5. a) Cost Variation b) Variation of Tolerance level

Fig. 5a shows the cost computed over a period of time and the classification based on threshold into one of the various states. Threshold value is calculated based on the weights assigned to each parameter. Here threshold values of -10 and 0 are used for classification into failure and unsteady states respectively. Based on the classified

state and the predicted state, trust is calculated. When a provider services in a consistent way and reaches a trust of 100, the surplus or tolerance level gets incremented. This tolerance level is the level till which a provider can stay at a trust value of 100. Fig. 5b shows the variation of the tolerance level with time.

4.4 Impact of Various Factors on Trust

Trust is affected mainly by the actual service or state and the expectation probability of the states. Fig. 6a shows the variation of trust with respect to the expectation and the actual state. The expectation can be high, medium or low as there are only 3 states. The expected occurrence of a state and unexpected occurrence of the same state affects trust in a different way. This is evident in Fig. 6a. Trust thus calculated is reported to the user in a detailed fashion with statistics formulated from continuously monitored data. This report is made available to the user on request or at the end of the service contract with the provider. Fig. 6b shows a snapshot of this detailed report generated for a user.

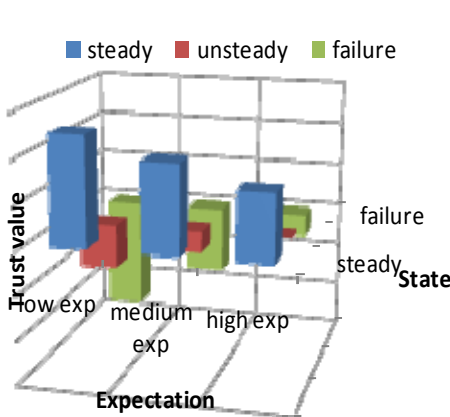
5 Related Work

Monitoring is very essential in the present cloud computing environment. There are a lot of software systems which provide monitoring functionality but most of them are self monitoring [7, 12, 14-17], which involves monitoring how their own resources are faring. Of late a few techniques for QoS monitoring has been proposed. QoS-MONaaS [2] is one such system which provides QoS Monitoring which relies on “as a Service” paradigm, so that it could be used seamlessly by any user to monitor the QoS. It provides the design and implementation of a QoS monitoring module, which works on top of the SRT-15 project [10]. It monitors the QoS parameters and sends a mail to the user whenever there is a violation.

Fault tolerance in mobile cloud using Markov model based monitoring [3] provides a markov model based fault prediction which helps in effective fault tolerance in mobile cloud. In this paper, the authors explain the volatility of mobile devices and resources in mobile cloud which makes fault tolerance difficult and indispensable. They have evaluated their model and accuracy of their fault prediction scheme.

Qos-MONaaS uses an approach similar to the proposed system for deriving parameters from SLA and monitoring the QoS. But it does not deal with any of the security threats involved and it does not elaborate on how the QoS is actually monitored. None of the existing monitoring techniques deal with the reduction of monitoring data which could become overwhelming at times.

Till now, there is no standard and established trust mechanism which dynamically evaluates the provider based on the services offered. With the extensive development in cloud computing, the need for a solid trust model is ever increasing. In recent times, a few trust models based on recommendation of previous users [5,19] and time delay in the offered services have been developed. Our trust model in comparison to



Trust Report

Provider Id : PRO102
 User Id : US200
 SLA Id : S101
 Report generated at : Tue Dec 11 2011 16:04:15
 Status : **Completed**

Provider Details	Least Trust attained	Number of failures	Trust over last 1 day	Overall Trust
Cloud-Pro	38	Last hour 0	89	86
		Last Day 2		
		Overall 7		

Violations
 SLA

Individual Parameter Report

	Max. Provided	Min. Provided	Failures
Memory in MB	546	502	3
CPU in %	100	69	4
Storage in GB	25	25	0

Fig. 6. a) Variation of trust with expectation and actual state b) Trust Report

the existing models is much more practical and fair to both the provider and the user. This is because it uses realistic expectations or predictions and the comparison that comes out of this becomes a fair one, rather than a much optimistic one as seen in other models.

6 Conclusions

QoS Monitoring component of the third party makes use of the state and derived monitoring techniques. But these two techniques does not apply for all the resources that are monitored. In most of the cases we need to send the exact data like in most other systems. To reduce the amount of monitoring data sent to further extent, we can make use of any compression technique that will suit our requirements. Obviously this involves an overhead due to the time delay involved during compression and decompression. So we opted out of using compression in our system. Making the Monitoring Agent safe from alteration does not mean that there is no other way to alter the monitored values sent to the third party. Man in the Middle and other attacks on the network are possible which can view this data or alter it and send it to the third party. This can be prevented by using secure means of transfer by employing SSL/TLS or HTTPS as transfer protocol. Trust establishment component takes the present level of service of the provider as the input and calculates trust dynamically. The trust value that a provider has achieved at the end of the previous service becomes the base trust when it starts to service the next user. By this the provider will be driven to perform better, so as to increase its number of customers and to position itself better in the market for the long run. Out of competition all the other providers will try and deliver better service, to retain their customers and to increase their clientele. This whole process leads to better service to the users and better value for their money which ultimately leads to customer satisfaction.

References

1. Khan, K.M., Malluhi, Q.: Establishing Trust in Cloud Computing. *IT Professional*, 20–27 (September 2010)
2. Romano, L., De Mari, D., Jerzak, Z., Fetzer, C.: A Novel Approach to QoS Monitoring in the Cloud. In: *First International Conference on Data Compression, Communications and Processing* (2011)
3. Park, J., Yu, H., Chung, K., Lee, E.: Markov Chain based Monitoring Service for Fault Tolerance in Mobile Cloud Computing. In: *Workshops of International Conference on Advanced Information Networking and Applications* (2011)
4. Lund, M., Solhaug, B., Stlen, K.: Evolution in Relation to Risk and Trust Management. *Computer*, 49–55 (May 2010)
5. Guo, Q., Sun, D., Chang, G., Sun, L., Wang, X.: Modeling and evaluation of trust in cloud computing environments. In: *3rd International Conference on Advanced Computer Control* (2011)
6. Markov-Chains, <http://www.cs.virginia.edu/~gfx/courses/2006/DataDriven/bib/texsyn/Chapter11.pdf>
7. Manvi, S.S., Birge, M.N.: Device Resource Status Monitoring System in Wireless Grids. In: *ACEEE Intl. ACT* (2009)
8. Eucalyptus, <http://www.eucalyptus.com/>
9. Xen-hypervisor, <http://xen.org/>
10. The SRT-15 project, <http://www.srt-15.eu/>
11. Gupta, D., Gardner, R., Cherkasova, L.: XenMon: QoS Monitoring and Performance Profiling Tool, <http://www.hpl.hp.com/techreports/2005/HPL-2005-187.pdf>
12. Wang, M., Holub, V., Parsons, T., Murphy, J., OSullivan, P.: Scalable Runtime Correlation Engine for Monitoring in a Cloud Computing Environment. In: *17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems* (2010)
13. Patel, P., Ranabahu, A., Sheth, A.: Service Level Agreement in Cloud Computing. In: *UKPEW* (2009)
14. Kung, H.T., Lin, C.-K., Vlah, D.: CloudSense: Continuous Fine-Grain Cloud Monitoring with Compressive Sensing. In: *USENIX HotCloud* (2011)
15. Huang, H., Wang, L.: P&P: a Combined Push-Pull Model for Resource Monitoring in Cloud Computing Environment. In: *IEEE 3rd International Conference on Cloud Computing* (2010)
16. Liu, F., Xu, Z., Wang, Q.: Grid Computing Monitoring Based on Agent. In: *1st International Symposium on Pervasive Computing and Applications* (2006)
17. Meng, S., Liu, L., Wang, T.: State Monitoring in Cloud Datacenters. *IEEE Transactions on Knowledge and Data Engineering* 23(9) (September 2011)
18. Stallings, W.: X.509 Authentication Service, 4th edn. *Cryptography and Network Security*, ch. 14, pp. 419–428 (2005)
19. Abawajy, J.: Establishing Trust in Hybrid Cloud Computing Environment. In: *TrustCom* (2011)
20. Massie, M.L., Chun, B.N., Culler, D.E.: The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing* 30, 817–840 (2004)
21. Legrand, I., Voicu, R., Cirstoiu, C., Grigoras, C., Betev, L., Costan, A.: Monitoring and Control of Large Systems with MonALISA. *Queue- Data* 7(6) (July 2009)

Multihop-Based Key Management in Hierarchical Wireless Sensor Network

Yiyang Zhang^{1,3,*}, Xiangzhen Li², Yan Zhen², and Lingkang Zeng²

¹State Grid Information & Telecommunication Company Ltd., Beijing, 100761, China

²State Grid Electric Power Research Institute, Nanjing 210003, China

³Beijing University of Posts and Telecommunications, Beijing, 100876, China
winzyy@163.com, {xzli, zhenyan, lkzeng}@sgcc.com.cn

Abstract. The vulnerable environment and open communication channel make it very necessary to protect wireless sensor networks (WSNs). The key management is the most important way to protect the communication in WSNs. In this paper, we design a Multihop-based Key Management (MKM) which can efficiently enhance the security and survivability in the hierarchical WSNs. Different from previous works, we present the key system as well as the cluster formation. The MKM generates and distributes keys based on hop counts, which not only localizes the key things but also has no overhead. The MKM provides the session keys among sensors and the cluster key between the cluster head and its member nodes. The different hops make the different keys. The MKM can protect the network from the compromised nodes by reducing the high probability of the common keys. The security analysis can effectively prevent several attacks.

Keywords: Key management, wireless sensor network, security, multi-hop.

1 Introduction

The wireless sensor networks (WSNs) are widely used with thousands of tiny sensors, such as in smart grid, smart city, internet of things (IoTs). However, due to limitations of wireless sensor networks in the computation, energy, storage and open wireless communication etc., WSNs are vulnerable to various attacks, and the security in WSNs is required [1, 2, 3, 4, 5, 6]. Since it is impossible and impracticable to utilize the single key to encrypt/decrypt message in the networks, the solutions, which trend to deploy keys in whole network, may cause the leak of key materials. Once the key things fall into the adversaries, the sensors are compromised, which threatens entire network.

Therefore, it is necessary to organize the sensors into clusters and localize the key things. In [2, 3], the authors presented RPKH and LDK schemes to provide the local key management. The RPKH and LDK utilize different nodes including the normal

* Corresponding author.

nodes and anchor nodes to generate keys by different transmission range. However, it also consumes large amount of energy, when the two kinds of nodes transfer messages and discovery common keys. Meanwhile, the adversary can eavesdrop on the key materials during nodes exchanging packets.

In this paper, we present a Multihop-based Key Management (MKM) in the hierarchical wireless sensor network [8]. Different from the previous works, we build our solution in normal network without any special nodes (e.g. high energy or high capability nodes), which makes it more practicable to deploy. In MKM, we generate the key system with the hierarchical architecture formation, which makes the scheme effective because of no overhead. When the clusters form, the cluster head gets the hop counts from cluster head to the member nodes with ACK packets and then uses the hop-count and nonces to generate the key system. Nodes in different distance have different keys.

In MKM, the nodes near the cluster head have more keys, which makes the nodes transfer messages on one-way routing. Meanwhile, with the cluster head reselection, the key system will be refreshed, because the new cluster head causes the hops changes and then the key should be reassigned. Moreover, the cluster head can use the cluster key to communicate with member nodes.

Considering about the security and the life time of WSNs, we will rekey to refresh the cluster and the keys. During the rekey phase, the cluster will elect new cluster head which calculate the new distance from CH to member nodes and then generate the new key system based on the old one.

Compared to previous works [2, 3, 5] in WSNs, our solution has the following scientific research contributions: 1) MKM utilizes the hierarchical architecture to localize the key things, which prevents the compromised nodes threat the entire network. 2) Without any overhead, MKM counts the hop count in the cluster formation, which can effectively reduce energy consumption. 3) MKM employs the normal wireless sensor network but not special nodes, which makes it more practicable.

The rest of this paper is organized as follow: Section 2 presents related work. Section 3 shows the system model. Section 4 will describe the key management in detail, and section 5 evaluates MKM using security analysis. Finally, we end the paper with a conclusion as well as the further work.

2 Related Work

Some literatures are researched in [2, 3, 5, 13, 14], these papers designed some schemes for local key management.

In [3], a location dependent key management (LDK) has been presented, which employs the heterogeneous sensors to build a clustered sensor network. In LDK, the sensors are static and considered the anchor nodes as the management nodes. The anchor nodes use the different location information to generate sets of keys. The adjacent nodes subjected to the same anchor node can establish secure communication links by exchanging all key materials to discover the common keys. Neighbouring

nodes can establish secure communication link by determining the common keys via exchanging their key materials. LDK significantly reduces the number of keys stored on each node without any pre-deployment knowledge of nodes. It also can increase the direct connectivity ratio among nodes.

However, the LDK makes nodes exchange all their key materials when determining the communicating key, which consumes more energy and increase attack chances. The transmitted message takes too many bytes. Therefore, it consumes lots of communicating energy and is not efficient for WSNs either.

In [2, 5], there are two similar key management schemes, the RPKH and ARPKH. The RPKH is based on random key distribution in the heterogeneous sensor networks, which used separate keys in different clusters and take into consideration distance of sensors from their cluster head.

The ARPKH, an improved version of RPKH, considers a multiple shared keys between pair-wise nodes on connectivity. When a key that used for establishing the secure link between two nodes is revealed, the link has been expired and then the connectivity is broken. ARPKH will change the alternative shared keys to replace the revealed key and establishes a new secure link between two nodes again.

Chan et al. [14] proposed q-composite random key pre-distribution scheme which needs q common keys ($q > 1$) to establish secure communication links between nodes. By increasing the value of parameter q, they can achieve high security level, because an adversary needs to know all q keys for attack. The drawback of this scheme is that if an adversary has compromised nodes in large scale, entire keys can be exposed.

In wireless sensor networks, nodes need to communicate with neighboring nodes only. Therefore, it can be more efficient to let nodes which are close together have common keys while between nodes which are far off have disjointed keys. Thus, W. Du et al [13] proposed a scheme that uses pre-deployment knowledge of expected location of nodes. Through allocating keys among adjacent nodes by using the pre-deployment knowledge, the memory requirement on a node can be decreased and the security impact of compromised nodes can be limited around the location.

3 System Model

3.1 Network Model

Given the WSN with n sensors as a graph G which consists of m clusters, that is, $G = C_1 \cup C_2 \dots \cup C_m$ and $C_i \cap C_j = \emptyset, i \neq j$, where C_i is a cluster with the cluster head (CH or CH_i) and member nodes. In a cluster, the CH collects and aggregates packets from its member nodes, and then forwards them to the base station (BS). Normally, a member sensor can transfer packets to CH through several hops. Note that there are some nodes in the range overlap of many cluster, they only choose one cluster to join. Each sensor has a unique ID. If a node is compromised, all of the information in this node will be compromised including the key materials [7].

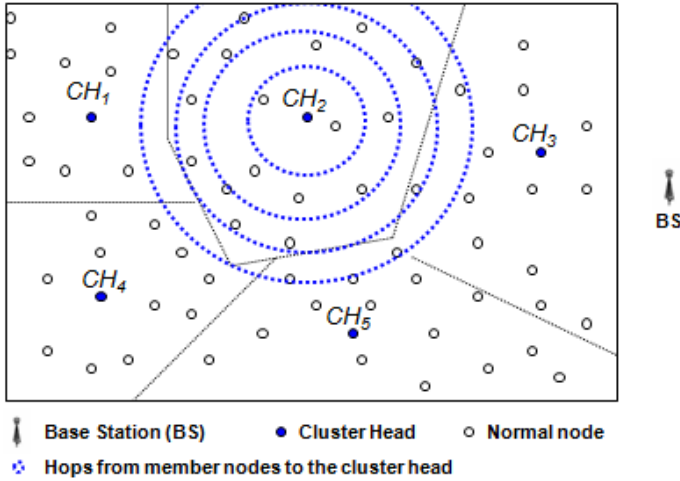


Fig. 1. The considered hierarchical WSN with 5 clusters

3.2 Notations

In Table 1, we list some notations used in this paper.

Table 1. Notations

Notation	Explains
K_I	The initial key shared by all nodes
ID_{CH}	ID of the cluster head
n_i	The i^{th} nonce in the set of nonces N
ID_{v_j}	ID of member node v_j
$f()$	The <i>one-way</i> function
TTL	Time To Live
C_i	The i^{th} cluster in the WSN
k_j^i	The i^{th} key for the member node v_j

4 Multihop-Based Key Management

In this section, we introduce the Multihop-based key management (MKM) in detail. Before the deployment of the sensor network, each sensor is pre-distributed an initial key K_I for the security in deployment phase, the initial key provides the communication during the formation phase [7].

4.1 The Cluster Head Election

As mentioned above, considering the energy efficiency and management facilitation of WSN, we adopt the hierarchical architecture for our network [9, 10, 11]. Firstly, a node itself decides whether it becomes a candidate CH or not according to the cluster head election algorithm as shown in equation (1):

$$T(n) = \begin{cases} \frac{p}{1-p \lfloor r \bmod (1/p) \rfloor} [\eta + (i \times p)(1-\eta)], & n \in G \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Where $T(n)$ indicates a probability function; p is the percentage; and r denotes the current round number and its default value is 1; η represents the left energy percentage; i is the sum of rounds that a node is idle, and it is reset to 0 when it is elected as CH. G is the set of nodes that have not been CHs in the $1/p$ rounds recently.

The node will announce the candidate information to other nodes according to the result of equation (1). And other nodes which maybe accept several election campaign messages, and they will choose one to join it as follow.

4.2 The Cluster Formation

Once a node becomes a cluster head, it will send a *beacon* message to other sensors to form a cluster. Each sensor may receive several different *beacon* messages from different candidate cluster heads, but it only can join one cluster.

When the *CH* broadcasts a *beacon* message encrypted by K_i contains ID of *CH*, *TTL* (time to live) and a set of *nonces* named N , the random numbers. And the *nonces* will be different with different *TTL*, that is, if the *TTL*= 3, then the sensor will get four (*TTL*+1) nonces, such as $N=\{ n_1, n_2, n_3, n_4 \}$. We generate more nonces (e.g. *TTL*+1) for the connectivity, especially the common keys. Where *TTL* is to limit the cluster size, e.g. *TTL*=3, and it will be decrease for each forwarding until it becomes 0 and the beacon message will be dropped.

Therefore, depending on the cluster size (*TTL*), other nodes can receive different sets of *beacon* messages from different *CHs* as the equation (2) in different distance (hop ranges) .

$$CH \rightarrow * : \{ ID_{CH}, n_1 \dots n_{TTL+1}, TTL \}_{K_i} \tag{2}$$

4.3 The Initial Key Generation

Assume $v_j \in C_i, v_j \neq CH$, we call v_j as member node. When member node v_j receives a *beacon* message and wants to join the cluster C_i , it counts the *TTL* and sends the ACK including the ID_{v_j} and the TTL_{v_j} back to its interest cluster, and then the cluster head knows the hops from ID_{v_j} to *CH*. *Beacon* messages are orderly transmitted at

different distance levels. And then, the member node v_j decrypts the *beacon* messages and obtains ID_{CH} and then set of nonces N_i .

And then, v_j calculates the candidate keys k_j^i based on the received *nonces* as follows:

$$k_j^i = f(k_I, n_i) \tag{3}$$

Where $f()$ denotes a one-way hash function. And the initial key generation process is as shown in fig.2.

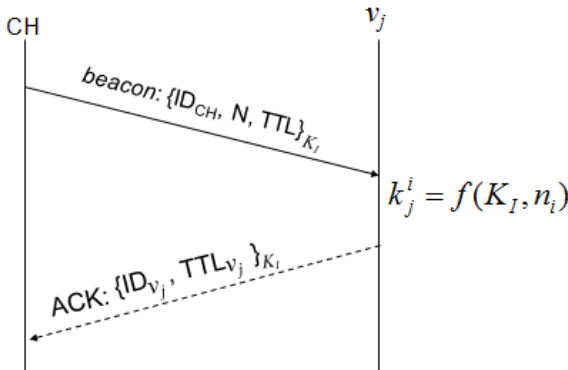


Fig. 2. The initial key generation process

After the calculations, nodes erase K_I . Consequently, v_j stores the key things as follows in table 2:

Table 2. Keys table of member nodes in different hop ranges ($TTL=3$)

Keys for 1 st hop	Keys for 2 nd hop	Keys for 3 rd hop
$k_j^1, k_j^2, k_j^3, k_j^4$	k_j^2, k_j^3, k_j^4	k_j^3, k_j^4
$k_j^1, k_j^2, k_j^3, k_j^4$	k_j^2, k_j^3, k_j^4	k_j^3, k_j^4
...
$k_j^1, k_j^2, k_j^3, k_j^4$	k_j^2, k_j^3, k_j^4	k_j^3, k_j^4

According to the table 2, we can find that the nodes can communicate with its neighbour nodes for the common keys. The specific algorithm of hop count and key information acquirement is as algorithm 1:

Table 2. Hop count and key information acquirement algorithm ($TTL=3$)

Algorithm 1 Hop count and key information acquirement	
1.	CH broadcasts <i>beacon</i> messages with different <i>nonces</i> : $CH \rightarrow * : \{ID_{CH}, n_1 \dots n_{TTL+1}, TTL\}_{K_I}$
2.	$v_j \rightarrow CH : \{ID_{v_j}, TTL_{v_j}\}_{K_I}$, <i>Key pool generation for v_j</i> by $k_j^i = f(k_I, N_i)$
3.	<i>Erase K_I.</i>
4.	CH obtains the hop count: $N_{hops} = TTL_{CH} - TTL_{v_j} + 1$
5.	According to the N_{hops} , nonces N , and oneway function $f()$, the cluster head can get the ID_{v_j} 's key information.
6.	end.

4.4 The Common Key Discovery

For communication, member node should establish secure link with its neighbouring nodes, which needs the common keys between them. According to those candidate keys, member nodes in the same distance receive the same *beacon* messages and they can also generate the same keys. Moreover, the nodes in the adjacent areas also have some duplicate candidate keys.

If a node can receive $\{ID_{CH}, N_i, TTL_i\}_{K_I}$, it also can receive $\{ID_{CH}, N_j, TTL_j\}_{K_I}$, where $TTL_i > TTL_j (i < j)$. Since the distance range of hops j covers the distance range of hops i , the node near cluster head has more keys than the one far away from cluster head.

