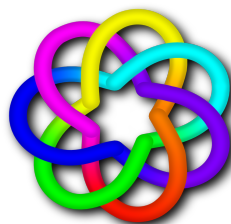


CONFERENCES IN RESEARCH AND PRACTICE IN
INFORMATION TECHNOLOGY

VOLUME 140

PARALLEL AND DISTRIBUTED COMPUTING 2013

AUSTRALIAN COMPUTER SCIENCE COMMUNICATIONS, VOLUME 35, NUMBER 6



PARALLEL AND DISTRIBUTED COMPUTING 2013

Proceedings of the Eleventh Australasian Symposium on
Parallel and Distributed Computing
(AusPDC 2013), Adelaide, Australia,
29 January – 1 February 2013

Bahman Javadi and Saurabh Kumar Garg, Eds.

Volume 140 in the Conferences in Research and Practice in Information Technology Series.
Published by the Australian Computer Society Inc.



Published in association with the ACM Digital Library.

Parallel and Distributed Computing 2013. Proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, Australia, 29 January – 1 February 2013

Conferences in Research and Practice in Information Technology, Volume 140.

Copyright ©2013, Australian Computer Society. Reproduction for academic, not-for-profit purposes permitted provided the copyright text at the foot of the first page of each paper is included.

Editors:

Bahman Javadi

School of Computing, Engineering and Mathematics
University of Western Sydney
Penrith, NSW 2751
Australia
Email: b.javadi@uws.edu.au

Saurabh Kumar Garg

IBM Research, Australia
University of Melbourne Precinct
Victoria 3010
Australia
Email: saurabhg@unimelb.edu.au

Series Editors:

Vladimir Estivill-Castro, Griffith University, Queensland
Simeon J. Simoff, University of Western Sydney, NSW
Email: crpit@scm.uws.edu.au

Publisher: Australian Computer Society Inc.
PO Box Q534, QVB Post Office
Sydney 1230
New South Wales
Australia.

Conferences in Research and Practice in Information Technology, Volume 140.
ISSN 1445-1336.
ISBN 978-1-921770-25-8.

Document engineering, January 2013 by CRPIT
On-line proceedings, January 2013 by the University of Western Sydney
Electronic media production, January 2013 by the University of South Australia

The *Conferences in Research and Practice in Information Technology* series disseminates the results of peer-reviewed research in all areas of Information Technology. Further details can be found at <http://crpit.com/>.

Table of Contents

Proceedings of the Eleventh Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, Australia, 29 January – 1 February 2013

Preface	vii
Programme Committee	viii
Organising Committee	x
Welcome from the Organising Committee	xi
CORE - Computing Research & Education	xiii
ACSW Conferences and the Australian Computer Science Communications	xiv
ACSW and AusPDC 2013 Sponsors	xvi

Contributed Papers

A GPU-based Method for Computing Eigenvector Centrality of Gene-expression Networks	3
<i>Ahmed Shamsul Arefin, Regina Berretta and Pablo Moscato</i>	
Non-blocking Parallel Subset Construction on Shared-memory Multicore Architectures	13
<i>Hyewon Choi and Bernd Burgstaller</i>	
Simseer and Bugwise - Web Services for Binary-level Software Similarity and Defect Detection	21
<i>Silvio Cesare and Yang Xiang</i>	
Cloud-Aware Processing of MapReduce-Based OLAP Applications	31
<i>Hyuck Han, Young Choon Lee, Seungmi Choi, Heon Y. Yeom and Albert Y. Zomaya</i>	
Tools and Processes to Support the Development of a National Platform for Urban Research: Lessons (Being) Learnt from the AURIN Project	39
<i>Richard Sinnott, Christopher Bayliss, Luca Morandini and Martin Tomko</i>	
A Web Portal for Management of Aneka-Based MultiCloud Environments	49
<i>Mohammed Alrokayan and Rajkumar Buyya</i>	
Author Index	57

Preface

Parallel and distributed computing has played a key role in enabling execution of several scientific applications over the past years. With advances in technology, it has changed its scope from small clusters of workstations to very large-scale datacenters, which providing Cloud computing services. This proceeding presents some of the current research in this area that have contributed to the 11th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), held between 29 January–1 February 2013 in Adelaide, Australia in conjunction with the Australasian Computer Science Week (ACSW 2013). In 2010, Australasian Symposium on Grid Computing and e-Research (AusGrid) was broadened to include all aspects of parallel and distributed computing and hence was called as Australasian Symposium on Parallel and Distributed Computing. Following a couple of successful events, AusPDC has become the flagship symposium for Grid, Cloud, Cluster, and Distributed Computing research in Australia and New Zealand.

Submissions were received, mostly from Australia, but also from New Zealand, China, Korea, India, and Indonesia. The full version of each paper was carefully reviewed by at least two referees, and evaluated according to its originality, correctness, readability and relevance. A total of 6 papers out of 13 submissions were accepted to present in the conference. The accepted papers cover topics from Cloud computing, system security, GPU computing, multi-processing systems, and e-Research tools. In addition to the technical papers, we are delighted to welcome an invited talk given by Professor Richard O. Sinnott from the University of Melbourne.

We are very thankful to the Program Committee members, and external reviewers for their outstanding and timely work, which was invaluable for taking the quality of this year’s program to such a high level. We also wish to acknowledge the efforts of the authors who submitted their papers and without whom this conference would have not been possible. Due to the competitive selection process, several strong papers could not be included in the program. We sincerely hope that prospective authors will continue to view the AusPDC symposium series as the premiere venue in the field for disseminating their work and results. We would like to acknowledge the leadership and untiring efforts of the conference General Chair, Dr. Ivan Lee and the guidance provided by the steering committee, in particular Professor Rajkumar Buyya, Associate Professor Jinjun Chen, and Dr. Rajiv Ranjan.

We are grateful to ACSW Organizing Committee and Professor Simeon Simoff from UWS representing CRPIT for his assistance in the production of the proceedings. Thanks to the School of Computing, Engineering, and Mathematics at University of Western Sydney for web support, advertising and refereeing for the conference.

Bahman Javadi

University of Western Sydney

Saurabh Kumar Garg

IBM Research, Australia

AusPDC 2013 Programme Chairs

January 2013

Programme Committee

Chairs

Bahman Javadi, University of Western Sydney, Australia
Saurabh Kumar Garg, IBM Research, Australia

Members

Jemal Abawajy, Deakin of University, Australia
David Abramson, Monash University, Australia
Peter Bertok, RMIT, Australia
Borzoo Bonakdarpour, University of Waterloo, Canada
Rajkumar Buyya, The University of Melbourne, Australia
Geffrey Fox, Indiana University, USA
Andrzej Goscinski, Deakin University, Australia
Kenneth Hawick, Massey University, New Zealand
John Hine, Victoria University of Wellington, New Zealand
Michael Hobbs, Deakin University, Australia
Zhiyi Huang, Otago University, New Zealand
Nick Jones, University of Auckland, New Zealand
Wayne Kelly, Queensland University of Technology, Australia
Kevin Lee, Murdoch University, Australia
Young Choon Lee, The University of Sydney, Australia
Laurent Lefevre, INRIA, University of Lyon, France
Weifa Liang, Australian National University, Australia
Farhad Mehdipour, Kyushu University, Japan
Paul Roe, Queensland University of Technology, Australia
Hong Shen, University of Adelaide, Australia
Jun Shen, University of Wollongong, Australia
Michael Sheng, University of Adelaide, Australia
Weisheng Si, University of Western Sydney, Australia
Gaurav Singh, CSIRO Mathematical and Information Sciences, Australia
Richard Sinnott, The University of Melbourne, Australia
Peter Strazdins, Australian National University, Australia
Kurt Vanmechelen, University of Antwerp, Belgium
Andrew Wendelborn, University of Adelaide, Australia
Yulei Wu, Chinese Academy of Sciences, China
Yang Xiang, Deakin University, Australia
Jingling Xue, University of New South Wales, Australia
Jun Yan, University of Wollongong, Australia
Yun Yang, Swinburne University of Technology, Australia
Rui Zhang, The University of Melbourne, Australia
Albert Zomaya, The University of Sydney, Australia

Steering Committee

Prof. David Abramson, Monash University, Australia
Prof. Rajkumar Buyya, University of Melbourne, Australia
A./Prof. Jinjun Chen (Vice Chair), University of Technology Sydney, Australia
Dr. Paul Coddington, University of Adelaide, Australia
Prof. Andrzej Goscinski (Chair), Deakin University, Australia
Prof. Kenneth Hawick, Massey University, New Zealand
Prof. John Hine, Victoria University of Wellington, New Zealand
Dr. Rajiv Ranjan, CSIRO ICT Centre, Australia Dr. Wayne Kelly, Queensland University of Technology, Australia

Prof. Paul Roe, Queensland University of Technology, Australia
Dr. Andrew Wendelborn, University of Adelaide, Australia
Dr. Bahman Javadi, University of Western Sydney, Australia
Dr. Saurabh Kumar Garg, IBM Research, Australia

Organising Committee

Chair

Dr. Ivan Lee

Finance Chair

Dr. Wolfgang Mayer

Publication Chair

Dr. Raymond Choo

Local Arrangement Chair

Dr. Grant Wigley

Registration Chair

Dr. Jinhai Cai

Welcome from the Organising Committee

On behalf of the Organising Committee, it is our pleasure to welcome you to Adelaide and to the 2013 Australasian Computer Science Week (ACSW 2013). Adelaide is the capital city of South Australia, and it is one of the most liveable cities in the world. ACSW 2013 will be hosted in the City West Campus of University of South Australia (UniSA), which is situated at the north-west corner of the Adelaide city centre.

ACSW is the premier event for Computer Science researchers in Australasia. ACSW2013 consists of conferences covering a wide range of topics in Computer Science and related area, including:

- Australasian Computer Science Conference (ACSC) (Chaired by Bruce Thomas)
- Australasian Database Conference (ADC) (Chaired by Hua Wang and Rui Zhang)
- Australasian Computing Education Conference (ACE) (Chaired by Angela Carbone and Jacqueline Whalley)
- Australasian Information Security Conference (AISC) (Chaired by Clark Thomborson and Udaya Parampalli)
- Australasian User Interface Conference (AUIC) (Chaired by Ross T. Smith and Burkhard C. Wünsche)
- Computing: Australasian Theory Symposium (CATS) (Chaired by Tony Wirth)
- Australasian Symposium on Parallel and Distributed Computing (AusPDC) (Chaired by Bahman Javadi and Saurabh Kumar Garg)
- Australasian Workshop on Health Informatics and Knowledge Management (HIKM) (Chaired by Kathleen Gray and Andy Koronios)
- Asia-Pacific Conference on Conceptual Modelling (APCCM) (Chaired by Flavio Ferrarotti and Georg Grossmann)
- Australasian Web Conference (AWC2013) (Chaired by Helen Ashman, Michael Sheng and Andrew Trotman)

In addition to the technical program, we also put together social activities for further interactions among our participants. A welcome reception will be held at Rockford Hotel's Rooftop Pool area, to enjoy the fresh air and panoramic views of the cityscape during Adelaide's dry summer season. The conference banquet will be held in Adelaide Convention Centre's Panorama Suite, to experience an expansive view of Adelaide's serene riverside parklands through the suite's seamless floor to ceiling windows.

Organising a conference is an enormous amount of work even with many hands and a very smooth cooperation, and this year has been no exception. We would like to share with you our gratitude towards all members of the organising committee for their dedication to the success of ACSW2013. Working like one person for a common goal in the demanding task of ACSW organisation made us proud that we got involved in this effort. We also thank all conference co-chairs and reviewers, for putting together conference programs which is the heart of ACSW. Special thanks goes to Alex Potanin, who shared valuable experiences in organising ACSW and provided endless help as the steering committee chair. We'd also like to thank Elyse Perin from UniSA, for her true dedication and tireless work in conference registration and event organisation. Last, but not least, we would like to thank all speakers and attendees, and we look forward to several stimulating discussions.

We hope your stay here will be both rewarding and memorable.

Ivan Lee

School of Information Technology & Mathematical Sciences

ACSW2013 General Chair

January, 2013

CORE - Computing Research & Education

CORE welcomes all delegates to ACSW2013 in Adelaide. CORE, the peak body representing academic computer science in Australia and New Zealand, is responsible for the annual ACSW series of meetings, which are a unique opportunity for our community to network and to discuss research and topics of mutual interest. The original component conferences - ACSC, ADC, and CATS, which formed the basis of ACSW in the mid 1990s - now share this week with eight other events - ACE, AISC, AUIC, AusPDC, HIKM, ACDC, APCCM and AWC which build on the diversity of the Australasian computing community.

In 2013, we have again chosen to feature a small number of keynote speakers from across the discipline: Riccardo Bellazzi (HIKM), and Divyakant Agrawal (ADC), Maki Sugimoto (AUIC), and Wen Gao. I thank them for their contributions to ACSW2013. I also thank invited speakers in some of the individual conferences, and the CORE award winner Michael Sheng (CORE Chris Wallace Award). The efforts of the conference chairs and their program committees have led to strong programs in all the conferences, thanks very much for all your efforts. Thanks are particularly due to Ivan Lee and his colleagues for organising what promises to be a strong event.

The past year has been turbulent for our disciplines. ERA2012 included conferences as we had pushed for, but as a peer review discipline. This turned out to be good for our disciplines, with many more Universities being assessed and an overall improvement in the visibility of research in our disciplines. The next step must be to improve our relative success rates in ARC grant schemes, the most likely hypothesis for our low rates of success is how harshly we assess each others' proposals, a phenomenon which demonstrably occurs in the US NFS. As a US Head of Dept explained to me, "in CS we circle the wagons and shoot within".

Beyond research issues, in 2013 CORE will also need to focus on education issues, including in Schools. The likelihood that the future will have less computers is small, yet where are the numbers of students we need? In the US there has been massive growth in undergraduate CS numbers of 25 to 40% in many places, which we should aim to replicate. ACSW will feature a joint CORE, ACDICT, NICTA and ACS discussion on ICT Skills, which will inform our future directions.

CORE's existence is due to the support of the member departments in Australia and New Zealand, and I thank them for their ongoing contributions, in commitment and in financial support. Finally, I am grateful to all those who gave their time to CORE in 2012; in particular, I thank Alex Potanin, Alan Fekete, Aditya Ghose, Justin Zobel, John Grundy, and those of you who contribute to the discussions on the CORE mailing lists. There are three main lists: csprofs, cshods and members. You are all eligible for the members list if your department is a member. Please do sign up via <http://lists.core.edu.au/mailman/listinfo> - we try to keep the volume low but relevance high in the mailing lists.

I am standing down as President at this ACSW. I have enjoyed the role, and am pleased to have had some positive impact on ERA2012 during my time. Thank you all for the opportunity to represent you for the last 3 years.

Tom Gedeon

President, CORE
January, 2013

ACSW Conferences and the Australian Computer Science Communications

The Australasian Computer Science Week of conferences has been running in some form continuously since 1978. This makes it one of the longest running conferences in computer science. The proceedings of the week have been published as the *Australian Computer Science Communications* since 1979 (with the 1978 proceedings often referred to as *Volume 0*). Thus the sequence number of the Australasian Computer Science Conference is always one greater than the volume of the Communications. Below is a list of the conferences, their locations and hosts.

2014. Volume 36. Host and Venue - AUT University, Auckland, New Zealand.

2013. Volume 35. Host and Venue - University of South Australia, Adelaide, SA.

2012. Volume 34. Host and Venue - RMIT University, Melbourne, VIC.

2011. Volume 33. Host and Venue - Curtin University of Technology, Perth, WA.

2010. Volume 32. Host and Venue - Queensland University of Technology, Brisbane, QLD.

2009. Volume 31. Host and Venue - Victoria University, Wellington, New Zealand.

2008. Volume 30. Host and Venue - University of Wollongong, NSW.

2007. Volume 29. Host and Venue - University of Ballarat, VIC. First running of HDKM.

2006. Volume 28. Host and Venue - University of Tasmania, TAS.

2005. Volume 27. Host - University of Newcastle, NSW. APBC held separately from 2005.

2004. Volume 26. Host and Venue - University of Otago, Dunedin, New Zealand. First running of APCCM.

2003. Volume 25. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Adelaide Convention Centre, Adelaide, SA. First running of APBC. Incorporation of ACE. ACSAC held separately from 2003.

2002. Volume 24. Host and Venue - Monash University, Melbourne, VIC.

2001. Volume 23. Hosts - Bond University and Griffith University (Gold Coast). Venue - Gold Coast, QLD.

2000. Volume 22. Hosts - Australian National University and University of Canberra. Venue - ANU, Canberra, ACT. First running of AUC.

1999. Volume 21. Host and Venue - University of Auckland, New Zealand.

1998. Volume 20. Hosts - University of Western Australia, Murdoch University, Edith Cowan University and Curtin University. Venue - Perth, WA.

1997. Volume 19. Hosts - Macquarie University and University of Technology, Sydney. Venue - Sydney, NSW. ADC held with DASFAA (rather than ACSW) in 1997.

1996. Volume 18. Host - University of Melbourne and RMIT University. Venue - Melbourne, Australia. CATS joins ACSW.

1995. Volume 17. Hosts - Flinders University, University of Adelaide and University of South Australia. Venue - Glenelg, SA.

1994. Volume 16. Host and Venue - University of Canterbury, Christchurch, New Zealand. CATS run for the first time separately in Sydney.

1993. Volume 15. Hosts - Griffith University and Queensland University of Technology. Venue - Nathan, QLD.

1992. Volume 14. Host and Venue - University of Tasmania, TAS. (ADC held separately at La Trobe University).

1991. Volume 13. Host and Venue - University of New South Wales, NSW.

1990. Volume 12. Host and Venue - Monash University, Melbourne, VIC. Joined by Database and Information Systems Conference which in 1992 became ADC (which stayed with ACSW) and ACIS (which now operates independently).

1989. Volume 11. Host and Venue - University of Wollongong, NSW.

1988. Volume 10. Host and Venue - University of Queensland, QLD.

1987. Volume 9. Host and Venue - Deakin University, VIC.

1986. Volume 8. Host and Venue - Australian National University, Canberra, ACT.

1985. Volume 7. Hosts - University of Melbourne and Monash University. Venue - Melbourne, VIC.

1984. Volume 6. Host and Venue - University of Adelaide, SA.

1983. Volume 5. Host and Venue - University of Sydney, NSW.

1982. Volume 4. Host and Venue - University of Western Australia, WA.

1981. Volume 3. Host and Venue - University of Queensland, QLD.

1980. Volume 2. Host and Venue - Australian National University, Canberra, ACT.

1979. Volume 1. Host and Venue - University of Tasmania, TAS.

1978. Volume 0. Host and Venue - University of New South Wales, NSW.

Conference Acronyms

ACDC	Australasian Computing Doctoral Consortium
ACE	Australasian Computer Education Conference
ACSC	Australasian Computer Science Conference
ACSW	Australasian Computer Science Week
ADC	Australasian Database Conference
AISC	Australasian Information Security Conference
APCCM	Asia-Pacific Conference on Conceptual Modelling
AUIC	Australasian User Interface Conference
AusPDC	Australasian Symposium on Parallel and Distributed Computing (replaces AusGrid)
AWC	Australasian Web Conference
CATS	Computing: Australasian Theory Symposium
HIKM	Australasian Workshop on Health Informatics and Knowledge Management

Note that various name changes have occurred, which have been indicated in the Conference Acronyms sections in respective CRPIT volumes.

ACSW and AusPDC 2013 Sponsors

We wish to thank the following sponsors for their contribution towards this conference.



CORE - Computing Research and Education,
www.core.edu.au



Australian Computer Society,
www.acs.org.au



**University of
South Australia**

University of South Australia,
www.unisa.edu.au/



University of Western Sydney,
www.uws.edu.au

CONTRIBUTED PAPERS

A GPU-based Method for Computing Eigenvector Centrality of Gene-expression Networks

Ahmed Shamsul Arefin

Regina Berretta

Pablo Moscato

Centre for Bioinformatics, Biomarker Discovery and Information-Based Medicine,
School of Electrical Engineering and Computer Science,
Faculty of Engineering and Built Environment,
The University of Newcastle, Callaghan, NSW 2308, Australia
Email: Ahmed.Arefin@uon.edu.au, {Regina.Berretta, Pablo.Moscato}@newcastle.edu.au

Abstract

In this paper, we present a fast and scalable method for computing eigenvector centrality using graphics processing units (GPUs). The method is designed to compute the centrality on gene-expression networks, where the network is pre-constructed in the form of k NN graphs from DNA microarray data sets.

Keywords: Eigenvector, centrality, k NN, CUDA.

1 Introduction

Centrality analysis measures the relative importance of the elements in a given network based on their connectivity within the network structure. In other words, centrality measures help to “rank” the network elements according to their importance within the network structure. Formally, the centrality of a network is defined as follows (Junker et al. 2006), let $G(V, E)$ be a directed or undirected graph (network), then the centrality on G is defined as a function $C : V \rightarrow \mathbb{R}$ that assigns a real number to each vertex. For a pair of two vertices, u and v , if $C(u) > C(v)$, one can say that u is more central than v . Although many of the popular centrality metrics are actually originated from the classical analysis of *social networks*, now they have successfully been investigated on many other practical networks, e.g., Internet (Page et al. 1998, Gkorou et al. 2011, Ou & Li 2011, Kleinberg et al. 1999), public transport networks (Kazerani & Winter 2009), power grid network (Jin et al. 2010), biological networks (Potapov et al. 2005, Bader & Madduri 2008), etc. A brief review of the existing centrality metrics and their applications can be found in (Junker & Schreiber 2011, Newman 2010). The main problem with many of the metrics is that their sequential implementations can often become very time consuming. For instance, the betweenness centrality computation of all nodes in a graph requires $\mathcal{O}(n^3)$ time with *Floyd-Warshall* algorithm, so for network with 1M nodes, a sequential method may take decades of computation on a general purpose computer. Even though there exists some faster approximate methods (Jacob et al. 2005, Eppstein & Wang 2001), their high error rates on larger networks can severely limit their applicability (Jia et al. 2008).

One feasible way to compute the centrality of such large-scale networks would be to parallelize the computation and interestingly, a number of parallelization approaches for such purpose have already been developed. Some of them are quite fast and scalable, but unfortunately, require highly sophisticated and expensive computer systems with parallel processing capabilities. For instance, Bader & Madduri (2006) implemented several parallel shortest path based metrics using shared memory multiprocessors on CRAY MTA-2. Madduri et al. (2009) presented a refinement of the same work by proposing a *lock-free* variant on CRAY- XMT system (Mizell & Maschhoff 2009). Jin et al. (2010) utilized the same system for computing the betweenness of power grid contingency measurements utilizing the same set of algorithms. Edmonds et al. (2010) presented a set of distributed memory algorithms for computing centralities using cluster computers with at least 100’s of compute nodes. Alternatively, there exist a few GPU implementations, which can be considered as relatively inexpensive approaches. However, a common problem with these implementations is their relatively lower scalability, when compared with the CPU based parallel approaches.

In this work, we present a scalable and fast method for computing a degree-based centrality metric, called eigenvector centrality. We apply it on gene-expression networks constructed from DNA microarray data sets. We use a GPU-based fast and scalable method for constructing the network. The proposed centrality computation method is an adaption of the classical *power iteration method* of computing eigenvector from a given matrix.

2 Literature Review

2.1 GP-GPUs and CUDA

The GPGPU is a powerful device that is devoted to parallel data processing rather than data caching and flow control as a general purpose CPU. Massive parallel processing capability of GPU makes it more attractive for algorithmic problem solving, where the processing of data (or a large block of data) can be handled in parallel. In general, the GPUs are organized in a streaming, data-parallel model in which the processors execute the same instructions on multiple data streams simultaneously. They are composed of a set of stream multi-processors (SM) with a certain number of stream processors (SP) each. At the software level, there exist several programming interfaces (e.g., CUDA, OpenCL, DirectCompute or the most recent innovation like OpenACC) that enable programmers to develop applications on GPUs. Among them, NVIDIA’s CUDA (Compute Unified

Copyright ©2013, Australian Computer Society, Inc. This paper appeared at the 11th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, South Australia, January-February 2013. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 140, Bahman Javadi and Saurabh Kumar Garg, Ed. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

Device Architecture) is one of the most widely used programming models that enable developing GPU-based applications using the C/C++ programming language. In CUDA, a parallel task is instantiated as a collection of *threads*, organized in *blocks* (a 1, 2 or 3- dimensional collection of threads, where a limited amount of shared memory is available to all the threads in a block), arranged in a *grid* (a 1 or 2-dimensional collection of blocks). A CUDA program typically consists of a host component that runs on the CPU, or host, and a smaller, but computationally intensive device component called *kernel*, that runs in parallel on the GPU. The kernel cannot access the main memory of the host directly; input data for the kernel must be copied to the GPU's on-board memory prior to its invocation, output data from the kernel must first be written to the GPU's memory and then copied back to the host CPU memory. For further details of GP-GPUs and CUDA technology we refer the reader to nVIDIA's "*CUDA Programming Guide*" (NVIDIA 2012).

2.2 Eigenvector Centrality Metric

Eigenvector centrality metric is a variant of degree centrality that measures relative influence of a node in a given network (Newman 2010). As we know, a degree-based centrality generally counts the number of neighbors of a vertex and decide the centrality accordingly (Figure 1). The main limitation of this centrality is that it only counts the number of neighbors, but does not consider the importance of the neighbors.

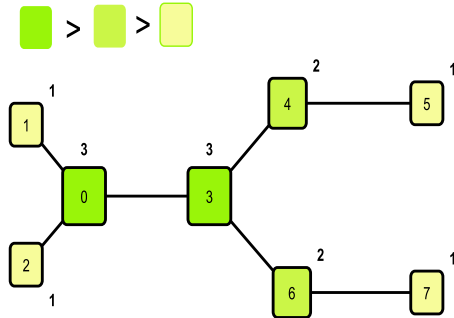


Figure 1: Degree centrality of the nodes in an undirected network.

For instance, the node 0 and 3 in Figure 1 have equal number of neighbors that gave them exactly same centrality on degrees. However, if we take a closer look, we can easily find that not all of their neighbors are equally connected to their neighbors, i.e., neighbors of neighbors of node 0 and 3. Therefore, we can predict that their centrality could be different in that context. To make this distinction, we need an extended degree-based metric that not only consider the neighbors but also the connectedness of the neighbors.

The degree-based centrality proposed by Katz (1953) is one of the first that idealised this neighborhood-based concept which is later adapted and improved by Hubbell (1965). Finally, Bonacich (1972) proposed a metric that assigns relative "scores" to all nodes, based on the concept that "*connections to high-scoring nodes contribute more to the score of the node in question than equal connections to low-scoring nodes*".

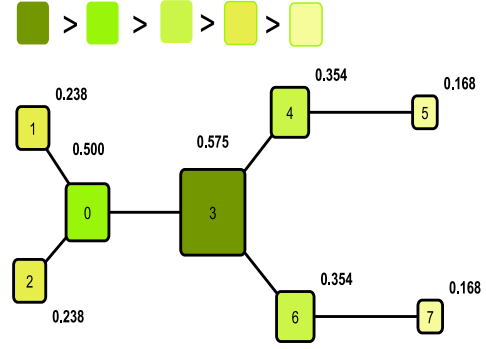


Figure 2: Eigenvector centrality of the nodes in the network shown in Figure 1.

The metric is popularly known as the *Bonacich's eigenvector centrality*, as it uses **eigenvector** and **principle eigenvalue** of the respective adjacency matrix. For the sample network presented in Figure 1, the eigenvector centralities are given in Figure 2, where the node 1 and 4 now have different values for centrality and hence $C(3) > C(0)$. The eigenvector centrality is one of the most important degree-based centrality, and a number of metrics are later derived from it. For instance, *Google's Page Rank* (Page et al. 1998) metric, is a widely known variant of the Eigenvector centrality. Formally, a non-zero vector \mathbf{e} is an eigenvector of a symmetric square matrix A , only if there is a scalar λ , such that

$$A\mathbf{e} = \lambda\mathbf{e} \quad (1)$$

For example,

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \text{ and } \mathbf{e} = \begin{bmatrix} 3 \\ -3 \end{bmatrix} \quad (2)$$

Here, \mathbf{e} , is an eigenvector with eigenvalue $\lambda = 1$, since,

$$A\mathbf{e} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 3 \\ -3 \end{bmatrix} = 1 \cdot \begin{bmatrix} 3 \\ -3 \end{bmatrix} \quad (3)$$

Now, for a given graph, $G = (V, E)$, if A is the respective adjacency matrix and $n \leftarrow |V|$, the eigenvector centrality (e_i) of a vertex $i \in \{1, \dots, n\}$ is obtained as follows (Bonacich 1972),

$$e_i = \frac{1}{\lambda} \sum_j a_{i,j} e_j \quad (4)$$

2.3 Related GPU-based Works

Even though a number of efforts have already been taken to parallelize the PageRank centrality on GPUs (Praveen et al. 2011, Wu et al. 2010, Cevahir et al. 2010), we found only one work, proposed by Sharma et al. (2011) that parallelizes the eigenvector centrality computation. The authors developed a CUDA-based approach that implements the centrality for NodeXL¹ (Hansen et al. 2010). Their method considers the input as an sparse graph and hence, use Compressed Row Storage (CRS) method² to store

¹NodeXL: Network Overview, Discovery and Exploration for Excel, <http://nodexl.codeplex.com/>

²By keeping the subsequent nonzero elements of the matrix rows in contiguous memory locations.