Therefore, each member node v_j generates a list which just stores the keys. Moreover, since the packets from members will be collected by the cluster head, the cluster head should have the ability to decrypt these messages. During this process, the member nodes should report its keys, which will increase the transmission. Because the nonces are sent by the cluster head, it also knows the function, and then it can calculate the keys of members as mentioned in algorithm 1.

Due to the keys are generated by hop count, which means the nodes can be connected in the same cluster. And the path key between v_i and v_j is calculated as follow:

$$k_{ij} = f^{abs(i-j)}(k_I, N_i) \tag{4}$$

The equation (4) makes it possible for any two nodes in the cluster to communicate with each other. Actually, there is another way to make every two nodes communicate, that is, the last but one nonce is same in a cluster and the key which they generate is same too (the last nonce is used to the cluster key).

The cluster key is the key which is used for communication between the CH and its members also for generating new key in next round. Since there are $TTL+1$

nonces, according to the algorithm 1, the last nonce in the set N will be transferred to every sensor.

4.5 The Rekey Process

For prolonging the lifetime of the whole network, it is necessary to change the cluster head. On the other hand, the key should be rekeyed for the security [12], otherwise, when CH receives a certain amount of encrypted messages (more than $2^{2k/3}$, k is the length of key), the keys will be no longer safe. According to the requirements above, we should recluster after a certain phase. Assume the process of recluster happens inside the cluster, which can reduce the energy consumption.

During the reselection of CH , we can rekey as the initial phase. When the new cluster head has been selected according to the equation (1), it will announce itself as the cluster head and recalculate the distance from its members.

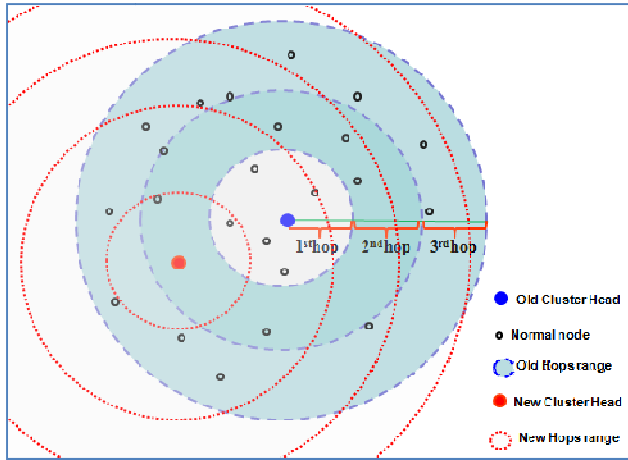


Fig. 5. The reselection of CH

As shown in fig 5, in this situation, the new cluster head changes not only the relative position but also hop counts from CH to members, which make the nonces as well as the key things different.

5 Security Analysis

Compared to previous works, the salient advantage of our solution is that we addressed challenging runtime security issues using localizing key things and design a dynamic key management.

During cluster formation phase, the cluster head can calculate the hop count from the CH and member nodes, and then the member nodes can generate keys by the

nonces and hops. According to the different hop count, the cluster is divided into several security belts as shown in fig 2, the nodes in different belts have different keys. Because the keys are generated by the set of nonces, the adjacent nodes have some common keys, which makes it possible to communicate with each other.

The nodes near CH have more keys than the nodes far away from CH, which means that the far nodes just can submit message to the CH. And then the messages just can be decrypted by near nodes. The one-way security model prevents the eavesdrop attack, selective-forwarding attack and hello flood attack as shown in Table 3.

Table 3. Analysis in local key management

Attacks Types	RPKH	LDK	MKM
Selective-Forwarding	×	×	√
Sink-Hole attack	×	√	√
Sybil attack	√	√	√
Worm-Hole	√	√	√
HELLO Flood	√	√	√
DoS	×	×	√

To communicate with members, the cluster head utilize the last key as the cluster key which is shared with all the sensors (including the CH) in the cluster. The cluster also can be used to rekey during the next round cluster. Furthermore, the key system forms during the cluster formation, which almost does not consume any energy overhead.

6 Conclusion and Future Work

In this paper, we propose a Multihop-based key management (MKM) protocol to enhance network security and survivability. Unlike previous works, we employ the hierarchical architecture but not fixed-node-network. In contrast to other clustered architectural security solutions, the salient advantage of this work is that we addressed challenging security issues by localizing key things. Also we present a rekey mechanism with low energy consumption. In the future, we will focus on how to enhance security in scalable WSN.

Acknowledgements. This work was supported by the foundation: Important National Science & Technology Specific Projects of China: Research, development, and application validation of sensor network for smart grid security monitoring, transmission efficiency, measurement and user interaction (2010ZX03006-005-02); the National Basic Research Program of China (973 Program): Basic theory and practice research of Internet of Things (2011CB302900); the Doctoral Start-up Fund of Liaoning Province (20101074).

References

- [1] Zhou, Y.: Securing Wireless sensor networks: A Survey. *IEEE Communications Surveys & Tutorials* (3rd Quarter, 2008)
- [2] Banihashemian, S., Ghaemi Bafghi, A.: A new key management scheme in heterogeneous wireless sensor networks. In: *Proceeding ICACT 2010, Korea* (2010)
- [3] Anjum, F.: Location dependent key management in sensor networks using without using deployment knowledge. In: *Proceedings of WiSe 2006* (2006)
- [4] Du, X., Xiao, Y., Guizani, M., Chen, H.-H.: An effective key management scheme for heterogeneous sensor networks. *Ad Hoc Networks* 5(1), 24–34 (2007)
- [5] Banihashemian, S., Bafghi, A.G.: Alternative shared key replacement in heterogeneous wireless sensor networks. In: *Proceeding IEEE Computer Society* (2010)
- [6] Luk, M., Mezzour, G., Perrig, A., Gligor, V.: MiniSec: A Secure Sensor Network Communication Architecture. In: *IPSN 2007* (April 2007)
- [7] Zhu, S., Setia, S., Jaodia, S.: LEAP+: Efficient Security Mechanisms for Large-Scale Distributed Sensor Networks. *ACM Transactions on Sensor Networks* 2(4), 500–528 (2006)
- [8] Abbasi, A.A., Younis, M.: A survey on clustering algorithms for wireless sensor networks. *Computer Communications* 30, 2826–2841 (2007)
- [9] Bandyopadhyay, S., Coyle, E.J.: An Energy Efficient Hierarchical Clustering Algorithm for Wireless Sensor Networks. In: *Proceeding of IEEE INFOCOM 2003, San Francisco* (April 2003)
- [10] Handy, M.J., Haase, M., Timmermann, D.: Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In: *Mobile and Wireless Communications Networks*, pp. 368–372. *IEEE Communications Society, Stockholm* (2002)
- [11] Manjeshwar, A., Grawal, D.P.: TEEN: A protocol for enhanced efficiency in wireless sensor networks. In: *PDPS 2001*, pp. 2009–2015. *IEEE Computer Society, San Francisco* (2001)
- [12] Abdalla, M., Bellare, M.: Increasing the Lifetime of a Key: A Comparative Analysis of the Security of Re-keying Techniques. In: *Okamoto, T. (ed.) ASIACRYPT 2000. LNCS*, vol. 1976, pp. 546–559. *Springer, Heidelberg* (2000)
- [13] Du, W., Deng, J., Han, Y., Chen, S., Varshney, P.: A key management scheme for wireless sensor networks using deployment knowledge. In: *INFOCOM 2004: Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 1. *IEEE* (2004)
- [14] Chan, H., Perrig, A., Song, D.: Random key predistribution schemes for sensor networks. In: *Proceedings of Symposium on Security and Privacy*, pp. 197–213. *IEEE* (2003)

A Bullet-Proof Verification Using Distributed Watchdogs (BPV-DW) to Detect Black Hole Attack in Mobile Ad Hoc Networks^{*}

Firoz Ahmed, Seok Hoon Yoon, and Hoon Oh^{**}

School of Electrical Engineering, University of Ulsan, P.O. Box 18, Ulsan 680-749,
South Korea

jewelraaz@yahoo.com, {seokhoonyoon, hoonoh}@ulsan.ac.kr

Abstract. In mobile ad hoc networks, a malicious node can respond with a false Route Reply (RREP) message to the source, when it receives a Route Request (RREQ). Then, it can absorb data packets destined for destination. We propose a Bullet-Proof Verification using Distributed Watchdogs (BPV-DW) in which a detection node uses an encrypted verification message whose hop-by-hop delivery is watched by multiple watchdogs. In this approach, firstly, every node examines suspicious nodes by collecting a message or inspecting the data transmission behaviors of its neighbors. Secondly, upon receiving RREP from any suspicious node, a node verifies whether the suspicious node is a black hole node or not. This two-step approach not only pins down the black hole nodes, but also reduces control overhead significantly. We prove by resorting to simulation that the BPV-DW is highly dependable against the black hole attack.

Keywords: Black hole attack, MANET, encryption, distributed watchdog, AODV.

1 Introduction

In mobile ad hoc networks (MANETs), every mobile node (MN) acts as a router and an MN can join or leave the network without permission from a management entity. Thus MANETs are vulnerable to various kinds of attacks such as black hole attack [1], worm hole attack [2] and so on. We address the problem to detect the black hole attack when AODV [3] is employed for routing in MANETs.

In AODV, a malicious node may respond with an RREP that includes a high sequence number so that it can absorb data packets on the way to the destination. This type of attack is called black hole attack. There are two type of black hole attacks: A single black hole attack in which an attacking node acts alone and a colluding black hole attack in which an attacking node collaborates with another one.

^{*} ULSAN Metropolitan City and the MEST (Ministry of Education, Science and Technology).

^{**} Corresponding author.

The single black hole attack has been tackled in various ways. Some focused on verifying the correctness of the obtained path through the downstream node of the RREP initiator [1], [4]. [5] Uses a watchdog mechanism such that a node watches the misbehavior of its downstream node. However, these approaches may not work appropriately if two black hole nodes cooperate with each other. Meanwhile, only a few methods have been proposed to tackle the colluding black hole attack. In SNV [6], an RREP initiator is required to issue a sequence number request (SREQ) message to the destination which is supposed to respond with SREP including its sequence number to the source. Then, the source node compares the sequence numbers in RREP and SREP. However, it always issue control message to destination and a node that detects an anomaly of malicious node always floods the network with an alarm message which is supposed to be delivered to a source.

The approaches discussed above suffer from high overhead by using flooding or additional messages as well as the failure to address colluding attack. We propose a new approach Bullet-Proof Verification using Distributed Watchdogs (BPV-DW). Our method uses an encrypted verification message whose hop-by-hop delivery is observed by multiple watchdogs. Thus, the malicious behaviors of a node on the path such as the fabrication, dropping or absorption of the message can be detected effectively.

Simulation results show that the BPV-DW can reduce control overhead and improve the reliability of a black hole detection considerably compared to SNV. Furthermore, the packet delivery rate of BPV-DW reached 0.81 even in the presence of five malicious nodes; that of AODV went down to 0.03. The BPV-DW can be equally applied to DSR [7].

2 Background

2.1 Network Model

The network consists of a number of mobile nodes with a limited transmission range where a mobile node can join or leave the network freely. Every node can act as a router and can communicate with any other node in the same network directly or via multiple wireless hops. A number of black hole nodes with a malicious intention can intrude the network. A routing protocol is used to establish a path between any two parties that want to communicate. We assume that the routing protocol implemented in this network is the AODV protocol.

2.2 Problem Identification

The watchdog mechanism [5] does not work effectively if multiple malicious nodes collaborate in order to deceive the watchdog such that one malicious node forwards the receiving packet to another malicious one on purpose. Another recently proposed method is the sequence number verification (SNV) [6] method in which source

always verifies the sequence number contained in RREP by receiving a current sequence number from destination. Accordingly, the SNV method works under the rule, “every RREP initiator sends a message to destination to ask for the destination to report its current sequence number to the source.” However, this method has some shortcomings in two ways:

- It can make a false decision such that either a normal node is determined to be malicious or a malicious node is to be normal; and
- It incurs high network overhead because the source that is multi-hop away from RREP-initiator always verifies the correctness of destination sequence number and a node that detects an anomaly of malicious node always floods the network with an alarm message which is supposed to be delivered to a source.

For convenience of discussion, we borrow some definitions given in the paper [8]. When a detection method is used to determine whether a node is malicious or not, it produces one of the following four decisions.

- TN (True Negative): The detection algorithm determines a normal node to be normal, thereby producing no alarm message
- FP (False Positive): The detection algorithm determines a normal node to be malicious, thereby producing an alarm message
- FN (False Negative): The detection algorithm determines a malicious node to be normal, thereby producing no alarm message
- TP (True Positive): The detection algorithm determines a malicious node to be malicious, thereby producing an alarm message

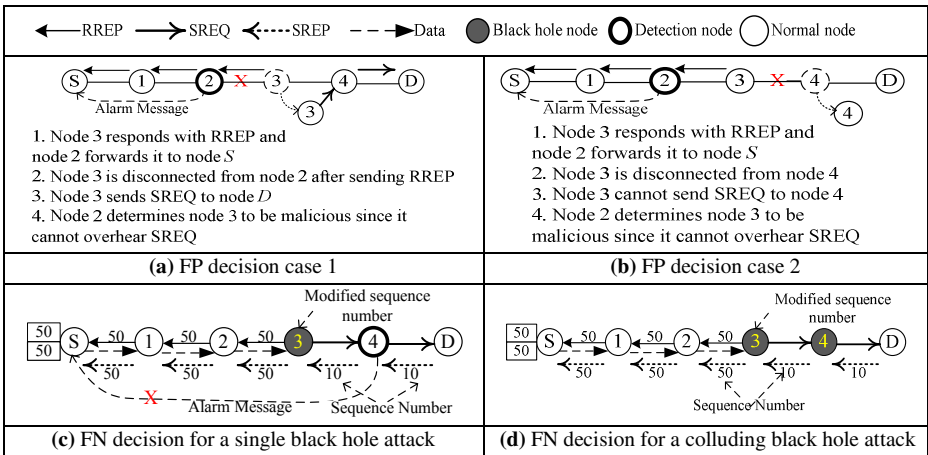


Fig. 1. Shortcomings of the SNV approach

1. It can make an FP decision frequently in relatively high mobility networks:
 - A. A node may determine a normal RREP originator to be malicious if it is disconnected from the RREP originator immediately after it has received RREP, resulting in the initiation of an alarm message erroneously (see Fig. 1-(a)).
 - B. A node may determine a normal RREP originator to be malicious if the RREP originator fails to send SREQ to its downstream node due to a link breakage as shown in Fig. 1-(b).
2. Even though a node detects a malicious node correctly, resulting in a TP decision, the alarm message can be lost easily by collision or link breakage since it has to go through multiple wireless hops. Thus, source may not notice the existence of a malicious node. So, this TP decision will turn to an FN eventually (see Fig. 1-(c)).
3. Consider the colluding attack as illustrated in Fig. 1-(d), If there does not exist a third node that connects commonly to the two colluding malicious nodes, the modification of sequence number cannot be detected, leading to an FN decision.

Problem 1-A and 1-B can be resolved if the upstream node of an RREP originator initiates a verification message and if the RREP originator sends error message to detection node. In problem (2), if the upstream node of a RREP originator is a verifier instead of source node it can be resolved easily. In case of problem (3), if we can prevent a node from modifying the test message and if the detection does not receive the test message with in the estimated time it would not take any action. In this case, the source will always try to consider other RREPs that it receives in order to establish a path. We propose a new method to detect a black hole based on the problems identified above.

3 BPV-DW Method

3.1 Local Decision Process

Even though any malicious node joins a network, if a node forwards data packets to a reliable node, the network will work safely. A node determines whether another node is reliable or not using the watchdog mechanism traditionally such that every node observes the communication behaviors of its neighbors by overhearing data packets transmitted by its neighbors. One simple way that decides the reliability of a node is to know whether or not the node has ever sent data packet to any reliable node. It starts with the fact "I am reliable." A node is said to be reliable if it has forwarded data packet(s) to some reliable node. Therefore, if a node has received data packet from its neighbor before, the neighbor is reliable. A node is said to be suspicious if it has never forwarded data packet to any node. So, initially nodes are suspicious to each other. Even though a node has never received data packet from one of its neighbors, it cannot say that the neighbor is suspicious, since the neighbor may have forwarded data packet to other reliable nodes.

However, suppose that a node knows that its neighbor has forwarded data packet to a third node, but the node does not know whether the third node is reliable or not. Whenever this situation occurs, if the node initiates any costly inspection algorithm, it

may cause too high overhead since those situations occur frequently. In this case, such a neighbor is said to be non-decidable. If a node determines its neighbor to be non-decidable, the node collects further data to inspect the receiver of the packets transmitted by the non-decidable node and examines whether or not the non-decidable node can turn to a reliable node based on the data. If the non-decidable node has forwarded packets to some nodes other than the node to which it forwarded the packet that I sent, it may turn to a reliable node. However, if a black hole node absorbs packets conditionally, the correctness of this initial decision will decrease. After the initial decision by the watchdog mechanism, if a node is still suspicious, it is verified by a global verification mechanism as the second step.

For a local decision, each node collects data by overhearing the packets that its neighbors transmit and maintains a data collection table (DCT) with those data as follows.

$$DCT_i = (j, From_j, Through_j, Suspicious_j), j \in i.N, \text{ where}$$

- $i.N$ is a collection of node i 's neighbors;
- $From_j$ indicates whether or not node i has received a packet from node j ever;
- $Through_j$ indicates whether or not node i has routed a packet via node j ever; and
- $Suspicious_j$ indicates whether or not node j is suspicious based on the combination of $From_j$ and $Through_j$ fields.

The values of $From_j$, $Through_j$, and $Suspicious_j$ are given true (1), false (0), non-decidable (x).

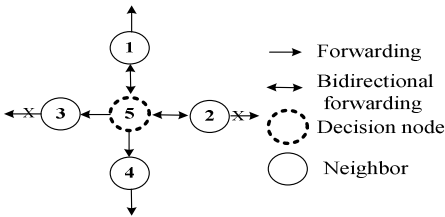


Table 1. An example of DCT_5

j	$From_j$	$Through_j$	$Suspicious_j$
1	1	1	0
2	1	0	0
3	0	0	1
4	0	1	x

Fig. 2. An example topology to define DCT

Fig. 2 shows an example of a small network topology and a data collection table for the network. Node 5 observes the data forwarding behaviors of its neighbors 1, 2, 3, and 4 and records them in its data collection table, DCT_5 as shown in Table 1. Node 5 has received data packet from node 1 and node 2 ($From_1 = 1, From_2 = 1$). Node 5 can determine that node 1 and node 2 are reliable ($Suspicious_1 = 0, Suspicious_2 = 0$). However, node 5 did not receive any data from node 3 and node 4. Since node 3 did not forward my data packet to anyone, it is determined to be suspicious ($Suspicious_3 = 1$). Although node 5 has routed data packet through node 4, it cannot know whether or not node 4 has forwarded to a reliable node. Thus, node 4 is not decidable ($Suspicious_4 = x$): It may collude with some other node for a black hole attack.

We employ a supplementary mechanism to further reduce the number of initiations for a global verification process. If multiple different sessions or paths go through the non-decidable node, multiple nodes may forward data packets to the non-decidable node. A decision node can count the number of different downstream nodes to which the non-decidable node has forwarded data packets by using the watchdog mechanism, as *cntDiffDownNodes*. If the *cntDiffDownNodes* is equal to and greater than *DiffDownNodes*, the non-decidable node turns to a reliable node. We can say that the correctness of the change of the decision will depend on the value of *DiffDownNodes*.



(a) Another path, but same downstream node (b) Another path, but different downstream node

Fig. 3. Data forwarding behavior of a non-decidable node

Fig. 3 shows two different data forwarding behaviors of a non-decidable node. In Fig. 3-(a) and (b), node *x* cannot decide the status of node *y* from its own forwarding. However, node *y* is more likely to be suspicious if it forwards the data packet of another session to the identical node *z* as in Fig. 3-(a) (*cntDiffDownNodes* = 1). If node *y* forwards data packet to another downstream node as in Fig. 3-(b) (*cntDiffDownNodes* = 2), the probability that node *y* is reliable may increase.

3.2 Global Verification Process

If a source or an intermediate node receives RREP from a reliable node, it takes the exactly the same process as AODV. That is, the source starts sending data packets while the intermediate node forwards the RREP toward the source. If a node receives RREP from a suspicious node, it becomes a detection node that initiates a verification process to check if the suspicious node is malicious.

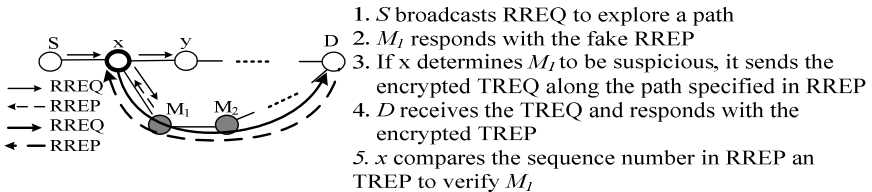


Fig. 4. Verification process

Detection node extracts destination sequence number from the RREP and stores it in its cache. It then generates a Test Request message, $TREQ = (detection\ node\ address, destination\ node\ address, timestamp)$ where the *timestamp* indicates a current time. The detection node encrypts TREQ using public key cryptosystem, being a bullet-proof message and sends it along the path specified in the RREP towards destination. A node that receives the TREQ relays it to next node. No node can alter the bullet-proof TREQ. If the destination receives the TREQ, it decrypts the message and creates a Test Reply message, $TREP = (detection\ node\ address, destination\ node\ address, timestamp, dsn)$ where *dsn* indicate current destination sequence number. The destination encrypts the TREP and sends it along the reverse path to the detection node. Upon receiving the TREP, the detection node decrypts it. If the detection node finds that the *dsn* in TREP is smaller than that in RREP, it judges the suspicious node is black hole and drops the RREP. If the suspicious node is determined to be a reliable node, the detection node starts sending data packets to the destination if the detection node is source; otherwise, it forwards the RREP towards the source node.

Fig. 4 illustrates a verification process. When a suspicious node M_1 responds with RREP, detection node x sends the encrypted TREQ to destination D along the path specified in RREP. Upon receiving the TREQ, Node D responds with the encrypted TREP along the reverse path towards x . M_1 and M_2 cannot alter the contents of the encrypted RREP. Detection node x judges whether M_1 and is reliable or not by comparing the sequence numbers in RREP and TREP.

In the verification process, even though a malicious node is prohibited from modifying the TREQ or TREP, it may drop the messages to hinder the verification process. Thus, we can consider two approaches to cope with such a behavior. One simple mechanism is to associate a timer with each verification process. If detection node does not receive TREP until the timer expires, it refuses to forward the RREP to its upstream node. Then, the source will consider other paths contained in some other RREPs that it has received. The second is a distributed watchdog mechanism.

3.3 Distributed Watchdog Mechanism

In this method, when the TREQ and TREP packet is forwarded every node in the network acts as a watchdog. A node is required to take one of the following responsive actions depending on its role when it receives TREQ or TREP:

- (a) A detection node sends a THANK message to the forwarder of TREP;
- (b) A destination node responds with TREP, upon receiving TREQ;
- (c) An intermediate node forwards TREQ or TREP toward the destination or detection node; and
- (d) If it fails to send the corresponding message, it broadcasts TERR.

If a node detects that some node does not obey the rules, it is immediately determined to be malicious. The purpose of these rules is to detect whether or not some node drops a test message since a malicious node can drop the test messages on purpose.

Note that the malicious node cannot modify the messages which are encrypted. When a certain node determines a node to be malicious, it does not have to inform the detection node. It simply generate a alarm message against the malicious node

4 Performance Evaluation

4.1 Simulation Environment and Performance Metrics

Using the NS-2 [9] simulator, BPV-DW was evaluated against the SNV approach. The used simulation parameters are given in Table 2. The simulation for each scenario and metric was performed five times, taking the average value. We use two metrics, packet delivery ratio (PDR) and control overhead (CO) for comparative evaluation.

Table 2. Simulation parameters

Parameter	Value	Parameter	Value
Mobility model	Random waypoint	Number of sessions	15
Number of nodes	50	Simulation time	300 sec.
Terrain range	1000 * 1000 m ²	Packet size	512 bytes
Maximum speeds	0, 5, 10, 15, 20, 25m/s	Packet transmission rate	4 packets/s
Pause time	30s	Number of malicious nodes	1, 2, 3, 4, 5

$$PDR = \frac{\sum_{i \in S} nPacketsReceived(i.d)}{\sum_{i \in S} nPacketsSent(i.s)}, \quad CO = \sum_{m \in M} \text{the number of } m\text{'s transmitted}$$

where S is a set of sessions created during simulation, and $nPacketsReceived(i.d)$ and $nPacketsSent(i.s)$ are the number of packets received at destination d and sent from source s for session i , respectively. M is all types of control message and the number of m 's transmitted include the messages initiated or relayed by the nodes in the network.

Furthermore, we use two more metrics to evaluate the reliability of two approaches, a true positive rate (TPR) and a false positive rate (FPR).

$$TPR = \frac{TP}{FN + TP} \quad \text{and} \quad FPR = \frac{FP}{TN + FP}$$

TPR indicates the ratio of the number of alarm messages generated for the attacks to the total number of black hole attacks, and FPR indicates the ratio of the number of counts that a normal node is determined to be malicious to the total number of counts that a decision is made for normal nodes.

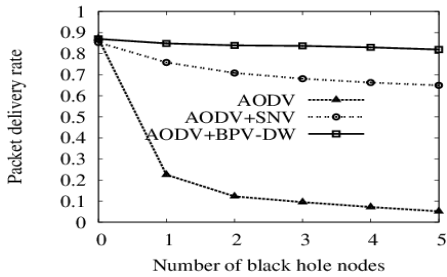


Fig. 5. Packet delivery rate versus Number of black hole nodes

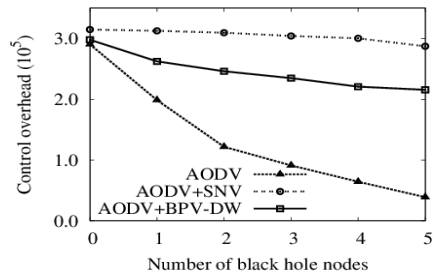


Fig. 6. Control overhead versus Number of black hole nodes

Fig.5 shows the packet delivery rate by varying the number of malicious nodes from 0 to 5. The packet delivery rate of all schemes performs well in case of no black hole node. A significant result is that the packet delivery rate of AODV dramatically drops from 88 percent to 21 percent in the presence of one black hole node and it becomes worse as the number of black hole nodes increase while BPV-DW shows a very stable outcome. BPV-DW has better performance than SNV because it can detect the black hole node more efficiently than SNV. Contrary to our expectation, Fig. 6 shows the control overhead of AODV sharply decreases as the number of malicious nodes increases while that of BPV-DW is not sensitive to the number of black hole nodes. Since the presence of a black hole node often hinders the normal operation of the protocol, the protocol in the network with a black hole node generates less control overhead than that in the network without a black hole node. However, SNV produces more overhead than BPV-DW; the reason is as explained in section 2.2.

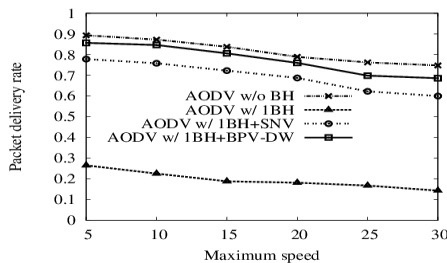


Fig. 7. Packet delivery rate versus Maximum speed

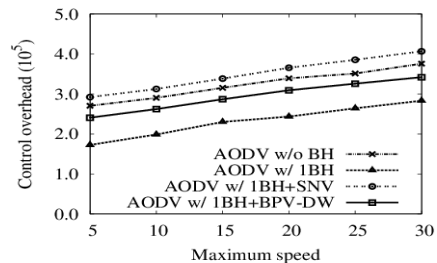


Fig. 8. Control overhead versus Maximum speed

In Fig. 7 and Fig. 8, we present packet delivery ratio and control overhead (CO) as a function of a maximum speed with one black hole node. As shown in Fig. 7, the packet delivery rate of all schemes decreases when the node mobility increases. On the other hand, the packet delivery rate of BPV-DW is over 68 percent even with

30m/s, whereas AODV with one black hole node went down to 15 percent. The packet delivery rate of SNV is less than that of BPV-DW because SNV cannot judge black hole attack effectively. Fig. 8 shows that $CO(BPV-DW \text{ w/ BH}) < CO(AODV \text{ w/o BH})$, the BPV-DW uses additional control messages, increasing control overhead; however, since control messages are unicast and conditional, the increase in overhead is not that serious. While the black hole node may limit the flooding of the RREQ partially, with the AODV in the network without a black hole node, the source floods the network with the RREQ without any hindrance. However, $CO(BPV-DW \text{ w/ BH}) < CO(SNV \text{ w/ BH})$, the reason is as explained in section 2.2.

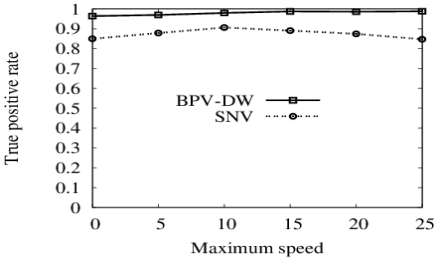


Fig. 9. True positive rate versus Maximum speed

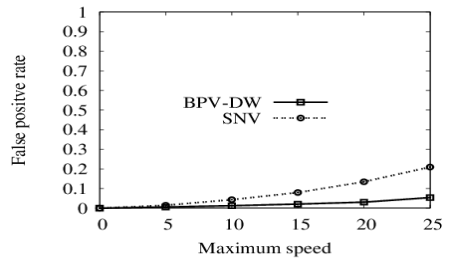


Fig. 10. False positive rate versus Maximum speed

Fig. 9 and Fig. 10 compare the true and false positive rate between BPV-DW and SNV when maximum speed varies from 0 to 25 m/s and 10% of the total nodes are black hole nodes. As shown in Fig.9, the true positive rate of SNV is lower than that of BPV-DW. This is due to the fact that in SNV scheme, only source node decides that a node is malicious. Therefore, if a certain node detects anomaly of a black hole node it broadcasts alarm message that is supposed to be delivered to the source. If the alarm message fails to reach the source, the scheme cannot detect black hole node. From the Fig. 10 we can observe that the false positive rate in both schemes increases when the nodes move more rapidly because links are broken frequently in a high mobility network. We also observe that the false positive rate of SNV is higher than that of BPV-DW. This is due to the fact that in SNV, if the RREP originator is disconnected from its immediate upstream node or it fails to send SREQ to its downstream node due to a link breakage, the upstream node determines that the RREP originator is black hole.

5 Concluding Remarks

The proposed BPV-DW method takes two steps: Identifying a suspicious node and verifying the suspicious node. The message for verification is encrypted so that it can be directly sent to the destination along the path reported from the suspicious node.

In this verification process, the nodes on the path are required to follow the specified rules and any third nodes can watch out the behaviors against the rules. We showed by simulation that the BPV-DW not only reduces control overhead but also it identifies a malicious node effectively.

References

1. Deng, H., Li, W., Agarwal, D.P.: Routing security in ad hoc networks. *IEEE Communications Magazine* 40, 70–75 (2002)
2. Hu, Y.-C., Perrig, A., Johnson, D.B.: Packet Leashes: A Defense against Wormhole Attacks in Wireless Ad Hoc Networks. In: *Proc. of IEEE INFOCOM* (2002)
3. Perkins, C.E., Royer, E.M., Das, S.: Ad-hoc On demand Distance Vector (AODV) Routing Protocol. RFC 3561 (2003)
4. Lee, S., Han, B., Shin, M.: Robust Routing in Wireless Ad Hoc Networks. In: *International Conference on Parallel Processing Workshops*, pp. 18–21 (2002)
5. Marti, S., Giuli, T., Lai, K., Baker, M.: Mitigating routing misbehavior in mobile ad hoc networks. In: *MOBICOM*, pp. 255–265 (2000)
6. Zhang, X., Sekiya, Y., Wakahara, Y.: Proposal of a method to detect black hole attack in MANET. In: *International Symposium on Autonomous Decentralized Systems*, pp. 1–6 (2009)
7. Johnson David, B., Maltz David, A.: Dynamic source routing in ad hoc wireless networks. *Mobile Computing* 353, 153–181 (1996)
8. Prathapani, A., Santhanam, L., Agrawal, D.P.: Detection of black hole attack in a wireless mesh network using intelligent honeypot agents. *The Journal of Supercomputing (JoS)*, 1–28 (2011)
9. NS-2, <http://www.isi.edu/nsnam/ns/>

Performance Analysis for Workflow Management Systems under Role-Based Authorization Control

Limin Liu¹, Ligang He², and Stephen A. Jarvis²

¹Department of Optical and Electronic Engineering, Mechanical Engineering College, Shijiazhuang, China

²Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK

Abstract. Role-Based Access Control (RBAC) remains one of the most popular authorization control mechanisms. Workflow is a business flow composed of several related tasks. These tasks are interrelated and context-dependent during their execution. Under many circumstances execution context introduces uncertainty in authorization decisions for tasks. This paper investigates the role-based authorization model with the runtime context constraints and dynamic cardinality constraints. The Generalized Stochastic Petri-net is used to model the authorization process. Moreover, due to the state explosion problem in the Petri-net formalism, the proposed modeling method combines the Queuing theory to analyze both system-oriented and user-oriented performance. Given the workflow information, its running context and the authorization policies, this work can be used to predict the performance of these workflows running in the system. The prediction information can give insight in how to adjust authorization policies to strike a better balance between security and performance.

Keywords: Workflow, Role, Authorization, Cardinality, Performance.

1 Introduction

Workflow is a business flow which has a certain target and composed of several related tasks (activities). These tasks are mutually dependent and are executed at a certain fashion. In order to guarantee each task to utilize the system resources under reasonable authorization, there should be an appropriate authorization mechanism in the workflow management system. The access control and authorization mechanism of workflow must not only guarantee that every task of the workflow be executed by the authorized subject, but also evolve with the execution of workflow. When the data is flowing among the tasks of workflow, the task on execution is changing and the privilege of the task is also changing. The privilege changing of tasks is related with the context environment of the data process and the implementation of the workflow authorization depends on the procedure context too.

In recent years the research of access control and authorization focuses on the model of Role Based Access Control 96 (RBAC 96) which is presented by Sandhu and Coyne, et al [1]. The RBAC model is an access control pattern that the subject's access privilege is determined by the given privilege of the role assigned to the subject. The RBAC model arouses researchers' interest more and more because of its flexible authorization mechanism, powerful management function and perfect safety policy. The researchers attach great importance to authorization constraints of the RBAC model too [2]. These studies become a motivation of the development of the RBAC model and a series of researching achievements about constraints are presented [5] [6].

The RBAC model has evolved to the RBAC2 model which is called constrained RBAC model [2] [3]. Authorization constraints specify the mandatory rules that must be observed when access permissions are assigned to roles, roles are assigned to users or a user activates a role at some time. These rules are established in order to satisfy the safety control principle and business needs. In recent years, the research about constraints has becoming an important research point related to the RBAC model. These researches extend the basic constraints and some new constraint types such as relationship constraint, prerequisite constraint and cardinality constraint etc. are presented.

Cardinality constraints can be divided into: (1)Static cardinality constraint: This constraint restricts the number of users of a role and the static cardinality constraint N means the maximum members of the users of a role at any time. For example, there is only one person as a chairman role in an organization. (2)Dynamic cardinality constraint: This constraint restricts the number of simultaneous active sessions of a role. This kind of constraint has more influence on system performance, so this paper will study the system performance under dynamic cardinality constraint.

In this paper, the application of the role-based authorization model under context and dynamic cardinality constraint to workflow system is studied. The queuing theory is utilized to analysis and to predict the system performance. The remainder of this paper is organized as follows: related work is discussed in Section 2; Section 3 presents the role-based authorization model under constraints and utilizes the Petrinet to express the authorization process; Section 4 presents the performance analysis of workflow management system under the model and discusses different parameters' affect on the workflow's performance; the paper concludes and presents future works in Section 5.

2 Related Works

The role-based authorization model and its utilization in workflow authorization has been extensively studied and well documented in related literature [4][6][8]. But the context among tasks in a workflow and the corresponding role-based authorization constraint is not considered in these works. [9] has studied the TBAC (Task Based Access Control) model which was an authorization model based on tasks of a workflow instance. This model has considered the context constraints among workflow tasks, but the role organizations, role logics, the roles' relationship with system resources and the model's effect on workflow performance were not included in their studies. Botha and Eloff et al. have studied the issue of Separation of duty (SoD) in workflow environment and

presented four types of needs for SoD: conflicting roles, conflicting permissions, conflicting users and conflicting tasks [4]. Binding of duty (BoD) is another type of authorization constraints. [13][14] introduced various constraints of the RBAC model, but they did not present the concrete scenarios of these constraints. [17] studied the relationship constraints and prerequisite constraints and also used the Petrinet to study SoD and BoD in workflow under role-based authorization mechanism. They presented the model implementation explicitly and used CTMC to study the workflow performance. The task context and cardinality constraint’s implementation in a role-based authorization system was not included in their study and they did not give these constraints’ effect on system performance too.

3 Role-Based Workflow Authorization Model

3.1 The Role Hierarchy in a System

As depicted in Fig.1, the relationship among the roles in a system or organization is a membership one as an inverted tree. If there are different membership relationships like a net because of different business interactions, the net membership can be transformed into tree membership according to the business interaction. The authorization service of the system can keep a role data structure according to membership relationship between roles. In a real system under role-based authorization mechanism, the authorization service will traverse the role tree-type data structure and find a spare role to assign it to the task T if a role and its father roles can be assigned to the task T. In principle, the role and all its father roles up to root node can be assigned to task T. The traversing order from the role to root node is the priority level to assign a role to task T and this level is the reverse of the tree-like membership order from the top of the tree.

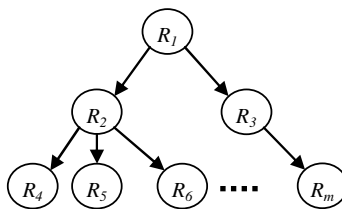


Fig. 1. The role hierarchy in a system

3.2 Authorization under Role Cardinality and Context Constraint

Suppose that the requested role of a task is R and the active cardinality constraint of the role R is n . The authorization rule to the task under R 's cardinality constraint is modeled by PN as in Fig.2. The diagram in the dotted block is the temporal constraint expression of roles [17]. Suppose the active cardinality of role R is n . The number of the tokens in

place $R-C$ stands for the remaining available number of the role R and the maximum number is n . When Task requests the authorization of R and the active task sessions of R are fewer than n , Task can obtain the authorization successfully. The place RT , \overline{RT} and the transitions between them construct the temporal constraints of the role's availability. The transition AoR is the assignment action of the role R to the task. This transition needs three input tokens: one from the place $Task$ which stands for the existence of an input task; one from the place $R-C$ when there are remaining available active role R ; one from place RT which stands for the temporal availability of the role. The arc between the place RT and the transition AoR is bidirectional and the transition AoR will not consume the token in the place RT . After the execution of transition AoR , there will be a token in the place $(T, Role)$, so the system can take the next $(Role, Res)$ authorization step. The token in place Res stands for the required resource and the transition AoE stands for the assignment of resource step. After the execution of AoE , there will be a token in place (T, Res) , the execution of the task can be initiated. After the execution of transition TC , there will be a token deposited into place $R-C$. Thus the assignment of a role to a task and the execution of the task constitute a closed loop.

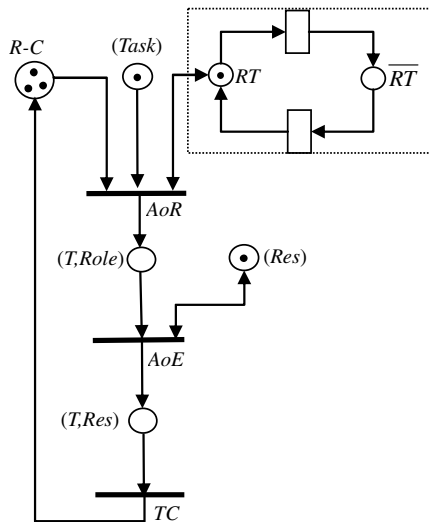


Fig. 2. Task-role assignment under active cardinality constraint

When the active sessions of R reach the maximum numbers of its cardinality limit or R is temporally unavailable, there should be a place or a state which indicates its unavailability in order to wait for the completion of one of the active tasks of R or apply for the authorization of father roles of R according to the organizational role inheritance relationship if the authorization policy permits. The figure showing the generation of the *RoleBusy* place is depicted as Fig.3.

The diagram in the dotted block is the added part which indicates the generating process of the *RoleBusy* state. The other part of the figure is same as Fig.2.

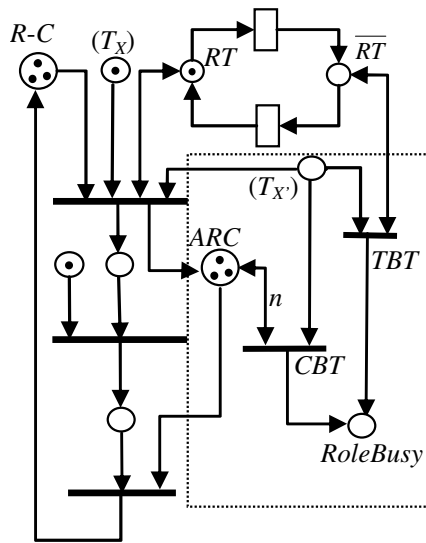


Fig. 3. Task-role assignment under active cardinality constraint with *RoleBusy* state

The place (T_x') can be identical to place (T_x) when a task can apply for the authorization of R and its father roles. Otherwise (T_x') can be a place indicating the arrival of another task which has task context constraint with the task indicating by the place (T_x) . Task context constraint will be discussed in the following section. The place ARC is the anti-place of $R-C$ and the tokens in it stand for the number of active sessions of R . Transitions TBT and CBT are transitions indicating R 's temporal availability and the number of R 's active session respectively. The arcs between \overline{RT} and TBT , ARC and CBT are bidirectional ones. The arc function of the arc between ARC and CBT is n , which is the active cardinality of R . Both transitions are triggered by the token in (T_x') . Thus when R is temporally unavailability or the active sessions of R have reached its cardinality, there will be a token in place *RoleBusy* if there is a task requesting token in (T_x') .

3.3 Authorization under Task Context Constraint

Because workflow is a business flow which has certain target and composed of several related tasks (activities), there are SoD, BoD and task context constraint between the roles assigned to the tasks of a workflow instance in terms of role authorization. SoD and BoD have been extensively studied and well documented in related literatures [2, 3, 4, 5, and 11], so the task context constraint is mainly studied in this paper.

From the view of role hierarchy, the context constraint of roles between tasks can be classified into the following two modes:

- 1) $T_a \xrightarrow{L} T_b$: the rank of the role assigned to T_a is lower than the one assigned to T_b .
- 2) $T_a \xrightarrow{H} T_b$: the rank of the role assigned to T_a is higher than the one assigned to T_b .

The implementation of the role hierarchy depicted in Fig.1 in an authorization service can be an appropriate data structure such as adjacency multilist.

The Authorization Process of $T_a \xrightarrow{L} T_b$. The context constraint of task T_b to T_a is modeled in Petrinet and the authorization process of $T_a \xrightarrow{L} T_b$ is illustrated as in Fig.4.

In the figure, the rank of the role requested by task T_b is higher than that of its antecedent task T_a . $R_{f1}-R_{fn}$ are the father roles of R which is the role assigned to the task T_a . $R_{f1}-R_{fn}$ are sorted from low to high rank in order to observe the least privilege principle. The transition in the Fig.4 is illustrated as bold solid line.

Add a place (T_a, R) in the Petrinet and deposit a token to it when role R is assigned to task T_a by the authorization service successfully. This place is also an input place of the authorization to the descendant task T_b , so the context constraint authorization constraint between task T_a and T_b can be guaranteed.

The execution of transition CBT and TBT deposits a token in place $RoleBusy$. The token in this place means that the current role is busy and can not be assigned to new-coming task. At this time, the neighbor higher rank role of the current one will be checked if can be assigned to the new-coming task. The Fig.3 is used as part of the Fig.4.

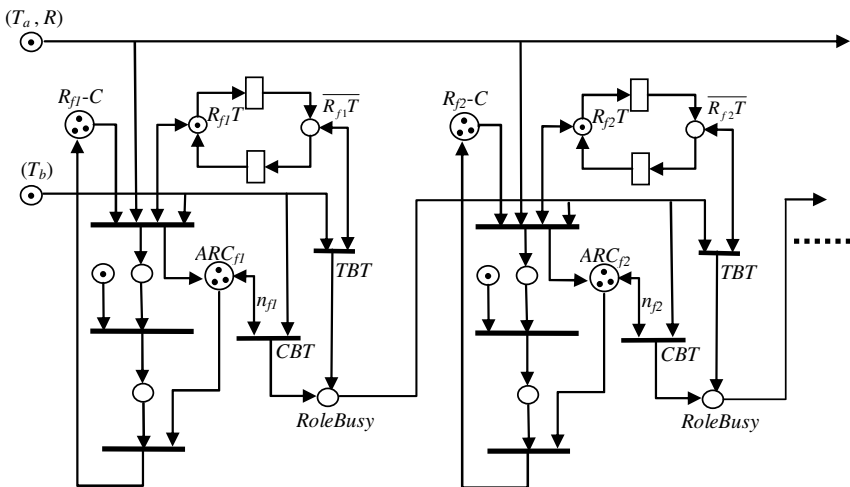


Fig. 4. Task-role assignment under task context constraint

The arcs from place $ARC_{(f_i)}$ to the transition CBT are bi-directional ones. The arc expressions of these arcs are n_{f_i} which is the active cardinality value of the roles $R_{f_1}-R_{f_n}$ respectively.