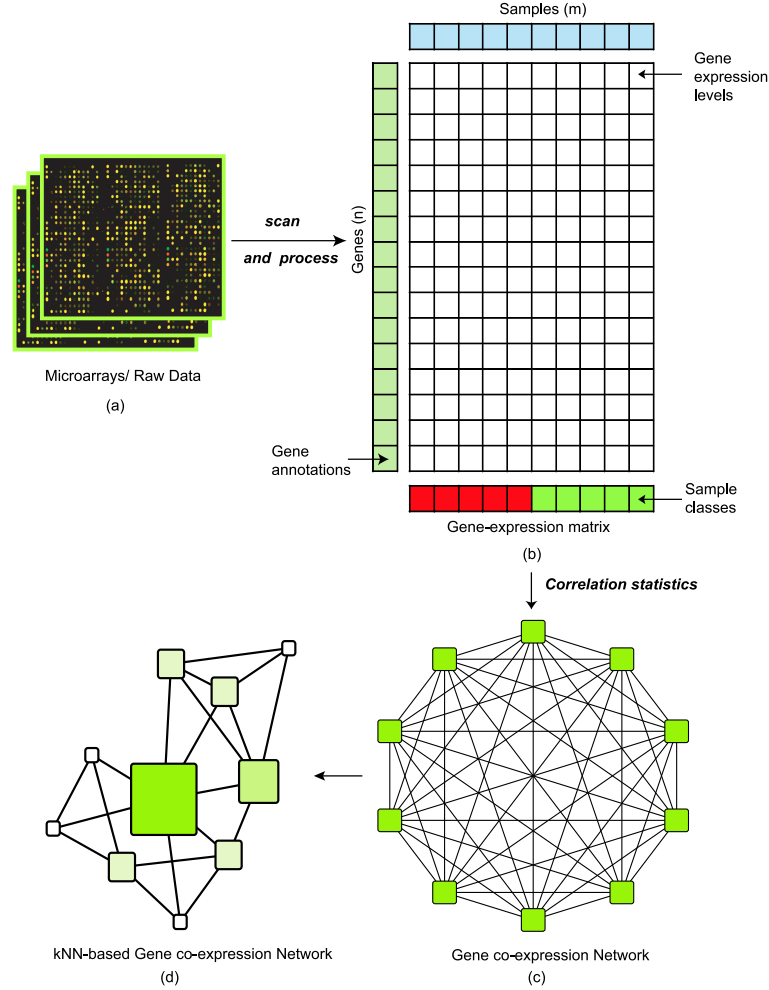


Figure 3: Identification of the node centrality in an undirected k NN-based gene-expression (gene co-expression) network. (a) Raw data are embedded in DNA microarray chips. (b) Arrays are scanned, processed and quantified into a $(n \times m)$ gene-expression matrix with n genes and m samples. (c) A fully connected gene co-expression network formed by taking pair-wise correlation of the gene-expressions in the above matrix (d) Analysis of centrality on the k NN graph extracted from the fully connected graph.

the adjacency matrix. They classified the computational problem related to eigenvector centrality as an “easy partitioning” problem (in contrast to shortest path-based centralities, which are classified as “hard to partition”) and proposed a data partitioning approach to make their implementation scalable on large instances. However, no results relevant to the data partitioning are provided in the relevant publication and furthermore, the implementation requires a number of tasks to be performed on the host CPU (e.g., vector norm, sum etc.). Therefore, the work can be recognised as a CPU-GPU hybrid approach.

3 Proposed Centrality Computation Method

The proposed centrality computation method is designed to identify central elements in DNA microarray gene-expression data sets. It works in two steps, first it constructs a k -nearest neighbor (k NN) graph from a given gene-expression data set. Next, it computes the centrality metric on that graph (Figure 3).

3.1 Construction of the k NN Graphs from Gene-expression Data Sets

From gene-expression data sets, the k NN graphs can be constructed in many ways, e.g., by using ex-

haustive search techniques, such as brute-force k NN search. However, the computation of the distances of the nearest neighbours for large-scale instances becomes very slow on general purpose computers. Fortunately, the nearest neighbours of each vertex can be computed and searched independently and hence, the brute force approach is highly parallelizable. This influenced us to develop a GPU based parallel implementation for tackling this problem. It can be noted that the most common problem with the existing GPU-based brute force k NN algorithms are two-fold, firstly, they can only work if all the distances between query and reference points, i.e., the distance matrix, can fit into GPU’s in-memory (e.g., see (Garcia et al. 2008)); secondly, they assume that the value of k is relatively small in comparison with the instance size (Liang et al. 2009). In contrast, we scaled and parallelized the simple brute force k NN algorithm and implemented a chunking-based approach called GPU-FS- k NN that can efficiently utilize GPUs and can handle instances with more than one million objects and fairly larger values of k (e.g., tested with k up to 64) on a single GPU. On multiple GPUs, if data partitioning is applied, then the method is capable of handling much larger instances and higher dimension sizes. Details of our GPU-based k NN graph construction method can be found in (Arefin et al. 2012a) and its other applications in (Arefin et al. 2012b).

3.2 Eigenvector Centrality of k NN Graphs

Our approach for computing eigenvector centrality works in two steps. First we construct a memory efficient bit-wise adjacency matrix of the k NN graph and then apply our data-parallel variant of a classical eigenvector computation method called, power iteration method.

3.2.1 Power Iteration Method

The *power iteration method* proposed by Hotelling (1936) is one of the many known algorithms that can be applied to obtain Eigenvector associated with a given adjacency matrix. The basic sequential method is presented in Algorithm 1.

Algorithm 1: Power Method

Input : A, n ;
Output: Eigenvector centrality of each vertex stored in e array;

```

1 Initialize  $e_i, \forall i \in n$ ;
2 repeat
3    $x_i = \sum_j A_{i,j} e_j, \forall i \in n$ ;
4    $\lambda \leftarrow ||x||$ ; // Euclidean norm
5    $e_i \leftarrow x_i / \lambda, \forall i \in n$ ;
6 until convergence criteria is met ;
```

The Power Method algorithm is simple to implement. Its convergence is slow except for certain special cases of matrices. To adapt the power method on GPUs, we create a set of CUDA kernels that can be called one after another to perform different steps of the method.

Names	Descriptions
Gk	The k NN graph (input graph)
e	An array to hold the eigenvector centrality of each node
x	An array to initialize the eigenvector
λ	Eigenvalue
λ_{max}	Principle eigenvalue
ϵ	A small constant to normalize the principle eigenvalue

Table 1: Variables and arrays used in the eigenvector centrality.

3.3 Bitwise Construction of the Adjacency Matrix

The k NN graph (Gk) is stored as an array of edge structure that has three members (**source**, **target** and **weight**) (other variables are given in Table 1). Unlike Sharma et al. (2011), we assume the input (i.e., k NN graph) can either be sparse or relatively dense, therefore we apply a different approach for storing the respective adjacency matrix. We use a memory-efficient representation of the “bit matrix” that significantly reduces the *space-complexity* of the algorithm. Furthermore, the bitwise operations (e.g., OR, AND, NOT) are very fast primitive actions that are directly supported by the hardware. Let us assume that we have a simple network of two vertices, as shown in Figure 4, where node 0 and 1 represents the source and target respectively. One can easily construct the respective (2×2) adjacency matrix from it (Equation

5) using a total of $4 \times 32 = 128$ bits of storage, considering each integer requires 32 bits (However, on 64-bit machines, each integer requires 64 bits of storage).

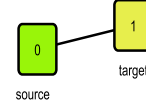


Figure 4: An illustrative network with two vertices.

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad (5)$$

Interestingly, the same matrix can simply be accommodated in a single `int` using only the first 4 out of its 32 bits. To achieve this, we designed two *macros* (Figure 5(a)) such that given the value of *source* and *target* of an edge, a bitwise OR operation between these two can set the respective bit in A , where A represented as a single dimensional integer array of size $((n^2 - 1)/32 + 1)$. An illustrative bit adjacency matrix is presented in Figure 5(b) and the bit adjacency matrix construction method (for k NN graphs) is presented in Algorithm 2.

Algorithm 2: Adjacency Matrix kernel

Input : Gk, A, n ;
Output: Adjacency matrix A initialized by Gk ;

```

1  $tid \leftarrow$  thread id;
2 if  $tid < n$  then
3    $source \leftarrow Gk[tid].source$ ;
4    $target \leftarrow Gk[tid].target$ ;
5    $u \leftarrow source \times n + target$ ;
6    $v \leftarrow target \times n + source$ ;
7   //  $A[source][target]=1$  ;
8   atomicOr ( $A+BIT\_POS(u)$ ,  $BIT(v)$ );
9   //  $A[target][source]=1$  ;
10  atomicOr ( $A+BIT\_POS(v)$ ,  $BIT(u)$ )
```

3.4 Data-Parallel Vector Operations

The adjacency matrix A can only have 0 or 1 as its entries, therefore the line 3 of the Power Method (Algorithm 1), can simply be implemented as a (conditional) vector addition, (i.e., **If** $A_{i,j} = 1$ **then** $x_i = x_i + e_j, \forall i, j \in n$) (Algorithm 3).

Algorithm 3: Add Kernel

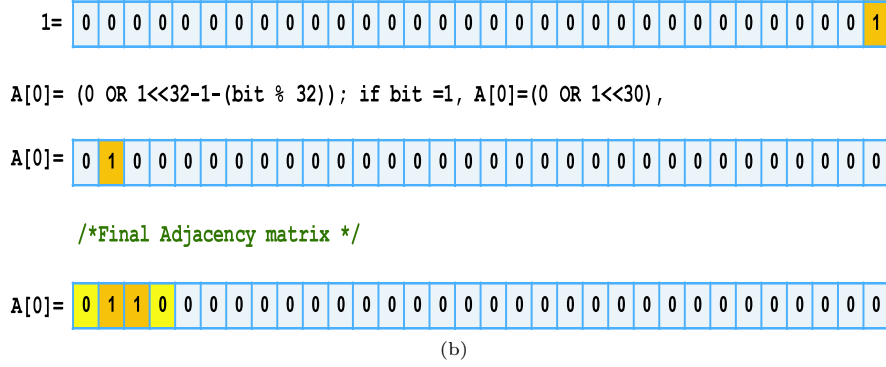
Input : x, A, e, n ;
Output: Vector x computed as $x_i += x_i + A_{i,j} \times e_j, \forall i, j \in n$.

```

1  $tid \leftarrow$  thread id;
2 if  $tid < n$  then
3    $x[tid] \leftarrow 0$ ;
4   for  $j \leftarrow 0$  to  $n - 1$  do
5      $bit \leftarrow tid \times n + j$ ;
6     if  $A[BIT\_POS(bit)] \ \& \ BIT(bit) = 1$  then
7       // if  $A[i,j]=1$  then
8        $x[tid] \leftarrow x[tid] + e[j]$ ;
```

```
# define BIT_POS (bit) ((bit) / 32 ) /* position of the bit */
# define BIT (bit) ( 1 << ( 32 - 1 - ((bit) % 32 ) ) ) /* shifted bit */
```

(a)



(b)

Figure 5: Bitwise construction of the adjacency matrix for the network shown in Figure 4.

Please note the line 6 of the Algorithm 3, where the bit defining macros (Figure 5(a)) are re-used (now with a bitwise $\&$) to investigate the content of the location defined by *bit*. To perform the line 4 in the Power method (Algorithm 1), we need to compute an Euclidean norm, which is also known by ℓ^2 -norm or distance. Given a vector $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$, the norm is computed as, $\|\mathbf{x}\| = \sqrt{\mathbf{x} \cdot \mathbf{x}} = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}$, which can be implemented as a **scalar dot product**. We implement it as a simple kernel using an **atomic operation**, as shown in Algorithm 4. Once, the product (*dot*) is computed, we get the $\lambda = \text{sqrt}(\text{dot})$.

Algorithm 4: Dot kernel

Input : x, y, dot, n ;
Output: Scalar dot product of the vectors x and y , stored in *dot*;
 1 $\text{tid} \leftarrow \text{thread id}$;
 2 **if** $\text{tid} < n$ **then**
 3 $\text{tmp} \leftarrow x[\text{tid}] \times y[\text{tid}]$;
 4 **atomicAdd**(*dot*, *tmp*); // $\text{dot} = \text{dot} + \text{tmp}$

To normalize the eigenvector (line 5, Algorithm 1) we implement a **scalar multiplication kernel** (termed as “Normalize Kernel”), as shown in Algorithm 5.

Algorithm 5: Normalize Kernel

Input : x, y, λ, n ;
Output: Vector y normalized by λ stored in x ;
 1 $\text{tid} \leftarrow \text{thread id}$;
 2 **if** $\text{tid} < n$ **then**
 3 $x[\text{tid}] \leftarrow y[\text{tid}] \times 1/\lambda$;

In addition to the eigenvector computation, we need to define a *convergence criteria* to stop the power iteration loop (line 2 - line 6). Thus, we need to determine how much the eigenvector changed from the previous iteration.

To do this, we implemented a variant of the vector addition (Algorithm 6) termed as “Substract Kernel”

that enables us to find the respective change using the principle eigenvalue.

Algorithm 6: Substract Kernel

Input : e, x, λ_{\max}, n ;
Output: Vector y normalized λ stored in x ;
 1 $\text{tid} \leftarrow \text{thread id}$;
 2 **if** $\text{tid} < n$ **then**
 3 $e[\text{tid}] \leftarrow e[\text{tid}] - \lambda_{\max} \times x[\text{tid}]$;

3.5 Eigenvector centrality on GPUs

By Perron-Frobenius theorem (Ninio 1976) we know that for the adjacency matrix A (which is a non-negative square matrix), there exists a largest unique eigenvalue λ_{\max} such that

1. λ_{\max} is positive;
2. λ_{\max} has a unique eigenvector, with all positive entries;
3. λ_{\max} is non-degenerate; and,
4. $\lambda_{\max} > \lambda$ for any eigenvalue $\lambda_{\max} \neq \lambda$.

Our goal is to apply the fourth condition ($\lambda_{\max} > \lambda$) as a stopping (or convergence) criteria for the Algorithm 1 (Power Method).

To do this, we slightly re-organise the original steps as shown in Algorithm 7 (**Eigenvector on GPUs**). First, we create the adjacency matrix $A(Gk)$ and initialize a vector as, $\mathbf{x} = \{1, 2, 3, \dots, n\}$, where n is the number of nodes in Gk (line 1-3). These variables are passed to the power iteration loop. Since, a parallelization of this outer-loop is not possible, we parallelize the computational tasks inside the loop, using the previously developed kernels for vector operations.

Inside the loop, we first find the Euclidean norm of the vector \mathbf{x} (line 5-8, Algorithm 7) using the **Dot Kernel** (Algorithm 4) which is stored in λ and subsequently we normalize the vector by λ and store the result in a separate vector \mathbf{e} (line 9, Algorithm 7 and see also the **Normalize Kernel** (Algorithm 5)). Then

we, re-initialize the vector \mathbf{x} by performing the following addition, $x_i = x_i + A_{i,j}e_i$, $\forall i, j \in 0$ using the **Add Kernel** (Algorithm 3). At this stage, we get the first eigenvector and so, before performing the matrix vector multiplication over and over again, we define a stopping criteria as follows,

$$\delta \times |\lambda_{max}| > \lambda \quad (6)$$

where, λ_{max} is the principle eigenvalue λ_{max} (computed in line 12 – line 13, Algorithm 7), ϵ is a small constant (e.g., $\epsilon \leftarrow 1 \times 10^{-6}$) that normalizes the principle eigenvalue (λ_{max}) and $\lambda = \|\mathbf{e}\|$ (line 15 – line 18, Algorithm 7).

When the convergence criteria is met, we derive the final eigenvector centrality of each node in vector \mathbf{e} as, $\mathbf{e} \leftarrow \mathbf{x}/\|\mathbf{x}\|$ (line 21-line 25 in Algorithm 7, see also line 5 in Algorithm 1 (Power Method)).

Algorithm 7: Eigenvector on GPUs

```

Input :  $Gk, \delta, n$ ;
Output: Eigenvector centrality of each
          vertex stored in  $\mathbf{e}$  array;
1 Initialize  $x[i] \leftarrow i, \forall i \in n$ ;
2 Remove Self-Cycles( $Gk, k, n$ );
3 Create Adjacency Matrix( $Gk, A, n$ );
4 repeat
5    $dot \leftarrow 0$ ;
6   device  $\rightarrow$  host ( $dot$ );
7   Dot ( $x, x, dot, n$ );
8    $\lambda \leftarrow \text{sqrt}(dot)$ ;
9   Normalize ( $e, x, \lambda, n$ );
10  Add ( $x, A, e, n$ );
11  /* Set-up the convergence criteria*/
12   $\lambda_{max} \leftarrow 0$ ;
13  Dot ( $x, e, \lambda_{max}, n$ );
14  Subtract ( $e, x, \lambda_{max}, n$ );
15   $dot \leftarrow 0$ ;
16  Dot( $e, dot, n$ );
17  device  $\rightarrow$  host ( $dot$ );
18   $\lambda \leftarrow \text{sqrt}(dot)$ ;
19 until  $\epsilon \times |\lambda_{max}| > \lambda$  ;
20 /*Finalize the Eigenvector*/
21  $dot \leftarrow 0$ ;
22 Dot( $x, dot, n$ ) ;
23 device  $\rightarrow$  host ( $dot$ );
24  $\lambda \leftarrow \text{sqrt}(dot)$ ;
25 Normalize ( $e, x, \lambda, n$ );
26 return  $e$ ;
```

4 Performance Evaluation

We evaluate the performance of our implementation on the following hardware setup: 4×NVIDIA Tesla C2050 GPU cards are installed on a X8DTG-Q Supermicro server that has 2 × Intel Xeon E5620 2.4GHz processors, 32GB of 1066 MHz DDR3 RAM and 800GB of Local Hard Disk.

The programs are written in C++ and CUDA (toolkit 4.0) and compiled using the g++ v4.4.4 and nvcc compilers on a Linux x86_64 version 2.6.9. The computational times are measured using CUDA timer utility (NVIDIA 2012).

4.1 Experimental Data Sets

We have utilized two categories of data sets in this evaluation process: synthetic and a real world DNA microarray data set. A brief description of these data sets are given below. Additionally, descriptions of the respective k NN graphs produced using GPU-FS- k NN (see Section 3.1) are given in Table 2.

Synthetic Data Sets. Synthetic data sets are created by randomly drawing points from a normal distribution $\mathcal{N}(0, 1)$ and generated utilizing the following function: `In = randn(n,m)` in MATLAB (v. 7.11.0), where n is the number of rows and m is the number of columns.

DNA Microarray Data Set. We used a renowned breast cancer gene-expression study data set provided by van de Vijver et al. (2002). The original data set is available at <http://bioinformatics.nki.nl/data.php>, and has a total of 24,479 biological oligonucleotides and 1,281 control probes in 295 breast cancer patients. For this experiment we utilized the published log ratio of a total of 24,158 probe sets (mainly targeting genes) for all the 295 samples (i.e., our data set had 24,158 rows and 295 columns).

Data set	#Vertices	k	#Edges
Synthetic	5,000 to 100,000	20	100,000 to 2,000,000
Breast Cancer	24,158	10	241,580
		15	362,370
		20	483,160
		25	603,950
		35	845,530

Table 2: Summary of the k NN graphs used in the experiments.

4.2 A Simple Case Study

Before proceeding to the larger data sets, we applied the proposed method to a toy data set for evaluating the feasibility of k NN graph as a prospective input. The toy instance i.e., the 16 Indo-European Languages data set is a simple 16×16 symmetric distance matrix which is extracted from 84 Indo-European Languages data set in (Dyen et al. 1992). Therefore, it is not possible to rank the nodes by any centrality metrics. One possible way could be to construct an MST and then compute the centrality (Figure 6).

Even though there exist certain applications of centralities metrics and their derivatives on the MST (e.g., (Correa et al. 2009)), on large-scale they may lead to significant information loss because of the sparsity of MST structure. On the other hand, a k NN graph contains more proximity information than the respective MST, therefore has more possibilities to give an accurate centrality estimation.

However, one potential problem is to determine “which value of k can give more accurate centrality ranking of the comprising nodes?”. Unfortunately, it is impossible to find such k .

Comparing the Figures 6 and 7, we can see that both the approaches at least identified a single node (i.e., **Italian**) as the most central one. The centralities increased with the value of k , however, such increments generally are proportional to the respective centrality of the nodes. We suggest to use the lowest k such that the network remains connected.

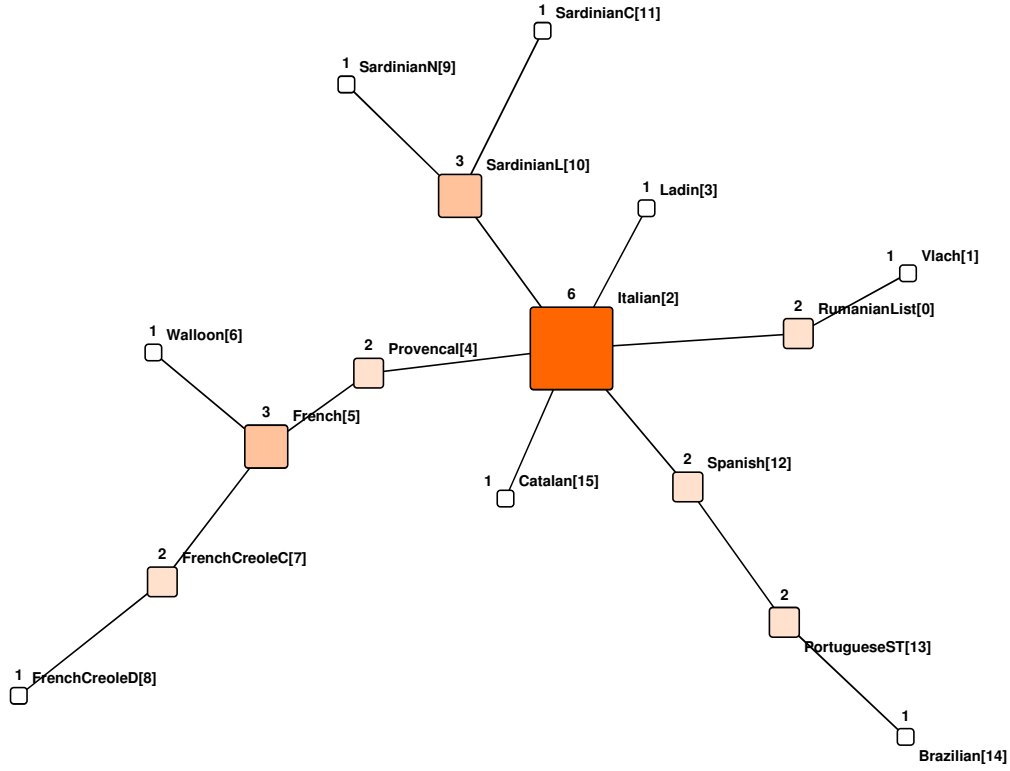
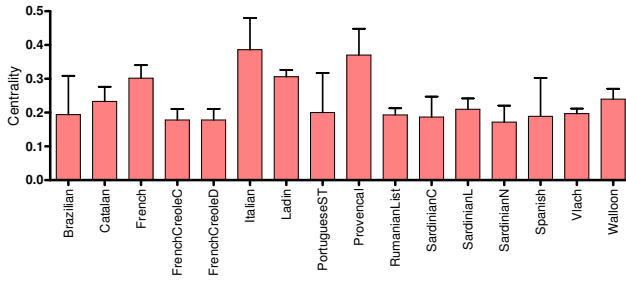


Figure 6: MST-based degree centrality computed from the 16 Indo-European Languages matrix.


 Figure 7: Average eigenvector centralities of k NN graphs created using $k = 2,4,6,8$ and 10 .

4.3 Speed-Up Gains

We applied the proposed GPU implementations to a set of k NN graphs, created using six synthetic data sets. The GPU computation times are given in Table 3.

#Vertices(n)	Times (Seconds)
5000	0.76
10,000	0.78
25,000	11.74
50,000	25.89
75,000	64.12
100,000	115.48

 Table 3: GPU computation times for the k NN graphs derived from the synthetic data sets (for a fixed value of $k = 20$).

The results in Figure 8 show that an increase in the data size, also increases the speed-up gains on GPUs. Higher arithmetic intensity improves the utilization of parallel hardware. However, we were unable to gain more than ten-times speed-ups, since the convergence of the power method is slow (Abraham 1974).

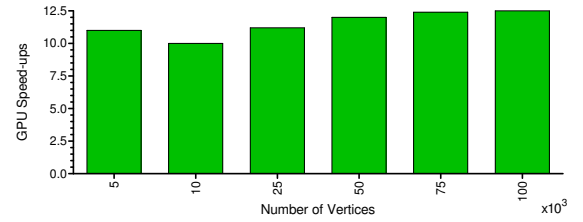
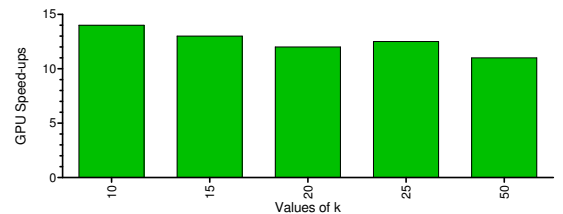


Figure 8: Speed-ups gained by the proposed GPU-based eigenvector centrality computation method over a standard CPU-based implementation of the Power method.

Next, we applied the proposed implementation to five different k NN graphs, created from the breast cancer study data set provided in (van de Vijver et al. 2002), by varying the value of k , i.e., for $k = 10,15,20,25$ and 50 . The main goal of this set of experiments is to investigate the change in speed-ups due to the change in the value of k . The respective results are presented in Figure 9.


 Figure 9: Variation of the GPU speed-ups due to the change in k (for a fixed value of $n = 25,158$).

The results in Figure 9 show that an increase in the value of k causes slightly decrease the speed-up gains (e.g., for $k=50$, the speed-up gain is decreased by at least 2 times). It means that denser graphs, can decrease the speed-ups.

5 Conclusion and Future Works

In this paper, we presented a GPU-based implementation of a popular centrality metric, called eigenvector centrality. The proposed method is an adaptation of classical power iteration method and unlike Sharma et al. (2011), it is complete GPU-based. In spite of the fact that power method converges slowly, our implementation showed at least ten times speed-ups over standard sequential implementation of classical power method. We have chosen the k NN graphs as input, as our main goal is to identify central elements in a given multi-variate data set (e.g., microarrays, time series data etc.). Generally, it is not possible to make any distinctions or rankings among the elements in a multi-variate data set if they are given in the form of a distance matrix or complete graph. As a solution, we proposed to use the k NN graphs (in contrast to the respective MST), as they contain the most important proximity relations. Our implementation is very generic and can be applied to other graph formats by accommodating simple modifications. The proposed method is tested using a single GPU device and with graphs having hundred thousands of nodes. A multi-GPUs implementation of the same method can further increase its scalability and speed-ups. In near future, we plan to develop a distributed framework for computing the eigenvector centrality metric. Moreover, we plan include other metrics (e.g., betweenness, random walk) in the same framework.

6 References

References

- Abraham, Z. (1974), 'Rate of growth and convergence factors for power methods of limitation', *Mathematical Proceedings of the Cambridge Philosophical Society* (76), 241–246.
- Arefin, A. S., Riveros, C., Berretta, R. & Moscato, P. (2012a), 'GPU-FS- k NN: A software tool for fast and scalable k NN computation using GPUs', *PLoS ONE*, **7**(8): e44000. doi:10.1371/journal.pone.0044000.
- Arefin, A. S., Riveros, C., Berretta, R. & Moscato, P. (2012b), k NN-Borivka-GPU: A fast and scalable MST construction from k NN graphs on GPU, in B. Murgante, O. Gervasi, S. Misra, N. Nedjah, A. M. A. C. Rocha, D. Taniar & B. O. Apduhan, eds, 'ICCSA (1)', Vol. 7333 of *Lecture Notes in Computer Science*, Springer, pp. 71–86.
- Bader, D. A. & Madduri, K. (2006), Parallel algorithms for evaluating centrality indices in real-world networks, in 'Proceedings of the 2006 International Conference on Parallel Processing', ICPP '06, IEEE Computer Society, Washington, DC, USA, pp. 539–550.
- Bader, D. A. & Madduri, K. (2008), 'A graph-theoretic analysis of the human protein-interaction network using multicore parallel algorithms', *Parallel Comput.* **34**(11), 627–639.
- Bonacich, P. (1972), 'Factoring and weighting approaches to status scores and clique identification', *Journal of Mathematical Sociology* **2**(1), 113–120.
- Cevahir, A., Aykanat, C., Turk, A., BarlaCam-bazoglu, B., Nukada, A. & Matsuoka, S. (2010), 'Efficient Pagerank on GPU Clusters', *IPSI SIG Notes* **2010**(21), 1–6.
- Correa, C. D., Crnovrsanin, T., Ma, K.-L. & Keeton, K. (2009), The derivatives of centrality and their applications in visualizing social networks, Technical report, University of California at Davis.
- Dyen, I., Kruskal, J. B. & Black, P. (1992), 'An indoeuropean classification: A lexicostatistical experiment', *Transactions of the American Philosophical Society* **82**(5), pp. iii–iv+1–132.
- Edmonds, N., Hoefer, T. & Lumsdaine, A. (2010), A Space-Efficient Parallel Algorithm for Computing Betweenness Centrality in Distributed Memory, in 'International Conference on High Performance Computing', pp. 1 – 10.
- Eppstein, D. & Wang, J. (2001), Fast approximation of centrality, in 'Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms', SODA '01, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, pp. 228–229.
- Garcia, V., Debreuve, E. & Barlaud, M. (2008), Fast k nearest neighbor search using GPU, in 'Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on', pp. 1–6.
- Gkorou, D., Pouwelse, J. & Epema, D. (2011), Betweenness centrality approximations for an internet deployed p2p reputation system, in 'Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum', IPDPSW '11, IEEE Computer Society, Washington, DC, USA, pp. 1627–1634.
- Hansen, D., Shneiderman, B. & Smith, M. A. (2010), *Analyzing Social Media Networks with NodeXL: Insights from a Connected World*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Hotelling, H. (1936), 'Simplified calculation of principal components', *Psychometrika* **1**(1), 27–35.
- Hubbell, C. H. (1965), 'An input-output approach to clique identification', *Sociometry* **28**(4), 377–399.
- Jacob, R., Kosch, D., Lehmann, K. A. & Peeters, L. (2005), 'Algorithms for centrality indices', *Network* **34**18, 62–82.
- Jia, Y., Hoberock, J., Garland, M. & Hart, J. C. (2008), 'On the visualization of social and other scale-free networks', *IEEE Trans. Vis. Comput. Graph.* **14**(6), 1285–1292.
- Jin, S., Huang, Z., Chen, Y., Chavarría-Miranda, D. G., Feo, J. & Wong, P. C. (2010), A novel application of parallel betweenness centrality to power grid contingency analysis, in 'IPDPS', IEEE, pp. 1–7.
- Junker, B., Koschutzki, D. & Schreiber, F. (2006), 'Exploration of biological network centralities with CentiBiN', *BMC Bioinformatics* **7**(1), 219+.
- Junker, B. & Schreiber, F., eds (2011), *Analysis of Biological Networks*, 1st edn, Wiley-Interscience, Secaucus, NJ, USA.
- Katz, L. (1953), 'A new status index derived from sociometric analysis', *Psychometrika* **18**(1), 39–43.
- Kazerani, A. & Winter, S. (2009), 'Can betweenness centrality explain traffic flow?', in 'Proceedings of the 12th AGILE International Conference', Springer, pp. 11–19.