The tokens in place (Ta, R) and Tb are the triggers of the task-role assignment under task context constraint.

The inputs of the transition *Role Assignment* of R_{f_1} need tokens from four places: (T_a, R) , T_b , $R_{f_1}-C$ and $R_{f_1}T$. The inputs of the transition *Role Assignment* of R_{f_2} need tokens from four places: (T_a, R) , $R_{f_2}-C$, $R_{f_2}T$ and *Rolebusy* of R_{f_1} . Similarly, the inputs of the transition *Role Assignment* of R_{f_i} needs token from four places: (T_a, R) , *Rolebusy* of $R_{f_{i-1}}$, $R_{f_i}-C$ and $R_{f_i}T$ respectively. With this method, the role assignment can observe the task context constraint and the least privilege principle.

The Authorization Process of $T_a \xrightarrow{H} T_b$. Suppose R is the role assigned to task T_a . When the rank of the role requested by task T_b is lower than that of its antecedent task T_a and is not claimed explicitly, the descendants of role R should be traversed from the role hierarchical data structure and the role authorization of task T_b should be conducted from low to high ranks of these descendants in order to observe the least privilege principle. The authorization process of $T_a \xrightarrow{H} T_b$ is same as that of $T_a \xrightarrow{L} T_b$ except that the $R_{f_1}-R_{f_n}$ in Fig.4 are substituted by $R_{s_1}-R_{s_n}$ which are the descendants of role R sorted from low to high ranks.

4 Performance Analysis of Workflow Management System under Constraints

4.1 System Parameters

Suppose the parameters of the workflow management system under analyzed are as the following list.

Workflow instance set (N elements): S_{WF} : $\{WF_1, \dots, WF_N\}$, there are N types of workflow instance in the system, WF_i is the i_{th} workflow instance;

Role set (M elements): S_R : $\{R_1, \dots, R_M\}$, there are M roles in the workflow management system;

Roles active cardinality set (M elements): S_{CR} : $\{CR_1, \dots, CR_M\}$, CR_i is the active cardinality value of R_i ;

The portion of every workflow instance type in the system: $\{\alpha_1, \dots, \alpha_N\}$ and $\alpha_1 + \dots + \alpha_N = 1$;

The Poisson arrival parameter set of every workflow instance type: $\{\lambda_{WF_1}, \dots, \lambda_{WF_N}\}$, λ_{WF_i} is the Poisson arrival parameter of the i_{th} workflow instance;

Task set (L elements): S_T : $\{T_1, \dots, T_L\}$, there are L types of tasks supported by system resources and authorization service. The tasks of this set constitute the N types of workflow instance;

The role's mean service time parameter set of each task: V_μ : $\{\mu_1, \mu_2, \dots, \mu_L\}$;

The mapping matrix of task set S_T and workflow instance set S_{WF} can be constructed as:

$$C_{N \times L}: \{ c_{ij} \}$$

In the matrix, c_{ij} stands for mean numbers of the task T_j appearing in the workflow instance WF_i and $c_{ij}=\varphi$, the value of φ is 0 or positive integer.

4.2 System Analysis under Role Cardinality Constraints

System performance is related with processing time of role and the service time of resource. In order to know which role is requested by which task, the mapping matrix $D_{M \times L}$ between tasks and requested roles is constructed and the value of the element d_{ij} in the matrix is:

$d_{ij}=\zeta$, ζ is the probability that role R_i is assigned to task T_j successfully before its execution and $0 \leq \zeta \leq 1$.

The Calculation of d_{ij} . The calculation of d_{ij} can be divided into two different cases:

1) Non-context constraint’s authorization

Tasks that have no context constraint’s authorization should have an explicit role authorization request. This request is derived from the workflow’s business procedure. The authorization request can be obtained from the authorization request file of the workflow instance (a XACML file, for example).

The value of d_{ij} is one when the task T_j has the explicit role authorization request of R_i and zero when it has not a role authorization request of R_i .

2) Context constraint’s authorization

The case of context constraint’s authorization of a task is presented by the Fig.4. In this case, the value of d_{ij} must be obtained from the calculation of authorization probability of each dependant roles. Take the Fig.4 as the example, the model presented by Fig.4 is a General Stochastic Petri-net model, and since the time associated with each transition is exponentially distributed, the underlying stochastic process is a Continuous Time Markov Chain[18]. We can construct the Tangible Reachability Graph (TRG) of the Petri-net model and also can construct the CTMC from the TRG[17]. After the CTMC is constructed, we can calculate the infinitesimal generator (which is a two dimensional matrix, denoted by Q) of the CTMC.

Suppose there are M states in the CTMC. p_i is the probability that the CMTC stays in state s_i . If we denote the row vector $P=[p_1, p_2, \dots, p_i, \dots, p_M]$, then the following linear equation system holds, where Q is the infinitesimal generator of the constructed GPSN model. P can be obtained by solving the equation system:

$$\begin{cases} PQ=0. \\ \sum_{i=1}^M p_i=1 \end{cases} \tag{1}$$

The nodes in the TRG are defines by the row vector:

$$N_s=[(T_a, R), T_b, R_{f1}, \overline{R_{f1}}, R_{f2}, \overline{R_{f2}}, \dots, R_{fn}, \overline{R_{fn}}]$$

In this vector, $R_{fi}=R_{fi}-C \cap R_{fi}-T$ which means the availability of role R_{fi} ; $\overline{R_{fi}}$ is *RoleBusy* of R_{fi} which means the unavailability of role R_{fi} .

We suppose that the role authorization of T_b will stay at the queue of all father roles of R and wait for the first spare role if all R 's father roles are busy. Though the first spare probability of all father roles of R can be calculated based on the whole workflow instances accomplished by the workflow management system, we take the uniform distribution of the father roles' spare probability in order to simplify the problem.

The following N_s state lists are the states that can lead to the successfully role authorizations of T_b :

N_s Value List	Probability
$S_1=(1,1,1,0,X,X,\dots,X,X)$	p_1
$S_2=(1,1,0,1,1,0,\dots,X,X)$	p_2
...	
$S_n=(1,1,0,1,0,1,\dots,1,0)$	p_n
$\overline{S}=(1,1,0,1,0,1,\dots,0,1)$	p_{bs}

S_1-S_n are the states that can lead to the successfully role authorizations of T_b to $R_{f1}-R_{fn}$ respectively and \overline{S} is the state that all roles are busy. p_1-p_n and p_{bs} are the probability of these states.

The values of p_1-p_n and p_{bs} can be obtained by the Eq.1. If T_j is a context constraint's authorization task as T_b and R_i is the i_{th} father role of R , the value of d_{ij} can be expressed as following:

$$\xi = p_1 + p_{bs}/n$$

The Queuing Model of Tasks Requesting a Role under Cardinality Constraint.

Suppose the arrival process of the workflow instances supported by workflow management system under studied is Poisson process. The arrival rate of all instances is λ , and then the arrival rate of instance WF_i is

$$\lambda_{WF_i} = \alpha_i \times \lambda;$$

α_i is the proportion of the number of instance WF_i to all workflow instances and we have

$$\sum_{i=1}^N \lambda_{WF_i} = \lambda$$

The arrival rate λ_{Tj} of every task supported by the workflow system can be calculated by the following equation:

$$(\lambda_{T1} \lambda_{T2} \dots \lambda_{TL}) = (\lambda_{WF1} \lambda_{WF2} \dots \lambda_{WFN}) \times C_{N \times L}$$

$$\lambda_{Tj} = \sum_{i=1}^N (\lambda_{WF_i} \times c_{ij})$$

Since all workflow instances' arrival rate are Poisson processes and each task in these instance only has a different parameter: λ_T , we can suppose that the arrival rate of the

R_i 's requesting tasks is a Poisson process. And we can obtain the total arrival rate parameter of these tasks using the former formula:

Suppose λ_{Ri} is the arrival rate of the tasks which request role R_i . The value of λ_{Ri} can be calculated by the Eq.2:

$$\lambda_{Ri} = \sum_{j=1}^L d_{ij} * \lambda_{Tj} \tag{2}$$

The mean service time of R_i to a certain task of different instances from a same workflow is exponential distributed. But the tasks requesting R_i come from different workflows or different transitions of a same workflow instance, so the mean service time of R_i to requesting tasks does not observe the exponential distribution but the independent and general distribution[20][21]. Through computing [16], the mean service time $\frac{1}{\eta_i}$ of R_i to its client is:

$$\frac{1}{\eta_i} = \frac{\sum_{j=1}^N \sum_{h=1}^L c_{jh} * d_{ih} * \lambda_{WFj}}{\sum_{j=1}^N \sum_{h=1}^L c_{jh} * d_{ih} * \lambda_{WFj}} \tag{3}$$

η_i is the service rate of R_i . For each role in the system, we can know both the arrival rate of requesting tasks and the role's service rate using the Eq.2 and Eq.3 respectively. We also know the role's cardinality value which can be used as the number of servers. The role and the requesting tasks constitute an $M/G/C$ queuing system. C is the number of servers in the queuing system and its value is the role's cardinality value.

4.3 System Performance Analysis

Performance Analysis Based on a Certain Role

1) The effect of role cardinality value CR on system performance

Three factors decide if a system will be a limitless waiting queue. These factors are: 1) the value of the role's cardinality, 2) the arrival rate of the requesting tasks and 3) the service rate of the role. λ_{Ri} is the number of clients who need the process of R_i during unit time. $CR_{(Ri)} * \eta_i$ is the number of clients who can be processed by R_i during unit time. In order to keep the workflow system stable, the processing speed of the role must be greater than the arrival rate of clients (tasks):

$$\lambda_{Ri} / (CR_{(Ri)} * \eta_i) < 1 \quad \text{and} \quad CR_{(Ri)} > \lambda_{Ri} / \eta_i$$

If the CR value of a role is determined, the arrival rate of tasks which request the role can be adjusted in order to avoid the limitless waiting queue.

2) Performance analysis

According to the theories of queuing system and stochastic service system [21] [22] [23] [24], the mean waiting time W_{qi} of a task which request Ri in the queue is [20] [26] [27]:

$$W_{qi} = \frac{CR_i \times \lambda_{Ri}^{CR_i}}{A}$$

Where $A = \left(\sum_{k=0}^{CR_i-1} \frac{\lambda_{Ri}^k}{\eta_i^k * k!} + \frac{CR_i * \lambda_{Ri}^{CR_i}}{(CR_i * \eta_i - \lambda_{Ri}) * \eta_i^{CR_i-1} * CR_i!} \right) \times (CR_i * \eta_i - \lambda_{Ri})^2 \times \eta_i^{CR_i-1} \times CR_i!$

By Little’s law, we have:

$$W_i = W_{qi} + 1/\eta_i \tag{4}$$

Where W_i is the sum of the task’s mean waiting time and R_i ’s mean service time and also is the mean processing time of the task under the authorization of R_i .

By the Eq.4, we can obtain the mean task processing times of all the roles in a system at their stable states. All the roles’ mean task processing time can constitute a vector V_W :

$$V_W: (W_1, W_2, \dots, W_M)$$

Performance Analysis of the Whole System

1) The delay time of a workflow instance

As stated in section 4.1, a workflow instance is possibly a business logic composed of different control structures of tasks. The delay time of a workflow instance should be analyzed based on these structures. These control structures can be divided into four basic structures: sequential, concurrent, selective and loop structures. The meaning of basic structure is that the components of the structure are not nested ones but only tasks. The tasks in a control structure can be denoted by the following two vectors:

$$CA: (ca_1, ca_2, \dots, ca_L)$$

$$CT: (ct_1, ct_2, \dots, ct_L)$$

The value of ca_i is 0 or 1 which means whether the task T_i from T_J-T_L is in the structure. The value of ct_i is 0 or positive integer which means the appearance numbers of the task T_i from T_J-T_L in the control structure.

· Sequential structure

The tasks in this structure are several ones from T_J-T_L and they execute sequentially. So the total delay time DT_s of this structure is the sum of the delay time of all the tasks:

$$DT_s = \sum_{i=1}^L (ct_i * \sum_{j=1}^M (d_{ji} * W_j))$$

· Concurrent structure

The tasks in this structure are several ones from T_J-T_L and they execute concurrently. So the total delay time DT_p of this structure is the maximum of the delay time of all the tasks:

$$DT_p = \text{Max} \left(ca_1 * \sum_{i=1}^M (d_{i1} * W_j), ca_2 * \sum_{i=1}^M (d_{i2} * W_j), \dots, ca_L * \sum_{i=1}^M (d_{iL} * W_j) \right)$$

· Selective structure

The tasks in this structure are several ones from T_1-T_L and only one of them executes in the system. The executing probability of T_1-T_L is denoted as p_1, p_2, \dots, p_L respectively and $p_1+p_2+\dots+p_L=1$, then the total delay time DT_o of this structure is:

$$DT_o = p_1 * (ca_1 * \sum_{i=1}^M (d_{i1} * W_j)) + p_2 * (ca_2 * \sum_{i=1}^M (d_{i2} * W_j)) + \dots + p_L * (ca_L * \sum_{i=1}^M (d_{iL} * W_j))$$

$$= \sum_{i=1}^L (p_i * ca_i * \sum_{j=1}^M (d_{ji} * W_j))$$

· Loop structures

The loop structures can be transformed into sequential structure by means of calculating the executing times of corresponding tasks in the loop structure. So the total delay time DT_L of loop structure is delay time of the sequential structure which is transformed from the loop structure.

Every workflow instance is a business logic composed of the task control structures listed above. When there are nested control structures, the delay time of the workflow instance should be calculated from innermost to outermost structures. The delay time of the neighbor inner control structure should be a same item as a task in the delay time calculation of neighbor outer control structure. For example, as in Fig.5:

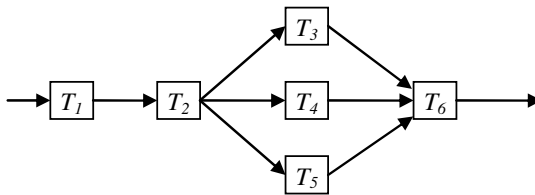


Fig. 5. A workflow instance composed of nested control structures

Suppose a workflow instance is composed of nested control structures as depicted in Fig.5. In order to simplification, the places in the Petri net are omitted and only the tasks as transitions are retained. T_3, T_4 and T_5 are concurrent tasks and they constitute a concurrent control structure: S_L . This structure is the neighbor inner control one of the outer sequential structure which is composed of T_1, T_2, S_L and T_6 . First the delay time DT_{S_L} of the S_L should be calculated:

$$DT_{S_L} = \text{Max} \left(\sum_{i=1}^M (d_{i3} * W_j), \sum_{i=1}^M (d_{i4} * W_j), \dots, \sum_{i=1}^M (d_{i5} * W_j) \right)$$

Then the total delay time DT_{WF} of the workflow instance can be calculated:

$$DT_{WF} = \sum_{i=1}^M (d_{i1} * W_j) + \sum_{i=1}^M (d_{i2} * W_j) + DT_{SL} + \sum_{i=1}^M (d_{i6} * W_j)$$

Using the method stated above, the processing time of a workflow instance with mixture of arbitrary control structures can be obtained.

2) The mean response time of the system

The mean response time MRT_{WF_i} of a workflow instance WF_i is the sum of WF_i 's waiting time and executing time. The sum of waiting time and executing time is the mean delay time. So the mean response time of a workflow instance WF_i is:

$$MRT_{WF_i} = DT_{WF_i}$$

The value of DT_{WF_i} can be obtained by the method introduced above.

With the mean response time of a workflow instance and other system parameters, the mean response time of the whole workflow management system is:

$$MRT_{WFMS} = \sum_{i=1}^N (\alpha_i * DT_{WF_i})$$

3) The occupation rate of roles under active cardinality constraint

The total number of active task sessions supported by the workflow management system under all roles' active cardinality constraint is:

$$ATS_{num} = \sum_{i=1}^M CR_i$$

According to queuing theory and the Little's law, the occupation rate ρ_i of a role R_i is:

$$\rho_i = \lambda_{R_i} / CR_i * W_i$$

Where λ_{R_i} is the arrival rate of the tasks requesting R_i , CR_i is the active cardinality constraint of R_i and W_i is the mean task processing time (service time) of R_i .

With ρ_i , the occupation rate of all roles of the system is:

$$\rho = \sum_{i=1}^M \rho_i / M = \sum_{i=1}^M \frac{\lambda_{R_i}}{CR_i * W_i * M}$$

With the result of ρ , we can adjust the parameters of the workflow system or the cardinality values of roles to maximum the system performance.

5 Conclusions and Future Works

In this paper, we take the workflow management system under role authorization as the studying object and emphasis on the study of system performance under task context and role cardinality constraints. We employ Petrinet to model the authorization procedure with role cardinality constraint and task dependence. The queuing theory is used to analysis the workflow instance delay time, mean response time, role occupation rate and other performance metrics of the workflow management system with multiple

kinds of workflow instances and tasks. Based on the context dependence among tasks in a workflow instance, our future works will focus on the study of uncertain authorization model of roles under task context and its performance analysis. We will employ the probability, fuzzy and queuing theories to analysis and predict the more generic and complicated workflow management under role authorization.

References

1. Sandhu, R.S., Coyne, E.J., Feinstein, H.L., Youman, C.E.: Role-Based Access Control Models. *IEEE Computer* 29(2), 38–47 (1996)
2. Ahn, G., Sandhu, R.: Role-Based Authorization Constraints Specification. *ACM Trans. Information and System Security* 3(4), 207–226 (2000)
3. Ahn, G., Sandhu, R.: The RSL99 Language for Role-based Separation of Duty Constraints. In: *Proceedings of the Fourth ACM Workshop on Role-based Access Control*, Fairfax, Virginia, United States, October 28-29, pp. 43–54 (1999)
4. Botha, R., Eloff, J.: Separation of Duties for Access Control Enforcement in Workflow Environments. *IBM Systems Journal* 40(3), 666–682 (2001)
5. Joshi, J., Bertino, E., Latif, U., Ghafoor, A.: A Generalized Temporal Role-Based Access Control Model. *IEEE Transactions on Knowledge and Data Engineering* 17(1), 4–23 (2005)
6. Wainer, J., Barthelmess, P., Kumar, A.: W-RBAC- A Workflow Security Model Incorporating Controlled Overriding of Constraints. *International Journal of Cooperative Information Systems* 12(4), 455–486 (2003)
7. Bertino, E., Bonatti, P.A., Ferrari, E.: TRBAC: a Temporal Role-based Access Control Model. In: *Proceedings of the Fifth ACM Workshop on Role-based Access Control*, Berlin, Germany, July 26-28, pp. 21–30 (2000)
8. Wang, Q., Li, N.: Satisfiability and Resiliency in Workflow Authorization Systems. *ACM Transactions on Information and System Security (TISSEC)* 13(4), 1–35 (2010)
9. Thomas, R.K., Sandhu, R.S.: Task-based Authorization Controls (TBAC): A Family of Models for Active and Enterprise-oriented Authorization Management. In: *IFIP TC11 WG113 11th International Conference on Database Security XI Status and Prospects*, vol. 11, pp. 166–181. Chapman & Hall, Ltd
10. Bertino, E., Ferrari, E.: An authorization Model for Supporting the Specification and Enforcement of Role-based Authorization in Workflow Management Systems. *ACM Transactions on Information and System Security* 2(1), 65–104 (1999)
11. Castano, S., Casati, F., Fugini, M.: Managing Workflow Authorization Constraints through Active Database Technology. *Information Systems Frontiers* 3(3), 319–338 (2001)
12. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers* 8(1), 21–66 (1998)
13. Ray, I., Li, N., France, R., Kim, D.K.: Using UML to Visualize Role-based Access Control Constraints. In: *Proceedings of the 9th ACM Symposium on Access Control Models and Technologies*, Yorktown Heights, New York, USA, June 02-04, pp. 115–124 (2004)
14. Tan, K., Crampton, J., Gunter, C.: The Consistency of Task-based Authorization Constraints in Workflow Systems. In: *Proceedings of 17th IEEE Computer Security Foundations Workshop*, pp. 155–169 (2004)
15. Liu, S., Fan, Y.S.: Workflow Model Performance Analysis Concerning Instance Dwelling Times Distribution. In: *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA*, pp. 601–605 (2009)

16. Li, J.Q., Fan, Y.S., Zhou, M.C.: Performance Modeling and Analysis of Workflow. *IEEE Transactions on System, Man, and Cybernetics A* 34, 229–242 (2004)
17. He, L., Calleja, M., Hayes, M., Jarvis, S.A.: Performance Prediction for Running Workflows under Role-based Authorization Mechanisms. In: *IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8 (2009)
18. Manolache, S.: *Schedulability Analysis of Real-Time Systems with Stochastic Task Execution Times*. Ph.D Thesis, Department of Computer and Information Science, IDA, Linköping University
19. Gallager, R.G.: *Discrete Stochastic Process*. Kluwer Academic Publishers Group (1996)
20. Adan, I., Resing, J.: *Queueing Theory*. Eindhoven University of Technology (2002)
21. Bunday, B.D.: *An introduction to queueing theory*. Arnold, London (1996)
22. Gross, D., Harris, C.M.: *Fundamentals of Queueing Theory*. Wiley, Chichester (1985)
23. Robertazzi, T.G.: *Computer Networks and Systems – Queueing Theory and Performance Evaluation*. Springer, New York (1994)
24. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: *Queueing Networks and Markov Chains – Modeling and Performance Evaluation with Computer Science Applications*. John Wiley and Sons, New York (1998)
25. Li, N., Tripunitara, M.V., Bizri, Z.: On Mutually Exclusive Roles and Separation-of-duty. *ACM Transactions on Information and System Security (TISSEC)* 10(2), 5-es (2007)
26. van Hoorn, M.H., Tijms, H.C.: Approximations for the Waiting Time Distribution of the M/G/C queue. *Performance Evaluation* 2(1), 22–28 (1982)
27. Boxma, O.J., Cohen, J.W., Huffels, N.: Approximations of the Mean Waiting Time in an M/G/C Queueing System. *Operations Research* 27, 1115–1127 (1980)

A Medical Image File Accessing System with Virtualization Fault Tolerance on Cloud*

Chao-Tung Yang^{1,**}, Cheng-Ta Kuo¹, Wen-Hung Hsu¹, and Wen-Chung Shih²

¹ Department of Computer Science, Tunghai University, Taichung, 40704, Taiwan ROC
ctyang@thu.edu.tw, {superada0923,aspecteric}@gmail.com

² Department of Applied Informatics and Multimedia, Asia University, Taichung, 41354,
Taiwan ROC
wjshih@asia.edu.tw

Abstract. In recent years, Cloud Computing and Virtualization has become one of the most popular computer science fields. Cloud Computing is enabled by the existing parallel and distributed technology, which provides computing, storage and software services to users. This paper focuses on the cloud storage virtualization technology to achieve high-availability services. We have designed and implemented a medical imaging system with a distributed file system. The technology of Distributed Replicated Block Device has been utilized to mirror an entire disk image. The experimental results show that the high reliability data storage clustering and fault tolerance capabilities can be achieved.