- Kleinberg, J. M., Kumar, R., Raghavan, P., Rajagopalan, S. & Tomkins, A. (1999), The web as a graph: Measurements, models, and methods, in 'COCOON', pp. 1–17.
- Liang, S., Wang, C., Liu, Y. & Jian, L. (2009), Cuknn: A parallel implementation of k-nearest neighbor on cuda-enabled gpu, in 'Information, Computing and Telecommunication, 2009. YC-ICT '09. IEEE Youth Conference on', pp. 415–418.
- Madduri, K., Ediger, D., Jiang, K., Bader, D. A. & Chavarria-Miranda, D. G. (2009), A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets, in 'IPDPS', IEEE, pp. 1–8.
- Mizell, D. & Maschhoff, K. (2009), Early experiences with large-scale cray XMT systems, in 'Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing', IPDPS'09, IEEE Computer Society, Washington, DC, USA, pp. 1–9.
- Newman, M. (2010), *Networks: An Introduction*, Oxford University Press, Inc., New York, NY, USA.
- Ninio, F. (1976), 'A simple proof of the perron-frobenius theorem for positive symmetric matrices', *Analysis* **9**(8), 1259–1259.
- NVIDIA, C. (2012), *NVIDIA CUDA C Programming Guide (Version 4.2)*, NVIDIA Corporation.
- Ou, P. & Li, Z. (2011), 'A variant betweenness centrality approach towards distributed network monitoring', *Parallel Architectures, Algorithms and Programming, International Symposium on* **0**, 340–344.
- Page, L., Brin, S., Motwani, R. & Winograd, T. (1998), The PageRank citation ranking: Bringing order to the web, in 'Proceedings of the 7th International World Wide Web Conference', Brisbane, Australia, pp. 161–172.
- Potapov, A. P., Voss, N., Sasse, N. & Wingender, E. (2005), 'Topology of mammalian transcription networks.', *Genome informatics. International Conference on Genome Informatics* **16**(2), 270–278.
- Praveen, K., Vamshi, K. K., Anil, S. H. B., Balasubramanian, S. & Baruah, P. (2011), Cost efficient pagerank computation using GPU, in 'Proceedings of the 18th IEEE International Conference on High Performance Computing', HiPC' 11, IEEE Computer Society, Washington, DC, USA, pp. 81–89.
- Sharma, P., Khurana, U., Shneiderman, B., Scharrenbroich, M. & Locke, J. (2011), Speeding up network layout and centrality measures for social computing goals, in 'Proceedings of the 4th international conference on Social computing, behavioral-cultural modeling and prediction', SBP'11, Springer-Verlag, Berlin, Heidelberg, pp. 244–251.
- van de Vijver, M. J., He, Y. D., van't Veer, L. J., Dai, H., Hart, A. A. & et al. (2002), 'A gene-expression signature as a predictor of survival in breast cancer.', *The New England Journal of Medicine* **347**(25), 1999–2009.
- Wu, T., Wang, B., Shan, Y., Yan, F., Wang, Y. & Xu, N. (2010), Efficient PageRank and SpMV computation on AMD GPUs, in 'Proceedings of the 2010 39th International Conference on Parallel Processing', ICPP'10, IEEE Computer Society, Washington, DC, USA, pp. 81–89.

Non-blocking Parallel Subset Construction on Shared-memory Multicore Architectures

Hyewon Choi

Bernd Burgstaller

Department of Computer Science

Yonsei University

Seoul, Korea

xizsmin@yonsei.ac.kr, bburg@cs.yonsei.ac.kr

Abstract

We discuss ways to effectively parallelize the subset construction algorithm, which is used to convert non-deterministic finite automata (NFAs) to deterministic finite automata (DFAs). This conversion is at the heart of string pattern matching based on regular expressions and thus has many applications in text processing, compilers, scripting languages and web browsers, security and more recently also with DNA sequence analysis. We discuss sources of parallelism in the sequential algorithm and their profitability on shared-memory multicore architectures. Our NFA and DFA data-structures are designed to improve scalability and keep communication and synchronization overhead to a minimum. We present three different ways for synchronization; the performance of our non-blocking synchronization based on a compare-and-swap (CAS) primitive compares favorably to a lock-based approach. We consider structural NFA properties and their relationship to scalability on highly-parallel multicore architectures. We demonstrate the efficiency of our parallel subset construction algorithm through several benchmarks run on a 4-CPU (40 cores) node of the Intel Manycore Testing Lab. Achieved speedups are up to a factor of 32x with 40 cores.

Keywords: Subset construction, shared-memory multicore architectures, non-blocking synchronization, concurrent data-structures

1 Introduction

The subset construction algorithm converts an NFA to the corresponding DFA. Subset construction is frequently applied with string pattern matching based on regular expressions. A standard technique to match a regular expression on an input text is to convert the regular expression to an NFA using Thompson's construction, perform subset construction to derive a DFA, and minimize the DFA. The DFA is then run on the input text. If the DFA accepts its input, the input is known

to be matched by the regular expression. This method has been described by Aho et al. (1986).

With multicores becoming the predominant computer architecture, it is desirable to parallelize the subset construction algorithm. Although algorithms for DFA state minimization and DFA matching (discussed in Section 7) exist, to the best of our knowledge this is the first attempt to parallelize subset construction.

We parallelize subset construction for shared-memory multicore architectures. We analyze the different potential sources of parallelism contained in the sequential subset construction algorithm and compare their profitabilities. We devise an efficient parallel version of the subset construction algorithm, which guarantees load-balance and provides good scalability. We state the data-structures devised for the parallelization, which help improve the efficiency of the operations performed on shared-memory multicore architectures. To ensure correctness of the algorithm while not compromising on parallelism, we developed efficient synchronization methods. The performance of our non-blocking synchronization based on a CAS primitive compares favorably to a lock-based approach. We consider structural NFA properties and their relationship to scalability on highly-parallel multicore architectures. We demonstrate the efficiency of our parallel subset construction algorithm through several benchmarks run on a 4-CPU (40 cores) node of the Intel Manycore Testing Lab (accessed Aug. 2012).

This paper is organized as follows. In Section 2, we present sequential subset construction and related background material. In Section 3, we identify potential sources of parallelism and their profitability in the sequential subset construction algorithm. In Section 4 we introduce the algorithm's data-structures for supporting scalability on shared-memory multicore architectures. Section 5 discusses synchronization and the algorithm's termination condition. Section 6 provides our experimental results. We discuss the related work in Section 7 and draw our conclusions in Section 8.

2 Background

Let Σ denote a finite alphabet of characters and Σ^* denote the set of all strings over Σ . Cardinality $|\Sigma|$ denotes the number of characters in Σ . A language over Σ is any subset of Σ^* . The symbol \emptyset denotes

Copyright ©2013, Australian Computer Society, Inc. This paper appeared at the 11th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, South Australia, January-February 2013. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 140, Bahman Javadi and Saurabh Kumar Garg, Ed. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

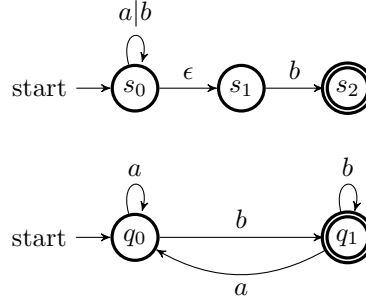


Figure 1: An example NFA (above), and its DFA converted through subset construction (below). DFA state q_0 represents NFA states s_0 and s_1 , while DFA state q_1 represents NFA states s_0 , s_1 and s_2 .

the empty language and the symbol ϵ denotes the null string. A finite automaton A is specified by a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \rightarrow 2^Q$ is a transition function, $q_0 \in Q$ is the start state and $F \subseteq Q$ is a set of final states. We define A to be a DFA if δ is a transition function of $Q \times \Sigma \rightarrow Q$ and $\delta(q, a)$ is a singleton set for any $q \in Q$ and $a \in \Sigma$. Otherwise, it is classified as an NFA. Let $|Q|$ be the number of states in Q . By the *density* of an automaton, we denote the ratio of the number of transitions in a given NFA to the number of transitions of a fully connected DFA of the same number of states and symbols (Leslie 1995). For a comprehensive background on automata theory we refer to (Hopcroft & Ullman 1979, Wood 1987).

2.1 Sequential Subset Construction

Algorithm 1 depicts the sequential subset construction algorithm from Hopcroft & Ullman (1979). Figure 1 depicts a sample NFA and the DFA computed by the subset construction algorithm. To begin with, we get the start state derived from s_0 of the NFA. i.e., we take $S_0 = \epsilon\text{-closure}(s_0)$ first. Then we compute $\epsilon\text{-closure}(\text{Move}(s_0, \sigma))$ for each $\sigma \in \Sigma$. If we get several states at once as a result of the computation, we make a set with them and treat it as a single DFA state. For a single DFA state T , we find all the states which can be reached by each $\sigma \in \Sigma$ from all the elements of T . Then we compute $\epsilon\text{-closures}$ for the results, and this creates a new DFA state. If this DFA state has not been appeared before, we add it to the DFA table. This process is iterated until no more DFA states are added.

To determine the time complexity for this algorithm, we consider the complexity of computing $\epsilon\text{-closure}(s_0)$ first. Because s_0 may have $(|Q| - 1)$ $\epsilon\text{-transitions}$, we conclude that computing $\epsilon\text{-closure}(s_0)$ takes $\mathcal{O}(|Q| - 1)$. Then the process without set equality test is computed with an

$$\mathcal{O}(|\Sigma| \times |Q| \times (|Q| - 1) \times (2^{|Q|} - 1))$$

time complexity. Now what we have to do is finding the factors which can be parallelized to reduce the complexity. We discuss the details in the following section.

Algorithm 1: Sequential Subset Construction

Input : NFA

Output: DFA states $Dstates$,
DFA transition function δ

- 1 $Dstates \leftarrow \{\}$;
 - 2 Add $\epsilon\text{-closure}(s_0)$ as an unmarked DFA state to $Dstates$;
 - 3 **while** there is an unmarked state T in $Dstates$ **do**
 - 4 mark T ;
 - 5 **for each** $\sigma \in \Sigma$ **do**
 - 6 $U \leftarrow \epsilon\text{-closure}(\text{Move}(T, \sigma))$;
 - 7 **if** $U \notin Dstates$ **then**
 - 8 add T as an unmarked DFA state to $Dstates$;
 - 9 $\delta[T, \sigma] \leftarrow U$;
-

3 Potential Sources of Parallelism With Sequential Subset Construction

In this section we identify sources of parallelism and their profitability with the sequential subset construction algorithm.

Source 1: The first opportunity for parallelization is on symbols $\sigma \in \Sigma$ (line 5 in Algorithm 1). Hence, this method is a task parallelization. After checking if there exists an unmarked state T in $Dstates$, what we have to do is taking its $\epsilon\text{-closure}(\text{Move}(T, \sigma))$ for each $\sigma \in \Sigma$, which is described in line 6 to line 9 of Algorithm 1. With the sequential version, this part must be computed $|\Sigma|$ times, i.e., for each $\sigma \in \Sigma$. However, because there is no dependency upon symbols, we may partition the symbols in Σ among the available processors to be processed in parallel. Thus, the time complexity becomes

$$\mathcal{O}\left(\left\lceil \frac{|\Sigma|}{p} \right\rceil \times |Q| \times (|Q| - 1) \times (2^{|Q|} - 1)\right).$$

Source 2: The second opportunity is parallelizing the outer while-loop (line 3) of the sequential algorithm. Until the algorithm terminates, $Dstates$ has at least one DFA state at the beginning of every iteration. Thus, for every iteration we partition the states in $Dstates$ among processors to parallelize the algorithm. This requires each processor to deal with the steps from line 4 to 9. A step for counting the number of unmarked states in $Dstates$

must be added right after entering the while loop. The result is used for determining the number of to-be-processed DFA states for each processor. For marking that follows after distribution, we should allow all processors to access *Dstates*. Time complexity for this algorithm becomes

$$\mathcal{O}(|\Sigma| \times |Q| \times (|Q| - 1) \times \left\lceil \frac{2^{|Q|} - 1}{p} \right\rceil).$$

Source 3: To exploit the final source of parallelism, we split a DFA state across its contained NFA states right after marking. The processors take the work described in line 5 through 9 in Algorithm 1. It should be noted that now processors work with the elements of a single DFA state (i.e., NFA states), not several DFA states. The granularity of concurrent computations is thus smaller than with Source 2. The partitioning of NFA states of an unmarked DFA state is conducted after we mark the unmarked DFA state. The work from line 5 to 8 is done in exactly the same way than the sequential version. Adding the result to a DFA state however, can be done in two different ways: we may let each worker thread add its NFA states to the DFA state, or have a dedicated master collect NFA states found by the workers and add them to the DFA state. Now it follows that the time complexity becomes

$$\mathcal{O}(|\Sigma| \times \frac{|Q|}{p} \times (|Q| - 1) \times (2^{|Q|} - 1)).$$

3.1 Profitability of Each Parallelism Source

Unfortunately, not all the suggested methods are equally profitable. Throughout our evaluated benchmarks (see also Section 6), we have found that the second and third opportunities are less effective than the first one (parallelizing on symbols). This phenomenon is caused by the following drawbacks of those two methods. To get a noticeable performance improvement by parallelizing the *Dstates*, it should be guaranteed that the number of DFA states at a certain moment is large enough. However, this number changes dynamically and cannot be known in advance. Even worse, larger automata sizes do not guarantee a larger number of *Dstates* at each point in time. Thus, even for large NFAs, a substantial performance gain cannot be expected: this observation implies that this algorithm would not be very useful in real situations.

In the case of parallelizing the NFA states of a DFA state, we need to collect the results from all the worker threads to construct a complete DFA state. For this work, the amount of synchronization is too high (e.g., from using barrier-synchronization) which eventually caused severe performance degradation with our evaluations.

On the contrary, parallelizing the algorithm on symbols is advantageous for load balance: because the number of symbols is constant and known a priori, a static partitioning of work among worker threads can be computed.

3	1	4	5			...
2	0	3				...
4	0	1	2	4	8	...

Table 1: Thread-local store of DFA states. The union of all workers' DFA states constitutes the *Dstates* set (see line 1 of Algorithm 1).

4 Data Structures for Parallel Subset Construction

We attempted to minimize overhead due to communication and synchronization between worker threads. In general, we kept data thread-local as much as possible, except for the *Dstates* set, which must be updated by all worker threads to collect DFA states.

Each worker thread maintains a thread-local array of DFA states as depicted in Table 1. The union over all thread-local DFA states constitutes the elements of the *Dstates* set. Similar to the sequential, deterministic state ADT by Leslie (1995), we store DFA states as linear arrays of NFA state members. The first array element of a DFA state contains the number of NFA states contained in a DFA state. Subsequent array elements represent NFA states. This representation facilitates efficient comparison of two DFA states, which is required for the set membership test (line 7 of Algorithm 1). The comparison of two DFA states is depicted in Algorithm 2.

To create new DFA states, each worker thread maintains a one-element DFA state scratch-space. Potentially new DFA states are computed by the ϵ -closure(*Move*(*T*, σ))-term in line 6 of Algorithm 1. NFA states are stored in the order of their appearance during this step. Before performing the set membership test in line 7, we sort the potentially new DFA state's NFA states in ascending order and copy the DFA state to the worker's thread-local DFA state array.

The *Dstates* set is a shared data-structure represented as an array of pointers to thread-local DFA states. Adding a DFA state to the *Dstates* set reduces to a single pointer update, i.e., the first (lowest-index) empty entry of *Dstates* is set to point to the new DFA state. Pointers are padded up to the CPU's cache-line size to avoid false sharing of cache-lines (see, e.g., (Lin & Snyder 2008)) that would otherwise occur if worker threads update adjacent array elements.

Algorithm 2: DFA State Equality Test

Input : 1D-array *Dstate1*, *Dstate2*
Output: True (if identical), False otherwise
1 for $i = 0$ to *Dstate1*[0] **do**
2 if *Dstate1*[i] = *Dstate2*[i] **then**
3 return False;
4 return True;

The *Dstates* set needs to maintain the following properties:

- The whole *Dstates* array is accessible by all worker threads.
- For the DFA states pointed by an entry of *Dstates*, duplication (entering a duplicate DFA state into *Dstates*) is not allowed.

After computing a potentially new DFA state in its thread-local store, a worker needs to determine whether the DFA state is allowed to be added to *Dstates*. As the first step of this decision, the thread performs the set membership test. Once it confirms that no identical DFA state has been added yet, it updates the next empty slot in *Dstates*.

Because *Dstates* allows access from any worker at any time, the set membership test followed by the subsequent pointer update is subject to potential race conditions. We will discuss three synchronization methods in the following section.

5 Synchronization and Termination

We are facing two synchronization problems: how to avoid race conditions with workers entering DFA states in the *Dstates* array, and when to terminate workers.

5.1 Synchronizing *Dstates* Updates

The first synchronization issue is due to the adding of DFA states to *Dstates*. We suggest three ways to synchronize access to the *Dstates* array: using a coarse-grained lock, a fine-grained lock, and making the algorithm non-blocking by guarding access to the *Dstates* array through a CAS instruction. CAS compares a data item in memory with a previous value *A* and replaces it with a newly provided value *B*, only iff the data item has not been updated by another thread. I.e., the data item's value in memory is identical with *A* (Herlihy & Shavit 2008). We use mutexes from the Pthread library for all lock-based synchronization and GCC's intrinsics for CAS operations.

Using a coarse-grained lock is a naive way to protect *Dstates*. Once a worker finds a new DFA state, it tries to acquire the mutex which is protecting *Dstates*. If successful, it performs the set membership test to confirm that its DFA state is not a duplicate of an existing DFA state in *Dstates*. During the whole set membership test, no other worker is allowed to update *Dstates*. After the set membership test, no matter the addition has been allowed or rejected, the worker thread releases the mutex.

A fine-grained lock, compared to the coarse-grained one, helps reducing the waiting time for worker threads waiting for acquiring the *Dstates* lock. The major factor which distinguishes this method from the previous one is that this time, during the set membership test, the lock for *Dstates* needs not be held by a worker thread. Instead, right after the worker thread finishes the set membership test and determines that no identical state exists in *Dstates* so far, it acquires the *Dstates* lock, and begins the set membership test again, from the point where it has stopped the test before locking *Dstates* until the newest element in *Dstates*. This second set membership test is for compensating a potential race condition that workers face after finishing the set membership test and before locking *Dstates*. Within this short time period, another thread might add a DFA state to *Dstates*. If this happens, the empty entry found by the previous thread is taken and the set membership test must be continued.

The final synchronization method does not lock *Dstates*: instead, it makes *Dstates* a concurrent data structure (Shavit 2011) by employing a CAS instruction. First, this method goes through the set membership test without locking *Dstates* as done when using a fine-grained lock. This time, however, we do not protect *Dstates* even after the worker thread has finished the set membership test. Instead, after finding the first empty entry in the *Dstates* array, it attempts to add the DFA state by executing a CAS instruction.

Algorithm 3: Access to *Dstates* guarded by CAS instruction

```

1 while !CAS (Dstates entry[i], NULL, DFA
   state) do
2   if DFA State Equality Test (Dstates
   entry[i], DFA state) then
3     return False;
4   ++i;
5 // At this point:
6 // Dstates entry[i] = DFA state
7 return True;
```

Algorithm 3 shows the basic concept of the non-blocking implementation using CAS. This form of execution confirms that only if the *Dstates* entry is empty, then the DFA state will be added there. It should be noted that after a failed CAS instruction, the set membership test must be continued, similar to our fine-grained locking method. Because we never delete entries in the *Dstates* set, the ABA-problem (see, e.g., (Herlihy & Shavit 2008)) does not apply.

5.2 Terminating Condition

The second problem is to decide when to make worker threads terminate. As stated in Algorithm 1, subset construction is supposed to terminate when no more new DFA states are added to *Dstates*. This implies that we need to enable the worker threads to determine when it is guaranteed that no new DFA state will appear anymore. If a worker thread cannot find any new DFA state, it needs to observe other threads if any of them is still processing DFA states. If it turns out that all workers have processed all DFA states, then it is safe for a worker to terminate.

To maintain the status of each worker thread wrt. processing of DFA states in *Dstates*, we use one status array per worker. In a worker's status array *L*, a worker keeps track how far it progressed in processing the DFA states in *Dstates*. The *n*th entry of status array *L* represents a worker's status wrt. the *n*th DFA state in *Dstates*.

Entries in the status array can have three values: *Not_accessed*, *Processing* and *Finished*. *Not_accessed* denotes the status that the worker thread has not begun to process the DFA state: Following the expression used in Algorithm 1, this DFA state has not been marked by this worker thread yet. Right after a worker begins processing within the while loop stated in line 3 of Algorithm 1 with the *n*th DFA state, it marks *L*[*n*] as *Processing*: if it finishes the work, it sets *L*[*n*] to *Finished*.

Now let us assume that a worker thread finds there is no new DFA state in *Dstates* after pro-

cessing the n th DFA state, i.e., the $(n+1)$ th entry in the $Dstates$ array is empty. Then it observes the $L[n]$ status values of the other workers: if any of the workers has not set its $L[n]$ status value as *finished* yet, then the idle worker needs to wait for a new DFA state to appear in the $n+1$ slot of $Dstates$. However, once all workers have set $L[n]$ to *finished* and no DFA state appeared in slot $n+1$ of $Dstates$, it is safe for all workers to terminate. It should be noted that workers write the *Finished* flag to the status array *after* they have entered a new DFA state to the $Dstates$ array.

6 Experimental Results

We demonstrate that our implementation of the sequential subset construction shows reasonable performance, compared to related tools. We chose the Grail tool by Raymond & Wood (1995) as our yardstick. Grail is a formal language toolbox which already provides a sequential version of the subset construction algorithm.

To determine the scalability of our parallel subset construction algorithm, we have conducted experiments on a 4-CPU (40 cores) shared memory node of the Intel Manycore Testing Lab. The POSIX thread library (see, e.g., Butenhof (1997)) has been used for worker threads and mutexes. We compared the scalability of two groups of NFAs that differ in their density, to show that density affects scalability.

Finally, we conducted experiments for all three $Dstates$ synchronization mechanisms: using a coarse-grained lock, a fine-grained lock, and the non-blocking method. We compare the efficiency of each method and we discuss the obtained results.

All execution times have been determined by reading the elapsed clock ticks from the x86 timestamp counter register (Paoloni 2010).

6.1 Performance Comparison with Grail

To confirm that our data-structures are efficient even for the sequential case, we compared our subset construction implementation to Grail. We used version 3.0 of the Grail code from the Grail+ Project Web Site (retrieved Aug. 2012), and we revised it to compile with g++ version 4.1.2. The performance comparison of a sequential version of our subset construction implementation with Grail is depicted in Figure 2.

The y axis represents relative time spent: we set the spent time for computing a 20-symbol automaton as 1. Thus, if the time spent for computing automata with x symbols takes five times longer than that of the 20-symbol automaton, the spent time for computing x is noted as 5. The number of states has been fixed to 20. This experiment has been conducted on an Intel Xeon E5405 CPU running Linux. As the NFA size increases, the performance gap increases remarkably: this implies that our NFA and DFA representations are efficient even with large NFAs.

6.2 Automata Density vs. Scalability

For this experiment, we have classified the NFA samples based on their *density*. In particular, we

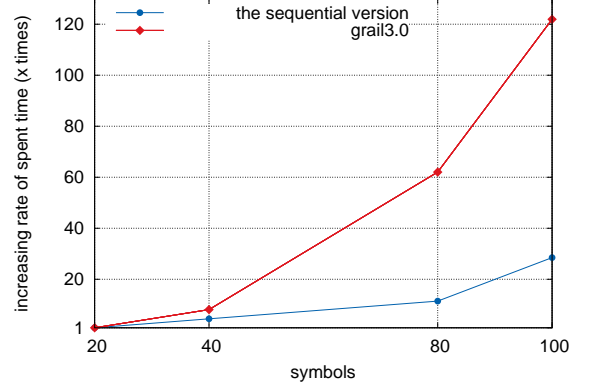
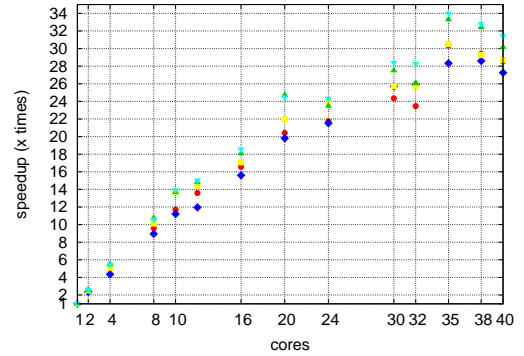
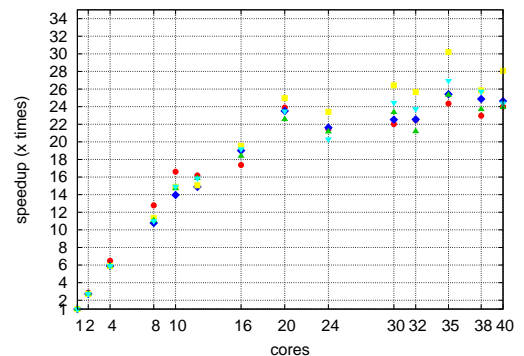


Figure 2: Performance comparison of proposed subset construction using custom NFA and DFA representations vs. the Grail tool.

considered two groups of density 0.3 and 0.4, respectively. We observed the speedups gained with both groups. As depicted in Figure 3, there is a clear gap in scalability between those two groups. We conjectured that this is related to properties of the DFAs converted from NFAs through our algorithm. In particular, we investigated the number and sizes of DFA states, as shown in Figure 5.



(a) NFAs with density 0.3



(b) NFAs with density 0.4

Figure 3: Scalability test with NFAs of density 0.3 and 0.4. NFAs from both groups have 20 states and 100 symbols.

To generate NFA samples, we have used an NFA random generator from Almeida et al. (2007). In this program, an automaton's density is used as a factor which determines the number of transitions. Let density be denoted by d , the number of states denoted by n , k denote the number

of symbols and t denote the temporary number of transitions. The NFA random generator computes the temporary number of transitions as

$$t = dn^2k - (n - 1).$$

The final number of transitions is determined by adding a small random factor to t . This formula suggests that if the number of states and symbols are not changed, as the density increases, the number of transitions also increases. As a result, each state gets more reachable states on average. From Algorithm 1, we can infer that if such an automaton undergoes subset construction, the average DFA state size would increase. Now we will show that this increment leads to a decrease in the number of DFA states. For an NFA, let Q denote the set of states. Then DFA states are subsets of Q . Let $n = |Q|$ and m the average DFA state size. Then we may approximate the number of possible DFA states by $\binom{n}{m}$.

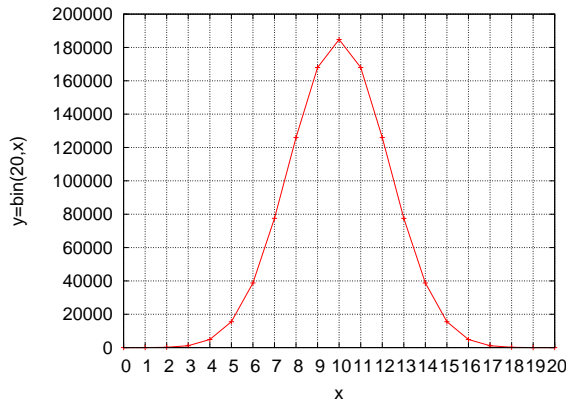


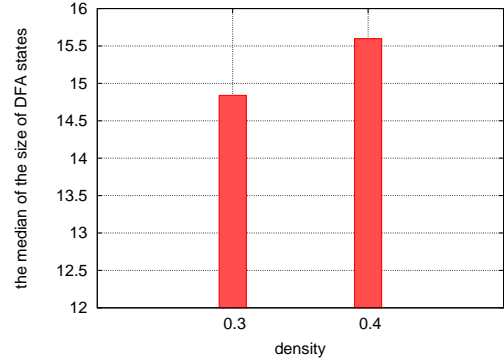
Figure 4: Number of possible DFA states $y = \binom{20}{x}$ for a 20-state NFA and $0 \leq x \leq 20$ NFA states per DFA state.

Figure 4 shows this tendency for n set to 20, and m ranging from 0 to n . As we may consider the x axis as the average size of the DFA states, we can infer that within a certain range of the DFA size, the number of DFA states would increase as the size increases, while the tendency could be reversed in some other range.

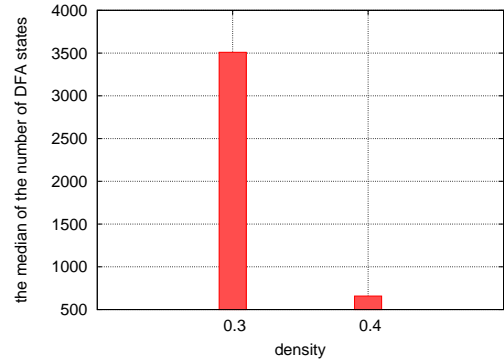
The DFAs from our experiment clearly follow this trend, as Figure 5 shows. To mitigate the effect of a few outliers, we have taken the median of the sample data instead of an arithmetic mean. The sizes and the number of DFA states collected from our NFA samples clearly match the approach suggested in Figure 4, which claims that between the average DFA state size of 14 to 16, the number of DFA states will decrease. This eventually reduces the amount of computation in the subset construction, which negatively affects the scalability as observed in Figure 3: compared to Figure 3(a), Figure 3(b) shows a clear performance degradation.

6.3 Comparing Scalability of the Three Synchronizing Methods

Figure 6 compares the scalability test results from three different versions of the parallelized algorithms: using a coarse-grained lock, a fine-grained



(a) the median of DFA state size



(b) the median of number of DFA states

Figure 5: Median over all DFA sizes and over the number of DFA states of the sample automata.

one, and the non-blocking mechanism. The graph clearly shows that using a coarse-grained lock causes severe serialization as the number of cores increases. With the coarse-grained lock, the set membership test is performed while holding the $Dstates$ lock. Thus, while a worker thread is doing the set membership test for a newly found DFA state, other threads who want to add their DFA states cannot proceed, which serializes the algorithm.

On the contrary, the parallelized algorithms which used a fine-grained lock and the non-blocking mechanism show good scalability. Both algorithms show a similar tendency, because their strategies to protect $Dstates$ are fundamentally similar to each other: they both let each worker thread perform the set membership test without locking $Dstates$, and once a worker finds an empty entry to add its DFA state, it begins the second test to confirm that it is safe to do the addition. What we need to focus here is that the most time-consuming part is the first set membership test, not the second one. Thus, this part affects on the whole processing time.

Both algorithms, i.e., using a fine-grained lock and the non-blocking mechanism, show an important phenomenon: Around 30 cores, the scalability becomes imperfect: this is due to the non-parallelized part of the algorithm, such as computing an epsilon-closure, the set equality test performed as a single step of the set membership test, etc. As the number of cores increases, time spent for the parallelized part decreases, but the non-parallelized part remains. In our experiment, this

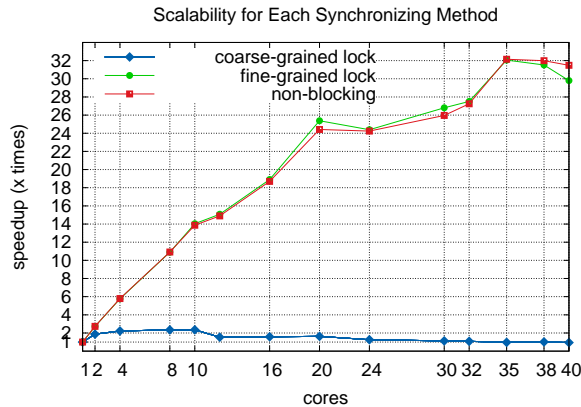


Figure 6: Scalability test result of the parallelized subset construction, with three synchronizing methods: using a coarse-grained lock, a fine grained lock and our non-blocking mechanism

limitation appears as the number of cores reaches around 30.

7 Related Work

Tewari et al. (2002) devised a parallel algorithm for DFA minimization. Approaches to parallelize DFA matching have been contributed by Luchaup et al. (2011), Wang et al. (2010), Holub & Štekr (2009), Jones et al. (2009), Luchaup et al. (2009), Scarpazza et al. (2007), Misra (2003), Hillis & Steele (1986), Ladner & Fischer (1980) and Ko et al. (2011). However, to the best of our knowledge this is the first attempt to parallelize subset construction.

Sequential subset construction has been described in the literature on automata theory by Hopcroft & Ullman (1979) and Aho et al. (1986). Leslie (1995) improves the efficiency of sequential subset construction through data structures for hashing, heaps and bitvectors. This implementation brings two major advantages: avoiding redundant checks for symbols, and taking the advantage of sorted transitions. Because their multi-way merging operation takes around 30% ~ 90% of the overall running time, techniques have been suggested to improve this operation. Representing the ADT for DFA states as a hash table is proposed. Leiss (1980) proposed a DFA construction method that reduces DFA size. Johnson & Wood (1996) discuss methods for instruction computation of DFAs. Liu et al. (2011) propose a method to construct a combined DFA for a set of regular expressions. They apply hierarchical merging of DFAs for individual regular expressions.

8 Conclusions

We have parallelized the subset construction algorithm, which is used to convert non-deterministic finite automata (NFAs) to deterministic finite automata (DFAs). We have discussed sources of parallelism in the sequential algorithm, and critically evaluated their profitability on shared-memory multicore architectures. Data-structures for NFAs

and DFAs have been chosen to improve scalability and keep communication and synchronization overhead to a minimum. Three different ways of synchronization have been implemented. The performance of our non-blocking synchronization based on a compare-and-swap (CAS) primitive compares favorably to a lock-based approach. We have shown that the amount of work and hence the scalability of parallel subset construction depends on the number of DFA states, which is related to automata density. We demonstrate the efficiency of our parallel subset construction algorithm through several benchmarks run on a 4-CPU (40 cores) node of the Intel Manycore Testing Lab. Achieved speedups are up to a factor of 32x with 40 cores.

9 Acknowledgements

Our research has been partially supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2010-0005234), and the OKAWA Foundation Research Grant (2009).

References

- Aho, A. V., Sethi, R. & Ullman, J. D. (1986), *Compilers: principles, techniques, and tools*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Almeida, M., Moreira, N. & Reis, R. (2007), On the performance of automata minimization algorithms, Technical Report DCC-2007-03, Departamento de Ciência de Computadores & Laboratório de Inteligência Artificial e Ciência de Computadores.
- Butenhof, D. R. (1997), *Programming with POSIX threads*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- Grail+ Project Web Site (retrieved Aug. 2012), ‘<http://www.csd.uwo.ca/Research/grail>’.
- Herlihy, M. & Shavit, N. (2008), *The Art of Multiprocessor Programming*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Hillis, W. D. & Steele, Jr., G. L. (1986), ‘Data parallel algorithms’, *Commun. ACM* **29**(12), 1170–1183.
- Holub, J. & Štekr, S. (2009), On parallel implementations of deterministic finite automata, in ‘Proceedings of the 14th International Conference on Implementation and Application of Automata’, pp. 54–64.
- Hopcroft, J. & Ullman, J. (1979), *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA.
- Intel Manycore Testing Lab (accessed Aug. 2012), ‘<http://software.intel.com/en-us/articles/intel-many-core-testing-lab>’.
- Johnson, J. & Wood, D. (1996), Instruction computation in subset construction, Technical report, Institute for Information Technology, National Research Council and Hong Kong University of Science and Technology.

- Jones, C. G., Liu, R., Meyerovich, L., Asanović, K. & Bodík, R. (2009), Parallelizing the web browser, *in* 'Proceedings of the First USENIX conference on Hot topics in parallelism', HotPar'09, USENIX Association, Berkeley, CA, USA, pp. 7–7.
- Ko, Y., Jung, M., Han, Y.-S. & Burgstaller, B. (2011), A speculative parallel DFA membership test for multicore, SIMD and cloud computing environments, Technical Report arXiv:1210.5093, Dept. Computer Science, Yonsei University, Seoul 120-749, Korea, <http://http://arxiv.org/abs/1210.5093>.
- Ladner, R. E. & Fischer, M. J. (1980), 'Parallel prefix computation', *J. ACM* **27**(4), 831–838.
- Leiss, E. (1980), 'Constructing a finite automaton for a given regular expression', *SIGACT News* **12**(3), 81–87.
- Leslie, T. (1995), Efficient approaches to subset construction, Master's thesis, University of Waterloo.
- Lin, C. & Snyder, L. (2008), *Principles of Parallel Programming*, Addison Wesley.
- Liu, Y., Guo, L., Guo, M. & Liu, P. (2011), Accelerating DFA construction by hierarchical merging, *in* 'Proceedings of the 2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications', ISPA '11, IEEE Computer Society, Washington, DC, USA, pp. 1–6.
- Luchaup, D., Smith, R., Estan, C. & Jha, S. (2009), Multi-byte regular expression matching with speculation, *in* 'Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection', RAID '09, Springer-Verlag, Berlin, Heidelberg, pp. 284–303.
- Luchaup, D., Smith, R., Estan, C. & Jha, S. (2011), 'Speculative parallel pattern matching', *IEEE Transactions on Information Forensics and Security* **6**(2), 438–451.
- Misra, J. (2003), 'Derivation of a parallel string matching algorithm', *Inf. Process. Lett.* **85**(5), 255–260.
- Paoloni, G. (2010), How to benchmark code execution times on Intel IA-32 and IA-64 instruction set architectures, Technical report, Intel Corporation.
- Raymond, D. & Wood, D. (1995), 'Grail: A C++ library for automata and expressions', *Journal of Symbolic Computation* **17**, 17–341.
- Scarpazza, D. P., Villa, O. & Petrini, F. (2007), Peak-performance DFA-based string matching on the Cell processor, *in* '21th International Parallel and Distributed Processing Symposium', pp. 1–8.
- Shavit, N. (2011), 'Data structures in the multi-core age', *Commun. ACM* **54**(3), 76–84.
- Tewari, A., Srivastava, U. & Gupta, P. (2002), A parallel DFA minimization algorithm, *in* 'Proceedings of the 9th International Conference on High Performance Computing', HiPC '02, Springer-Verlag, London, UK, UK, pp. 34–40.
- Wang, X., He, K. & Liu, B. (2010), Parallel architecture for high throughput DFA-based deep packet inspection, *in* '2010 IEEE International Conference on Communications', pp. 1–5.
- Wood, D. (1987), *Theory of Computation*, John Wiley & Sons, Inc., New York, NY.

Simseer and Bugwise - Web Services for Binary-level Software Similarity and Defect Detection

Silvio Cesare and Yang Xiang

School of Information Technology

Deakin University

Burwood, Victoria 3125, Australia

{scesare, yang}@deakin.edu.au

Abstract

Simseer and Bugwise are online web services that perform binary program analysis: 1) Simseer identifies similarity between submitted executables based on similarity in the control flow of each binary. A software similarity service provides benefit in identifying malware variants and families, discovering software theft, and revealing plagiarism of software programs. Simseer additionally performs code packing detection and automated unpacking of hidden code using application-level emulation. Finally, Simseer uses the similarity information from a sample set to identify program relationships and families through visualization of an evolutionary tree. 2) Bugwise is a service that identifies software bugs and defects. To achieve this end, it performs decompilation and data flow analysis. Bugwise can identify a subset of use-after-free bugs and has already found defects in Debian Linux. Bugwise and Simseer are both built on Malwise, a platform of binary analysis.

Keywords: computer security, software similarity, software theft detection, plagiarism detection, bug detection, cloud computing.

1 Introduction

Software similarity is an important topic with a number of applications. It can be used in the areas of malware detection, software theft detection and plagiarism detection. These are the applications for which Simseer was designed to address.

Software similarity analysis is built upon a platform of program analysis that performs the relevant aspects of feature extraction. This process of software analysis can be used not only for software similarity tasks, but also to detect software bugs and defects.

Defect Detection is the problem of finding software bugs. Examples of bugs that defect detection can identify are buffer overflows, divide-by-zeros, and dynamic memory management problems such as use-after-frees.

Malware variant detection is the problem of identifying malware that is a replicated, obfuscated, or an evolved copy of a known malicious sample. Malware

variant detection can be used to attribute a sample to a particular author or family of malware. Malware variant detection is the problem of identifying similarity between known malware and unknown programs.

Software theft detection identifies the unauthorized duplication or copying of software. The purpose of this area is to have automated ways to discover or verify copyright infringement of software or intellectual property. Software theft detection is the problem of identifying unauthorized similar software.

Plagiarism detection detects student cheating in assignments where the submission is a piece of software. Students copying each other's work can be broken down into the problem of identifying similar copies of software in the students' submissions

1.1 Motivation

Defect Detection can reduce the cost of maintaining software by identifying problems during quality and assurance testing and not after the public software release is made. Identifying software defects that impact on the security of software means that producers of software can stay ahead of attackers who actively try to discover these defects themselves. Bug detection in binaries is important to external auditors who need to validate the security of software they are given. Binary auditing is also important to verify the compiler and linker are working properly without introducing new defects.

Malware detection is an important problem on the internet today. According to the Symantec Internet Threat Report (Symantec 2008), 499,811 new malware samples were received in the second half of 2007. The same vendor reported over 1.5 billion malicious code detections in 2010 (Symantec 2011). F-Secure published, "As much malware [was] produced in 2007 as in the previous 20 years altogether" (F-Secure 2007). This growth continues today.

Malware variant detection can be used to enhance the traditional approach of signature based malware detection by providing more predictive power to those signatures. Most malware today is a variant of existing malware, so identifying variants is effective in detecting a significant amount of malicious code that traditional approaches fail to identify.

Software theft is a problem with significant consequences. In 2005, a federal court determined that the independent software vendor Compuserve be paid \$140 million by IBM to license its software or \$260 million to purchase its services because it was discovered that IBM products had illegitimately used code from

Compuware without authorization (Wang et al. 2009). Software theft detection is an important area that helps protect the high worth of intellectual property.

Plagiarism detection is important to maintain integrity in educational environments. If students believe they will be caught if they cheat then they are unlikely to proceed with that unethical practice. If educators receive a high number of assignment submissions then it may be hard to recognize that cheating has occurred, so automated methods are an important tool.

1.2 Innovation

Simseer is a tool that can detect similar software and identify malware variants, discover software theft, and reveal plagiarism. Bugwise can detect some classes of software defects in binaries. The contributions of this paper are as follows:

- We propose an online web service to address the issues of malware variant detection, software theft detection, and plagiarism detection.
- We propose an online web service to address the issue of closed source software defect analysis.
- We use state-of-the-art algorithms in our novel service.
- We implement and make public our services.

1.3 Structure of the Paper

The structure of this paper is as follows: Section 2 examines related work in software similarity and bug detection. Section 3 describes a high level overview of our aims and approach. Section 4 discusses the design and implementation of our system as a cloud service. Section 5 evaluates different aspects of our system. Section 6 gives details on how to access our service. Section 7 looks at future work. Finally, Section 8 concludes the paper.

2 Related Work

Detecting defects in software has a long history in formal methods. Data flow analysis is used by compilers (Aho, Sethi & Ullman 1986) and is what Bugwise uses to perform binary analysis. Abstract interpretation, which formalizes data flow analysis was introduced in (Cousot & Cousot 1977). Theorem proving has been used to prove the absence of bugs (Dijkstra 1975; Hoare 1969). Satisfiability over Modulo Theories (SMT) extends SAT and has been used to perform bug detection (Cadar et al. 2008; Molnar & Wagner 2007) and symbolic execution (King 1976). Decompilation has been used to analyse binary programs in the past including work in (Cifuentes 1994; Van Emmerik 2007) which used compilation techniques to aid the decompilation process.

The areas relating to software similarity are malware variant detection, software theft detection, plagiarism detection, and code clone detection. A unified approach to the software similarity problem is to divide the problem into feature extraction to construct fingerprints, known as birthmarks, and then to calculate birthmark similarity using mathematical distance and similarity functions. Birthmarks can be considered as strings, vectors, sets, trees, graphs and other objects.

In malware variant detection, raw code has been used to construct string based signatures, which is common in Antivirus software (Griffin et al. 2009; Kephart & Arnold 1994). Kolmogorov complexity of raw code has been used in (Wicherski 2009). The Normalized Compression Distance was used in (Wehner 2007). Opcode distributions are another feature used in (Bilar 2007). N-grams were used on instructions in (Karim et al. 2005) and evolutionary trees were constructed. Static and dynamic API call features were used in (Ye et al. 2007) and (Kolbitsch et al. 2009) respectively. Control flow and data flow were used as a feature in (Christodorescu & Jha 2003; Christodorescu et al. 2005).

Control flow is the approach that Simseer uses to construct birthmarks. Interprocedural control flow was proposed as a feature in (Briones & Gomez 2008; Carrera & Erdélyi 2004; Dullien & Rolles 2005; Gerald & Lori 2007; Hu, Chiueh & Shin). Simseer uses intraprocedural control flow of a program's procedures and similar techniques have been applied in (Cesare & Xiang 2010b) (Cesare & Xiang 2010a) (Bonfante, Kaczmarek & Marion 2008) (Kruegel et al. 2006).

In software theft detection, similar techniques have been used. Instruction sequences were used in (Park et al. 2008). K-grams of instruction sequences were used in (Myles & Collberg 2005). Control flow was used in (Lim et al. 2009a, 2009b) and static API calls used in (Choi et al. 2008, 2009).

In plagiarism detection systems such as JPlag (Prechelt, Malpohl & Philippsen 2002) and YAP3 (Wise 1996) have used the text of raw source code as a feature. Parse trees were used in (Son, Park & Park 2006) allowing tree based distances to calculate similarity. Program Dependence Graphs (PDGs) were used in (Liu et al. 2006).

Code clone techniques are based on the software similarity problem. It is the problem of identifying duplicate or similar fragments of code in a piece of software. Approaches have included using raw source code as a birthmark in (Ducasse, Rieger & Demeyer 1999) and for large scale applications in (Kamiya, Kusumoto & Inoue 2002; Livieri et al. 2007). Abstract Syntax Trees (ASTs) were used in (Baxter et al. 1998). PDGs were proposed in (Krinke 2001).

Birthmark similarity is the next step after feature extraction and birthmark creation. Distance metrics for strings, vectors, sets, trees, and graphs exist. For strings, the Levenshtein distance is the minimum number of insertions, deletions, and substitutions to transform one string to another. Sequence alignment is often used in bioinformatics including the optimal local sequence alignment, known as the Smith-Waterman algorithm. Vector distance metrics include the Manhattan distance or the classic Euclidean distance. Cosine similarity is a popular vector similarity measure. Set similarity includes the Dice Coefficient and the Jaccard Index. Trees and graphs have edit distances to describe the number of basic operations to transform one object to another. Maximum common subtrees or subgraphs are other measures used to identify similarity and distance.

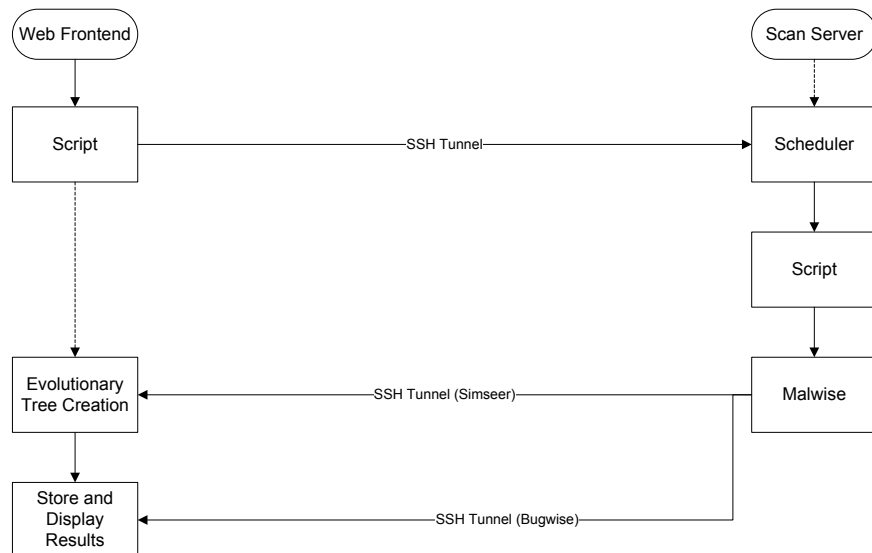


Figure 1. Web services work flow.

3 Our Approach

The aims of this work are to provide a web service to score and visualize similarity between executable binaries, and also to provide a web service to detect software defects in binaries. To perform software similarity scoring and defect detection, we employed some of our previous work, Malwise, to do the backend processing.

Malwise performs software similarity scoring by using control flow within a binary as a signature. Control flow is considered invariant under common program transformations and is effective at detecting program variants.

Malwise can also perform general static analysis of binaries. It does this by disassembling the binary, translating the disassembly to an intermediate language, and then performing decompilation, data flow, and other analyses. Data flow analysis combined with decompilation is capable at detecting some defects from some classes of bugs in binaries.

4 System Design and Implementation

The system uses two Virtual Private Servers (VPS) in the cloud and could potentially be scaled into larger server farms. One server is the web frontend and one server is the scan server. The servers have 1GB of memory each. The workflow for the web service involving all components is shown in Fig. 1. The user interface is a submission system that returns a results page.

4.1 The Web Frontend

Both Simseer and Bugwise are accessed by web frontends. Bugwise is almost functionally equivalent in its processing, so Simseer will be explained in depth.

The web frontend is the user interface to the Simseer cloud service and the landing page and the final result is shown in Fig. 2. And Fig. 3. A user of the service can

submit a ZIP archive of executables which are subsequently transferred to and processed by the server. Our implementation is coded in the server side PHP programming language. The PHP code is responsible for rate limiting the number of submission requests per IP address by maintaining a record of submissions to the system in a MySQL database.

The PHP code launches a shell script which takes over handling of the archive submission. The script checks that the ZIP archive is valid, does not contain an excessive number of samples, does not contain symbolic links as archive members, and does not contain archive member names using special characters.

The system logs that a submission to the system has been made and makes a copy of the submission content into storage. The script launches a C++ compiled program that acts as a client in a client-service protocol with the scan server. The protocol enables transmission of files that will be processed by the scan server. Communication with the scan server is performed over an SSH port forwarded tunnel which allows security in the client-server protocol.

4.2 The Scheduling Work Queue

The scan server listens locally on a TCP port which is connected via an SSH tunnel back to the web frontend. The C++ implemented server component launches the Malwise backend to process files received. However, scheduling must occur so that the server does not consume excessive resources. Thus receipt and processing of files is queued so that only 1 job is active at any given time. The number of parallel jobs can be arbitrary, however due to the single core nature of our Virtual Private Server (VPS) scan server, running jobs in parallel does not result in an increase in performance. Additionally, running multiple jobs in parallel places more restrictions on memory usage per instance which we wanted to avoid. Once a job has been scheduled and the ZIP archive or binary received from the web frontend host, a script is launched to process the file and launch the

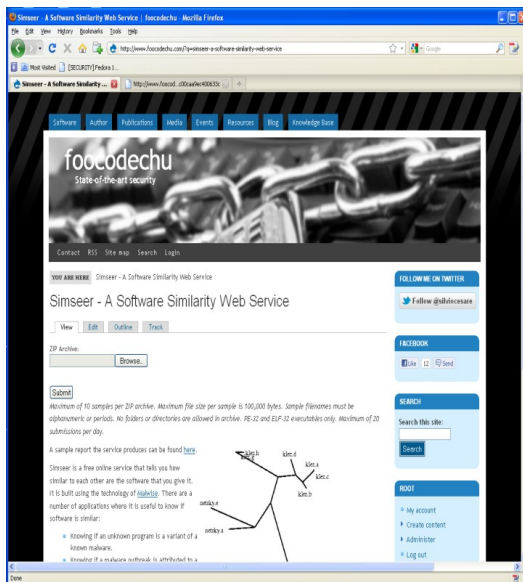


Figure 2. The Simseer landing page.

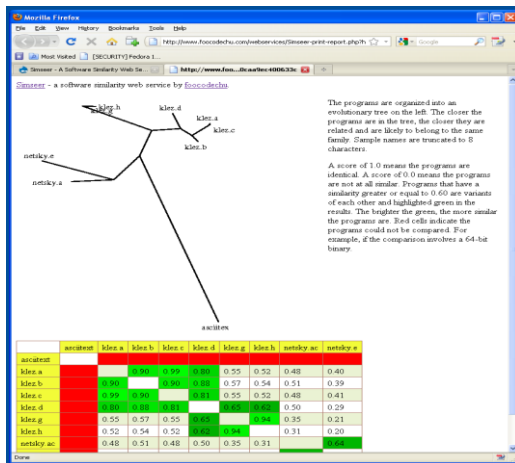


Figure 3. Simseer results.

Malwise system. For Simseer, the script unpacks the archive ready for Malwise to process. For Bugwise, the binary is passed on directly.

4.3 Malwise Backend

Simseer and Bugwise both use the Malwise backend. The difference between Simseer and Bugwise is the module list that Malwise uses. The Malwise backend is coded in C++ and consists of 100,000 Lines of Code (LOC). Malwise is launched as a standalone program from the scheduler launched script. It is possible to use Malwise as a daemon and avoid the cost of repeated program loading when submitting jobs. However, the reliability of the system as a whole is increased when we launch Malwise as a standalone program for each job because if a scan then causes a crash it is contained to an individual job. If Malwise was run as a daemon and allowed jobs to be queued then all jobs would be lost if the program failed. Even though we launch jobs separately, the service allows for scalability because jobs could potentially be launched on server farms behind the interface. Likewise,

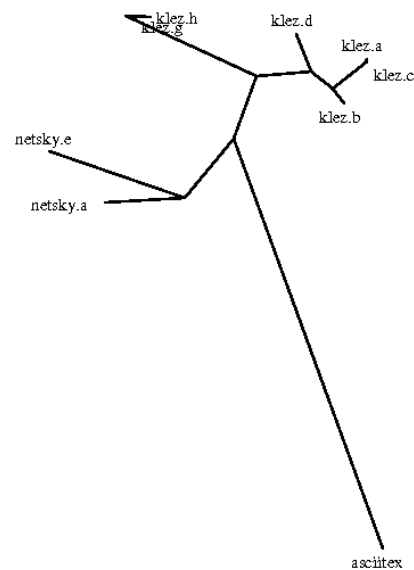


Figure 4. Program relationship visualization.

the system still maintains a global view of jobs being launched - it stores copies of the binaries submitted to the service. This allows us to perform offline analysis and correlation to determine if novel samples are being submitted to the service or if known samples are the primary source of submissions.

The backend is modular and allows for loading of modules at program startup defined by an XML configuration file. A sample of the differences between the configuration for Simseer and Bugwise is shown in Fig. 8 and Fig. 9. Malwise returns its results in XML. This XML is transferred back across the SSH tunnel to the client on the web frontend host where it is stored for processing.

4.3.1 Simseer

The modules we have deployed to implement Simseer are:

- Packer Detection using Entropy Analysis
- Automated Unpacking using Application-level Emulation.
- Control Flow Decompilation
- Software Similarity Detection using Q-Grams of Decompiled Control Flow Graphs

The automated unpacker is a module to remove obfuscations and encryptions by revealing the hidden code (Cesare & Xiang 2010a). Packing is common in most malware. To deploy the automated unpacker we needed to make available the Windows system libraries. The reason for this is that the emulator requires libraries to implement dynamic linking of the emulated guest programs.

We used two types of configurations to Malwise with the above module list. In the first configuration, processing executables creates a signature for the software similarity detection. In the second configuration, the signature database is assumed to be already filled.

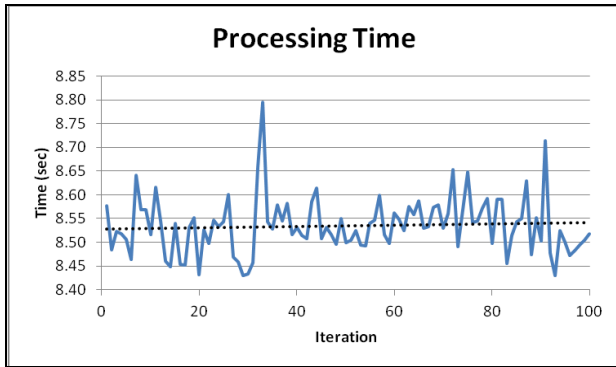


Figure 5. Simseer processing time.

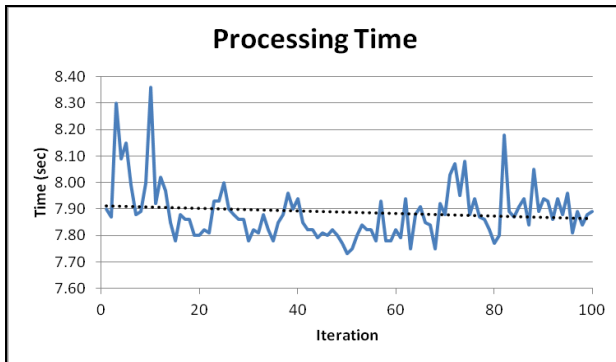


Figure 6. Malwise processing time.

Thus Simseer is split into two phases – signature database creation and software similarity detection. The script handling the launching of Malwise calls Malwise once for each phase, and therefore two times in total.

4.3.2 Bugwise

The modules we have deployed to implement Bugwise are:

- Intermediate Language Optimisation
- Decompilation Modules
- Linux
- Data Flow Analysis
- Double-free Detection

The intermediate language optimisations are a set of compiler style optimisations that operates over the intermediate language Malwise uses to represent x86 assembly code. The optimisations that are implemented are:

- Dead Code Elimination
- Copy propagation
- Constant folding
- Constant propagation

The decompilation modules translate stack based local variables to native variables in the intermediate language. This allows the data flow analysis to identify problems such as use-after-frees and double-frees.

A Linux specific module is used to identify the beginning of the main() function via the __libc_start_main library call.

The data flow analysis module enables a variety of analyses such as:

- Reaching Definitions
- Upwards Exposed Uses

```
klez.a
klez.b
klez.c
klez.d
klez.g
klez.h
netsky.aa
netsky.e
asciitext
```

Figure 7. Simseer samples.

```
<ModuleGroup>
  <Name>Scan</Name>
  <Run>Packer Detection Using Entropy</Run>
  <Run>Unpacker Using Application Level Emulation</Run>
  <Run>Structuring</Run>
  <Run>NGram Structuring</Run>
</ModuleGroup>
```

Figure 8. Simseer configuration.

```
<ModuleGroup>
  <Name>Scan</Name>
  <Run>Code Optimisation 1</Run>
  <Run>Linux Arch</Run>
  <Run>Pre Decompiler Data Flow Analysis</Run>
  <Run>X86 Decompiler Data Flow Analysis</Run>
  <Run>Decompiler Data Flow Analysis</Run>
  <Run>Code Optimisation 2</Run>
  <Run>IRDataFlowAnalysis</Run>
  <Run>Double Free Detection</Run>
</ModuleGroup>
```

Figure 9. Bugwise configuration.