Keywords: Medical Image, PACS, Virtual Machine, DRBD, Fault Tolerance.

1 Introduction

More and more long-term costs control an onsite medical imaging archive. In 2005, the typical U.S. provider estimated that 50 MB of storage were required for an average study. According to the U.S. diagnostic imaging archive storage demand forecasts, it attempting to arrive at an estimate of the total storage volume required to store medical image data. The first assumption that one has to make is to consider only the primary copies of images, that is, to assume that there is no data duplication taking place for the sake of redundancy (e.g. RAID) or disaster recovery planning [1-4].

It is worth noting that while storage cost per Terabyte is declining, the overall cost to manage storage is growing. The common misperception is that storage is inexpensive, but the reality is that storage volumes continue to grow faster than hardware price declines. Using of cloud computing promises to reduce costs, high

* This work was supported in part by the National Science Council, Taiwan ROC, under grant numbers NSC 100-2218-E-029-001, NSC 100-2218-E-029-004, and NSC 100-2622-E-029-008-CC3.

** Corresponding author.

scalability, availability and disaster recoverability, can we solve the long-term face the problem of medical image archive [5-9].

Build cloud computing in this large-scale parallel computing cluster is growing with thousands of processors [3-5, 9-14]. In such a large number of compute nodes, faults are becoming common place. Current virtualization fault tolerance response plan, focusing on recovery failure, usually relies on a checkpoint/restart mechanism. However, in today's systems, node failures can often be predicted by detecting the deterioration of health conditions [15-25].

For over a decade, the majority of all hospital and private radiology practices have transformed from film-based image management systems to a fully digital (filmless and paperless) environment but subtly dissimilar (in concept only) to convert from a paper medical chart to an HER. Film and film libraries have given ways to modern picture archiving and communication systems (PACS). And they offer highly redundant archives that tightly integrate with historical patient metadata derived from the radiology information system. These systems may be not only more efficient than film and paper but also more secure as they incorporate with safeguards to limit access and sophisticate auditing systems to track the scanned data. However, although radiologists are in favor of efficient access to the comprehensive imaging records of our patients within our facilities, we ostensibly have no reliable methods to discover or obtain access to similar records which might be stored elsewhere [6-8].

According to our research, there were few Medical Image implementations on cloud environment. However, a familiar research presented the benefits of Medical Images on cloud were: Scalability, Cost effective and Replication [9]. In the same study, they also presented a HPACS system but lacked of management interface.

We build a HDFS as a platform for Medical Image File Access System (MIFAS), and to do fault-tolerant for HDFS. Instead of a reactive scheme for Virtualization Fault Tolerance (VFT), we are promoting a one where processes automatically migrate from "unhealthy" nodes to healthy ones. Our approach relies on operating system virtualization techniques exemplified by Xen. It leverages virtualization techniques combined with health monitoring and load-based migration. We exploit Xen's live migration mechanism for a guest operating system (OS) to migrate a Namenode from a health-deteriorating node to a healthy one without stopping the Namenode task during most of the migration.

Our VFT daemon orchestrates the tasks of health monitoring, load determination and initiation of guest OS migration. The results showed that the actual cost of relocation hidden cost of live migration has been seamless transfer can be done. Furthermore, migration overhead is shown to be independent of the number of nodes in our experiments indicating the potential for scalability of our approach. Overall, our enhancements make VFT a valuable asset for long running HDFS task, particularly as a complementary scheme to reactive VFT using full checkpoint/restart schemes. In the context of OS virtualization, we believe that this is the first comprehensive study of fault tolerance where live migration is actually triggered by health monitoring.

2 Background

2.1 Medical Technologies

PACS is an acronym that stands for Picture Archiving and Communication System. PACS has revolutionized the field of radiology, which now consists of all digital, computer-generated images as opposed to the analog film of yesteryear. Analog film took up space and time for filing and retrieval and storage, and was prone to being lost or misfiled. PACS saves time and money, and reduces the liability caused by filing errors and lost films. A PACS consists of four major components: imaging modalities such as CT and MRI, a secure network for transmitting patient information, workstations for interpreting and reviewing images, and archives for storing and retrieving images and reports. Combined with available and emerging Web technology, PACS can deliver timely and efficient access to images, interpretations, and related data. PACS breaks down the physical and time barriers associated with traditional film-based image retrieval, distribution, and display. PACS is primarily responsible for the inception of virtual radiology, as images can now be viewed from across town, or even from around the world. PACS also acts as a digital filing system to store patients' images in an organized way that enables records to be retrieved with ease as needed for future reference.

DICOM is short for Digital Imaging and Communications in Medicine, a standard in the field of medical informatics for exchanging digital information between medical imaging equipment and other systems, ensuring interoperability. The standard specifies a set of protocols for devices communicating over a network the syntax and semantics of commands and associated information that can be exchanged using these protocols a set of media storage services and devices claiming conformance to the standard, as well as a file format and a medical directory structure to facilitate access to the images and related information stored on media that share information. The standard was developed jointly by the American College of Radiology (ACR) and National Electrical Manufacturers Association (NEMA) as an extension of an earlier standard for exchanging medical imaging data that did not include provisions for networking or offline media formats [26].

2.2 Virtualization

Virtualization is simply the logical separation of requests for some services from the physical resources where the service is actually provided. In practical terms, virtualization allows applications, operating systems, or system services in a logically distinct system environment to run independently of a specific physical computer system. Obviously, all of these must run on a certain computer system at any given time, but virtualization provides a level of logical abstraction that liberates applications, system services, and even the operating system that supports them from being tied to a specific piece of hardware. Virtualization, focusing on logical operating environments, makes applications, services, and instances of an operating system portable across different physical computer systems. Virtualization can

execute applications under many operating systems, manage IT more efficiently, and allot computing resources with other computers [2].

Virtualization has hardware imitate much hardware through a Virtual Machine Monitor, and each virtual machine functions as a complete individual unit. A virtual machine is composed of memories, CPUs, unique complete hardware equipment, and so on. It can run any operating system as Guest OS without affecting other virtual machines. In general, most virtualization strategies fall into one of two major categories:

Full virtualization also called native virtualization is similar to emulation. As in emulation, unmodified operating systems and applications run within a virtual machine. Full virtualization differs from emulation because operating systems and applications run on the same architecture as the underlying physical machine. This allows a full-virtualization system to run many instructions directly on raw hardware. The hypervisor in this case monitors access to the underlying hardware and gives each guest operating system the illusion of having its own copy.

For Para-virtualization, the hypervisor exports a modified version of the underlying physical hardware. The exported virtual machine has the same architecture, which is not necessarily the case in emulation. Instead, targeted modifications make it simpler and faster to support multiple guest operating systems. For example, the guest operating system might be modified to use a special hyper called application binary interface (ABI) instead of using certain architectural features. This means that only small changes are typically necessary in the guest operating systems, but any changes make it difficult to support closed-source operating systems that are only distributed in binary form, such as Microsoft Windows. As in full virtualization, applications are still in run without modifications.

2.3 Related Work

HDFS servers (i.e., Data nodes) and traditional streaming media servers are both used to support client applications that have access patterns characterized by long sequential reads and writes. As such, both systems are architected to favor high storage bandwidth over low access latency [34].

Recently “Cloud” became a hot word in this field. S. Sagayaraj [9] proposes that Apache Hadoop is a framework for running applications on large clusters built of commodity hardware. The Hadoop framework transparently provides both reliability and data motion. Hadoop implements a computational paradigm named Map/Reduce, where the application is divided into many small fragments of work. Each fragment of work may be executed or re-executed on any node in the cluster. So, by just replacing the PACS Server with Hadoop Framework can lead to good, scalable and cost effective tool for the Imaging solution for Health Care System. In the same study, they also presented a HPACS system but lacked of management interface.

It is complex for kind of performance issues, but J. Shafer, et al. [13] proposed “The Hadoop distributed file system: Balancing portability and performance” had a good view in this field. The poor performance of HDFS can be attributed to challenges in maintaining portability, including disk scheduling under concurrent

workloads, file system allocation, and file system page cache overhead. HDFS performance under concurrent workloads can be significantly improved through the use of application-level I/O scheduling while preserving portability. Further improvements by reducing fragmentation and cache overhead are also possible, at the expense of reducing portability. However, maintaining Hadoop portability whenever possible will simplify development and benefit users by reducing installation complexity, thus encouraging the spread of this parallel computing paradigm.

In our previous reproaches [35-40] use co-allocation to solve the data transfer problem in grid environment. It is the foundation of this paper. But it is for grid not implement on cloud, so in this paper we have a significant change from previous works, because we implement it on the cloud environment and use virtualization fault tolerance techniques.

3 System Design and Implementation

3.1 DRBD

This paper applied DRBD (“Distributed Replicated Block Device”) with Heartbeat to be a good fault-tolerance solution technology [27]. DRBD is a software-based, shared-nothing, replicated storage solution mirroring the content of block devices (hard disks, partitions, logical volumes, etc.) between servers. DRBD technology, a block-device component, forms a high-availability (HA) cluster. This is done by mirroring a whole block device through a specified network. DRBD technology can be understood as a RAID-1 network.

DRBD’s core functionality implements a Linux kernel module. DRBD also constitutes a driver for a virtual-block device, so DRBD is situated “right near the bottom” of a system’s I/O stack. DRBD is extremely flexible and versatile; a replication solution is suitable for adding high availability to any other applications. Heartbeat [28] is a daemon that provides cluster infrastructure (communication and membership) services to its clients. This allows clients to know of the presence (or disappearance!) of peer processes on other machines and to easily exchange messages with them [30].

3.2 Virtualization Fault Tolerance Design

Virtualization technology [1, 5, 18, 29-32] implements a cluster-based server to overcome these problems. Cluster nodes can develop through some virtualization platforms (Xen, KVM, VMWare, etc.) with an efficient virtual-machine manager. It incorporates a provisioning model to dynamically deploy new virtual cluster nodes when user demand increases, and consolidates virtual nodes when demand decreases. This approach, called Virtualization Fault Tolerance (VFT), is designed for management virtual machines with an efficient mechanism to reach high availability under limited resources. Apart from this, this paper investigates fault tolerance on virtualization machines and raising reliability. To provide continuous availability for applications in the event of server failures, a detection methodology is necessary.

Virtualization Fault Tolerance (VFT) has three main phases: virtual-machine migration policy, information gathering, and maintaining constant service availability (Fig. 1). However, the physical host number must be bigger than three to achieve VFT methodology. Information Gathering presents a detection mechanism to retrieve all Hosts and checks whether Hosts are alive or not, using a “ping” command every five minutes by running Linux schedule through “crontab”.

Maintaining constant service availability: assuming VM *m* is under a Heartbeat plus with DRBD mechanism, and then Host *n* becomes an unavailable physical machine. Once the Host *n* is shut down, if VM *m* is secondary node, then it moves to an on-line Host and boot automatically. If VM *m* is a primary node then the secondary node replaces the VM *m* to primary node immediately. Next pre-primary node boots on available host(s) and become secondary. In OpenNebula, command `onevm` is to submit, control and monitor virtual machines (Fig. 2). This helps control dead VM to deploy on other available physical hosts.

This flow is one of the scheduled programs and deployed on the front end. It is reasonable to enhance this function on OpenNebula’s front end, because it controls all VM operations. One example can explain a single-failure event triggered in a VFT approach (Fig. 3). First, Host A is shut down by unexpected matters; a few minutes later, the front end detected it and also triggered VFT. Next, the secondary node VM 2 becomes primary and hands over all services from pre-primary (FAIL-OVER). Finally, VM 1 boots on Host C automatically and becomes the secondary node (FAIL-BACK).

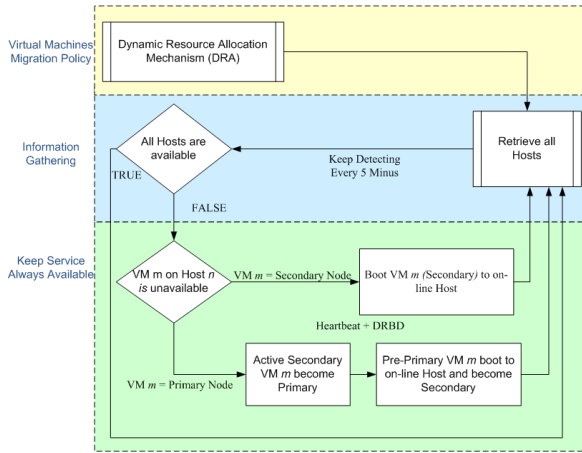


Fig. 1. Virtualization Fault Tolerance Flow

3.3 System Architecture

MIFAS was developed on cloud environment, Detailed System Components, such as Figure 4. The distribution file system was built on HDFS of Hadoop environment. This Hadoop platform could be described as PaaS (Platform as a Service). We

extended a SaaS (Software as Service) based on PaaS. As the shown illustration, the top level of MIFAS was web-based interface. And now we do more things between Pass and IaaS, in order to achieve the Hadoop HA.

IaaS: in our previous work, we used OpenNebula to manage our VMs [22]. We can migrate the physical servers into OpenNebula environment. And it is also a key feature to develop the VFT approach on virtualization.

PaaS: HDFS was a well-known could platform in this field, so we will save the introduction of HDFS in this section. But, as we mentioned the VFT approach in section 3.1, the Hadoop Namenode was also under HA control by our VFT approach.

SaaS: in this layer we called Middleware. It is the core of MIFAS. From the top level a web-based system was provided a GUI interface that users or administrators could manage patient’s data on it also including the quick view of medical images. NanoDICOM [41] was a component that made by PHP, it could convert DICOM file into JPEG without complex process.

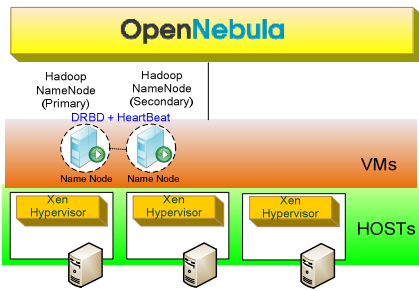


Fig. 2. Deploy Namenode HA on Virtualization

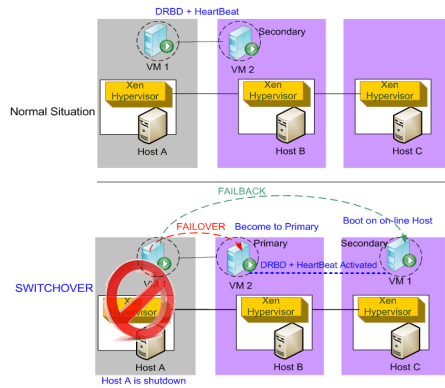


Fig. 3. How to Trigger VFT

We applied it while uploading DICOM files, therefore the system would convert it into JPEG format automatically. In another hand, consider to the gap between Hadoop and general platform, we provided a good solution for it which is Hadoop-over-ftp [42] and CurlFTP [43]. Both of these components are open source based software. The Hadoop-over-ftp could convert HDFS as FTP service, and then CurlFTP could mount the FTP service as a local level storage. Thus, we could provide kind of solution for the enterprise or those people whom are not familiar with Hadoop as a remote storage. In generally, we split entire system into two different levels, one is the DICOM viewer and patient management, and the other is storage level.

This Middleware also collected necessary information such as bandwidth between server and server, the server utilization rate, and network efficiency. The information provided entirety MIFAS Co-allocation Distribution Files System to determine the best solution of downloading allocation jobs.

Information Service: To obtained analysis in the status of host. The Middleware of MIFAS had a mechanism to fetch the information of hosts called Information Service.

In this research, we installed the Ganglia [44] in each member of Hadoop node to get the real-time state from all members. Therefore, we could get the best strategy of transmission data from Information Service which is one of the components of MIFAS Middleware.

Co-allocation: Co-allocation mechanism could conquest the parallel downloading from Datanodes. Besides, it also sped up downloading and solved network faults problems. Due to user using MIFAS to access Medical Images, the co-allocation will be enabled automatically. In order to reached parallel downloading approaches, the system will split those file in to different parts and obtain data from different Cloud depend on Cloud health status. Therefore, we can get the best downloading strategy. In our earlier research [35-37] was also provided our co-allocation mechanism.

Replication Location Service: In this research, we built three groups of HDFS in different locations, and each HDFS owned an amount of Datanodes. The Replication Location Service means that the Service would automatically make duplication from private cloud to one another when medical images uploaded to MIFAS.

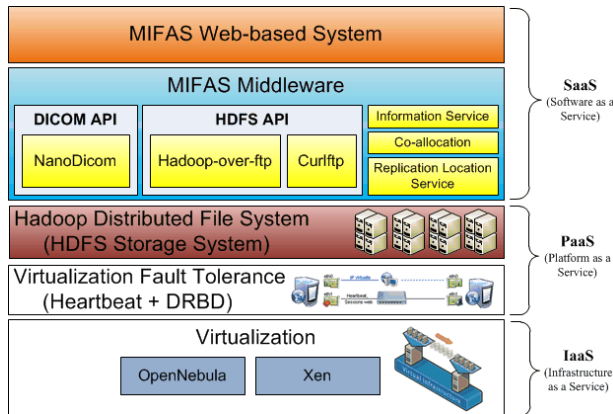


Fig. 4. System Components of MIFAS on cloud

4 Experimental and Results

In our MIFAS environment, there are three HDFS nodes (THU1, THU2 and CSMU). For each Namenode are done in two VMs configuration, and use the DRBD with heartbeat sync to do this part of the configuration, were configured for each Namenode four Datanodes. Figure 5 show the details and the environment.

At this part we do stress testing with JMeter. We set 10 Threads, and Loop count 5 times more physical machines and virtual machine on the environment were to download 1MB, 10MB and 50MB file sizes, etc., the resulting throughput and the ability to download data. The result of Figure 6 shows that the smaller of file size will enable greater throughput, and physical machines and VMs will be more obvious differences. Figure 7 shows we download a small file, VMs transmission performance will be better than physical machines.

In this experiment, we download the same files from each PACS and MIFAS. The purpose of this experiment is to compare of PACS and MIFAS Networking Performance. Figure 8 shows the results, a smaller download file, MIFAS better transmission capacity, whereas in downloading large files, PACS has better transmission performance.

In this system, we used three HDFS nodes to access data. When a HDFS node which network disconnection, through the co-allocation mechanism, Information service will note that the current network node there is a problem. When users access to data, system will make the current surviving HDFS nodes to do distribution of the current file transfer request to the user. In Figure 9, we interrupt the THU2 HDFS and CSMU HDSF network, and then the system will transfer data THU1 HDFS as the main node.

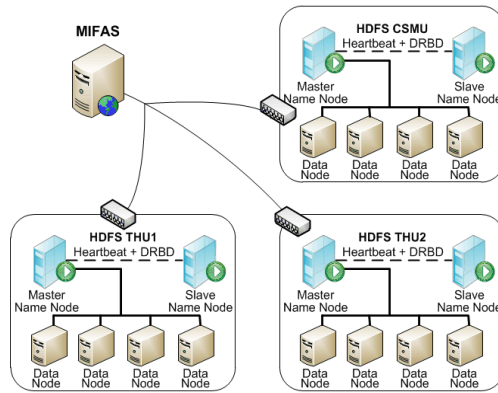


Fig. 5. The Experimental environment

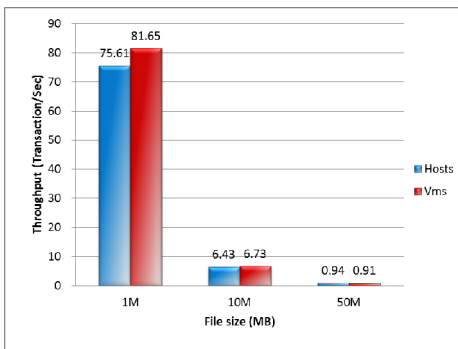


Fig. 6. Compare of Physical Host and Virtual Machine Throughput

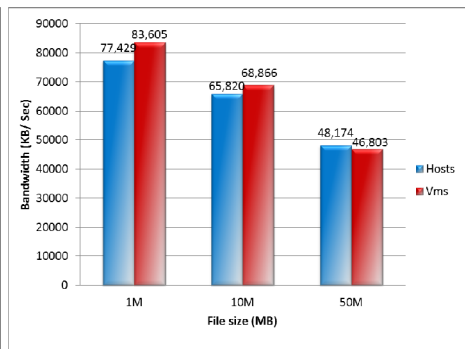


Fig. 7. Compare of Physical Host and VM Networking Performance

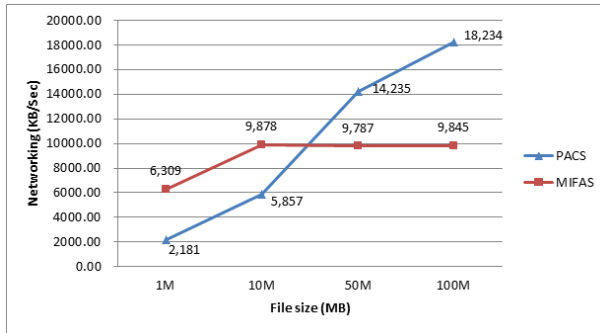


Fig. 8. Compare of PACS and MIFAS Networking Performance

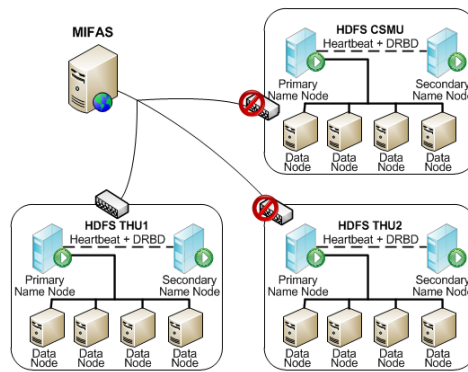


Fig. 9. Network Fault Tolerance

5 Conclusions

At present the computer node failure can usually be detected through the monitoring mechanism for system health. Compared to passive solutions, the recovery has occurred in response to failure, we are actively promoting the virtualization fault tolerance (VFT). Systems that exhibit truly continuous availability are comparatively rare and higher priced, and most have carefully implemented specialty designs that eliminate any single point of failure and allow online hardware, network, operating system, middleware, and application upgrades, patches, and replacements. Zero downtime system design means that modeling and simulation indicates mean time between failures significantly exceeds the period of time between planned maintenance, upgrade events, or system lifetime. We use VFT mechanism to build a high availability of HDFS. Combining virtualization techniques, load balancing, health monitoring and live migration, to be able to run virtual machines from hardware failure on one machine and restart on another machine without losing any state.