• Reaching Copies

Finally, the double-free detection module uses the data flow analysis to discover use of the free pointer after a free() without a reassignment of the pointer. In practice, Bugwise has found software defects in Debian Linux given only access to the binary executables.

4.4 Simseer Evolutionary Tree Visualization

A phylogenetic or evolutionary tree is a visual representation of the evolutionary relationships between species based on similarity between features or characteristics. Species closer to the tree in relation to the number of branches or branch lengths are more closely related. Simseer uses evolutionary trees to visualize the relationships between programs and their variants. This visualization is useful because program variants are typically derivatives and modified versions of their upstream source.

The web frontend host is responsible for processing the XML results returned by Malwise. One of the responsibilities of the script launched on the web host is to create and render an evolutionary tree of the submissions. The XML returned by Malwise scores the similarity between each sample. The script transforms the XML into a distance matrix. Distance is calculated as 1 –

similarity. This distance matrix is then analysed to create an evolutionary tree using the PHYLIP software package (Felsenstein 2005). The PHYLIP package uses the neighbour joining method (Saitou & Nei 1987) to construct an evolutionary tree. The evolutionary tree is described by the Newick tree format which gives such information as branch lengths in the tree. The Newick tree file is processed to render a figure suitable for display. The figure is then transformed to a PNG image and stored on the web host. An example of the tree visualization is shown in Fig. 4.

4.5 Results Processing

The results shown to the user are different depending on whether Simseer or Bugwise is being used.

4.5.1 Simseer

To display the results, the Malwise XML similarity results are displayed as an HTML table. The background colour of the table cells are proportional to how similar the samples are. The lighter the colour, the more similar the programs are. If the programs are not variants of each other, the table cell is left unshaded. The evolutionary tree image of the programs is shown on the same page. The results processing is performed after submitting an archive to the system and may also be accessed at a later time. Later viewing of the results is achieved by accessing a PHP page to reprocess the Malwise XML results and displaying the permanently stored evolutionary tree image. To specify which archive is requested to be processed, an MD5 digest of the ZIP archive is passed as a parameter to the web page using the GET HTTP method.

4.5.2 Bugwise

Bugwise lists the double frees detection in a HTML table. The double free detector returns the address of the code in the disassembly for both frees that are involved in the bug. To be able to use the results effectively, an analyst must be familiar with reverse engineering. For people performing binary analysis without source this skill is expected.

5 Efficiency of Malwise as a Web Service

We performed an evaluation of the time it takes to process 9 samples using the Simseer web service. We did this by writing a Python script to submit the samples to the web service over HTTP and read the results. The samples we used to perform this test are shown in Fig. 9. Eight samples were malware and 1 sample was some ASCII text which should not be found similar to any of the executables. We submitted the 9 samples as a ZIP archive to a local machine running the Simseer web service. We performed this test 100 times. A mean time of 8.53 seconds was recorded with a standard deviation of 0.06 seconds. The results are shown graphically in Fig. 5.

We performed a similar evaluation on the samples, but this time we ran the tests by command line and without performing the program visualization using evolutionary trees. This test gives us a base line for Malwise, upon which Simseer is based. The comparison between Malwise (Fig. 9) and Simseer (Fig. 8) demonstrates how

effective the web service is (Fig. 8) when compared to using the system without the web interface (Fig. 9). The mean processing time for 100 iterations was 7.89 seconds with a standard deviation of 0.11 seconds. The results are shown graphically in Fig. 6.

The overhead of Simseer as a web service, excluding varied upload times of different speed networks, is 0.64 seconds. These results show that providing Simseer as a web service is efficient and does not add significant overhead to Malwise.

We take the previous results into account when considering Bugwise. Bugwise is much slower than Simseer due to the data flow analysis that is required for bug detection. We see no significant overhead in launching Bugwise since it uses the same web frontend and scheduling code as Simseer.

6 Availability

The Simseer service is free to use. It can be accessed on the web at <http://www.foocodechu.com/?q=simseer-a-software-similarity-web-service>. The Bugwise service is also free to use and can be accessed on the web at <http://www.foocodechu.com/?q=node/19>. We have implemented rate limiting to restrict the number of scans per day per IP address. We have also limited the number of samples that can be submitted per ZIP archive to the Simseer, and limited the size of the binary that can be submitted to the Bugwise service. As the service grows, we may relax some of these constraints.

7 Future Work

One thing we would like to do is replace our custom scheduling work queue with an enterprise messaging system such as RabbitMQ. Enterprise-level messaging systems have guarantees on reliabilities in the case of transmission or network failures. Using such a system would improve our reliability. Enterprise messaging also leads to an easy solution to distributed scan servers as we can have a single producer of messages on the web front end, and consumers in multiple scan servers.

We would also like to implement more flexibility in which modules are used in launching Simseer and Bugwise. Malwise has many modules available, and multiple options are available for software similarity scoring and defect detection.

Another possibility is using any-time clustering on the stream of samples that are given to Simseer. In this approach, cluster analysis is performed incrementally as objects are given to the system sequentially. An any-time phylogenetic tree analysis could follow on from any-time clustering. Any-time clustering could provide intelligence into new families of malware that are given to Simseer. This could benefit analysts in determining if a new sample relates to an existing family is something never seen before or relatively new.

Bugwise could be extended by treating bug detection instead as bug management. An automated bug reporting system could be used to submit, remove, and verify bugs that it discovers. This type of approach has been used successfully in network vulnerability management and we think that there exists many parallels.

8 Conclusion

In this paper we have demonstrated novel services to 1) score and visualize the software similarity of executable binary programs 2) detect software defects in binaries. The Simseer and Bugwise services are deployed as cloud services and are free to use. Simseer can be used to identify malware variants, detect software theft, and reveal plagiarism of software programs. Bugwise has already found real defects in Debian Linux. Simseer and Bugwise are built as a modular extension to our Malwise binary analysis platform. It demonstrates the versatility of our system that we can launch both services using only slightly different parameters with separate configurations. We performed an evaluation on the overhead incurred by making our Malwise platform using web services. We found that such an overhead was minimal and not significant. We are the first to make a public service that analyses executable binaries in these contexts and see the area of cloud based software analysis and similarity detection as having future growth.

9 References

- Aho, AV, Sethi, R & Ullman, JD 1986, *Compilers: principles, techniques, and tools*, Addison-Wesley, Reading, MA.
- Baxter, ID, Yahin, A, Moura, L, Sant'Anna, M & Bier, L 1998, 'Clone detection using abstract syntax trees', in p. 368.
- Bilar, D 2007, 'Opcodes as predictor for malware', *International Journal of Electronic Security and Digital Forensics*, vol. 1, no. 2, pp. 156-68.
- Bonfante, G, Kaczmarek, M & Marion, JY 2008, 'Morphological Detection of Malware', in *International Conference on Malicious and Unwanted Software, IEEE*, Alexandria VA, USA, pp. 1-8.
- Briones, I & Gomez, A 2008, 'Graphs, Entropy and Grid Computing: Automatic Comparison of Malware', in *Virus Bulletin Conference*, pp. 1-12.
- Cadar, C, Ganesh, V, Pawlowski, PM, Dill, DL & Engler, DR 2008, 'EXE: automatically generating inputs of death', *ACM Transactions on Information and System Security TISSEC (2008)*, vol. 12, no. 2, pp. 10:1-:38.
- Carrera, E & Erdélyi, G 2004, 'Digital genome mapping—advanced binary malware analysis', in *Virus Bulletin Conference*, pp. 187-97.
- Cesare, S & Xiang, Y 2010a, 'Classification of Malware Using Structured Control Flow', in *8th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2010)*.
- Cesare, S & Xiang, Y 2010b, 'A Fast Flowgraph Based Classification System for Packed and Polymorphic Malware on the Endhost', in *IEEE 24th International Conference on Advanced Information Networking and Application (AINA 2010)*.
- Choi, S, Park, H, Lim, H & Han, T 2008, 'A static birthmark of binary executables based on API call structure', *Advances in Computer Science—ASIAN 2007. Computer and Network Security*, pp. 2-16.
- Choi, S, Park, H, Lim, H & Han, T 2009, 'A static API birthmark for Windows binary executables', *Journal of Systems and Software*, vol. 82, no. 5, pp. 862-73.
- Christodorescu, M & Jha, S 2003, 'Static analysis of executables to detect malicious patterns', paper presented to Proceedings of the 12th USENIX Security Symposium.
- Christodorescu, M, Jha, S, Seshia, SA, Song, D & Bryant, RE 2005, 'Semantics-aware malware detection', in *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P 2005)*, Oakland, California, USA.
- Cifuentes, C 1994, 'Reverse compilation techniques', Queensland University of Technology.
- Cousot, P & Cousot, R 1977, 'Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints', in *Sixth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, Los Angeles, California, pp. 238-52.
- Dijkstra, EW 1975, 'Guarded commands, nondeterminacy and formal derivation of programs', *Communications of the ACM*, vol. 18, no. 8, pp. 453-7.
- Ducasse, S, Rieger, M & Demeyer, S 1999, 'A language independent approach for detecting duplicated code', in p. 109.
- Dullien, T & Rolles, R 2005, 'Graph-based comparison of Executable Objects (English Version)', in *SSTIC*.
- F-Secure 2007, 'F-Secure Reports Amount of Malware Grew by 100% during 2007', retrieved 19 August 2009, <http://www.f-secure.com/en_EMEA/about-us/pressroom/news/2007/fs_news_20071204_1_eng.html>.
- Felsenstein, J 2005, *PHYLIP (phylogeny inference package), version 3.6*, Joseph Felsenstein.
- Gerald, RT & Lori, AF 2007, 'Polymorphic malware detection and identification via context-free grammar

- homomorphism', *Bell Labs Technical Journal*, vol. 12, no. 3, pp. 139-47.
- Griffin, K, Schneider, S, Hu, X & Chiueh, T 2009, 'Automatic Generation of String Signatures for Malware Detection', in *Recent Advances in Intrusion Detection: 12th International Symposium, RAID 2009*, Saint-Malo, France.
- Hoare, CAR 1969, 'An axiomatic basis for computer programming', *Communications of the ACM*, vol. 12, no. 10, pp. 576-80.
- Hu, X, Chiueh, T & Shin, KG 'Large-Scale Malware Indexing Using Function-Call Graphs', in *Computer and Communications Security*, Chicago, Illinois, USA, pp. 611-20.
- Kamiya, T, Kusumoto, S & Inoue, K 2002, 'CCFinder: a multilingual token-based code clone detection system for large scale source code', *IEEE Transactions on Software Engineering*, pp. 654-70.
- Karim, ME, Walenstein, A, Lakhotia, A & Parida, L 2005, 'Malware phylogeny generation using permutations of code', *Journal in Computer Virology*, vol. 1, no. 1, pp. 13-23.
- Kephart, JO & Arnold, WC 1994, 'Automatic extraction of computer virus signatures', in *4th Virus Bulletin International Conference*, pp. 178-84.
- King, JC 1976, 'Symbolic execution and program testing', *Communications of the ACM*, vol. 19, no. 7, pp. 385-94.
- Kolbitsch, C, Comparetti, PM, Kruegel, C, Kirda, E, Zhou, X, Wang, XF & Santa Barbara, UC 2009, 'Effective and efficient malware detection at the end host', in *18th USENIX Security Symposium*.
- Krinke, J 2001, 'Identifying similar code with program dependence graphs', in p. 301.
- Kruegel, C, Kirda, E, Mutz, D, Robertson, W & Vigna, G 2006, 'Polymorphic worm detection using structural information of executables', *Lecture notes in computer science*, vol. 3858, p. 207.
- Lim, H, Park, H, Choi, S & Han, T 2009a, 'A method for detecting the theft of Java programs through analysis of the control flow information', *Information and Software Technology*, vol. 51, no. 9, pp. 1338-50.
- Lim, H, Park, H, Choi, S & Han, T 2009b, 'A Static Java Birthmark Based on Control Flow Edges', in *Computer Software and Applications Conference (COMPSAC '09)*, pp. 413-20.
- Liu, C, Chen, C, Han, J & Yu, PS 2006, 'GPLAG: detection of software plagiarism by program dependence graph analysis', paper presented to Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, Philadelphia, PA, USA.
- Livieri, S, Higo, Y, Matushita, M & Inoue, K 2007, 'Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder', in *Proceedings of the 29th international conference on Software Engineering (ICSE '07)*, pp. 106-15.
- Molnar, DA & Wagner, D 2007, *Catchconv: Symbolic execution and run-time type inference for integer conversion errors*, Technical Report UCB/EECS-2007-23, EECS Department, University of California, Berkeley.
- Myles, G & Collberg, C 2005, 'K-gram based software birthmarks', paper presented to Proceedings of the 2005 ACM symposium on Applied computing, Santa Fe, New Mexico.
- Park, H, Choi, S, Lim, H & Han, T 2008, 'Detecting code theft via a static instruction trace birthmark for Java methods', in pp. 551-6.
- Prechelt, L, Malpohl, G & Philippsen, M 2002, 'Finding plagiarisms among a set of programs with JPlag', *Journal of Universal Computer Science*, vol. 8, no. 11, pp. 1016-38.
- Saitou, N & Nei, M 1987, 'The neighbor-joining method: a new method for reconstructing phylogenetic trees', *Molecular biology and evolution*, vol. 4, no. 4, pp. 406-25.
- Son, J-W, Park, S-B & Park, S-Y 2006, 'Program Plagiarism Detection Using Parse Tree Kernels', in Q Yang & G Webb (eds), *PRICAI 2006: Trends in Artificial Intelligence*, Springer Berlin / Heidelberg, vol. 4099, pp. 1000-4.
- Symantec 2008, *Symantec internet security threat report: Volume XII*, Symantec.
- Symantec 2011, 'Internet Security Threat Report', vol. 16.
- Van Emmerik, MJ 2007, 'Static Single Assignment for Decompilation', The University of Queensland.

Wang, X, Jhi, Y-C, Zhu, S & Liu, P 2009, 'Behavior based software theft detection', paper presented to Proceedings of the 16th ACM conference on Computer and communications security, Chicago, Illinois, USA.

Wehner, S 2007, 'Analyzing worms and network traffic using compression', *Journal of Computer Security*, vol. 15, no. 3, pp. 303-20.

Wicherski, G 2009, 'peHash: A Novel Approach to Fast Malware Clustering', in *Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET'09)*, Boston, MA, USA.

Wise, MJ 1996, 'YAP3: improved detection of similarities in computer program and other texts', *SIGCSE Bull.*, vol. 28, no. 1, pp. 130-4.

Ye, Y, Wang, D, Li, T & Ye, D 2007, 'TMDS: intelligent malware detection system', in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*.

Cloud-Aware Processing of MapReduce-Based OLAP Applications

Hyuck Han¹ Young Choon Lee² Seungmi Choi¹ Heon Y. Yeom¹
 Albert Y. Zomaya²

¹ School of Computer Science and Engineering,
 Seoul National University, Seoul, 151-742, Korea.
 Email: {hhyuck, smchoi, yeom}@dcslab.snu.ac.kr

² Centre for Distributed and High Performance Computing, School of Information Technologies,
 University of Sydney, NSW 2006, Australia.
 Email: yclee@it.usyd.edu.au, albert.zomaya@sydney.edu.au

Abstract

As the volume of data to be processed in a timely manner soars, the scale of computing and storage systems has much trouble keeping up with such a rate of explosive data growth. A hybrid cloud combining two or more clouds is emerging as an appealing alternative to expand local/private systems. However, the effective use of such an expanded cloud system is limited primarily by low network bandwidth and high latency between clouds (i.e., large intercloud data transmission overheads) when applications/services span across clouds, and they deal with large data in particular. Online analytical processing (OLAP) applications are a typical class of data-intensive application. These applications process multi-dimensional analytical queries dealing with ‘big data’ (or data warehouses). In this paper, we address the effective processing of MapReduce-based OLAP applications in a hybrid-cloud environment, and present a (hybrid) cloud-aware OLAP system incorporating data filtering techniques. Our system filters out unnecessary data for intercloud transmission with the ultimate goal of optimizing the performance to cost ratio, or cost efficiency. Based on experimental results obtained using two large-scale data analysis benchmarks, our system demonstrates its efficacy in improving the cost efficiency with the reduction in intercloud network traffic from 76%-99%.

Keywords: Cloud Computing; Hybrid Cloud; MapReduce; On-Line Analytical Processing (OLAP); Cost Efficiency

1 Introduction

Cloud computing with the support of virtualization technologies and utility computing (or pay-as-you-go) has emerged as a cost-effective solution for many computing tasks including large-scale data processing (Amazon Web Services 2012). For example, *CycleComputing*’s Amazon EC2 (Elastic Compute Cloud) powered 51,132 core high-performance cluster performed massive molecular modeling simulations—21 million chemical compounds—that used the equivalent of 12.5 CPU years for less than \$4,900 an hour

(CycleComputing 2012). In essence a cloud is classified as private or public based primarily on the availability to the public. A hybrid cloud can be formed as a mixture of two or more clouds of these two categories. Services offered in clouds can be classified into three types: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). This study takes a particular interest in IaaS clouds. An IaaS cloud provisions virtual resources such as computing nodes, storage, and networks, e.g., Amazon EC2 and S3 (Simple Storage Service).

In the recent past, we have witnessed dramatic increases in the volume of data literally in every area—business, science, and daily life to name a few. Today, some claim that data (more specifically, data-intensive science) are the fourth paradigm in scientific research after experimentation/observation, theory, and computational simulation (Hey et al. 2009). The storage and processing of such an overwhelming amount of data is a challenging task in the current computing environments. What’s more, the timeliness of data processing is the key to judicious decision making, particularly in business.

The MapReduce framework (Dean & Ghemawat 2008) proposed by Google is a parallel-programming model primarily for (large) data processing specifically designed with the consideration of large-scale distributed computing systems, such as clusters and data centers. A MapReduce application (or job) typically consists of large numbers of map and reduce tasks. Each map/reduce task deals with a chunk of data independently, and thus tasks in the job can be readily parallelizable and effectively processed in a large-scale computing environment like a cloud. Recently, MapReduce have been used in not only the analysis of a homogeneous data set (e.g., log processing) but also that of a heterogeneous data set such as data warehouses and data marts. Particularly in businesses, online analytical processing (OLAP) applications can take great advantage of the MapReduce framework because these applications process multi-dimensional analytical queries dealing with data warehouses (or ‘big data’).

It is often the case that the capacity of a single computer system (e.g., a private cloud) cannot keep up particularly with the growth rate of data volume to be processed by large-scale data processing applications, such as OLAP applications. Besides, private clouds occasionally get overloaded due to workload surges. The expansion with a public cloud (or hybrid cloud) is an appealing alternative to complement private clouds (Celesti et al. 2010, Buyya et al. 2010, Johnston 2009).

Copyright ©2013, Australian Computer Society, Inc. This paper appeared at the 11th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, South Australia, January-February 2013. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 140, Bahman Javadi and Saurabh Kumar Garg, Ed. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

Although adequate computational resources are available for MapReduce-based OLAP jobs in hybrid cloud environments, performance is not as high as expected due primarily to low bandwidth and high latency between private and public clouds. This intercloud (or cloud to cloud) transmission imposes an economic burden on users since network usage is also charged in public clouds. Thus, in hybrid cloud environments data movement is a crucial factor not only for performance, but also for cost.

In this paper, we address the problem of processing MapReduce-based OLAP applications in hybrid cloud environments. The work in this paper is designed to mitigate the performance and cost issues of MapReduce jobs in hybrid cloud environments by reducing the amount of intercloud data transfer using data filtering techniques. We exploit two types of filters—static and dynamic—deployed on the distributed file system; and they are evaluated with Hadoop. Our experimental results with data analysis workloads in Amazon EC2 demonstrate performance improvement and cost-cutting effects. Specifically, our OLAP system reduces network traffic as much as 99% (and at least 76%), and improves application performance (reduction in processing time) by 13-71%; together, 84% of total costs when processing without our system (default) is reduced.

The main contributions of this paper are as follows:

- We identify the impact of intercloud data transfer overheads on the performance of MapReduce-based OLAP applications.
- We develop a cloud-aware OLAP system based on the MapReduce framework for hybrid clouds.
- We demonstrate the effective usage of data filtering techniques to reduce intercloud data transmission overheads.
- Our system using two large-scale data analysis benchmarks has been evaluated in terms of both performance and cost efficiency.

The remainder of the paper is organized as follows. Section 2 presents background and related work. Section 3 describes the problem we address in this paper. Section 4 details the design and implementation of our system with description of data filtering technique incorporated. Section 5 evaluates the efficacy of our system in terms of performance (running time) and cost efficiency. Then, Section 6 concludes the paper.

2 Background and Related Work

In this section, we begin by describing the MapReduce framework and OLAP applications in the context of Hadoop (Apache 2012a), the open-source counterpart of MapReduce. We then discuss the deployment of OLAP applications in a hybrid cloud and issues related to such deployment.

2.1 MapReduce and OLAP

MapReduce is derived from functional programming concepts and is composed of two basic computation units/functions: Map and Reduce.

- Map takes an input and produces a set of intermediate key/value pairs. The MapReduce runtime classifies all intermediate values according to the same intermediate key k and passes them to the Reduce function.

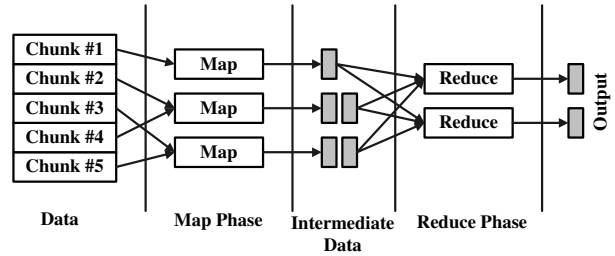


Figure 1: MapReduce model.

- Reduce receives an intermediate key k and a set of values associated with the key. Reduce merges the values to form a set of new values. Typically, one output value is produced per one Reduce invocation.

Figure 1 shows the data flow in the Map/Reduce phases with IO for reading and writing data indicated by arrows. These IO activities are handled by the Hadoop Distributed File System (HDFS) resided in the private cloud. The existing Hadoop filtering is performed in worker nodes on which Map/Reduce functions execute instead of HDFS nodes; and some of these worker nodes are in the public cloud in our hybrid cloud model. Thus, for these worker nodes, the existing filtering approach has no effect on intercloud data traffic.

The MapReduce programming model has many advantages, such as high throughput/performance, use of commodity clusters, and fault tolerance. MapReduce is used in not only index construction for search engines (Dean & Ghemawat 2008) but also data analysis of both homogeneous and heterogeneous sets (Yang et al. 2007, Apache 2012c). Data join processing, which is very important for complex analysis in data warehouses, is addressed in (Yang et al. 2007) and (Pike et al. 2005) using MapReduce. Recently, Hadoop-based implementations, such as Hive (Apache 2012b) and CloudBase (Business.com 2012), have been developed for data warehouse workloads. In (Stonebraker et al. 2010), the authors compare MapReduce and parallel DBMS in various viewpoints such as performance and system management. Based on their conclusion, parallel DBMSs are suitable for efficient querying of large structured data, whereas MapReduce has advantages at complex analytics and extract-transform-load (ETL) tasks. This means that MapReduce can be useful for OLAP processing in large data warehouses. For example, Facebook has implemented a large data warehouse system using MapReduce instead of DBMSs (Monash 2009).

A data warehouse is an online repository for data from operational systems of an enterprise (W.H. Inmon 1996). A data warehouse is usually maintained using a star schema that is composed of a single fact table and any number of dimension tables. A fact table contains atomic data or records for business areas such as sales and production. Dimension tables have a large number of attributes that describe records of the fact table. Figure 2 shows an example of a star schema derived from the Star Schema benchmark database (O’Neil et al. 2007). The fact table is the LINEORDER table, and the dimension tables are CUSTOMER, SUPPLIER, PART, and DATE tables. The LINEORDER table has several foreign keys such as CUSTKEY, PARTKEY, SUPPKEY, ORDERDATE, and COMMITDATE to refer to each dimension table. Generally, queries in data warehouses are complex and ad hoc. The star-join query, in which the fact table

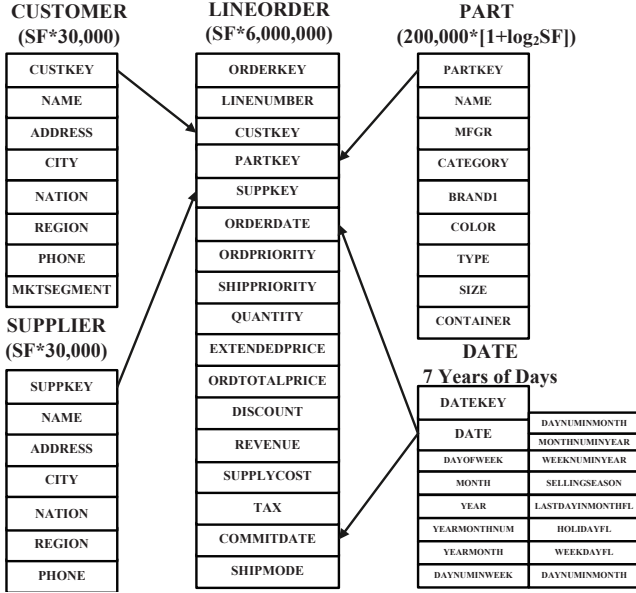


Figure 2: Example of the star schema (O’Neil et al. 2007); SF: scale factor in the databases.

is joined with one or more dimension tables, is one of the well-known queries in OLAP. For another example, TPC-H (Transaction Processing Performance Council 2012) provides a set of ad hoc queries for decision support systems that primarily use data in a data warehouse. In MapReduce, all data in data warehouses are stored as a form of a chunk in distributed file systems such as HDFS and Google File System (GFS) (Ghemawat et al. 2003). In this paper, we focus on MapReduce-based OLAP applications in data warehouses.

2.2 Hybrid Cloud Deployment

In (Vaquero et al. 2008), authors define a cloud as a large pool of easily usable and accessible virtualized resources. Clouds can also be regarded as data center hardware and software being served, and these resources are exposed in a pay-as-you-go style to enable public utility computing (Armbrust et al. 2010, Paul 2008). Cloud computing users take advantage of this pool of resources by paying only for the resources as their need grows or shrinks. The elasticity and scalability are the key characteristics of cloud computing platforms. Scalability is one rising issue in large-scale cloud deployments particularly with multi/many core machines (Boyd-Wickizer et al. 2010, Song et al. 2011). Authors in (Vaquero et al. 2011) extensively surveyed cloud scalability issues and classified them in three different levels.

As the scale of data constantly increases timely data processing and analysis is of great practical importance in business activities and scientific research communities such as HPC (Humphrey 2011, Zhai et al. 2011). If the data size is very large to the extent that a single cloud system cannot process the data, the cloud system would not be able to satisfy further requests from clients. In such a situation, the hybrid cloud plays a crucial role in resolving the scarcity of resources. If a private cloud in an enterprise becomes saturated, the enterprise may provision resources by renting from public cloud service providers. Thus, large-scale data processing in data warehouses will be an important target of hybrid clouds. However, current MapReduce frameworks do not consider hy-

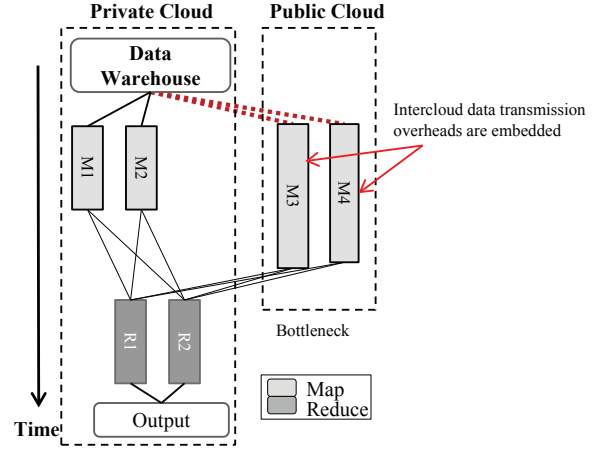


Figure 3: MapReduce example in a hybrid cloud environment.

brid clouds yet, and MapReduce-based OLAP applications on hybrid clouds experience low performance. This expansion of running OLAP application to multiple clouds offers a type of scalability solution, i.e., the platform level scalability as classified by authors in (Vaquero et al. 2011).

3 Problem Statement

In this section, we use an illustration to state the problem of processing MapReduce-based OLAP applications in hybrid clouds.

Supposing there was a private cloud running a data warehouse, it may occasionally require more computing capacity beyond its own. If new physical machines are added to the private cloud, the heavy burden of both capital and operating costs would be inevitable. Thus, renting additional computing units from public clouds is naturally a cost-effective alternative. In this study, we do not consider a situation that the private cloud needs a public cloud for storage (e.g., security issues of internal data), and data are not partitioned or distributed over between private and public clouds.

In hybrid cloud environments, networking between clouds has low bandwidth and high latency since their communication essentially relies on the Internet. This is a major limiting factor to the adoption and prevalence of hybrid clouds, particularly for the MapReduce framework for which the data movement is frequent and large. Figure 3 demonstrates the effect of MapReduce applications when data are transferred from one cloud to another. While map tasks (M_1 and M_2) receive their inputs from the data warehouse of their local cloud, map tasks (M_3 and M_4) in a public cloud get their inputs from the private cloud, i.e., intercloud data transmission. Although all map tasks start almost at the same time, the map tasks in the public cloud take a longer time than the other two map tasks in the private cloud to execute due to the intercloud data transmission. Besides, each reduce task performs only after it gets corresponding output from every map task. Thus, no reduce tasks can start until the map tasks in the public cloud are finished, even if the other map tasks are completed. As a result, the map tasks in the public cloud are bottlenecks of total MapReduce processing, and the MapReduce program is slowed down.

4 Cloud-aware OLAP

In this section, our system design and implementation are articulated with the description of data filtering techniques incorporated. Then, cost efficiency metrics associated with our approach are presented.

4.1 Intercloud Transmission Reduction using Data Filtering

We propose a simple filtering technique that avoids transmission of unnecessary data particularly to public cloud nodes in which Map and Reduce tasks execute. And, we modify HDFS to be aware of data (e.g., record layout).

In our system, filters are configured in MapReduce programs by users (MapReduce Program Generation in Figure 4), and our system uses filters for better performance (MapReduce Task Running in Figure 4). When users write a MapReduce program, they add static/dynamic filters into the MapReduce program. Then, our system uses filters that users specified. When map processes start, they request data for their jobs with filter information to file systems (3 in Figure 4). File systems send filtered data to map processes on the public cloud (4 in Figure 4).

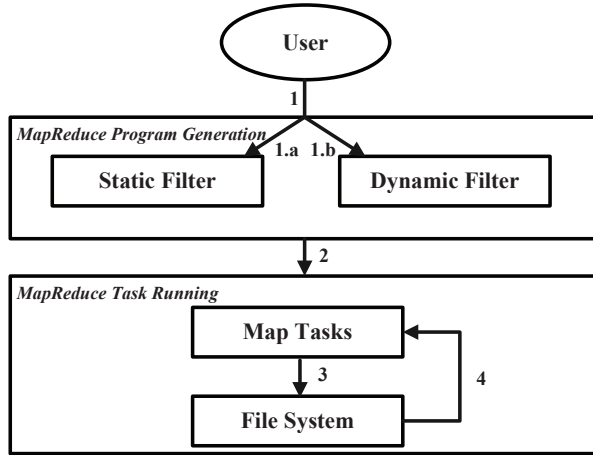


Figure 4: Overview of our technique.

Currently, our system supports two types of filter: static (shown as 1.a in Figure 4) and dynamic (shown as 1.b in Figure 4) filters. A static filter, such as a relational algebraic operator, is recognized when a MapReduce task starts. Figure 5 indicates a sample database and its example query. A static filter uses a fixed constraint, such as $DT1.PK_1 = a_0$ and $DT2.PK_2 = b_0$ in Figure 5. This information is included as a job configuration parameter in the MapReduce Program (shown as 1.a Figure 4). It will be used to filter data from dimension tables ($DT1$ and $DT2$) if map tasks are placed outside the private cloud.

On the other hand, a dynamic filter, such as a bloom filter (Bloom 1970), can be used after filter construction is performed. For example, a user writes an efficient MapReduce-based join program by using a bloom filter. To this end, the user typically writes the program considering a filter construction phase (Business.com 2012, Han et al. 2011). During this phase, records of $DT1$ and $DT2$ are processed to produce bloom filters for all join keys (PK_1 and PK_2) and bloom filters are stored to the distributed file system. In the next step, records of the fact table (FT) and its corresponding dimension table are processed

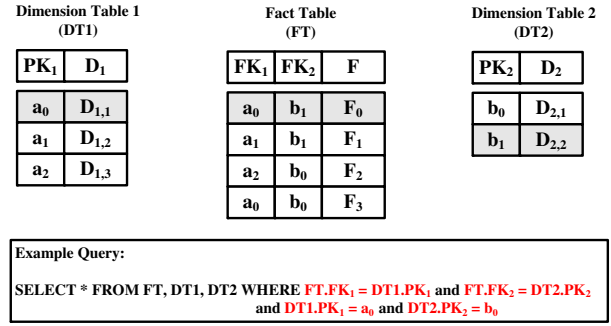


Figure 5: Query example.

to perform the join processing. In this phase, the distributed file system first checks whether each foreign key (FK_1 and FK_2) of each record in the fact table, which usually is the largest table in the data warehouse, is contained in bloom filters from the previous phase or not. Then, if a record passes through the check, it is sent to map processes when they are far from the data warehouse cloud. Otherwise, it is dropped in the file system node. Due to the nature of the dynamic filters, the address of each bloom filter is saved as a configuration in MapReduce Program, (1.b) in Figure 4. It is noted that all bloom filters can be used in map processes if users do not use our system, and this technique can improve the performance of star-join queries in a single cloud (Han et al. 2011).

4.2 System Implementation

In this section, we give implementation details of our system including filter configuration as part of the OLAP MapReduce configuration and the actual filtering operation of data.

4.2.1 Filter Configuration

The filter information is stored as part of a job configuration in Hadoop. The information includes a type of filter (dynamic or static, or both), the use of filter (true, false), required arguments (e.g., addresses of bloom filters), and the location of the file system (whether inside or outside the private cloud).

Table 1: Filter property.

Dynamic filter	Static filter
filter type (e.g., bloom filter)	filter type (e.g., operators such as < and >)
filter address	number of filters number of columns line or only column

Table 1 shows that each setting of filters needs a different configuration. The dynamic filter needs a type of filter, notifies its own type, and specifies addresses of bloom filters. The static filter also demands a filter type, the number of columns for the table of data, the number of filters, and a method to eliminate a line or column.

4.2.2 Operation on HDFS

In this section, we propose a transmission algorithm that applies a filter at each time after reading data. Since we consider the `LineRecordReader` class as

a default reader¹, the first line and last line in the block (or chunk) may be overlapped with the previous or next chunk and may not be complete. We exclude these lines from the filter operation. The rest of the lines are passed on to the filtering step. If those lines are necessary data, they are added to the output; otherwise, they are discarded. When the output size is larger than or equal to the predefined packet size (e.g., 64KB), the file system nodes transfer the packet to map tasks in the computational nodes. The filtering operation (Algorithm 1) repeats until the end of the block is reached.

Algorithm 1 HDFS Filter.

```

while not the end of block do
    read data 64KB at a time
    if data are the first or last line of block then
        continue
    end if
    apply filter to data
    concatenate filtered data to output
    if output is larger than packet size (64KB) then
        transmit output
    end if
end while
transmit remaining output
    
```

We modify client and server code of original HDFS to realize filters. In the **BlockSender** class of the server side, data is recognized as a set of tuples, and is sent to the client side applying filters. The **DFSClient.BlockReader** class of the client side receives filtered data. For prototype implementation, we do not use existing checksum data stored in HDFS node but **BlockSender** re-computes checksum values when it sends filtered data. Additionally, we insert a one-byte value (boolean value) that indicates the last packet for the chunk. Since existing communication protocols use only the length of static data, the value is used to mark the completion of data transmission. To configure filters in MapReduce programs, we introduce new key/value pairs to **JobConf** objects, and the key/values pairs describe filter information shown in Table 1.

4.3 Cost Efficiency

The performance to cost ratio (cost efficiency) from the user's perspective is an important metric when considering the use of public clouds in particular. In this section we characterize cost efficiency of running OLAP applications in hybrid clouds. We explicitly take into account intercloud network traffic and usage of resources (or instances in Amazon EC2) in our cost efficiency metrics.

The perfect linearity or even decent direct proportionality between the number of public resources rented and performance improvement is only in the ideal scenario (theory). Although this non-linearity is not only present with the use of public clouds, the cost related to public cloud usage makes such non-linearity more serious. This non-linearity is sourced from two main factors, particularly in our study with OLAP applications: (i) data transmission overhead between clouds and (ii) 'hourly-base' rate for public cloud resource rental. In the following, we devise a cost efficiency metric considering both factors.

For a given MapReduce-based OLAP job with M map tasks, we estimate the completion time of map

phase T_m as follows:

$$T_m = (T_i \cdot M) / (N_p + sd \cdot N_b) \quad (1)$$

where T_i is the average execution time of map tasks, N_p and N_b are the numbers of resources in the private cloud and the public cloud, respectively, and sd is a slowdown rate. sd in our study is primarily estimated based on the first non-linearity factor described above, i.e., intercloud data transmission. Clearly, the performance of public cloud resources is affected/decreased by the amount of data to be transferred. We do not consider the reduce phase because it is performed in a private (local) cloud.

The expense of renting public cloud resources C_b is calculated by the product of unit resource cost C_i , total map phase time (hour) T_m , and the number of rented resources N_b .

$$C_b = C_i \cdot \lceil T_m \rceil \cdot N_b \quad (2)$$

The cost efficiency of running a MapReduce job in a hybrid cloud is the reduced time per unit price, i.e., the performance improvement to the public cloud cost ratio. It is determined by T_m and T_d where T_d is the map phase time without the support of the public cloud. More formally,

$$CE = (T_d - T_m) / C_b. \quad (3)$$

5 Experimental Evaluation

In this section we detail experiments to evaluate our system and present results. Specifically, the performance improvement capability of our system is verified with experiments in the real cloud setup using Amazon EC2. Then, we show cost saving implications and discuss how to optimize cost efficiency.

5.1 System Performance

In this section, we show that our system reduces the amount of data transmission and this reduction leads to performance improvement effectively.

5.1.1 Experimental Environment

For our experiment, we rented resources from Amazon EC2 (Standard Small, m1.small) in two different available regions: US East and West zones as shown in Figure 6. We used four instances in each region as computing nodes, and four more instances were added to the East zone as storage nodes. Map tasks in the US East zone receive their input data from the same zone (as in the private cloud). However, map tasks in the US West zone receive their input from the US East zone through an inter-zone (intercloud) network.

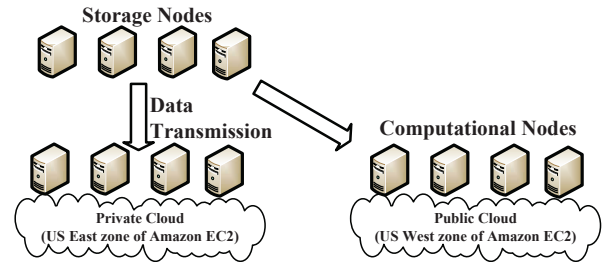


Figure 6: Experimental environment.

¹The **LineRecordReader** class recognizes a line as a record or a tuple.

Our system was evaluated with well-known, large-scale data analysis benchmarks:

- TPC-H - a business-oriented decision support benchmark, which simulates an online production database environment; we used a scale factor of 20 (i.e., a data set of 20GB) in this study.
- SSBM (Star Schema Benchmark) - a benchmark for data warehousing applications; we used a scale factor of 40, which also gives us a database size of about 20GB. SSBM has four query groups, and each group has a couple of queries with different selectivity of the *LineOrder* table (fact table). We used the MapReduce-based SSBM benchmark suite (Han et al. 2011).

These benchmarks provide dedicated data generator programs (*db_gen*). Each *db_gen* program produces data files (*.tbl* files). For example, data for the *lineitem* table is stored in the *lineitem.tbl* file. A tuple of a table corresponds to a line of a data file, and each column in a tuple is separated by a special identifier ('|'). In our experiments, we uploaded all data files to HDFS or filter-enabled HDFS and Map/Reduce programs process data from distributed storage line by line. We ran each test case five times and measured the average running time and average traffic volume. The data are transferred between clouds in two different ways:

- Default - default Hadoop transmission
- Filter_hdfs - filtering data in storage nodes, where the data are filtered before the transmission of the data.

We report results of only dynamic filters (e.g., bloom filter) for this study since static filters do not show significant performance improvement (less than 10%). Because OLAP applications as our target applications have many complex join procedures, our dynamic filters can improve query performance and reduce network usage significantly. It is noted that all results include costs of additional filter construction phases (computation, network, and storage costs).

5.1.2 Results

Results are presented in two aspects: traffic volume and running time. Clearly, these two performance metrics are negatively correlated.

Figure 7 shows the efficacy of our system using *Filter_hdfs* for processing the TPC-H benchmark. Figure 7(a) also shows that performance using the default transmission—without explicit intercloud data filtering—is improved to a certain degree with the use of public cloud resources; however, this degree of performance improvement is not quite align with extra costs related to public cloud usage. Our system leads to further performance improvement through filtering-enabled HDFS. That is, *Filter_hdfs* reduces data transmission by 76-99% compared with the default hadoop transmission (Figure 7(b)), and this leads to performance improvement of 13-56% (Figure 7(a)).

Similar results were obtained from experiments with processing SSBM queries (Figure 8). From Figure 8(b), we can see that our system significantly reduces intercloud data transmission for star join queries (i.e., 95-99% reduction). This leads to superior performance of the *Filter_hdfs* transmission by 49.5-70.5% in terms of running time (Figure 8(a)).

More concrete data on performance improvement our system delivered in experiments are shown in Table 2.

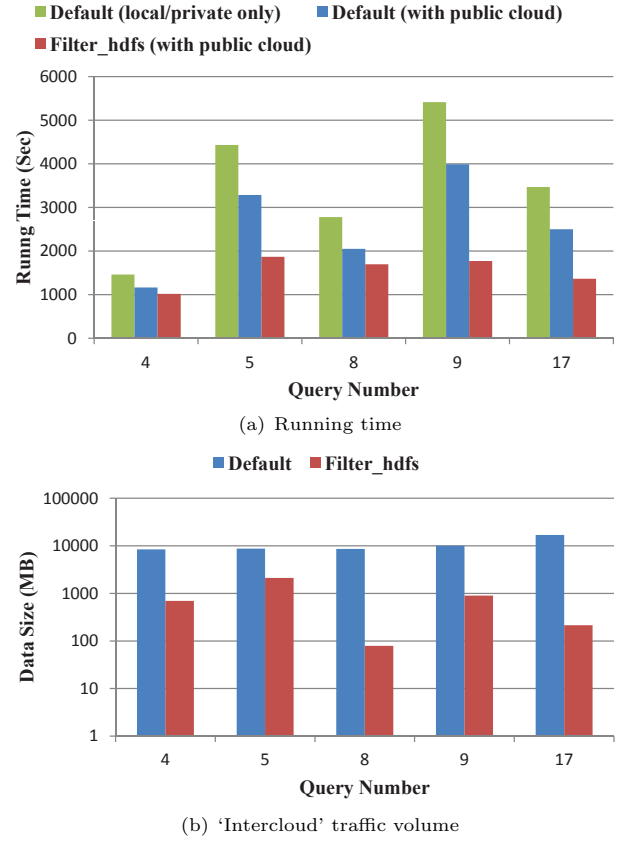


Figure 7: TPC-H results.

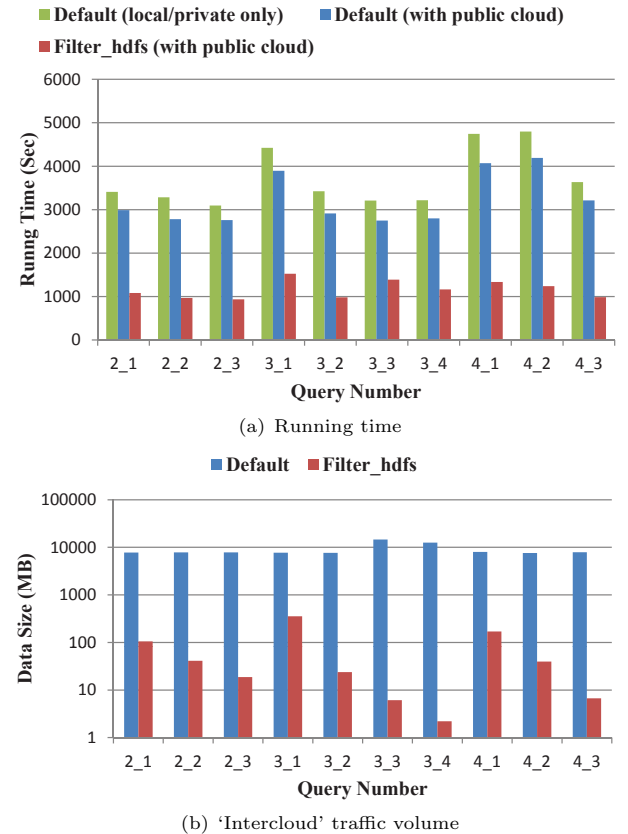


Figure 8: SSBM results. Query id (x_y) in x-axis represents group id x and the y th query within the x query group in SSBM (Han et al. 2011).

5.2 Cost Efficiency

This section begins by demonstrating actual cost savings obtained from our system and discusses cost efficiency implications. All costs are calculated based on the actual rates of Amazon EC2.²

5.2.1 Cost Savings

We summarize total running times and intercloud traffic volumes of TPC-H and SSBM—with and without (default) using our system—in Table 2. Corresponding cost savings sourced from those reductions in Table 2 are plotted in Figure 9.³ Red bars indicate cost savings from reduction in intercloud network traffic while blue bars show cost savings from improved performance (fewer instance rental hours). Specifically, cost savings from intercloud traffic reduction are \$4.8 and \$8.7 for TPC-H and SSBM, respectively. Such cost savings can afford additional 56 and 102 default compute instances (m1.small) for 1 hour; and this implies further performance improvements ‘recursively’. Cost savings in instance rental from improved performance are \$0.34 and \$1.7 for TPC-H and SSBM, respectively. Overall, 84% of costs were reduced using our system, i.e., 96% in data transfer costs and 40% in instance rental.

Table 2: Total running time & intercloud traffic volume.

	Total running time (hour)		
	Filter_hdfs	default	reduction
TPC-H	2.1	3.6	1.5
SSBM	3.9	8.9	5.0

	Total intercloud traffic (GB)		
	Filter_hdfs	default	reduction
TPC-H	3.9	51.6	47.7
SSBM	0.75	87.4	86.65

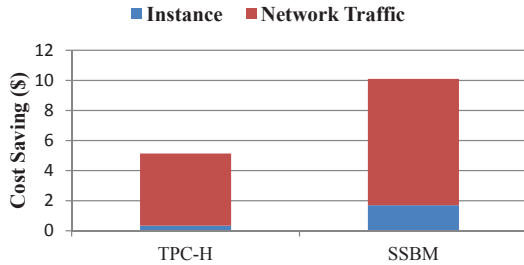


Figure 9: Cost savings.

5.2.2 Optimization of Cost Efficiency

As the usage of public cloud resources is charged based on the number of whole hours, the cost efficiency of cloud deployment is largely dependent on the compactness of tasks with a given number of resources. We have verified this cost efficiency characteristic using the example below. Given a MapReduce-based OLAP application, we consider that $M = 3000$ map tasks, $T_i = 55$ sec, $C_i = \$0.2$, $sd = 0.45$, and $N_p = 4$.

²Resource rental: \$0.085/hr for Linux/Unix m1.small, and data transfer in/out: \$0.1 and \$0.15 per GB, respectively.

³Total costs in our experiments do not include costs for the private cloud since the management or pricing policy of the private cloud may vary.

Table 3: Cost efficiency.

# public resources (N_b)	Cost (C_b)	Running time (T_m)	Cost efficiency (CE)
25	18.32	3.02	27.65
26	16.30	2.91	31.44
27	16.88	2.85	30.61
28	17.46	2.78	29.83
29	18.04	2.71	29.08
30	18.63	2.64	28.39
31	19.21	2.58	27.74
32	19.80	2.51	27.13
33	20.38	2.44	26.55
34	20.97	2.37	26.00
35	21.56	2.34	25.38
36	22.14	2.27	24.90
37	22.73	2.24	24.34
38	23.32	2.17	23.90
39	23.91	2.14	23.40
40	24.50	2.10	22.92
41	25.09	2.07	22.46
42	17.28	1.99	32.84
43	17.67	1.97	32.23
44	18.06	1.93	31.65

Figure 10 shows the relationship between cost efficiency and public cloud cost with respect to different volumes of public resource rental. In this experiment, we have identified eight ‘cost efficiency’ points for the number of resources to be rented (resource count or N_b), i.e., they are good candidates for running the application in terms of cost efficiency. These candidates are 4, 6, 9, 12, 17, 26, 42, and 95 of resources in this experiment as indicated by vertical lines in Figure 10; they are the points where running times are a multiple of whole (or very close to whole) hours, i.e., 7.90, 6.85, 5.69, 4.88, 3.93, 2.91, 1.99, and 0.98. Specifically, each of these points is the case that an additional public resource contributes to the reduction of running time by one hour. Thus, public cloud cost drops as shown in Figure 10 and Table 3 despite the increase in the number of public cloud resources. Table 3 highlights this hourly rate originated pattern exhibited between 25 and 44 of public resources in our experiment. Clearly, the global maximum and its corresponding number of resources are the best interest of the user. In this particular experiment, renting 42 instances is the best choice in terms of cost efficiency (performance improvement per dollar). However, one may select another point (a local maximum) due to budget or time constraints. The cost efficiency characteristic presented in this section can greatly facilitate the design of scheduling and resource allocation policies for the user. Note that since good candidates for the number of public resources to be rented appear around multiples of hours, the search for the global maximum terminates at around one hour of running time.

6 Conclusion

To date, a majority of use cases of hybrid cloud deployments lie in with computationally intensive applications. Yet, applications and services deployed in cloud environments are increasingly data intensive. Cloud sourcing—delegating the entire IT solution to public clouds—might be an alternative; however, it is often not quite possible due to various reasons including security. Thus, reducing the amount of intercloud data transmission is of great practical importance in hybrid cloud deployments. In this paper, we have studied on the intercloud data transmission of OLAP applications, and presented a cloud-aware OLAP system using data filtering techniques. We have shown that our system is capable of reducing intercloud data transmission significantly; that is, experimental re-

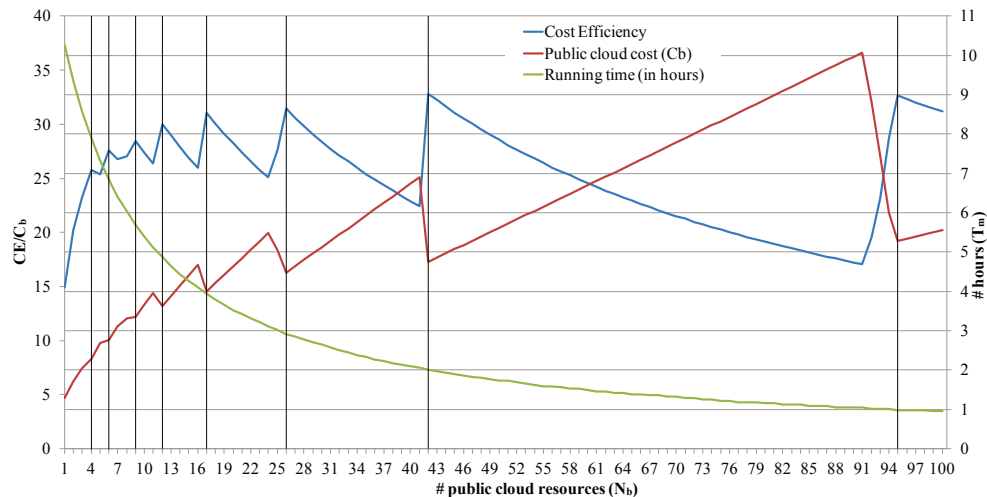


Figure 10: Cost efficiency with respect to different numbers of public cloud resources.

sults verified this claim with improvements in both running time and cost efficiency. We also have explicitly taken into account the current practice of (public) cloud service pricing, and devised cost efficiency metrics to discuss about judicious resource rental decisions.

Acknowledgements

This work was supported by Mid-career Researcher Program through NRF grant funded by the MEST (No. 2010-0014387). The ICT at Seoul National University provided research facilities for this study.

References

- Amazon Web Services (2012), *Customer Success*, <http://aws.amazon.com/solutions/case-studies/>.
- Apache (2012a), *Hadoop*, <http://hadoop.apache.org>.
- Apache (2012b), *Hive*, <http://hive.apache.org>.
- Apache (2012c), *Pig*, <http://pig.apache.org>.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. & Zaharia, M. (2010), 'A view of cloud computing', *Commun. ACM* **53**, 50–58.
- Bloom, B. H. (1970), 'Space/time trade-offs in hash coding with allowable errors', *Commun. ACM* **13**, 422–426.
- Boyd-Wickizer, S., Clements, A. T., Mao, Y., Pesterev, A., Kaashoek, M. F., Morris, R. & Zeldovich, N. (2010), 'An analysis of linux scalability to many cores', in 'Proceedings of the 9th USENIX conference on Operating systems design and implementation', OSDI'10, pp. 1–8.
- Business.com (2012), *CloudBase*, <http://cloudbase.sourceforge.net>.
- Buyya, R., Ranjan, R. & Calheiros, R. (2010), 'Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services', in 'Algorithms and Architectures for Parallel Processing', Vol. 6081 of *Lecture Notes in Computer Science*, pp. 13–31.
- Celesti, A., Tusa, F., Villari, M. & Puliafito, A. (2010), 'How to Enhance Cloud Architectures to Enable Cross-Federation', *Cloud Computing, IEEE International Conference on* pp. 337–345.
- CycleComputing (2012), *CycleCloud Achieves Ludicrous Speed! (Utility Supercomputing with 50,000-cores)*, <http://blog.cyclecomputing.com/2012/04/cyclecloud-50000-core-utility-supercomputing.html>.
- Dean, J. & Ghemawat, S. (2008), 'MapReduce: simplified data processing on large clusters', *Commun. ACM* **51**(1), 107–113.
- Ghemawat, S., Gobioff, H. & Leung, S.-T. (2003), 'The google file system', in 'SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles'.
- Han, H., Jung, H., Eom, H. & Yeom, H. (2011), 'Scatter-gather-merge: An efficient star-join query processing algorithm for data-parallel frameworks', *Cluster Computing* **14**, 183–197.
- Hey, T., Tansley, S. & Tolle, K., eds (2009), *The Fourth Paradigm: Data-Intensive Scientific Discovery*, Microsoft.
- Humphrey, M. (2011), 'Cloud, hpc, or hybrid: A case study involving satellite image processing', in 'Cloud Futures 2011'.
- Johnston, S. (2009), 'Intercloud is a global cloud of clouds'.
- Monash, C. (2009), *Cloudera presents the MapReduce bull case*, <http://www.dbms2.com/2009/04/15/cloudera-presents-the-mapreduce-bull-case>.
- O'Neil, P., O'Neil, E. & Chen, X. (2007), 'The Star Schema Benchmark'.
- Paul, M. (2008), 'The cloud is the computer', *IEEE Spectrum Online*.
- Pike, R., Dorward, S., Griesemer, R. & Quinlan, S. (2005), 'Interpreting the data: Parallel analysis with sawzall', *Scientific Programming Journal*.
- Song, X., Chen, H., Chen, R., Wang, Y. & Zang, B. (2011), 'A case for scaling applications to many-core with os clustering', in 'Proceedings of the sixth conference on Computer systems', EuroSys '11, pp. 61–76.
- Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A. & Rasin, A. (2010), 'MapReduce and parallel DBMSs: friends or foes?', *Commun. ACM*.
- Transaction Processing Performance Council (2012), *TPC-H*, <http://www.tpc.org/tpch>.
- Vaquero, L. M., Roderio-Merino, L. & Buyya, R. (2011), 'Dynamically scaling applications in the cloud', *ACM SIGCOMM Computer Communication Review* **41**(1), 45–52.
- Vaquero, L. M., Roderio-Merino, L., Caceres, J. & Lindner, M. (2008), 'A break in the clouds: towards a cloud definition', *SIGCOMM Comput. Commun. Rev.* **39**, 50–55.
- W.H. Inmon (1996), *Building the Data Warehouse*, J. Wiley & Sons, Inc.
- Yang, H.-c., Dasdan, A., Hsiao, R.-L. & Parker, D. S. (2007), 'Map-reduce-merge: simplified relational data processing on large clusters', in 'SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data'.
- Zhai, Y., Liu, M., Zhai, J., Ma, X. & Chen, W. (2011), 'Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications', in 'State of the Practice Reports', SC '11.

Tools and Processes to Support the Development of a National Platform for Urban Research: Lessons (Being) Learnt from the AURIN Project

Richard O. Sinnott, Christopher Bayliss, Luca Morandini, Martin Tomko

The Australian Urban Research Infrastructure Network (AURIN)

University of Melbourne, VIC, 3052 Australia

`rsinnott@unimelb.edu.au`

Abstract

The development of large-scale software systems remains a non-trivial endeavour. This is especially so when the software systems comprise services and resources coming from multiple distributed software groups, and where they are required to interoperate with heterogeneous, independent (and autonomous) distributed data providers. The use of software development and management tools to support this process is highly desirable. In this paper we focus on the software development and management systems that have been adopted within the national Australian Urban Research Infrastructure Network (AURIN - www.aurin.org.au) project. AURIN is tasked with developing a software platform to support research into the urban and built environment - a domain with many diverse software system and data needs. In particular, given that AURIN is tasked with integrating a large portfolio of sub-projects offering both software and data that needs to be integrated, deployed and managed by a core team at the University of Melbourne, we illustrate how tooling and support processes are used to manage the software development lifecycle and code/data integration from the distributed teams and data providers that are involved. Results from the project demonstrating the ongoing status are presented.

Keywords: Code Management, Collaborative Development Environment, Software Testing, Urban Research.

1 Introduction

Despite many years of experience, the development of complex software infrastructure and systems remains a challenge (Stojanovic 2005). This is especially so when the software systems are developed by distributed teams of software engineers from a multitude of organisations and where stakeholders beyond the software development teams are protagonists in the infrastructure efforts. For many major organisations such efforts are commonplace and systems and processes are well embedded into the way in which software is developed, managed and

ultimately released as products with each iteration (software release) building upon the previous version. However in many circumstances, this building upon a common platform with extensive experiences and feedback from the end users/customers is simply not possible. This is often the case in the area of research-focused projects dependent upon IT. For many research projects, e.g. those sponsored by governments, the software development activities required to support the particular research endeavour are, what can best be described as a greenfield, i.e. with no pre-existing systems already running that require enhancing/tuning, but largely building from scratch. In such circumstances, the architecture of complex research-oriented systems benefits greatly from re-use of existing software components, typically these will come from a variety of sources and/or require implementation without any existing prototype in place. Even with this recycling of software systems however, the adaptation, integration and management of various subsystems to meet the needs of the research community is a far from trivial process – especially when the research needs evolve with the development of the infrastructure itself. That is, often researchers are unaware of the capabilities required of the underlying research platform until the platform exists and facilitates their research.