References

1. Prepare for Disasters & Tackle Terabytes When Evaluating Medical Image Archiving, <http://www.frost.com/>
2. Teng, C.C., Mitchell, J., Walker, C., Swan, A., Davila, C., Howard, D., Needham, T.: A medical image archive solution in the cloud. In: IEEE International Conference on Software Engineering and Service Sciences, ICSESS 2010, pp. 431–434 (2010)
3. Silva, L.A.B., Costa, C., Oliveira, J.L.: A PACS archive architecture supported on cloud services. *International Journal of Computer Assisted Radiology and Surgery*, 1–10 (2011)
4. Macedo, D.D.J.D., Wangenheim, A.V., Dantas, M.A.R., Perantunes, H.G.W.: An architecture for DICOM medical images storage and retrieval adopting distributed file systems. *Int. J. High Perform. Syst. Archit.* 2, 99–106 (2009)
5. Yang, C.T., Chen, L.T., Chou, W.L., Wang, K.C.: Implementation of a Medical Image File Accessing System on Cloud Computing. In: 2010 IEEE 13th International Conference on Computational Science and Engineering, CSE, pp. 321–326 (2010)
6. Faggioni, L., Neri, E., Castellana, C., Caramella, D., Bartolozzi, C.: The future of PACS in healthcare enterprises. *European Journal of Radiology* (2010)
7. Bellon, E., Feron, M., Deprez, T., Reynders, R., de Bosch, B.V.: Trends in PACS architecture. *European Journal of Radiology* 78, 199–204 (2010)
8. Sutton, L.N.: PACS and diagnostic imaging service delivery—A UK perspective. *European Journal of Radiology* 78, 243–249 (2011)
9. Ganapathy, G., Sagayaraj, S.: Circumventing Picture Archiving and Communication Systems Server with Hadoop Framework in Health Care Services. *Journal of Social Sciences* 6, 310–314 (2010)
10. National Institute of Standards and Technology (NIST), <http://www.nist.gov/>
11. Venner, J.: *Pro Hadoop*, 1st edn (2009)
12. Grossman, R.L., Gu, Y., Sabala, M., Zhang, W.: Compute and storage clouds using wide area high performance networks. *Future Generation Computer Systems* 25, 179–183 (2009)
13. Shafer, J., Rixner, S., Cox, A.L.: The Hadoop distributed filesystem: Balancing portability and performance. In: 2010 IEEE International Symposium on Performance Analysis of Systems & Software, ISPASS, pp. 122–133 (2010)
14. Liu, J., Bing, L., Meina, S.: The optimization of HDFS based on small files. In: 2010 3rd IEEE International Conference on Broadband Network and Multimedia Technology, IC-BNMT, pp. 912–915 (2010)
15. van Hagen, W.: *Professional Xen Virtualization: Wrox*, 1st edn. (January 29, 2008)
16. Walters, J.P., Chaudhary, V., Cha, M., Guercio Jr., S., Gallo, S.: A Comparison of Virtualization Technologies for HPC. In: 22nd International Conference on Advanced Information Networking and Applications, AINA 2008, pp. 861–868 (2008)
17. Zhu, J., Jiang, Z., Xiao, Z., Li, X.: Optimizing the Performance of Virtual Machine Synchronization for Fault Tolerance. *IEEE Transactions on Computers*, 1–1 (2010)
18. Bressoud, T.C., Schneider, F.B.: Hypervisor-Based Fault-Tolerance. *SIGOPS Oper. Syst. Rev.* 29, 1–11 (1995)
19. Walters, J., Chaudhary, V.: A fault-tolerant strategy for virtualized HPC clusters. *The Journal of Super Computing* 50, 209–239 (2009)
20. Clinical Data Update System (CDUS), <http://www.cdus.org>
21. Rafael, M.V., Montero, R.S., Llorente, I.M.: Elastic management of cluster-based services in the cloud. Presetend at the Proceedings of the 1st Workshop on Automated Control for Data Centers and Clouds, Barcelona, Spain (2009)
22. Yang, C.T., Cheng, H.Y., Chou, W.L., Kuo, C.T.: A Dynamic Resource Allocation Model for Virtual Machine Management on Cloud. In: *Symposium on Cloud and Service Computing* (2011)

23. Varghese, B., McKee, G., Alexandrov, V.: Implementing intelligent cores using processor virtualization for fault tolerance. *Procedia Computer Science* 1, 2197–2205 (2010)
24. Villa, O., Krishnamoorthy, S., Nieplocha, J., Brown, D.M.J.: Scalable transparent checkpoint-restart of global address space applications on virtual machines over infiniband. Presetend at the Proceedings of the 6th ACM Conference on Computing Frontiers, Ischia, Italy (2009)
25. Simons, J.E., Buell, J.: Virtualizing high performance computing. *SIGOPS Oper. Syst. Rev.* 44, 136–145 (2010)
26. DICOM, <http://medical.nema.org/>
27. Yang, C.-T., Tseng, C.-H., Chou, K.-Y., Tsaur, S.-C., Hsu, C.-H., Chen, S.-C.: A Xen-Based Paravirtualization System toward Efficient High Performance Computing Environments. In: Hsu, C.-H., Malyshev, V. (eds.) *MTPP 2010*. LNCS, vol. 6083, pp. 126–135. Springer, Heidelberg (2010)
28. VMware, <http://www.vmware.com/>
29. Nagarajan, A.B., Mueller, F., Engelmann, C., Scott, S.L.: Proactive fault tolerance for HPC with Xen virtualization. Presetend at the Proceedings of the 21st Annual International Conference on Super Computing, Seattle, Washington (2007)
30. Open Nebula, <http://www.opennebula.org>
31. Apache Hadoop Project, <http://hadoop.apache.org/hdfs/>
32. DRBD Official Site, <http://www.drbd.org>
33. Pla, P.: Drbd in a heartbeat. *Linux J.* 2006, 3 (2006)
34. Reddy, A.L.N., Wyllie, J.: Disk scheduling in a multimedia I/O system. Presetend at the Proceedings of the First ACM International Conference on Multimedia, Anaheim, California, United States (1993)
35. Yang, C.T., Wang, S.Y., Lin, C.H., Lee, M.H., Wu, T.Y.: Cyber Transformer: A Toolkit for Files Transfer with Replica Management in Data Grid Environments. Presetend at the Proceedings of the Second Workshop on Grid Technologies and Applications, WoGTA (2005)
36. Yang, C.T., Wang, S.Y., Fu, C.P.: A Dynamic Adjustment Strategy for File Transformation in Data Grids. In: Li, K., Jesshope, C., Jin, H., Gaudiot, J.-L. (eds.) *NPC 2007*. LNCS, vol. 4672, pp. 61–70. Springer, Heidelberg (2007)
37. Yang, C.T., Chi, Y.C., Han, T.F., Hsu, C.H.: Redundant Parallel File Transfer with Anticipative Recursively-Adjusting Scheme in Data Grids. In: Jin, H., Rana, O.F., Pan, Y., Prasanna, V.K. (eds.) *ICA3PP 2007*. LNCS, vol. 4494, pp. 242–253. Springer, Heidelberg (2007)
38. Yang, C.T., Yang, I.H., Wang, S.Y., Hsu, C.H., Li, K.C.: A Recursively-Adjusting Co-allocation scheme with a Cyber-Transformer in Data Grids. *Future Generation Computer Systems* 25, 695–703 (2009)
39. Yang, C.T., Yang, I.H., Li, K.C., Wang, S.Y.: Improvements on dynamic adjustment mechanism in co-allocation data grid environments. *J. Supercomput.* 40, 269–280 (2007)
40. Yang, C.T., Wang, S.Y., Chu, W.: Implementation of a dynamic adjustment strategy for parallel file transfer in co-allocation data grids. *The Journal of Supercomputing* 54, 180–205 (2009)
41. Nanodicom, <http://www.nanodicom.org/>
42. Hadoop-over-ftp, <http://www.hadoop.iponweb.net/Home/hdfs-over-ftp>
43. Curlftp, <http://curlftpfs.sourceforge.net/>
44. Ganglia, <http://ganglia.sourceforge.net/>

Enhanced Password-Based User Authentication Using Smart Phone*

Inkyung Jeun, Mijin Kim, and Dongho Won**

Sungkyunkwan University,
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do, 440-746 Korea
{ikjeun,mjkim,dhwon}@security.re.kr

Abstract. Today, an internet environment has become a single society in which all of the services required for daily living are implemented online. To reliably use such an internet environment for our daily living, safe user identification and authentication are required. Of course, there are many ways to perform authentication online. However, thus far, in many e-services include cloud services, passwords are mainly used for user authentications. Although there are many safer authentication methods than a password, which has the risk of personal information leakage and is a relatively poor authentication method, passwords are frequently used due to their high user convenience and ease of implementation. In this article, to resolve the weaknesses of the current password environment, we suggest a new user authentication method that can offer both safety and convenience combined with the recent mobile internet environment. For this purpose, in this article, a smart phone is used as the storage space for the user password. A user can conveniently use passwords in the wireless e-service as well as the wired e-service.

1 Introduction

Smart phones represented by Steve Jobs in Apple inc. have recently been gaining a global attention. They are intelligent terminals combining the functions of mobile phones with features such as internet communication and information search. The representative aspect that distinguishes smart phones from other mobile phones is that users can install applications ("apps" hereinafter) in it[1]. The smart phones aren't just means of communication, but they provide us functions like a PC. As a result, the mobile cloud environment is established in our world. We can see and modify our document on the PC as well as our smart phone using the mobile cloud services. So, the internet is changing from a simple means of information exchange to a cyber society. We call this e-society. The convenience that we can connect to the internet anytime and anywhere also increased the use of Social network service(SNS) like Facebook or Twitter, so

* This research was supported by the KCC(Korea Communications Commission), Korea, under the RnD program supervised by the KCA(Korea Communications Agency)"(KCA-2012-12-912-06-003).

** Corresponding author.

the world has established an e-society. E-society in a cyber space has some good points, as it facilitates communication among users and can serve as a vital source of information in the event of a disaster or emergency, as was recently seen during the Japanese earthquakes. Nevertheless, the more the e-society is becoming a giant and the e-services which are members of e-services have been increased, the more threats are also increasing. Among them, one of the leading threats is the problem of a personal information exposure. When the use of e-services increases, the threats of a personal information exposure also increase. There have been many cases reported related to the distribution of false information using stolen ID, or causing damages to others by misrepresenting celebrities.

SNS Profiles have also become a channel for ID theft by hackers. The user information contained in an SNS Profile (e.g. address, place of birth, school, phone number, date of birth, color of vehicle and name of pet) can help hackers find questions and answers for ID tracking. Hackers can use the information contained in an SNS Profile to answer the ID search question, and abuse the account by identifying ID [2]. This is possible because users use multiple SNSs with a single ID, allowing the ID linkability of multiple SNSs.

In order to avoid these risks, each e-services are using variety user authentication method like a password as well as a digital certificate, one time password (OTP), bio information. Nevertheless, the most e-services still use a password as a user authentication method for reasons of user-friendliness and ease of implementation. But, the problems of password are steadily being pointed. The password can be guessed, forgotten, written down and stolen, eavesdropped or deliberately being told to other people. Most users tend to use a simple password even though they know the importance of his/her password for the inconvenience of memory. Also even if they use a secure password, the password can be exposed to others by key logging program of malicious hackers. In addition, despite the increase of smart phone users, the use of wired and wireless environments for e-services has not been lined. That is, even if the same e-service, user authentications are performed separately in wired and wireless environment, and especially smart phone users tend to use weak password for user authentication, because the input is inconvenient compared to the input of keyboard in PC. And sometimes they use automatic login function in smart phone, which can increase the risk of password exposure.

Against this backdrop, this paper proposed a way to safely and conveniently enjoy e-services by using smart phones. The content of this paper are follows. Chapter 2 featured the problem of password for an user authentication and chapter 3 proposed the enhanced password-based user authentication protocol using smart phone. Chapter 4 analyzed our proposed model and chapter 6 provided conclusions.

2 Problems of Password for an User Authentication

Our world is being tied into the huge e-society as the off-line based services such as an e-health service, e-business and e-banking are available through on-line internet. Recently, as many people use the excellent wireless devices such as smart

phones, e-society that we can access to e-services at anytime and anywhere is being built. To use this e-service safety, the trusty worth user authentication method is our priorities. In an e-society environment, the proper authentication method should be provided considering the risk level of each services. The authentication methods are divided into various types such as user holding, user itself and user knows.

The most widely used means of authentication among various authentication methods is clearly a password. This is easier to implement, as well as easy to use, so it is used in the most e-services. However, the vulnerability of passwords has been raised in the past until now. Passwords can be exposed to hackers by key logging programs, as well as hackers can find out the passwords if we use a short and simple password. Thus, the e-services take place some financial transactions such as e-banking, e-payment use multi-factor authentication. For that, the password is used in conjunction with digital certificate, OPT, etc. methods for user authentication

The specific issues if we use a password as an authentication method are as follows.

(1) Difficulty of memory

The more we use e-services, the more the passwords we should remember increase. We should use different passwords in all services due to the possibility of the risk of exposing the password, but it's impossible to set up the different passwords according to the e-services. Secure passwords have a long length including letters, number, as well as special characters, but this also limits to remember. As a result, it's very difficult to configure easy-to-remember passwords for users. So most users use the weak and same password in all e-services, and besides they write their password on the post-it or the notebook. A recent study by the security firm Trustwave indicated that the most common password is Password1, a variation on the historically common default password, Password3. As we can see in this study, most users tend to use easy password.

(2) Key logging

Recently, the key logging tools that installed on a PC and can expose an input password via keyboard are exploited. The services that require high security technology such as e-banking service installed and used some anti-hacking software to prevent key logging attack, but it is impossible that all e-services use that software. So, we can say that the risks of exposing a password exist anywhere.

(3) Phishing Attack

Hackers can seizure the input password as they built a fake web-site instead of actual web-site. Phishing attack is to be used in order to seize your financial information to earn the money, but simply it can be used means to steal user's password and some personal information.

(4) Non-Repudiation

Password user can deny their act on the internet. In other words, if a user access the SNS and write some wrong comments, the user can repudiate his acts and

say that it is performed by others due to the password theft. In this case, the service provider is hard to prove it.

Despite these many problems, the password is used widely and easy to use, so it is very necessary to strengthen its safety. Accordingly, this paper proposed an enhanced user authentication scheme using smart phone considering recent e-society environment. To this, we proposed password-based authentication model, one of the most commonly used means for user authentication and suggested using of smart phone as a storage of password. So we hope it can be classified as a multi-factor authentication, and it can increase security level of password.

3 Enhanced Password-Based User Authentication Scheme Using SmartPhone

In this section, we proposed the enhanced authentication method that will minimize the changes of the password-based authentication mechanism which is the most used in the current e-services. Our proposed method can solve the problems of password such as the difficulties of memory, the risk of personal information exposure due to use of weak password, etc. E-services that need a high level authentication method such as e-banking service use an additional authentication information for user authentication like PKI certificate or bio information.[\[4\]](#) But, these methods are impractical in general e-society services like SNS because they require excessive costs for an additional implementation. Another high level authentication method is OTP. Google web-site uses OTP to who wants to use stronger user authentication.[\[5\]](#) It is two-factor and strong authentication method based on one time password. Instead of authenticating with a simple and weak password, each user carries a OPT generator ("token") to generate passwords that are valid only one time. To generate one time password, the user has to enter his personal PIN into the OPT token. And the one-time password which is generated in the token is used for user authentication on the e-services. So OPT authentication is two factor authentication method using the OTP token and a PIN ("something you have and something you know"). This is obviously more secure than just simple password, because an attacker should know the PIN as well as the token device. This can be quite expensive and it is very inconvenient to the users. To solve this problem, the mobile-OTP was developed using a smart phone instead of a token device.[\[7\]](#) But, Mobile-OTP also needs an additional implementation cost to recognize and process the OTP.

In this paper, we proposed the password-based user authentication mechanism similar with current e-services. The e-service providers don't need any additional implementation, and they can just process the password. And the user can use their own password without difficulty of memory using smart phone and mobile application. Also, the security level of the password would be increased.

3.1 Terms and Abbreviations

- ESP(E-Service Provider) : It is e-service which is configure the e-society like as e-banking, e-payment, cloud service provider, etc.

- Password token : The data element containing the personal information of a user, including password, ID, etc. It will be saved in smart phone safety, and sent to e-service provider for an user authentication if necessary.
- SmartID_App : It is application installed in the smart phone of users, and it saves, manages, and transfers the password token.
- ID(Identity) : It is identifier for user identification in ESP, and it is generated by users.
- salt : It consists of random bits, creating one of the inputs to a one-way hash function with password.
- SS(Shared Secret) : It is shared secret between ESP and user's smart phone, and used as encryption key when they transfer confidence information between them.

3.2 Symbols and Notions

- $T(s, r, a)$: For $s \in \{ESP, USER, SmartID_App\}$, $r \in \{ESP, USER\}$ and a , it means transfer a from s to r .
- $S(i, a)$: For $i \in \{ESP, USER\}$ and a , it means to save a to i .
- $E(a)$: For $a \in \{SmartID_App\}$, it means to execute a by user on the smart phone.
- $ENC_{KEY}(p)$: It means to encrypt p using encryption key KEY .
- $DEC_{KEY}(p)$: It mean to decrypt p using encryption key KEY .
- $PKEY(PublicKey)$: It means a public key for encryption in the public encryption algorithm, $PKEY_{ESP}$ is a public key of ESP .
- $SKEY(SecretKey)$: It means a secret key for decryption in the public encryption algorithm, $SKEY_{ESP}$ is a secret key of ESP .
- PW_{ESP} : It is a password for user authentication in ESP , input strings by user.
- PW_{APP} : It is a password for user authentication when the app starts, input strings by user's smart phone.
- HPW_{ESP} : It is a hashed password value saved in ESP .
- $KDF_i(PW)$: It means a Password-based key derivation function. It generates SS based on the input PW . On here, i means a key length.
- $a \parallel b$: It is a concatenation of a and b .
- TK_{ESP} : It means a password token include PW_{ESP} , ID used in ESP .
- $TKG(PW_{ESP}, ID)$: It is a function to generation TK_{ESP} using PW_{ESP} , ID .
- $P(a, b)$: For a, b , If $a = b$, then it proceed to the next stage of the protocol, but if $a \neq b$, then the protocol is stopped.

3.3 Assumptions

Before describe the details, the assumptions of this paper are as follows.

1. Users should pre-install SmartID_App on their smart phone to use the password token in the e-services. This app has a roll like a password wallet, so

the SmartID_App should be developed and installed via safety method. [8] Also, before the SmartID_App is run in a user's smart phone, the user of the app should be authenticated using a password. This password is set when the SmartID_App is installed first on the smart phone

2. SmarID_App should store ESP lists which can use the SmartID_App for user authentication and the public keys of each ESPs. Users can select the ESP name that the user wants to access now on SmartID_App, and if there are changes in the status of ESP lists supporting SmartID_App, the status should be updated through an application upgrade method on the smart phone. The public key of ESP is used to transfer the secret key which is needed between SmarID_App and ESP server.

3.4 Service Protocol

This mechanism uses the current authentication method used in e-service but the users use their smart phone instead of key board to input the password. That is, users can use the e-services same as the past, but they just select and transfer the password token in smart phone instead of entering the password via the keyboard in PC. From now on, we call this authentication method as SmarID_App-based method. The process of SmartID_App-based method consists of three major steps for user authentication. First process is an user registration to ESP and the password stored ESP as well as smart phone. And the second process is an user authentication process using password token which is store in smart phone. Last process is an user management process which is used when the user lost or change his smart phone.

Now, the detailed protocols of our SmartID_App -based authentication method are as follows.

Step1. User Registration Protocol

The process for user registration to ESP is similar with a general ESP registration process, but the password that used for an user authentication in ESP is stored in the smart phone as well as ESP server as a password token type. The password that stored in the user's smart phone is used when the user try to access an e-service instead of entering the password on PC. The password token selected by user in his smart phone would be transferred from the smart phone to the ESP for user authentication. The detailed protocol is as follows.

1. User applies for membership in ESP that the user intends to use. The user should enter his ID, password and his smart phone number to ESP after ESP performed an identity proofing of the user. The detailed identity proofing method is not mentioned here.
 - (1) $T(USER, ESP, (ID, PW, PhoneNum))$
2. ESP server sends an authentication code which is randomly generated to the smart phone for the smart phone authentication. After the user confirms this

authentication code, he inputs this authentication code to ESP, and then the ESP can confirm the reality of smart phone number. User runs SmartID_App in his smart phone, and input the password for app operation. This password is set when the SmartID_App is installed in the smart phone by a user.

(2) $(ESP, SmartID_App, authenticationcode)$

(3) $T(USER, ESP, authenticationcode)$

(4) $E(SmartID_App)$

(5) $T(USER, SmartID_App, PW_{App})$

3. The User selects the ESP name in the SmartID_App, and input the ID used on his smart phone. At this time, SmartID_App generates the shared secret key(SS) which is needed for secret communication between ESP and SmartID_App. SS is made with PW_{app} by key derivation function and its length is 128 bits. SS is sent from SmartID_App to ESP after encrypt using the public key of $ESP(PKEY_{ESP})$ which is stored in SmartID_App with ESP name.

(6) $SS = KDF_{128}(PW_{App})$

(7) $T(USER, SmartID_App, (Name\ of\ ESP, ID))$

(8) $T(SmartID_App, ESP, (ID, ECN_{PKEY_{ESP}}(SS)))$

4. ESP decrypts the encrypted secret key which is sent from SmartID_App using his private key($SKKEY_{ESP}$). And he sent the hashed password which is matching with ID to SmartID_App after encrypt the hashed password using SS .