There have been ranges of software engineering approaches and methodologies that have been explored historically to manage such efforts (Boehm 1988, Filman 2004, Booch 1996). More recently the area of agile software development has gained widespread endorsement as the best way of developing complex systems and ensuring that they meet the end user research community needs through a rapid prototyping and release driven approach (Martin 2003). However for many agile software approaches, the assumption is often that the software developers themselves are physically co-located and directly interacting with each other for iterations of the code and infrastructure release. Often this is not the case especially in large-scale distributed software engineering activities. In such circumstances, tool support for managing the software engineering process of multiple software teams is essential. This is the focus of this paper. In particular the paper focuses on the tools and processes that have been adopted within the Australian Urban Research Infrastructure Network (AURIN) project (www.aurin.org.au).

The rest of the paper is structured as follows. Section 2 provides an introductory overview of the AURIN

project. Section 3 describes the software development framework that represents the AURIN systems architecture. Section 4 describes the tools and process that have been adopted to manage the process of developing the AURIN infrastructure and the lessons learnt in their usage. Section 5 describes the history of development of the AURIN platform and example use cases that have been supported of increasing complexity. Finally section 6 offers some conclusions and identifies future work plans for the AURIN project as a whole.

2 AURIN Overview

The Australian Urban Research Infrastructure Network (AURIN) project (www.aurin.org.au) has been funded through the Australian Government's Department of Industry, Innovation Science, Research and Tertiary Education (DIISRTE). The project is lead out of the University of Melbourne. The project formally commenced in July 2010 with the overarching remit for the '*establishment of facilities to enhance the understanding of urban resource use and management*'. AURIN is a large and complex project with government investment of \$20m to run over the project lifetime. AURIN was initially expected to run to mid-2014, but has since been agreed with DIISRTE to run to mid-2015.

The AURIN project is tasked with providing urban and built environment researchers with a research environment offering seamless access to data and tools for interrogating a wide array of distributed data sets crossing government, industry/commercial and academic domains. The intention is to support multiple research activities that will enhance the understanding of key issues of Australia's past, current and future major urban settlements. This will allow better understanding of a range of phenomenon including (but not restricted to): the impact of population growth and changing demographic profiles of cities; the nature and context of urban environments in which diverse people live, e.g. the future challenges on transport networks, housing, employment, through to the health and well-being of individuals and societal groups, e.g. the elderly.

Prior to AURIN no such national urban research facility existed. Rather a wide range of largely independent silos of data and information existed with no possibility to support the interconnected and multifaceted research challenges associated with urban settlements. Similarly a range of bespoke tools and processes has often been used for the analysis of these data sets. Pockets of expertise in how to use these tools and data sets have been the norm. AURIN is tasked with development of a common data platform with associated analytical tools to provide a "*lab in a browser*" offering seamless access to distributed and heterogeneous data sets and associated tools.

It is essential to note that in developing an infrastructure to tackle such multi- and inter-disciplinary demands, it is paramount that the infrastructure is developed to be flexible, scalable and extensible. Thus there is no fixed (closed) set of data providers, data sets and tools that represent urban and built environment research. Rather, the AURIN infrastructure has to be developed to accommodate the flexible access to and use

of data from a range of diverse organisations including new data providers and data sets almost *on the fly*.

To structure and scope the work on AURIN, the first year of the project (June 2010-June 2011) focused on community engagement and outreach on what the urban and built environment research community would like AURIN to do, be and ultimately deliver.

It was widely accepted that the heart of the AURIN project would be providing programmatic access to urban and built environment data sets in a manner that supported the researchers and their associated research processes. To overcome the data deluge and associated research processes adopted by many which can be classified as "Google-like", i.e. searching for relevant research data using search engines, which typically return masses of relevant and irrelevant data to the researchers, it was identified that targeted access to specific data for specific urban phenomenon was required. To this end the first year of detailed requirements and community engagement resulted in the identification of a key set of strategic urban and built environment research areas to be realized through targeted implementation stream (lenses). Each of these lenses has their own data sets, services and tools that need to be brought together. Ten aspirational lenses were identified including:

1. Population and demographic futures and benchmarked social indicators;
2. Economic activity and urban labour markets;
3. Urban health, well-being and quality of life;
4. Urban housing;
5. Urban transport;
6. Energy and water supply and consumption;
7. City logistics;
8. Urban vulnerability and risks;
9. Urban governance, policy and management;
10. Innovative urban design.

Each of these lenses has an associated expert panel that have (are) directly shaping the focus of the lens activities. Typically these panels identify core data sets and tools that are required to support the particular urban research of interest. Once identified, a typical scenario is that a range of lens-specific sponsored projects is funded through AURIN. These projects have their own software development activities. However a foundational principle of AURIN was that support for multi- and inter-disciplinary research would be possible. Thus rather than having ten separate lens subprojects, it was identified that the inter-connection and interoperability across these lenses was essential. To this end, a common unifying e-Infrastructure was needed. A core technical team at the University of Melbourne is tasked with implementing and coordinating this overarching e-Infrastructure.

At the time of writing, the current core e-Infrastructure has been undergoing development since September 2011 (with the full complement of staff in place since April 2012); the first three of these lens areas have started implementation and each of these has a multitude of lens-specific subprojects occurring. Lenses 4-6 are now at the formal contracting stage (with

3.3 AURIN Analytical Process Library

Many of the needs of AURIN researchers are driven by access to and usage of analytical tools and routines. The AURIN core e-Infrastructure offers a range of *basic* analytical routines such as linear regression, however for many researchers access to richer analytical routines is essential. These can cover algorithms that allow performing geospatially-oriented weighted measures or a variety of cluster analysis. Often domain experts using statistical packages such as R, SAS or STATA to realize such routines and algorithms combining advanced statistical knowledge with urban and build environment experiences. Incorporation of such expertise (routines and how best to utilize them) into the AURIN e-Infrastructure is essential to the overall success of the AURIN platform for collaboration.

4 Distributed Code Development Tools and Processes

To support the AURIN project and its evolving set of needs and requirements both regarding the core technical e-Infrastructure and the multitude of external subprojects that are occurring, software development tool support is essential. To this end the AURIN project has adopted processes and a range of tools that are shaping the overall software engineering efforts including: distributed code versioning tools; coordination, bug tracking and feedback tools; software documentation tools; integrated testing tools, and deployment and management tools.

4.1 AURIN Agile Process

The AURIN project has adopted an agile methodology for its software development plan. Agile software development is an iterative method of determining requirements based on rapid prototyping efforts as the key way to elaborate software requirements and as a model to move towards systems that meet customer's needs (in this case the AURIN research community). An agile methodology is particularly suitable for AURIN since the requirements are extremely complex with a multitude of end users with varying expectations. Put another way, there is no single documented specification of what the AURIN e-Infrastructure should do or be, rather these requirement specifications are growing with the prototype versions of the platform. As such other development models such as sequential design ala Spiral or Waterfall models (Boehm 1988) are not suitable, since they are typically not able to cope with constant changes in requirements from end users and the associated AURIN service/data providers. Indeed, despite the year spent by AURIN on enumerating the needs of the community on the AURIN platform, the level of abstraction identified was not at an implementation level. Instead, the AURIN agile process is one based largely upon real-time reactive design, build, test and deploy.

The core AURIN team themselves are physically co-located at the University of Melbourne in a single office space. This co-location has been specifically and deliberately established to support this project and the team-based coordination efforts. That is, there is no single team member that has the complete infrastructure responsibility (at an implementation level) nor the skill sets to deliver all of the AURIN needs. Rather, it is the

pooling of efforts and resources across the team that is needed.

The actual core AURIN technical team comprises a range of targeted roles including: Portal / User Interface e-Enabler; Security e-Enabler; Data/Metadata e-Enabler; Data architect; Platform infrastructure support; Statistical Geospatial e-Enabler; Geospatial e-Enabler; Workflow e-Enabler and a Middleware/business logic e-Enabler.

The full complement of staff on the core technical team has been in place since April 2012. A senior project manager responsible for the information infrastructure design supervises the day-to-day activities of the core AURIN technical team. The agile methodology that has been adopted utilizes SCRUM-based (Schwaber 2009) systems development, where the senior project manager represents the *ScrumMaster* tasked with the day-to-day efforts of the team. A key goal of a SCRUM-based methodology is organized around the SCRUM-based concept of *sprints*, which involve rapid prototyping to complete the next iteration of the AURIN e-Infrastructure and/or its components. The SCRUM product owner is the AURIN Technical Architect and weekly meetings are organized where results of the latest sprint are discussed and/or demonstrated.

The physical co-location of the core AURIN e-Infrastructure staff allows for immediate feedback and discussions on software development issues that arise. Even with this however and the associated tools that are adopted (see below), there is a need to ensure that the efforts of the team are properly coordinated and synchronized. To support this process, the AURIN e-Infrastructure team runs a daily stand-up session where their daily plans and development issues are identified and discussed. This daily process is augmented with a white-board tracking of efforts and issues as shown in Figure 2 where the horizontal rows represent the individual team members and the vertical columns the specific activities that have been identified for completion as part of the current sprint. The left hand column represents the starting point of a given activity through to the right hand column, which indicates when a particular activity has been successfully completed. As well as providing an overview of the activities of the individual team members, this approach allows to see where bottlenecks and delays are arising for individuals and across the team, with the added (and inadvertent!) advantage of visibly incentivizing team members.



Figure 2: White-board based Work and Activity Scheduling

This model of software development has major advantages for rapid prototyping but does obviously not cater for remote software engineers. To this end, the project has organized a series of *CodeSprints* where the distributed software teams working on lens specific projects come together with the core technical team and, through close coordination of the AURIN senior project manager, work on joint development and integration activities. Thus far three *CodeSprints* have taken place with a fourth planned in November 2012.

The AURIN e-Infrastructure requires far more interaction between the technical teams than physical face-to-face meetings every quarter however. To this end, the AURIN project has adopted a portfolio of distributed software development and management systems.

4.2 AURIN Code Versioning Tools

For many distributed software development projects, network-accessible code versioning systems have been widely recognized as an essential component for the implementation of distributed systems. A range of code management systems currently exist including for example: Code Versioning System (CVS) (<http://sourceforge.net/apps/trac/sourceforge/wiki/CVS>), Mercurial (<http://mercurial.selenic.com>), Subversion (http://en.wikipedia.org/wiki/Apache_Subversion) and Git (<https://github.org>). The AURIN project has adopted Git for its code management and versioning system.

Git is the fastest growing source and revision management system, originally developed for the management of code commits by the open source development community contributing to the development of the Linux kernel. Git provides the ability to develop code in a collaborative manner without the need for a single centralized repository (but allowing the use of one). As such, it is particularly appealing for use in AURIN, where elements of the code may have to be forked and shared with large numbers of external providers for extensive periods of time, before being tested, evaluated and subsequently committed (rolled in) to the core platform.

GitHub (www.github.com) is a commercial code repository based on Git. AURIN opted to host its code in a private GitHub repository to reduce the load on the internal system administrators and leverage access to the wealth of functions provided. It also allows the checking in of code from external parties without having to expose the internal infrastructure at AURIN. At present AURIN has paid for ten Github instances that are used extensively across the project for the core e-Infrastructure development and the associated subprojects.

One of the most important features of Git for AURIN is its code branching and merging model. Instead of cloning software into a separate directory, as is the case with many code versioning systems, Git allows switching between branches in a single working directory. Thus instead of only having branches for major development Git allows routine creation, merging and destroying of multiple, *ad hoc* branches. Indeed each feature or bug can have its own branch, merged in only when it is resolved. This model allows the AURIN software developers to experiment quickly and encourages a rapid development

cycle, where they can work in parallel without always having immediate dependencies on each other.

4.3 AURIN Coordination, Bug Tracking and Feedback Tools

AURIN uses a project management Web application called Redmine (<http://www.redmine.org/>) for the assignment and tracking of development tasks. This environment provides an important capability for tracking bugs, support and feature requests. These so-called “issues” can be assigned to particular developers, versions of the system, and internal deadlines, the progress followed by other collaborators and managers, and statistics about the progress can be collected. It also provides an environment where collaborative documentation is built and maintained in a wiki. It is important to make this environment available as much as possible for externally contracted teams who are interacting with AURIN during the outsourced phases of development – to this end, the first steps following the inception of a new subproject are the activation of the GitHub and Redmine accounts.

4.4 AURIN Software Documentation Tools

To improve collaboration and support the automated documentation of AURIN software subsystems based upon ReST-ful APIs, the AURIN project has adopted Swagger (<http://swagger.wordnik.com>). Swagger is a both source code-level a tool for documenting ReST-ful APIs, and a web-based user interface to browse and test the APIs by sending API requests; indeed, Swagger allows software engineers to test an API without having to write a single line of code. As such Swagger provides a key communications tool that supports collaborative development of APIs. To achieve this, a first prototype API is developed with Swagger - typically including representative data. This allows the API's users to play with the API, and the API's developers to gather users' feedback and modify the API prototype accordingly. Once the API prototype matures (is accepted), the full API can be implemented. This is especially useful when dealing with distributed software teams.

Swagger is available for multiple languages and frameworks. Within AURIN it is used with the Node.js and Spring frameworks. Whilst it works in language-specific ways, the same annotation-driven mechanism is used throughout. Swagger works by defining request parameters, routing and descriptions as JSON objects defined within the code itself, alongside the function definition. In the case of Node.js this is highlighted below (italics representing the Swagger annotations).

```
exports.putGraph = {
  "spec": {
    "description": "Adds a graph to database",
    "path":
      "/graph/datasets/{datasetid}/graphs/{graphid}",
    "notes": "",
    "summary": "Inserts a graph in BPnet or
      JSONGraph format",
    "method": "PUT",
    "params":
      [param.path("datasetid", "ID of dataset", "string"),
        param.path("graphid",
```



```

    "ID of graph", "string"),
    param.query("format",
        "Format of graph to be inserted (bpnet|
        jsongraph)", "string")],
    "responseClass": "Response",
    "errorResponses": [],
    "nickname": "putGraph"
  },
  "action": function (request, response) {
    var dsid= request.params.datasetid;
    var graphid= request.params.graphid;
    var format= request.query.format;

```

Figure 3: Swagger Annotations for Bi-Partite / JSON Graphs

The Swagger web-based user interface that allows automated testing of such APIs is shown in Figure 4 with the request URL, the body and response for testing a Graph-oriented API.

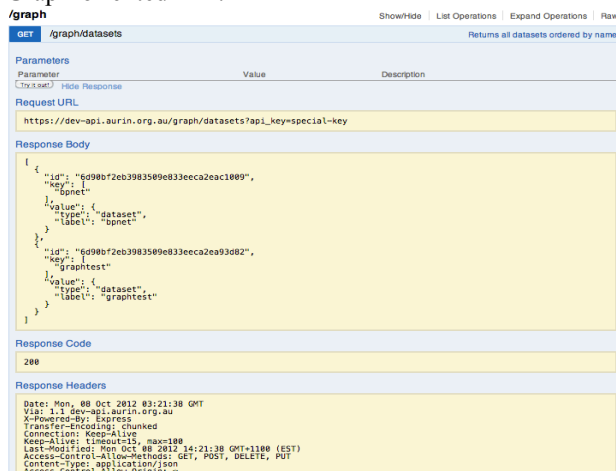


Figure 4: AURIN Swagger-based ReST Testing

It is noted that Swagger is still evolving and does not yet represent a completely mature ReST-based testing and documentation framework. Thus not every feature is used consistent across languages, e.g. full functional support of JSON Schema is not yet supported.

4.5 AURIN Integrated Testing Tools

To support the development and build environment and associated activities in software testing and integration, the AURIN project has adopted the Jenkins software environment (<http://jenkins-ci.org/>). Through Jenkins it is possible to automatically highlight the status of overall builds comprised of many independent software systems including those from AURIN lens projects and open source systems upon which these are often built as shown in Figure 5.

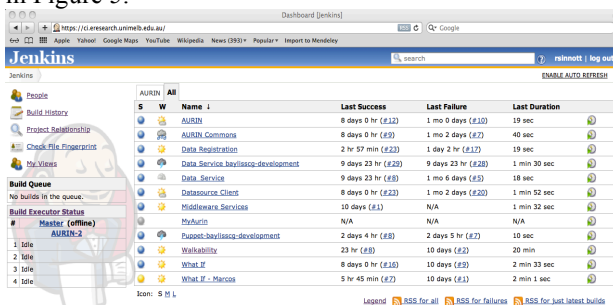


Figure 5: AURIN Jenkins-based Continuous Integration

Specifically, the AURIN project has adopted Jenkins to automate many of the typically manual processes associated with continuous software testing and integration. Advanced capabilities to support code coverage and usage are supported in Jenkins. This allows for streamlining of codes that are developed or contributed to the AURIN environment. With Jenkins, every time a new revision of the code is committed it automatically downloads, builds and tests the code in a clean environment. This ensures that any problems introduced, e.g. due to eccentricities in an individual developer's personal working environment are identified before the new code is circulated.

4.6 AURIN Common Build Environment Tools

To provide a common software build development, the AURIN core technical team has adopted the Maven (<http://maven.apache.org>) software build and management system. Maven is an open source tool that supports the building and management of Java-based projects.

Maven and its project object model (POM) utilizes a set of plugins that are shared by all AURIN projects. This provides a uniform AURIN build environment for all AURIN software developers that addresses many common challenges facing distributed software systems including support for tackling software dependencies, configuration challenges and unit tests. The project also includes core components in Maven. This provides an easy way for Maven clients to update their installations so that they can take advantage of any changes that been made to Maven itself. This latter feature allows support for installation of new or updated plugins from third parties or Maven itself.

4.7 AURIN Deployment and Management Tools

A key part of the AURIN development and management environment is to provide integrated deployment and configuration of phased implementations of systems. At present two versions of the e-Infrastructure are supported: *production* (accessible at <https://porta.aurin.org.au>) and an on-going development version of the e-infrastructure which is used for prototyping purposes and maturing the software systems to production level. This *development* version is available at <http://portal-dev.aurin.org.au>. It is planned that a further staging environment will also be rolled out in due course to help in the transition from development to production versions. The production version is deployed within the Australia Access Federation whilst the development version is available through the Australian test federation.

To support this process the AURIN project has adopted the Chef configuration and management software (<http://www.opscode.com/chef>). Chef provides a coherent management approach for the specification and delivery of deployment of e-Infrastructure components through *recipes* and *cookbooks*. These allow specification and bundling of the underlying software systems e.g. the OS versions required, the prototyped versions of software components and their dependencies and indeed the database resources and how they should be deployed onto particular virtual machines. A key advantage of Chef is

that it allows association of software bundles onto resources (VMs) with roles assigned for future access, usage and monitoring of these resources. Firewalls are also used to sandbox systems to ensure that systems developed in development and separated from production. In AURIN a single dedicated software engineer is responsible for the deployment and configuration of systems.

5 AURIN Phased Implementations

The proof of the AURIN development methodology, as with any other software engineering project can best be assessed by the successful software systems that are developed and ultimately used by the end users. The project has had two major releases of the e-Infrastructure with a third planned for mid-October 2012.

5.1 AURIN e-Infrastructure Mark-I

The first AURIN platform was largely a proof of concept system to demonstrate the viability of the approach that was to be taken. This system was described in (Sinnott 2011) and completed in a 3-month rapid prototyping effort by a small subset of developers - since at this time the full complement of the AURIN technical team was not yet in place. This Mark-I prototype version of the e-Infrastructure allowed access to a small subset of the data providers with basic visualization and analytical tools. One of the primary data resources in the Mark-I version of the AURIN e-Infrastructure was from Landgate in Western Australia. Work was also undertaken in this release using streamed social media from Twitter. The user interface and visualization/analytical capabilities for this Mark-I version of the e-Infrastructure are shown in Figure 6 where data from Landgate is being analyzed and visualized.

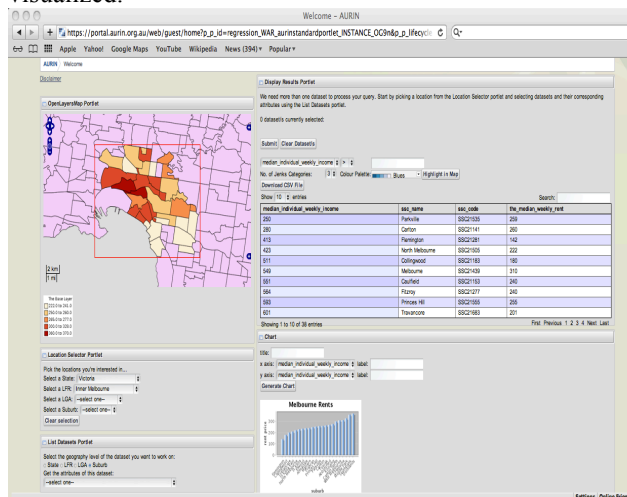


Figure 6: AURIN e-Infrastructure Mark-I (circa July 2011)

Despite its somewhat limited functionality, this initial implementation was extremely informative and influential to AURIN as a whole from a variety of perspectives. Firstly it allowed establishing a grounding and understanding of the data sets that would be dealt with in AURIN, and the geospatial information systems through which many of these data sets would be delivered. Secondly and importantly this version of the e-Infrastructure helped to bring an implementation-oriented

focus to the numerous technical and management boards involved in overseeing the project. To that point, a huge effort had been expended on discussions, documentation and planning on what the system might be. This provided a basis for customer engagement far beyond the more abstract discussion that had hitherto been taking place. Thirdly, this version of the system allowed assessment of the integration of the AURIN portal (which at the time was based upon the LifeRay portal framework) with the Australian Access Federation (www.aaf.edu.au) - the federated authentication system adopted by the project. In a similar vein this version of the e-Infrastructure enabled exploration of a suitable workflow environments and their prototyping - this was based upon the Object Modeling System version 3 (OMSv3). Experiences in developing and using OMSv3 in the AURIN context and results of applying OMSv3 on Cloud based infrastructures described in (Javadi 2012).

The majority of the software systems that formed the basis for the Mark-I version of the AURIN e-Infrastructure were largely discarded when worked commenced on the Mark-II version. There were several reasons for this, but the most important one was that the initial work was primarily to understand and articulate the problem through implementation, and to demonstrate that the overall vision was realistic and achievable. In doing this, the user interface had specifically developed/crafted user interface components (portlets) targeted to individual data sets and tools. It was rapidly recognized that this model would not scale given the volume of data to be made accessible. Similarly data and metadata considerations both in terms of access, usage storage and provenance were identified as crucial, but not supported in Mark-I. Instead it was recognized that the whole AURIN e-Infrastructure itself had to be data-driven (Sinnott 2012). To this end, a major focus was focused on the definition and realization of the architecture shown in Figure 1.

5.2 AURIN e-Infrastructure Mark-II

Driven by user demand, the Mark-II e-Infrastructure offered a variety of ways to geospatially drill into urban and built environment data and hence target the data sets of associated with particular regions (Figure 7).

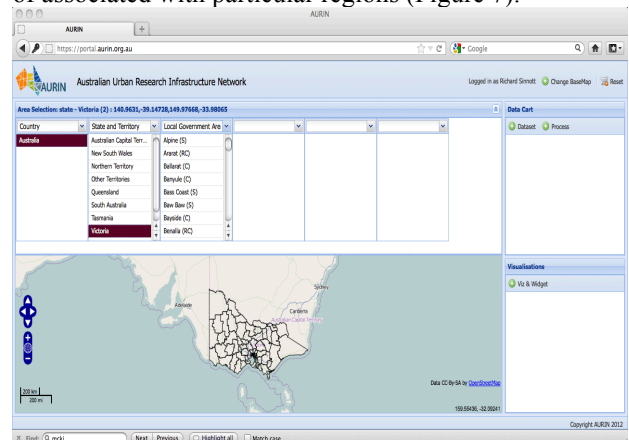


Figure 7: AURIN Mark-II Geospatial Filtering (May 2012)

This filtering of data is essential in the urban domain given the proliferation of data and information available from multiple organisations. This data selection can be

done graphically (using zoom features of the data visual interface given as a map of Australia) or through use of query interfaces that allow direct specification of the region of interest, e.g. Australia/Victoria/Melbourne as shown in Figure 7. A Google-like search interface was also offered to select particular regions or data of interest, e.g. search for data sets associated with “employment”.

To improve the overall performance of the user experience in accessing and using data, the geometrical boundaries of spatial regions, e.g. Census Districts, Statistical Local Areas, Local Government Authorities, are stored in topologically correct representations at multiple resolution levels. In particular whilst the detailed boundaries are always used for analytical purposes, generalized boundaries are used for client-side display. This radically increases the overall responsiveness of the user interface and hence user experience. Details of how this has been achieved are described in (Tomko 2012).

The Mark-II version of the AURIN e-Infrastructure incorporated a range of services and tools developed through the core technical team and through the associated externally funded subprojects. Some of the core subproject capabilities included in AURIN e-Infrastructure Mark-II release was the data registration service developed by the Centre for Spatial Data Information and Land Administration (CSDILA). This service provides an automated mechanism to harvest metadata from OGC-compliant web feature services. The service also allows for manual additions and refinements of metadata from data providers.

Several lens specific data-oriented subprojects were incorporated into the Mark-II release including some of the data sets from the Population Health Information Development Unit (PHIDU – www.publichealth.gov.au) at the University of Adelaide. PHIDU has a rich source of health and other aggregated data sets from across Australia. Voting and a range of associated classification data were included in this release from the University of Queensland eResearch Group – drawing on work previously undertaken by the ARC funded Research Network in Spatially Integrated Social Science (www.siss.edu.au). Data sets from the Centre of Full Employment and Equity (CoffEE – <http://e1.newcastle.edu.au/coffee/>) were also included in this release. Some of the data providers and the associated data sets along with their associated variables are shown in Figure 8.

Search keywords:	Available Datasets	Organisation	Attributes
Search keywords:	PSMA Product (VISA) (AUS-001)	Landgate	<input type="checkbox"/> Attribute
Area filter:	PSMA Product (CD) (Australia) (AUS-028)	Landgate	<input type="checkbox"/> sscc_code
	PSMA Product (CD) (Australia) (AUS-029)	Landgate	<input checked="" type="checkbox"/> sscc_name
	Aurion Twitter Dataset	Aurion	<input type="checkbox"/> ss_wkms
	PSMA Street Network of Victoria	PSMA	<input checked="" type="checkbox"/> total_population
	Australia by COFEE Functional Economic Region	Newcastle	<input type="checkbox"/> aged_45_to_54
	Australia by Labour Force Region	Newcastle	<input type="checkbox"/> aged_55_to_64
	Children's Public Health Data	PHIDU	<input type="checkbox"/> aged_65_to_74
	Deakin Quality of Life Data	Deakin	<input type="checkbox"/> aged_75_to_84
	Australia by Statistical Local Area	Newcastle	<input type="checkbox"/> aged_85_and_over
	PSMA Product (VISA) (AUS-001)	Landgate	<input type="checkbox"/> total_number_of_families
Filter	PROPERTY Product (SC) (Western Australia) (AUS-044)	Landgate	<input checked="" type="checkbox"/> married_persons
	Household Income by Census District (2006 Census) (AUS-068)	Landgate	<input type="checkbox"/> separated_persons
	Household Income by Census District (2006 Census) (AUS-071)	Landgate	<input type="checkbox"/> divorced_persons
	Language by Census District (2006 Census) (AUS-074)	Landgate	<input type="checkbox"/> widowed_persons
	Population by Census District (2006 Census) (AUS-077)	Landgate	<input type="checkbox"/> indigenous_persons
	Employment Statistics by Suburb (2006 Census) (AUS-078)	Landgate	<input type="checkbox"/> persons_never_married
	Population by Suburb (2006 Census) (AUS-075)	Landgate	<input type="checkbox"/> persons_never_married_marriage
			<input type="checkbox"/> persons_in_duo_duo_marriage

Figure 8: Mark-II Data Providers and Variables from a Provider

With this *shopping* interface, as with the Mark-I AURIN e-Infrastructure, data could be accessed from a range of federated (distributed) data providers and brought into the AURIN research space. Following the data shopping, i.e.

once data had been returned to the AURIN environment a range of charting (Figure 9), mapping (Figure 10) and basic analytical capabilities (Figure 11) were offered.

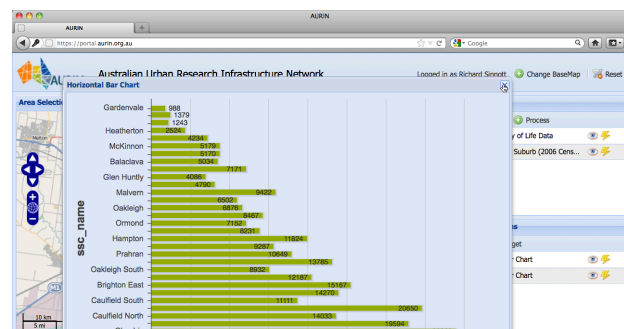


Figure 9: AURIN Mark-II Charting (May 2012) showing the total population of Statistical Local Areas in the Local Government Authority Glen Eira (from Landgate and based on the 2006 Census)

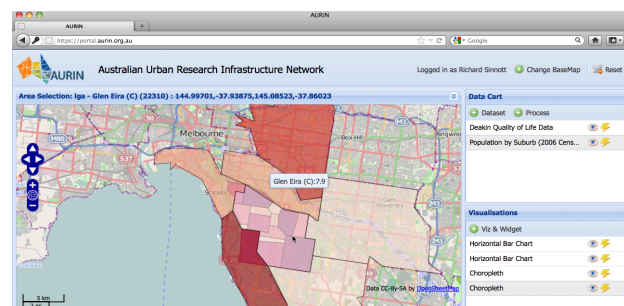


Figure 10: AURIN Mark-II Visualisation (May 2012) showing a choropleth overlaying data from the Australian Unity Quality of Life survey with specific focus on population safety, i.e. how safe do you feel in your suburb and population density for the Local Government Authority of Glen Eira

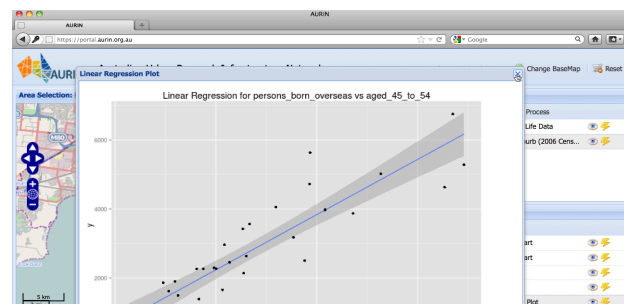


Figure 11: AURIN Mark-II Analytics (May 2012) showing the correlation (linear regression) between people born overseas and the age group 45-54 for the Local Government Authority of Glen Eira

5.3 AURIN e-Infrastructure Mark-III

The Mark-III AURIN e-Infrastructure is currently still under development (with the next formal release scheduled for mid-October 2012). This next release has been increasingly extended based upon the agile methodology that has been adopted. Included in the next release is a major increase in the number of data providers and data sets that are now provisioned. This includes a vastly extended set of data from PHIDU (with 156 separate data sets now incorporated); a variety of health survey data from VicHealth; data and services

from the Public Sector Mapping Agency (PSMA – www.pdma.com.au) including access to the Geocode National Address File (GNAF) which allows to convert a valid Australian address into geospatial coordinates (latitude and longitude). A typical scenario illustrating these advanced data sets is shown in Figure 12.