(9) $SS = DECS_{KEY_{ESP}}(ECN_{PKEY_{ESP}}(SS))$

(10) $T(ESP, SmartID_APP, ENC_{SS}(Hash(PW_{ESP} \parallel salt)))$

5. SmartID_App will store the encryption value of the hashed password with ID and this is called password token. SmartID_App will store the password token with an icon to distinguish the token from others.

(11) $TK_{ESP} = TKG(ENC_{SS}(Hash(PW_{ESP} \parallel salt)), ID)$

(12) $S(SmartID_App, (TK_{ESP}))$

6. ESP should save the user's registration information such as ID, the smart phone number and hashed password. PW is stored in ESP server after hashing using one-way hash function. For a password hash, the salt is used with the password. This makes the hash value of password to strong from some dictionary attack to stole the password.

(13) $(ID, PhoneNum, Hash(PW_{ESP} \parallel salt))$

Step2. User Authentication Protocol

To use ESP in the PC e-service environment, the user just inputs the ID without the password, and the password is sent from the user's smart phone after the user select the ESP icon in the SmartID_App. The detailed protocol is as follows.

1. The user inputs the ID on the ESP via PC. Then ESP sends SMS to the smart phone to request the password which is used to access the ESP site for user authentication.
 - (1) $T(USER, ESP, (ID))$
 - (2) $T(ESP, SmartID_App, (PasswordRequestSMS))$
2. The user runs the SmartID_App after reading the SMS from ESP in his smart phone, and he inputs the password(PW_{APP}) to run the SmartID_App. The password token(TK_{ESP}) list is showed as icon type, the user selects the ESP name.
 - (3) $E(StartID/_{App})$
 - (4) $T(USER, SmartID/_{App}, PW_{App})$
3. SmartID_App generates the shared secret key(SS) based on the user's input password(PW_{APP}). This secret key is used to decrypt the encrypted hashed password and the length of it should be longer than 128 bits. In here, we assume the length of the secret key is 128 bits. The user can select the password token SmartID_App to send a password to ESP. The hashed password which is included in the password token is sent to ESP after encrypted using SS and the SS is sent after encrypted by a ESP' public key.
 - (5) $SS = KDF_{128}(PW_{APP})$
 - (6) $SELECT(TK_{ESP})$
 - (7) $T(SmartID/_{App}, ESP, (ENC_{PK_{KEY_{ESP}}}(SS), TK_{ESP}))$
4. ESP decrypts the SS using his private key and encrypts the hashed password using SS . And then ESP confirms whether the decrypted hashed password is same with the stored hashed password in ESP. If the two values are same, the user's login to ESP is allowed.
 - (8) $SS = DEC_{SK_{KEY_{ESP}}}(ECN_{PK_{KEY_{ESP}}}(SS))$
 - (9) $P(Hash(PW_{ESP} || salt), DEC_{SS}(ENC_{SS}(Hash(PW_{ESP} || salt))))$

As we can see this protocol, ESP just uses the password same with the past e-services. But the password is sent from the user's smart phone instead of keyboard input. In other words, ESP improves the password-based user authentication method with a minimal cost using a smart phone, which is widely used recently. Also the user don't need to remember his password to access ESP, and he can use the ESP just run SmartID_App and click the password token in SmartID_App. The password stored in the smart phone is hashed and encrypted, so the risk of exposure can be minimized. The password which is used to encrypt the hashed password is not save anywhere, and just entered by a user when he run a SmartID_App. So the possibility of exposure is close to zero.

Step3. Management Protocol of Smart Phone Number

If the user wants to change the smart phone number, he should login to the ESP specified in Step.2 and then he can changed the smart phone number. At that

time, the protocol of Step.1 is run again, and the new TK_{ESP} is generated and saved in the new smart phone.

But, if the user lost or stole the smart phone, the user must lock the login function by smart phone in ESP. For this, ESP should build the emergency login function in it. For this function, ESP can use to confirm the smart phone number after the user input the ID and password in ESP.

4 Analysis

In this section, we look our proposed SmartID_App-based user authentication scheme which is proposed to solve the problem of password-based authentication problems and further it will increase the level of trust of the password-based user authentication method.

1. SmartID_App-based-authentication method reduces the user's inconvenience, possibility of key logging, and phishing attack and provides user's non-repudiation.
 - Users Inconvenience : To use the normal password, the user should remember and input the password whenever the user access to the ESP, but in this method, there is no need to input of password. The user authentication is completed when the user select the password token which is saved in smart phone and generated during registration process of users. Only thing that the user should memory is just the password of SmartID_App. In other words, the user can access all e-services after login SmartID_App.
 - Key Logging : The password can be exposed by a malicious key logging tools during the user input the password in his PC even though the password is very safety consist of long and complex characters. This problem occurs when the user enters his password via a keyboard. But in our proposed method, there is no process that the user inputs his password, so it is safe from the key logging attacks.
 - Phishing Attack : Phishing attack occurs when the hackers built a fake site to know the user's password and he user input his password on that fake site. At this case, the hacker can hijack the password, and it can be used on the real site illegally. But on our proposed method, the fake site can't send SMS to the user's smart phone because it doesn't know the user's smart phone number. And the user doesn't need to input his password on the site. So the phishing attack to seize a password is impossible.
 - Non-repudiation : To use our proposed method, the user should have a smart phone before access to e-services. In other words, the user who access to the e-service must have had his smart phone, so he can't repudiate his action except the smart phone is stolen or lost.

2. SmartID_App based-authentication method improves the security level of password-based user authentication method.

Our proposed method uses the password token instead of a normal password from the smart phone to ESP, so it provides two-factor authentication function. In other words, if the user doesn't have a smart phone, he cant use ESP, because the password is stored in the smart phone as a form of token. Thus, it's security level is enhanced to the multi-factor authentication capabilities such as OPT.

3. SmartID_App based-authentication method is interoperable with the wireless and wired e-service environment.

E-services such as SNS, etc. are operated on the wireless network using smart phone as well as wired network using PC. So, many ESPs are providing the wireless services in the form of app and mobile web browsing services. In the case of wireless services using smart phone, the using of ID and password is very inconvenient due to the constrained mobile phone environment. So, some apps use the auto login function where their ID and PW are store in smart phone and then used in log-in, but if the smart phone is been rooting or hacking that information could be exposed to the other person. But, in the our proposed method, the ID and password are stored in the smart phone as a form of token, so the password token is used on the wireless servive as itself and it can avoid the risk of lost.

As a results, SmartID_App based-authentication method can be used for an user authentication in the wireless e-services as well as wired e-services.

5 Conclusion

This paper proposed SmartID_App based password as a model to improve the safety of e-Society. Many services is opeated in wireless as well as wired environment. For user authentication using mobile phone like as smart phone, users tend to use easy password due to dufficulty of memory. So the security level of password can be low. But our proposed method store the password to the smart phone. So the password can be used on the PC and mobile internet using smart phone. As a result, it is expected that users will be able to conveniently use e-services using their smart phone. The password is not any more weak authentication method using the SmartID_App by this paper.

References

1. Ballagas, R., et al.: The smart phone: a ubiquitous input device. IEEE Pervasive Computing (2006)
2. ENISA, Security Issued and Recommendations for Online Social Networks, ENISA Position Papter No.1
3. The daily caller, Ten ways your smartphone is vulnerable to hackers (March 2012)

4. Housley, R., et al.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile (April 2002)
5. Google, <http://support.google.com>
6. Zhang, L., et al.: A Dynamic Password Identity Authentication System Based on Mobile Phone Token. *Communications Technology* (2009)
7. Cheng, F.: A Secure Mobile OTP Token. In: Cai, Y., Magedanz, T., Li, M., Xia, J., Giannelli, C. (eds.) *Mobilware 2010*. LNCS, vol. 48, pp. 3–16. Springer, Heidelberg (2010)
8. Jeun, I., Lee, K., Won, D.: Enhanced Code-Signing Scheme for Smartphone Applications. In: Kim, T.-H., Adeli, H., Slezak, D., Sandnes, F.E., Song, X., Chung, K.-I., Arnett, K.P. (eds.) *FGIT 2011*. LNCS, vol. 7105, pp. 353–360. Springer, Heidelberg (2011)
9. Mulliner, C.R.: *Security of Smart Phone*, Master's Thesis of University of California (June 2006)

Development of *m*-TMS for Trusted Computing in Mobile Cloud

Hyun-Woo Kim¹, Eun-Ha Song¹, Jun-Ho Kim¹, Sang Oh Park²,
and Young-Sik Jeong^{1,*}

¹ Department of Computer Engineering, Wonkwang University,
Iksan, S. Korea

khw0121@gmail.com, {ehsong, ysjeong}@wku.ac.kr,
fate1204@naver.com

² School of Computer Science and Engineering, Chung-Ang
University, Seoul, S. Korea
sj1st@cs.cau.ac.kr

Abstract. In this rapidly changing IT society, computer system security is very crucial. This system security applies not only to individuals' computer systems but also to cloud environments. "Cloud" concerns operations on the web; therefore it is exposed to a lot of risks and security of its spaces where data are stored is vulnerable. Accordingly, in order to reduce factors of threat to security, the TCG (Trusted Computing Group) proposed a highly reliable platform based on a semiconductor-chip, TPM (Trusted Platform Module). Therefore, this paper proposes a *m*-TMS (Mobile Trusted Monitoring System) that monitors trusted state of a computing environment on which TPM chip-based TPB (Trusted Platform Board) is mounted and the current status of its system resources in a mobile device environment resulting from the development of network service technology. *m*-TMS is provided to users so that system resources of CPU, RAM, and process, the objects of monitoring in a computer system, may be monitored. Moreover, converting and detouring of single entities like PC or target addresses, which are attack pattern methods that pose a threat to computer system security, are combined. Branch instruction trace function is monitored using a BiT (Branch Instruction Trace) Profiling tool through which processes attacked or those suspected of being attacked may be traced, enabling users to actively respond.

Keywords: TPM, TPB, Mobile Cloud, System Behavior Monitoring, BiT Profiling.

1 Introduction

System security is an essential element whose importance is proportional to the development of IT society. Vulnerability of software-based security in existing

* Corresponding author.

system security mechanisms has grown due to varying hacking attacks. As a result, TC technology has emerged in order to complement weakness of system security through hardware-based security in addition to soft-ware based security. TC technology is to provide reliability so that computers may consistently behave in intended ways. Nonetheless, there is no technology to monitor TCG’s TPM-based computer system security on a real-time basis in a mobile device environment [1-3].

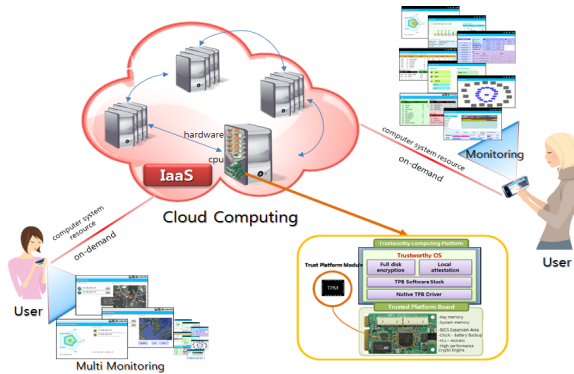


Fig. 1. Relationship of *m*-TMS for IaaS

Accordingly, this paper proposes easy, fast, and intuitional monitoring functions for notification of abnormal conditions and trust evaluation. This study develops *m*-TMS that evaluates and monitors system conditions in a mobile-device environment and may operate in any environment where network services are provided in order to provide high reliability on a client level. Mobile device-based *m*-TMS monitoring supports the following technologies in a cloud environment regarding trusted state of TPM-chip mounted TPB. In other words, users may properly respond through single monitoring of targeted computer systems, multiple monitoring of targeted multiple computers, and branch instruction trace function through a BiT Profiling tool [4-6].

Single monitoring provides monitoring of the current status of resources such as CPU, memory, process, user, and virtual memory and trusted state of a targeted computer system and monitoring of branch instruction trace function through the BiT Profiling technique. For multiple monitoring, the IP of targeted computer system is added. All functions of single monitoring may be applied to the added IP and the location where the monitored computer system operates is notified. Under multiple monitoring, a constraint of providing single monitoring to users is overcome and multiple computer systems may be rapidly monitored, raising time efficiency.

The idea on the BiT originated from the fact that malicious execution traces may be identified on an instruction level despite increase in the variety of methods to attack systems from outside and that instruction level behaviors under program implementation are influenced by system module operation. In other words, BiT may early detect the location of logical errors through branch instruction trace of the program, enabling the user to actively cope with them.

The services provided by this cloud computing includes SaaS (Software as a Service) that provides applied software, PaaS (Platform as a Service) that provides software development environment, and IaaS (Infrastructure as a Service) that provides hardware resources of a computing system. IaaS is a service that provides storage devices through cloud platform or computer capabilities through the Internet and highly focuses on security. This service is provided through the web and therefore in case reliability of security in computer storage devices and computer capabilities is lowered, its user's all information may be exposed. In this study, the user monitors on a realtime basis trusted state of IaaS that uses a TPB-mounted computer system through *m*-TMS and therefore may deal with malicious behaviors, thereby increasing system stability.

2 Related Works

2.1 TPM

The TCG is a not-for-profit organization formed in 2003 to develop, define, and promote open standards using reliability computing and security technologies based on hardware across multiple platforms or peripheral devices. The TCG that is proceeding with entire standardization of trust computing defines TPM as a method to guarantee software-based reliability. A TPM provides storage domains that can protect data, keys, certificates, etc. and encryption engines such as RAS (named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman) and Secure Hash Algorithm 1 (SHA-1). When each TPM chip is manufactured, unique keys—endorsement key (EK) and storage root key (SRK) are granted and they are kept within the chip, not disclosed. By utilizing such trait of TPM, weakness of software-based security methods may be complemented [8-9].

2.2 BiT Profiling

The idea of BiT originated from the fact that malicious execution traces may identified on an instruction level despite increase in the variety of methods to attack systems from outside and that instruction level behaviors under program implementation are influenced by system module operation. The BiT measures runtime behaviors at the processor instruction level, thereby resolving performance decrease in software-based security that can only cope with already known attack patterns or methods and are incapable of adequate defense against new types of attacks. Unlike existing security methods through approaches to find memory execution traces of control flow attacks, the BiT even performs profiling of control data such as execution paths and branch targets and combine converting and detouring of single entities such as PC or target addresses, making it fairly hard to initiate damage to these without modification [10].

The BiT monitors instruction trace and for processing selects a conditional branch in relation to each indirect branch instruction. Different standards for each indirect

branch are defined for security purposes. The BPC(Branch Program Counter) is a linear address for indirect branch instructions, TPC(Target Program Counter) and EP(Execution Path) refer to consecutive results (true or false) of conditional branches that lead to indirect branches indexed by BPC values. A program includes multiple indirect branches (multiple BPCs). These indirect branches follow different execution paths according to program execution. Therefore, inclusion of multiple TPCs in each BPC is possible and multiple conditional branches (EP) result from indirect branch targets. The hierarchical structure is shown in Figure 2.

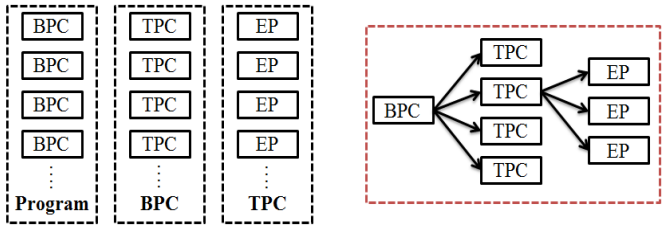


Fig. 2. BiT Architecture

2.3 IssS of Cloud Computing

Cloud computing is a service in which a user borrows computing resources as a lot as necessary through the Internet. It includes SaaS that provides software as a service, PaaS that provides software development environment as a service, and IaaS in which users rent virtual server instances from cloud data centers as a lot as necessary. IaaS does not sell hardware to server users but provides them with hardware computing capabilities only. IaaS has made it possible to transcend the limit of storage devices and capabilities. Nonetheless, IaaS is vulnerable in terms of security from server managers' perspectives. Accordingly, this study intends to combine a computer system that operates based on TPM-mounted TPB with m-TMS that monitors the current status of the system's resources and its reliability. A representative IaaS is Amazon Web Services' Simple storage Service and Elastic Compute Cloud (S3 & EC2).

3 Description of TPB

TPB is a hardware board for extension of TPM functions including the design of the hardware board and TPB interface. First, the hardware board specifies the hardware components, extended components of TPM, and digital circuit. The TPB interface specifies the hardware interface, extended interface, device driver design, and API design. There are many technologies in TPB including: safe high capacity storage, boot code for expansion of system BIOS, high performance encryption co-processor, and additional services for trusted computing. The high performance encryption co-processor supports a damper area for key storage, high speed operation for the encryption data algorithm, and hash function.

For improving trustworthiness with TPM, it researched three different issues: (1) program behavior verification, (2) the trustworthy computing platform, and (3) hardware technologies for the trustworthy platform board. The goal of program behavior verification is to develop a program counter (PC) encoding compiler and monitoring system for system behavior. The PC encoding compiler is the trusted compiler for hardening the control flow of the program and is the visual monitoring system for system behavior. The purpose of the trustworthy computing platform is improving the existing virtualization technology and hardening the operating system with respect to the TPM chip for trusted computing. The goal is to monitor the safety of booting, operating system and application programs. A trustworthy platform board is a good complement to hardware support technology for verifying the reliability, that is, it improves the primitive functions of TPM.

The trustworthy computing platform has the following two aims for enhancing the system reliability: (1) improving the existing virtualization technology and hardening the operating system with respect to the TPM chip for trusted computing, and (2) monitoring the safety of booting, operating system and application programs. The trusted boot loader supports the root of trust for measurement and extends trusted areas of the kernel on a computing platform based on hardware. Environments for reliable execution of applications can include local attestation and trusted areas of the kernel for applications. Local attestation is a storage technology of databases for integration of program code and verification. This platform uses encryption of the whole disk and provides disk access authority to authentication users only. It also specifies the design of the TPB software stack, TPB service class, and TPB device driver library API specification. As shown in Figure 3, we make a comparison between the software layer hierarchy of TPM and TPB [6].

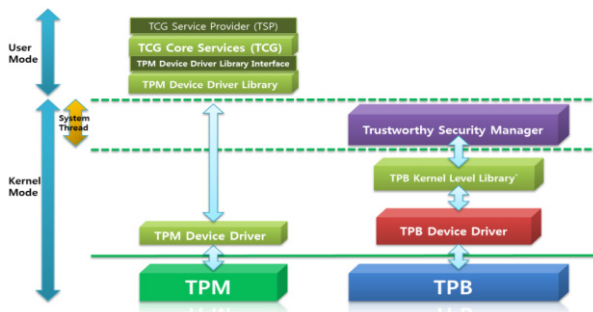


Fig. 3. Software Hierarchy of TPM and TPB

4 Design of *m*-TMS

m-TMS consists of three sides: TPB, TM-Interface, and Mobile-Device, and Figure 4 shows its whole structure chart.

On the TPB-side, computer system resources of the monitored target are regularly investigated and data on the most recent status of the resources are sent to XML Converter. XML Converter converts the data received from the system into XML data on BiT, integrate, peripheral central resource (PCR), process, CPU, memory, and user.

The TM-Interface-side plays the role of a mediator that connects the TPB-side and the Mobile-Device-side. The Mobile-Device-side receives data on the current status of resources and trusted state of the TPB-side and provides monitoring through its mobile devices to users.

The Mobile-Device-side is composed of four internal components: User Interface that receives input of selected monitoring from users; Activity Update to regularly bring data on the selected monitoring and make analysis to reflect them to Activity; Handler to inform the analyzed data to Activity Viewer; and Activity Viewer that receives notification of data update through Handler and provides visualized information on the monitoring to users. The Mobile Device selects desired monitoring through User Interface regarding the relevant monitored computer system. Activity Viewer visualizes the information on the monitoring selected by User Interface. Activity Update brings data on the current status of computer system resources through the TM-Interface-side in relation to the selected monitoring and analyzes them in line with the Activity form. Handler notifies Activity Viewer that such data collected and analyzed have been updated, and Viewer is continuously updated in such a way.

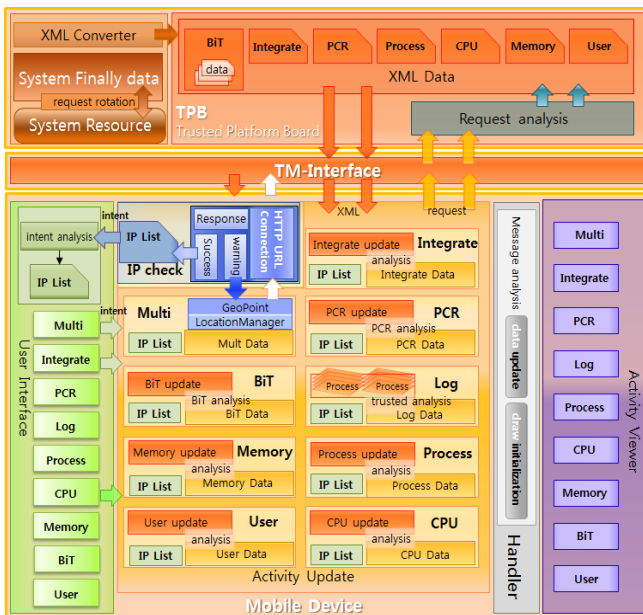


Fig. 4. m-TMS Architecture

In particular, in the case of multiple monitoring, multiple IPs are input and `HttpURLConnection` examines whether the relevant servers are operating. The conditions of the servers are received and when they are not operating, messages are delivered to the user. If a server is operating, the IP is added to the list and the location is received from the server and the physical location of the computer system is provided to the user; monitoring is made based on the IP. The added IP is shared among all Activities and all single monitoring functions are provided to them.

5 Implementation of *m*-TMS

5.1 Single Monitoring

Figure 5 displays a screen that visualizes single monitoring by *m*-TMS and execution of each Activity.

m-TMS consists of 8 different Activities: Activity(1) that visually shows trusted state of monitored computer system targets—RAM, CPU, user, process, and virtual memory through the pentagon graph; Activity(2) about trusted state Log of the process, Activity(3) that provides memory information, Activity(4) that divides trusted state of the current process into trusted, distrusted, and unknown and provides monitoring of each; Activity(5) that visualizes branch instruction trace through the BiT Profiling tool; Activity(6) that provides monitoring of PCR values of TPM; Activity(7) that provides comparative monitoring of basic information about CPU and utilization rates of each core through bar and line graphs; and Activity(8) that provides information about user IDs, user group identifiers (GIDs), and resource utilization amount of each user in connection with information about sessions employed by users.



Fig. 5. Single Monitoring of *m*-TMS

5.2 Multiple Monitoring

For multiple monitoring by *m*-TMS, the relevant server’s IP is added. Activity (1) in Figure 6 concerns the operation screen when multiple IPs are added. An IP is added to the relevant Activity by pressing Add button, and the computer system server of the input IP determines whether execution is made and then brings the address of the system. Latitude and longitude of the address through Geocoder provided by Android are calculated and the derived value is synchronized with MapView, enabling monitoring of the location. Under Activity (2) in Figure 6, the IP may be selected from the list on the left side and the utilization amount of relevant system CPU and RAM resources may be identified by such selection. The locations of all systems under monitoring may be pinpointed through the All View mode.

When the IP is added by menu change through Save button in Multi Activity, the same monitoring function in Figure 5 is provided. Activity (3) in Figure 6 shows the IP added by Multi Activity on the right side, supporting monitoring and visualization of CPU utilization rate, RAM occupancy rate, virtual memory occupancy rate, process, and user through the pentagon graph. When each Activity monitors two or more IPs, a button through which a selection may be made is dynamically generated on the Menu button. When an IP desired by the user is selected from the IP list, the relevant computer system’s resources and its trusted state are monitored.



Fig. 6. Multiple Monitoring of *m*-TMS

6 Conclusions

In order to provide a higher-level of reliability than existing software security methods, *m*-TMS embodied in this study provides real-time monitoring of information and resources of a computer system on which TPM chip-based TPB is mounted and visualizes information and trusted state of the computer system regarding multiple IPs, not a single IP. *m*-TMS provides monitoring to users by visualizing the current status of resources and trusted state of IaaS in a cloud computing system, enabling them to cope with errors in the computer system.

Meanwhile, BiT Profiling technique provides monitoring of branch instruction tracing. Each BPC, TPC, and EP are the result of each branch instruction tracing, and visualized information on the early locations of suspected logical errors is provided to a user through markings in red and graphs; Accordingly, the user can actively respond. Moreover, trusted state of IaaS through the TPB-mounted computer system in a cloud environment may be monitored through mobile devices, thereby enhancing security stability.

Future research will concern *m*-TMS's functions other than monitoring—functions to respond to attacks and logging functions to notify users of possible locations of logical errors caused by attacks, the possibility of attack on each process, and occurrence of problems with trusted state

Acknowledgments. This work was supported by the IT R&D Program of MKE/KEIT [KI002090, Development of Technology Base for Trustworthy Computing] and [10033915, Adaptive Fusion Technology for Large-scale Sensor node based Intelligent Surveillance Systems].

References

1. Trusted Computing Group Web Site, <http://www.trustedcomputinggroup.org>
2. TCG Specification Architecture Overview Specification Revision 1.4, Trusted Computing Group (TCG) (2007)
3. Common Criteria, Trusted Computing Group (TCG) Personal Computer (PC) Specific Trusted Building Block (TBB) Protection Profile and TCG PC Specific TBB With Maintenance Protection Profile (2004)
4. Lin, H., Lee, G.: Micro-Architecture Support for Integrity Measurement on Dynamic Instruction Trace. *Journal of Information Security* 1(1), 1–10 (2010)
5. IBM, Integrity Measurement Architecture (IMA), <http://domino.research.ibm.com/comm/researchpeople.nsf/pages/sailer.ima.html>
6. Jeong, Y.-S., Park, J.H.: Visual Trustworthy Monitoring System (v-TMS) for Behavior of Trusted Computing. *Journal of Internet Technology* 11(6), 731–741 (2010)
7. Suh, G., O'Donnell, C., Sachdev, I., Devadas, S.: Design and Implementation of the AEGIS Single-Chip Secure Processor Using Physical Random Functions. Technical Report, MIT CSAIL CSG Technical Memo 483 (2004)
8. Alves, T., Felton, D.: Trustzone: Integrated Hardware and Software Security. ARM white paper (2004)
9. Halfhill, T.: ARM Dons Armor: TrustZone Security Extensions Strengthen ARMv6 Architecture. *Microprocessor Report* (2003)
10. Crandall, J., Chong, F.: Minos: Control Data Attack Prevention Orthogonal To Memory Model. In: *Proc. the 37th Int'l. Symp. on Micro Architecture* (2004)

An Efficient Cloud Storage Model for Cloud Computing Environment

HwaYoung Jeong¹ and JongHyuk Park^{2,*}

¹ Humanitas College of Kyunghee University, Hoegi-dong, Seoul, 130-701, Korea

² Department of Computer Science and Engineering, Seoul National University of Science
and Technology

hyjeong@khu.ac.kr, parkjonghyuk1@hotmail.com

Abstract. Cloud computing is a new trend of information technology and computing system. In the traditional computing infrastructure, operating system software, applications and data are typically stored and managed on an individual user's computer. Cloud-computing has a different form from traditional way. All kinds of service and software are stored, accessed, and used via third party servers connected to the internet. If cloud computing was used, data and information can be easily shared and managed by user and service provider. Furthermore the user can do their works without application or software. When the user wants work using the software or application, the user just access cloud computing system via the internet. On the other hand, cloud computing has large data and information. Therefore the efficient method for large data/information management for cloud computing is necessary. In this paper, we proposed a cloud storage model for cloud computing environment. The model is to consider the relation between the cloud structures; service provider, application, and user.

Keywords: Cloud computing, Cloud storage, Data center, Service oriented system.

1 Introduction

A new business model is currently being adopted which is changing the way that Information Technology (IT) resources and services are deployed and used. In this model, the acquisition of resources and services occurs whenever and wherever needed, and the amount charged is related to the amount of resources and services that are actually used. This model of IT sold as a service has been called cloud computing. One of its main selling arguments is the possibility of substantial reductions on the total cost of ownership of IT infrastructures [1]. The latter term denotes the infrastructure as a “Cloud” from which businesses and users are able to access applications from anywhere in the world on demand. Thus, the computing world is

* Corresponding author.

rapidly transforming towards developing software for millions to consume as a service, rather than to run on their individual computers. At present, it is common to access content across the Internet independently without reference to the underlying hosting infrastructure. This infrastructure consists of data centers that are monitored and maintained around the clock by content providers. Cloud computing is an extension of this paradigm wherein the capabilities of business applications are exposed as sophisticated services that can be accessed over a network [2].

With rapid development of cloud computing, more and more enterprises will outsource their sensitive data for sharing in a cloud. To keep the shared data legal against untrusted cloud service providers, a natural way is to store only the encrypted data in a cloud. The key problems of this approach include establishing access control for the encrypted data, and revoking the access rights from users when they are no longer authorized to access the encrypted data. Compared with traditional storage, cloud storage is not just hardware, but network equipment, storage equipment, servers, applications, public access interface, the access network and the client program and other parts of the system. Cloud storage is provided storage services, storage services through the network data stored in local storage service provider (SSP) to provide online storage space. Need to store the service users no longer need to build their own data centers, storage services only apply to the SSP, thus avoiding the duplication storage platform, saving the expensive hardware and software infrastructure investments.

Current cloud storage infrastructures are focused on providing users with easy interfaces and high performance services. However, there are some classes of storage services for which the current cloud model may not fit well [3].

In this paper, we propose a cloud storage model that is to consider the relation between cloud service provider, application and user. This model includes the service connection with application, platform, and infrastructure. The remainder of the paper is structured as follows. Section 2 presents related work with cloud computing. Section 3 describes the proposed cloud storage model for cloud computing considering their relation, application, platform, and infrastructure. Section 4 concludes with a discussion on this research.

2 Related Work

2.1 Cloud Computing Environment

Cloud computing has been referred to as an architecture, a platform, an operating system, and a service, and in some senses, it is all of these. A basic definition of cloud computing is using the Internet to perform tasks on computers. It is an approach to computing in which resources and information are provided through services over the Internet, in which the network of services is collectively known as “the cloud” [4]. The transient nature of cloud computing is also reflected in the various business models used to sell the service. They include [5]:

- Cloud software as a service (SaaS), where the customer uses applications provided by the seller. One example that has been in use for some time is web-based e-mail. In this respect, the customer uses the network, servers, operating systems, storage facilities, and possibly individual applications provided by the seller.
- Cloud platform as a service (PaaS), by which the seller provides the infrastructure (network, servers, operating systems, storage facilities) to enable a customer to use their own applications that they create by using any programming languages and tools supported by the seller. The seller will not necessarily offer its own or a single infrastructure to provide the service. It may act as an ‘aggregator’ by which the seller uses a number of third parties to provide separate applications and sets of hardware, but the buyer is given the impression that that the service they are paying for is one consolidated infrastructure.
- Cloud infrastructure as a service (IaaS) (sometimes called a ‘hosted’ service), where the seller provides the infrastructure (network, servers, operating systems, storage facilities) to enable the customer to use and run software of their choice, which can include operating systems and applications.

These services define a layered system structure for cloud computing as shown in Fig. 1.

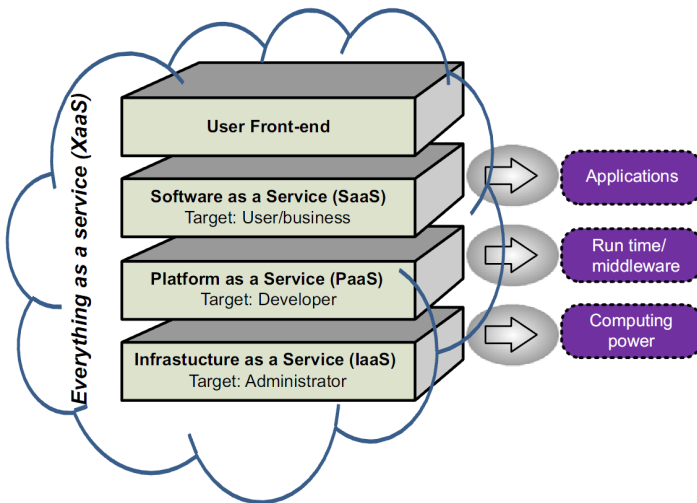


Fig. 1. Cloud computing: everything is a service

At the Infrastructure layer, processing, storage, networks, and other fundamental computing resources are defined as standardized services over the network. Cloud providers’ clients can deploy and run operating systems and software for their

underlying infrastructures. The middle layer, i.e. PaaS provides abstractions and services for developing, testing, deploying, hosting, and maintaining applications in the integrated development environment. The application layer provides a complete application set of SaaS [6].

2.2 Cloud Storage

Cloud storage is typically where a business stores and retrieves data from a data storage facility via the Internet. Storing data in this way offers near unlimited storage and can provide significant cost savings as there is no need for the business to buy, run, upgrade or maintain data storage systems with unused spare capacity [6].



Fig. 2. A simple structure of cloud storage

With cloud storage to accelerate growth showing a surprising trend, more traditional dual-controller storage controller or, when capacity and performance expansion, it is often only a simple increase in the number of back-end disk when the disk to a certain number of when the front-end controller, the backplane can not follow the expansion, the formation of a performance bottleneck. Traditional storage architecture is characterized by: usually with a single powerful, comprehensive, complete and have personalized slightly expensive computing resources. But the overall structure is not easy to adopt new technologies or external, and when the whole system to a certain extent, the need to carefully match the calculation of the current system resources, not very easy to achieve the overall expansion of the entire system. Cloud storage in the cloud computing, and developed an extension of the concept of a new concept, refers to the application through the cluster, grid or distributed file systems and other functions, the network in a large variety of different types of storage devices together through the application of software work together to provide outside access to data storage and business functions of a system. Cloud storage is applied through the cluster, grid or distributed file systems and other functions, the network in a large variety of different types of storage devices through the application of software work together, a common external data storage and access capabilities of a business data storage and management as the core of the cloud computing system [3].

In this structure, data may be transferred between many computers across a number of continents during the time a person or legal entity decides to use a cloud computing service. As a result, there are at least three possibilities in relation to the data: there might be multiple copies of the data on each storage device it is stored upon as it is moved around the globe, or the data might be securely erased as it is

moved from one computer infrastructure to another, leaving no trace; alternatively, residual copies of data might be created that a user has an obligation to delete. Copies of data might not only be stored in an unknown number of computers across the globe, but there might be an unknown number of copies of the same digital document in different iterations across different jurisdictions. This could affect the identification of relevant data for criminal proceedings [5]. Rajkumar et al. [2] proposed cloud service framework with cloud storage as shown in Fig 3.

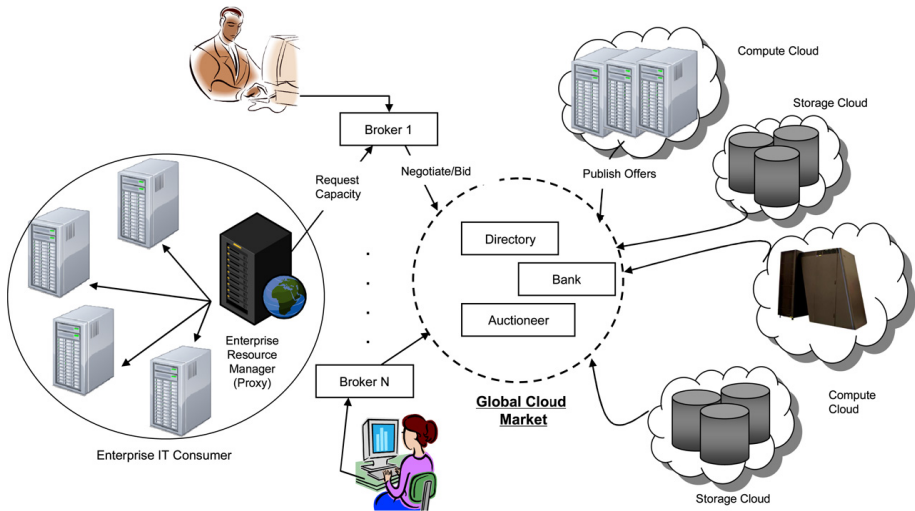


Fig. 3. Framework of cloud service with cloud storage

3 Cloud Storage Model for Cloud Computing

Most of the cloud computing infrastructure is transmitted through a reliable data center services and create different levels on the server virtualization technology. It can be any place to provide network infrastructure to use these services. “Cloud” usually presents the computing needs of all users a single access point. Cloud storage is the core software applications combined with the storage device, through the application software to achieve the storage service to store the changes. Compared with traditional storage, cloud storage is not just hardware, but network equipment, storage equipment, servers, applications, public access interface, the access network, and client and other components of the complex system [3]. Wang [3] proposed cloud storage system architecture model as four layers; Storage layer, Basic management, Application interface layer and Access layer.

- Storage layer: Storage layer is the most basic part of the cloud storage. Cloud storage is often a huge number of storage devices and the distribution of many different regions, each other through wide area network, Internet. Cloud storage system offers a variety of external

storage services, services of data stored in a unified cloud storage system to form a huge data pool.

- Basic management layer: Cloud-based storage management is the core part is stored in the cloud part of the most difficult to achieve.
- Application interface layer: Cloud storage application interface layer is the most flexible part. Different operating units can cloud the actual storage type of business; develop a different application service interface provided by different applications.
- Access layer: Any authorized user via a standard application interface to log public cloud storage system, enjoy cloud storage service.

Through this structure, we propose a cloud storage model for their environment. The model typically consists of three parts; Storage and management layer, Application layer, and Access layer. These parts also was related cloud computing conceptual structure; SaaS, PaaS, and IaaS. Fig 4 shows our storage model.

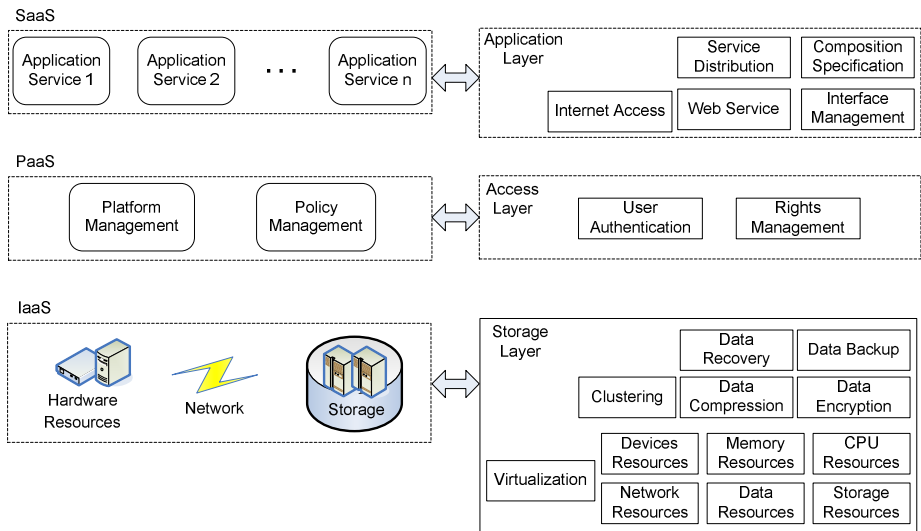


Fig. 4. Proposed cloud storage model by three layers

Application layer consists of 5 factors; Internet Access, Service Distribution, Web Service, Composition Specification, and Interface Management. Application layer has 2 factors; User Authentication and Rights Management. The most important factor of these is Storage layer. It has 12 factors; Virtualization, Devices Resources, Network Resources, Memory Resources, Data Resources, CPU Resources, Storage Resources, Clustering, Data Recovery, Data Compression, Data Backup, and Data Encryption. The Storage layer deal with the process to manage, service, handling, interface between the cloud service and data/information. Therefore this layer influence to hardware resources, Network, and Storage resources in IaaS of cloud computing structure.

4 Conclusion

Cloud computing is a new paradigm of information and service oriented technology for today's and nearly future perspective. Cloud storage is not just storage but more applications. In the cloud computing environment, cloud storage deals with the process to control and manage all kind of data and information with hardware, service (software), and platform resources. Therefore, in order to support cloud service to user efficiently, to identify and organize the cloud storage model can be very important work.

In this paper, we propose an efficient cloud storage model for cloud computing environment. Basically, cloud computing has three factors in the structure; SaaS, PaaS, and IaaS. In order to support the process with data and information storage, the storage model has to consider the relation between their factors and storage model. The proposed model connects their factors in cloud computing structure. And our model consists of three layers; Storage layer, Application layer, and Access layer. Storage layer connect to IaaS in cloud computing structure, Application layer connect to SaaS, and Access layer connect to PaaS. Through this model, we can shows the relation and behavior between the factor in cloud computing structure which is IaaS, SaaS, and PaaS, and the each of storage layer.

References

1. Maciel Jr., P.D., Brasileiro, F., Santos, R.A., Candeia, D., Lopes, R., Carvalho, M., Miceli, R., Andrade, N., Mowbray, M.: Business-driven short-term management of a hybrid IT infrastructure. *J. Parallel Distrib. Comput.* 72, 106–119 (2012)
2. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25, 599–616 (2009)
3. Wang, D.: An Efficient Cloud Storage Model for Heterogeneous Cloud Infrastructures. *Procedia Engineering* 23, 510–515 (2011)
4. Andriole, K.P., Khorasani, R.: Cloud Computing: What Is It and Could It Be Useful?, *Bits and Bytes. American College of Radiology* (2010), doi:10.1016/j.jacr.2010.01.009
5. Mason, S., George, E.: Digital evidence and 'cloud' computing. *Computer Law & Security Review* 27, 524–528 (2011)
6. Joint, A., Baker, E., Eccles, E.: Hey, you, get off of that cloud? *Computer Law & Security Review* 25, 270–274 (2009)

Author Index

- Abidi, Leila 74
Ahmed, Firoz 312
- Banditwattanawong, Thepparit 1
Ben, Ke-rong 221
- Cao, Jian 137
Cérin, Christophe 74
Chandrasekar, Ashok 152, 289
Chandrasekar, Karthik 152, 289
Chen, Junliang 89
Chen, Shizhan 109
Chen, Yanjun 16
Cheng, Bo 89
Chin, Sung-Ho 31
Chourou, Lilia 264
Chung, Kwang-Sik 31
Cui, Yabing 231
- Dong, Xiaoju 198
- Elleuch, Ahmed 264
- Feng, Zhiyong 109
- Gmach, Daniel 16
Gong, Bin 254
Gu, Pingli 89
Guo, Bin 122
- Han, Jikwang 162
He, Ligang 323
Hsu, Wen-Hung 338
Hu, Chunming 231
Hua, Rui 16
Huang, Chih-Lin 64
- Jarvis, Stephen A. 323
Jemni, Mohamed 264
Jeong, HwaYoung 370
Jeong, Young-Sik 361
Jeun, Inkyung 350
Jiang, Yan 89
- Jin, Hai 244
Joe, Inwhee 183
- Kakkad, Jignesh 210
Kim, Hyun-Woo 361
Kim, Jennifer 172
Kim, Jun-Ho 361
Kim, Mijin 350
Klai, Kais 74
Kuo, Cheng-Ta 338
Kwak, Ho-Young 162
- Lee, Junghoon 162
Li, Feibo 54
Li, Kenli 279
Li, Minglu 137
Li, Ruixuan 279
Li, Xiangzhen 302
Liang, Yunji 122
Lim, JongBeom 31
Liu, Limin 323
- Mahadevan, Malairaja 289
Mannava, Vishnuvardhan 98
- Oh, Hoon 312
- Pan, Weisen 109
Parameswaran, Nandan 210
Park, Gyung-Leen 162
Park, JongHyuk 370
Park, Sang Oh 361
- Qin, Zhuoran 46
- Rahim, Rafica Abdul 152
Ramasatagopan, Harini 152
Ramesh, T. 98
Ruan, Li 54
- Sallam, Ahmed 279
Shang, Yanlei 89
Shih, Wen-Chung 64, 338
Song, Eun-Ha 361
- Tang, Zhuo 279

- Varalakshmi, P. 289
- Wan, Jiguang 16
- Wang, Dongping 254
- Wang, Hanwen 231
- Wang, Huixiang 54
- Wang, Jianzong 16
- Wang, Jun 221
- Wei, Juan 279
- Wo, Tianyu 231
- Won, Dongho 350
- Wu, Yihua 137
- Xiao, Limin 54
- Xie, Changsheng 16
- Xie, Xia 244
- Xue, Jianxin 198
- Yang, Chao-Tung 64, 338
- Yang, Yue 122
- Yoon, Seok Hoon 312
- Yu, Heon-Chang 31
- Yu, Zhiwen 122
- Yuan, PingPeng 244
- Zeng, Ling kang 302
- Zhang, Fu-Quan 183
- Zhang, Jixian 46
- Zhang, Xuejie 46
- Zhang, Yiyi 302
- Zhao, Guoling 254
- Zhen, Yan 302
- Zheng, Di 221
- Zhou, Bin 244
- Zhou, Xingshe 122
- Zhu, Mingfa 54