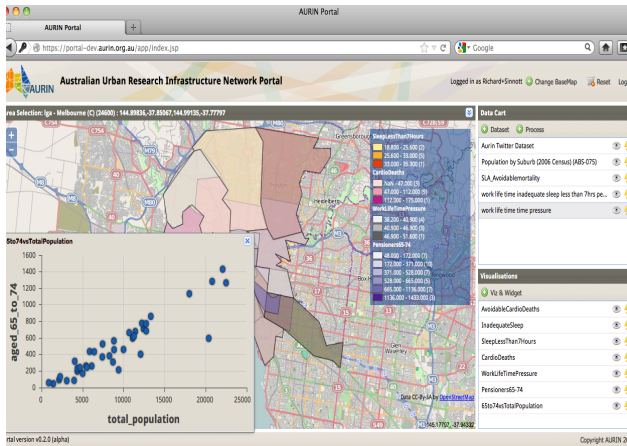


Figure 12: AURIN Mark-III and Enhanced User Interface (October 2012) showing data related to avoidable cardiovascular mortalities (PHIDU), those who sleep <7 hours and have increased work time pressure (VicHealth 2011 survey), population statistics (total population and those 65-74 years of age Landgate) for Melbourne

The user interface to the Mark-III version of the AURIN e-Infrastructure is also evolving. For example, advanced brushing techniques now allow data from map-based regions to be highlighted (and vice versa) as shown in Figure 13.

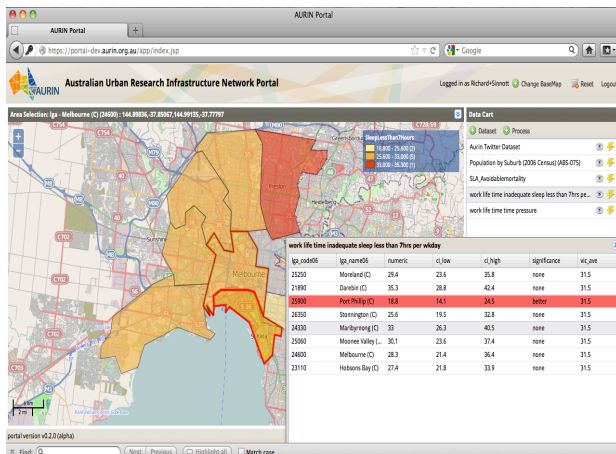


Figure 13: AURIN Mark-III and use of Brushing Techniques (October 2012) showing responses to survey questions on sleeping < 7 Hours (VicHealth 2011 survey) for Melbourne. PortPhilip is selected on the map and the associated data is highlighted.

Many of the AURIN subprojects for the first three lenses are now deep into their development activities and using the core collaboration and software management tools identified in section III. These include a range of advanced analytical capabilities from the CoFFEE group at the University of Newcastle; advanced walkability tools from the McCaughey Centre; health-based

demonstrators with VicHealth and Western Health (in Perth) amongst many others.

A major enhancement in Mark-III of the e-Infrastructure is the enhanced utilization of workflow tools (based upon OMSv3). These workflow tools now allow definition of rich workflows coupling data and analytical services reflecting and capturing scientific processes. These workflows have been initially focused upon the walkability tools and use of PSMA data for geocoding addresses, and associated simulation and analytical services (implemented as part of the walkability services).

Many of the lessons learnt within the AURIN core technical team in development of the architecture identified in Figure 1, are now transferring to the collaborating partners. However this is still a non-trivial problem. The systems that are being developed are by their very nature, complex software engineering tasks for many urban research-oriented groups. To address this, the project has attempted to provide a core team technical buddy to the remote software engineering efforts. Given the number of projects expected to be running concurrently in 2013, this model will be stress tested with single technical staff members having to work across a multitude of domain-specific lens projects. Furthermore, to ensure that the expertise of any given core team technical developer is shared, the Project Manager/*ScrumMaster* has deliberately paired team members to work on each others individual software activities. This provides redundancy to the team and a shared understanding of the overall development activities.

In moving from the Mark-II to the Mark-III e-Infrastructure release, the access to and use of the AURIN e-Infrastructure has remained largely consistent, i.e. access is through the Australian Access Feature and users have to shop for data once a particular geospatial region has been selected. This consistency is an important feature to maintain to ensure that returning end users can benefit from improvements in the e-Infrastructure capabilities and not have to learn new user interfaces or techniques to access and use the system more generally.

6 CONCLUSIONS

One of the major challenges of distributed systems development that is often overlooked is not the distributed systems and hardware/software resources themselves, but the distributed teams that are often involved in these development activities. Tools to optimize the way in which these teams can coordinate their activities are essential yet are surprisingly not well recognized and adopted. This is in particular true as the types of *human* distributed collaborations vary, as much as software distributed collaborations. From teams where the members are (spatially) distributed, but belong to the same project and their resources, skills, and tools are matched, through teams that are contracted to deliver a specific type of software component based on well defined acceptance criteria, to merely *federated* contributors, i.e. software development resources that are leveraged because of their availability without the possibility to influence their direction or adherence to common project management and coding standards. This

last case represents many voluntary contributors from the open source development community, to large data providers that provide data resources to the general public, where AURIN cannot mandate the APIs and protocols that are used.

The AURIN work is far from complete – as noted the project runs to mid-2015. However the foundations for distributed collaborations and the processes that have been adopted from the project outset are now bearing fruit. Without these software development and coordination foundations, major risks would arise that could threaten the success of the project as a whole. The requirement to adopt key tools by the internal and external groups has meant that the overall software integration, management and coordination effort has been greatly simplified. Thus it is directly possible to check when a delivered piece of software meets the required integration testing for inclusion into the e-Infrastructure. As noted, it should be emphasized that these tools do not remove the overall challenges in developing and delivering distributed systems involving distributed teams, rather they are a mechanism to help to manage these challenges.

The AURIN project is running contemporaneously with many major e-Infrastructure investment activities that are currently taking place across Australia. Most notably are the \$50m Research Data Storage Infrastructure (RDSI – www.rdsi.uq.edu.au), which has a specific focus on supporting storage of nationally significant research data sets, and the \$47m National eResearch Collaboration Tools and Resources (NeCTAR – www.nectar.org.au) project, which has a specific focus on eResearch tools, collaborative research environments and Cloud infrastructures. The AURIN project has been engaging directly with these projects and related projects, e.g. the Australian Access Federation, in delivery of much of its underpinning infrastructure. For example, the AURIN portal and many of the associated services have been made available on virtual machines made available through NeCTAR. However given the ramping up of these projects, early issues with these projects has already arisen. To mitigate these risks and avoid the total reliance on VMs from NeCTAR or storage from RDSI, the AURIN project has purchased its own hardware systems, which are now used to augment the offerings of NeCTAR and RDSI.

The AURIN e-Infrastructure is very much a supporting activity. That is, the work in the e-Infrastructure development is not targeted at delivering novel IT solutions per se nor exploring research challenges in e-Infrastructure development, but on supporting the urban and built environment research community in *their* research needs. The AURIN research community is extremely diverse (with over 500 registered individuals and organisations) crossing a multitude of research disciplines. Whilst their feedback (positive and/or negative) will ultimately shape the AURIN e-Infrastructure, it is hoped that the underlying agile software engineering and tools described here will persist throughout the AURIN project lifetime and allow rapid evolution of systems in a tool supported manner.

6.1 Acknowledgments

The authors would like to thank the AURIN groups and committees that are directly shaping these efforts. The AURIN project is funded through the Australian Education Investment Fund SuperScience initiative. We gratefully acknowledge their support. In addition to the co-authors, the AURIN team comprises Ivo Widjaja (Portal/User Interface); Gerson Galang (Data e-Enabler); Jos Koetsier (Data/Metadata e-Enabler); William Voorsluys (Workflow e-Enabler); Damien Mannix (Infrastructure Support); Philip Greenwood (Statistical Geospatial Developer); Marcos Nino-Ruiz (Geospatial e-Enabler) and Sulman Sarwar (Middleware/Business Logic).

7 References

- Stojanović, Z., Dahanayake, A., *Service-oriented software system engineering: challenges and practices*, Idea Group Publishing, 2005.
- Boehm, B.W., *A spiral model of software development and enhancement*, Computer, vol. 21, Issue 5, May 1988.
- Filman, R., Elrad, T., Clarke, S., *Aspect-oriented software development*, Addison-Wesley Professional, 2004.
- Booch, G., *Object-oriented Development*, IEEE Transactions on Software Engineering, vol. 12, Issue: 2, Feb. 1986.
- Martin, R.C., *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall 2003.
- Sinnott, R.O., Galang, G., Tomko, M., Stimson, R., *Towards an e-Infrastructure for Urban Research Across Australia*, IEEE e-Science Conference, Stockholm, Sweden, December 2011.
- Schwaber, K., *Agile Project Management with Scrum*, Microsoft Publishing, 2009.
- Javadi, B., Tomko, M., Sinnott, R.O., *Decentralized Orchestration of Data-centric Workflows Using the Object Modeling System*, 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012), Ottawa, Canada, May 2012.
- Sinnott, R.O., Bayliss, C., Galang, G., Greenwood, P., Koetsier, G., Mannix, D., Morandini, L., Nino-Ruiz, M., Pettit, C., Tomko, M., Sarwar, M., Stimson, R., Voorsluys, W., Widjaja, I., *A Data-driven Urban Research Environment for Australia*, IEEE e-Science Conference, Chicago USA, October 2012.
- Tomko, M., Sinnott, R.O., Bayliss, C., Galang, G., Greenwood, P., Koetsier, G., Mannix, D., Morandini, L., Nino-Ruiz, M., Pettit, C., Sarwar, M., Stimson, R., Voorsluys, W., Widjaja, I., *The Design of a Flexible Web-based Analytical Platform for Urban Research – Systems Paper*, ACM International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2012), Redondo Beach, USA, November 2012.

A Web Portal for Management of Aneka-Based MultiCloud Environments

Mohammed Alrokayan and Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory,
Department of Computing and Information Systems,
The University of Melbourne, Parkville, Victoria 3010, Australia

Emails: a@alrokayan.com, rbuyya@unimelb.edu.au

Abstract

Many Cloud providers offer services with different sets of configurations and settings. This makes it difficult for their clients to seamlessly integrate various services from different Cloud providers. To simplify, we developed an extendible Cloud Web Portal (CWP), a comprehensive open source Cloud management portal that aims to deliver a foundation for researchers and developers to prove a research concept, test a code or deliver a product in a fast and easy to use graphical user interface. Also, it aims to seamlessly integrate different Cloud services by providing a flexible architecture and design system. CWP is based on Aneka, which allows developers to use a set of .Net-based APIs for monitoring, billing/accounting, scheduling, and provisioning. Aneka provision services from private, public or hybrid Clouds. Our evaluation results show that Aneka scheduling algorithm performs efficiently for executing tasks in distributed machines.

Keywords: Cloud Management, Scheduling, Provisioning, MultiCloud, Portal, Cloud Services.

1 Introduction

Contemporary Cloud computing solutions, both research projects and commercial products, have mainly focused on Infrastructure as a Service (IaaS) model due to the uncertainty in the other models like Platform as a Service (PaaS). This uncertainty is caused by the lack of proper standards in the IaaS level especially in terms of APIs for federated Clouds. This drives the users to develop their own platforms and portals from scratch trying to use multiple IaaS providers' APIs.

As a result, users need an open portal that is flexible to adapt this uncertainty and regular variations in Cloud infrastructures. Our Cloud Web Portal (CWP) aims to provide an open source portal for easy adjustment to adapt the frequent changes in Cloud computing technology. CWP has been developed using Microsoft ASP.NET MVC 4¹ with Razor syntax (Galloway et al. 2011). It has the capability to build, test, deploy, and scale applications easily and rapidly.

There are three main common types of Cloud PaaS: Firstly, PaaS for application deployment, where

a product is deployed in distributed resources over the Cloud, for example: web applications, which is the most common type of application to be deployed on the Cloud. Secondly, PaaS for batch processing and scalable grid computing over the Cloud, where a batch of files is processed in distributed machines to accelerate the application performance. Thirdly, PaaS for multi and hybrid Cloud management where users manage multiple resources from different providers through one interface. CWP is not for application deployment as in type one of Cloud PaaS. However, it supports the other two types. It supports the second type of Cloud PaaS by using the Aneka framework² for three different kind of programming models: Thread, Task and MapReduce. Also, it supports the third type of Cloud PaaS by using the provisioning library of Aneka which support three different providers: Amazon AWS, Microsoft Windows Azure and GoGrid (Wei et al. 2011).

CWP provides several services and components for developers, researchers and deployment teams to integrate their work through a graphical web portal. Those different services and components will be discussed in Section 4. Our evaluation method, encompassing cases with up to 80 experiments using three different parameters, shows that the Aneka scheduling algorithm perform efficiently for batch processing in distributed machines, especially when the number of workers is increased. Surprisingly, with all the network latency and overhead to send and receive data, the 49 image rendering tasks does not have significant effect on Aneka performance as shown in Section 8.

The key **contributions** of our paper are: 1) an extensible architecture for Web portal for cloud computing environments, 2) a methodology for creation of adapters or widgets for monitoring or interacting with different cloud platforms, 3) a prototype software system demonstrating these capabilities and their mapping to the Aneka cloud application platform, and 4) a detailed evaluation and demonstration of our portal functionalities by deploying in a hybrid cloud environment by utilizing Melbourne private cloud and Amazon EC2 resources.

The rest of the paper is organized as follows: In next section, we present various open source cloud projects and how they are compared to CWP. Then in Section 3 we discuss the motivations behind CWP. Section 4 describes the different components and services of CWP and how they are integrated with Aneka. Then we discuss the CWP graphical user interface usability for the end-users and the flexibility of the underlying code for developers in Section 5. An example of a page request has been illustrated in a sequence diagram in Section 6. Then in Section 7, based on the NIST definition of Cloud Deployment

Copyright ©2013, Australian Computer Society, Inc. This paper appeared at the 11th Australasian Symposium on Parallel and Distributed Computing (AusPDC 2013), Adelaide, South Australia, January-February 2013. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 140, Bahman Javadi and Saurabh Kumar Garg, Ed. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

¹ASP.NET MVC 3: <http://www.asp.net/mvc/mvc3>

²Manjrasoft Pty Ltd <http://manjrasoft.com>

Models, we summarize how CWP and Aneka support all levels of deployment models. In Section 8, an evaluation of CWP shows the Aneka scheduling performance using several parameters and statistics. We close in Section 9 with conclusions and future work.

2 Related Works

Cloud computing has been driven mainly by the industry; as a result the most common and useful services can be found there. This section is a result of a study researching seven Cloud computing projects. It shows two main features: the service and deployment models. A summary of the related works is shown in Table 1 along with Cloud Web Portal for comparison. In our research, we focus on the deployment model which will be explained in Section 7. The following is a summary of the seven projects:

CloudFoundry³

It is a portal to deploy applications in distributed machines over a Cloud. It supports limited number of public Cloud providers. Also, it supports OpenStack⁴ private Clouds. A commercial public Cloud version of CloudFoundry project by VMWare can be found at CloudFoundry.com⁵, which supports only one public Cloud provider even though the open source version of the project supports Multi-Public Cloud. CloudFoundry is for application deployment, not for MultiCloud management like Delta Cloud project.

Delta Cloud⁶ and jCloud⁷

These two projects are different than the others, they are libraries to manage Multi-Public Cloud providers and manage hybrid infrastructure services using one API instead of dealing with multiple APIs for different providers. Delta Cloud is written in Ruby and it supports wide range of private Cloud projects and public Cloud providers, which allows users to manage several instances through one REST-based API for simple any-platform access. jCloud is almost the same as Delta Cloud except than it is written in Java.

3 Motivations

Not all Cloud platforms and portals fall into one category, CloudFoundry, Microsoft Windows Azure⁸, CloudBees⁹, and Google App Engine¹⁰, for example, are platforms for application deployment and they vary according to which programming language the user want to deploy on the Cloud. Other platforms, like RightScale¹¹, is for Multi-Public Cloud management where multiple VM instances from multiple providers and monitoring can be provisioned via one management console.

Cloud Web Portal (CWP) is different from the others, on top of Multi-Public Cloud management, it allows users to run batch processing jobs over the Cloud in parallel and distributed manner leveraging Aneka framework (Vecchiola et al. 2012). Aneka supports

three different programming models: Task, Thread and MapReduce. CWP provides a flexible user interface for Cloud administrators, developers and researchers to manage multiple nodes in a Cloud. Also, it aims to give non-technical users an easy to use fully functional dashboard to view a summary of the current system status, and allow them to apply changes to the Cloud system according to their granted permissions. CWP allows the Cloud developers to add any feature to the portal or develop an application by implementing what we call "widgets" (which is Detailed in Section 5) and add it to the portal to be used instantly.

4 CWP Architecture and Services

CWP has been designed and developed to provide developers with an easy to understand code structure. Also, it has a flexible architecture for any future changes or adjustment to the portal. CWP gives portal users a Multi-Public Cloud support to provision public VMs from multiple different providers. Also, it has the capability to execute batch processing jobs over the Cloud according to the scheduling policy that the user chooses using Aneka APIs.

The CWP architecture in Figure 1 shows the different CWP services that users and developers can use and integrate. Aneka provides the main services for CWP such as monitoring, SLA-Resource management and resource provisioning whether those resources are local desktop machines, private Cloud, public Cloud or hybrid Cloud. CWP graphical user interface and its usability is described in the next section.

4.1 Provisioning

CWP provisions resources from all different deployment models using Aneka (?). Aneka uses different scheduling algorithms, for example, the deadline scheduling algorithm (Vecchiola et al. 2012) which allows Aneka to provision to public Clouds dynamically when the local desktop grid is not enough to execute the job within its given deadline. Another example of an algorithm is the budget-based algorithm that aims to limit the resources that are provisioned and chooses a set of resources that can finish the job within the specific budget.

4.2 Monitoring

Aneka monitors the running nodes through a heartbeat-based approach that performs periodic checks on the availability of the node. If one of the worker nodes fails executing a task, Aneka migrates that task to another node to be executed. If a master node fails, a discovery system looks for another master within the defined Cloud domain to take over and continue scheduling the tasks starting from where the previous master node stopped.

Service Measurements

CWP measures the services from different Cloud providers. CWP suggests the best service(s) to provision based on the users' QoS, budget, and monitored data. The SMICloud (Garg et al. 2011) algorithm has been used for multi-criteria selection of different Cloud services but has not been integrated with Aneka yet.

³Cloud Foundry Open Source: <http://cloudfoundry.org>

⁴Apache OpenStack: <http://www.openstack.org>

⁵Cloud Foundry: <http://cloudfoundry.com>

⁶Apache Deltacloud API: <http://deltacloud.apache.org>

⁷Apache jclouds: <http://www.jclouds.org>

⁸Microsoft Windows Azure: <http://www.windowsazure.com>

⁹CloudBees Java PaaS: <http://www.cloudbees.com>

¹⁰Google App Engine: <https://cloud.google.com/products>

¹¹RightScale Cloud Management: <http://www.rightscale.com>

Table 1: Related Works Projects

Project	Service Model	Deployment Model	Language	Licence
CloudFoundry	Portal for application deployment	Private/Multi-Public Cloud management	Java	Apache
Delta Cloud	Ruby Library for Multi-Public Cloud support	Multi-Public Cloud	Ruby	Apache
jCloud	Java Library for Multi-Public Cloud support	Multi-Public Cloud	Java	Apache
Cloud Web Portal	Cloud portal for Multi-Public Cloud management and batch processing	Private/Multi-Public Cloud/Hybride-Cloud	.Net	Apache

SLA-Based Resource Allocation and Provisioning

Aneka can allocate resources based on the users' QoS. Unfortunately, due to the current limitations, almost all public Cloud service providers provide a static SLA, mainly for availability, that can not be negotiated. However, an Aneka prototype performance results show the feasibility and effectiveness of SLA-based resource provisioning in Clouds (Buyya et al. 2011).

4.3 Scheduling

Aneka support three different programming models: Task, Thread, and MapReduce associated with different scheduling algorithms based on time (Vecchiola et al. 2012) and budget. A user or a developer can implement or execute applications mixing any of the programming models with any of the scheduling algorithms. This kind of scheduling is important, for example, for scientist to execute batch processing type of applications and for graphical designers to render images or videos in a shorter time.

4.4 Billing/Accounting

Aneka is Market-oriented system, which can be used by a Cloud broker. Aneka has a fully functional billing and accounting system mainly for desktop grids. There are two different models: Pay-Per-Task or Pay-Per-Resource. The first one is to set a price for a task on each node to charge the users or the brokers depending on how many tasks they have executed. The second model is to set a price for the hourly usage on a node regardless on how many tasks the user will execute.

5 CWP Interface and Component Usability

CWP helps developers to create Cloud applications with decoupled components (input logic, GUI logic, and business logic). The loose coupling among the three main components of CWP provides the ability for parallel development, flexibility in changes and fast debugging. CWP infrastructure has been designed and implemented not just for the developers and researchers but also for any non-expert users to use the portal easily. Also, it brings flexibility in development and usability for the end-users to use its graphical user interface especially the concept of widgets and dashboard. This flexibility and usability in design allows the Cloud developers to edit any existing feature, widget or application and add almost any desired one easily.

The error messages are shown on the top right corner of the portal, which we call Issues Centre. It

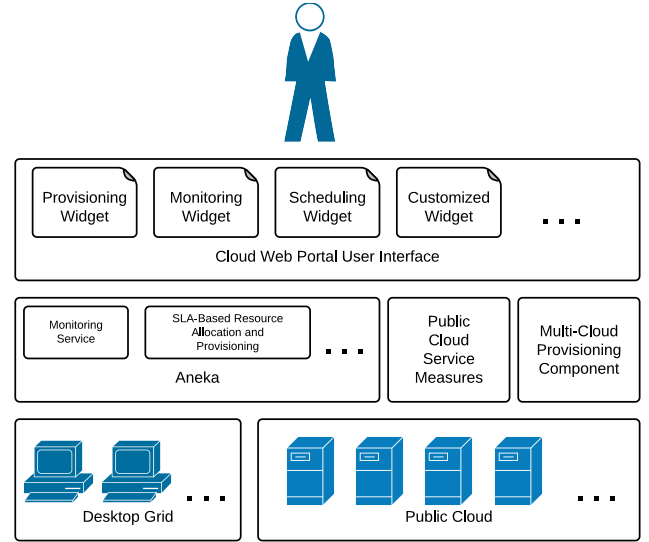


Figure 1: CWP Architecture

gives the portal users a summary and a quick overview of what are the current errors and warnings. As soon as the user clicks on any of the errors or warnings it shows a dialog to solve the problem easily. In addition to the Issues Centre, CWP has the Activity Centre that shows to users a summary and a quick overview on the current running tasks. Next we discuss further the Widgets, Dashboard, Issues Centre and Activity Centre.

5.1 Widgets and Dashboard

Widgets give the developers a quicker method to develop Cloud applications leveraging all the components that have been mentioned in Section 4. The concept of widgets gives the users the flexibility to adjust the graphical user interface and to modify the business logic easily. Each widget is designed to be wrapped-up with `<article>` and `<section>` HTML tags; those wrapped-up tags specify the widgets' configurations, such as the width. A widget creates a box of information or form depending on the developer design. The width of a widget is between 1 and 12, so 12 is the full width of the browser window, see Figure 2 for example.

The HTML code in Figure 2 displays two widgets, each one fills half the width of the user's browser window (`class="grid_6"`). The first widget that will be shown is the `_Clouds` controller and "Details" action to shows details for a specific Cloud as shown in Figure 4 (we are also passing the Cloud id `id = @Model.CloudId`). The code of this widget can be found in the file `"Controllers_CloudsController.cs"`

```

<article class="container_12">
  <section class="grid_6"><br />
    <div class="block-border">
      @Html.Action("Details", "_Clouds",
        new { id = @Model.CloudId })
    </div>
  </section>

  <section class="grid_6"><br />
    <div class="block-border">
      @Html.Action("CPU_Utilization_Range",
        "_Charts",
        new { id = @Model.CloudId })
    </div>
  </section>
</article>

```

Figure 2: An example of an HTML code to display two widgets

```

<div class="block-content">
  <h1>New CWP Widget Title</h1>
  <div class="infos">
    //Any HTML or ASP.Net/RAZOR code
  </div>
</div>

```

Figure 3: An example of a widget code

in function *"Details"*, which returns a View object. The second widget is almost the same as the first one, it calls the *"CPU_Utilization_Range"* controller and *"_Charts"* action to draw a CPU utilization chart for a specific Cloud as shown in Figure 5. This shows how easy it is to customize and edit any widget in CWP. A new widget can be added by creating a new HTML file and following a specific format to match the CSS of CWP, for example, an HTML file content is shown in Figure 3

5.2 Issue and Activity Center

The issues and activity centers were designed to monitor the Cloud resources and to keep checking the status of Aneka workers and masters. Also, it shows the ongoing tasks for the provisioned machines, masters and/or workers.

Issue Center

The issues centre, as shown in Figure 6, checks the status of the CWP resources, whether they are active or failed last execution. Also, it displays an error or a warning message according to how critical the issue is. Issue center gives the users the ability to fix any issue easily by clicking on any of the listed issues to popup a dialog box to solve it.

Activity Center

The activity centre, as shown in Figure 7, checks if there is any tasks in progress. Each one of those tasks is clickable to open a popup dialog box to show the users more information about the selected task.

6 CWP Sequence Diagram

CWP sequence diagram shows how its components operate with each other and in what order. The sequence diagram in Figure 8 was simplified to show details of only one widget (*_Clouds\Details*). The rest of the widgets follow almost the same approach.

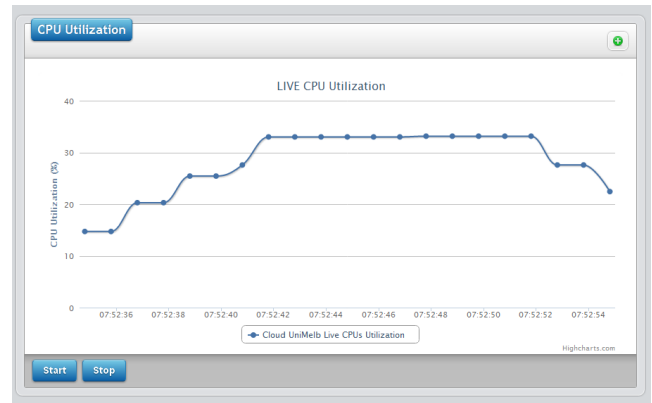


Figure 5: Live CPU utilization widget

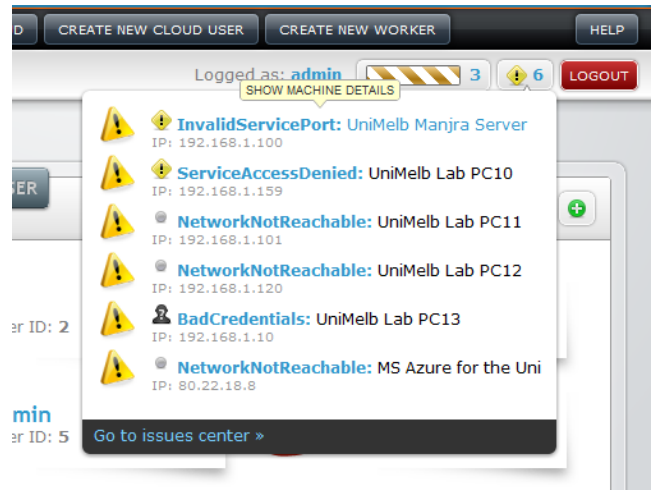


Figure 6: Different issues have been addressed by CWP Issue Center

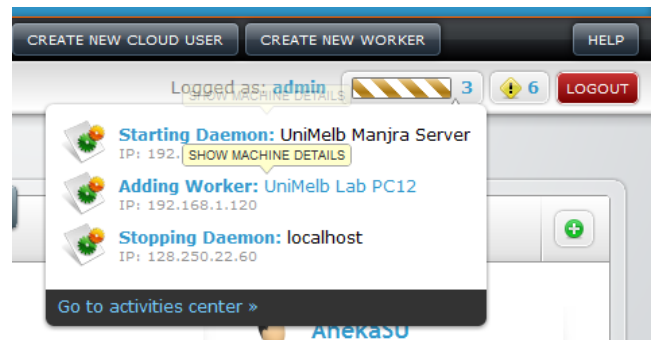


Figure 7: Activity Center shows different stages of ongoing tasks

The sequence diagram in this section is an example of a request to this page: *"\CloudManagement\CloudDetails\1"*, where the web server calls *"CloudDetails"* action in *"Cloud-Management"* controller passing the Cloud id *"1"*. The controller requests a Cloud object from the Entity Framework (EF) to send it to the CWP view: *"CloudManagement\CloudDetails"*. This view calls three widgets as shown in Figure 8. The first widget is: *"_Clouds\Details\1"*, which shows the master and a list of workers along with some information about the selected Cloud, like running services. The widget sends several requests to the Entity Framework to get such information. Then the widget returns

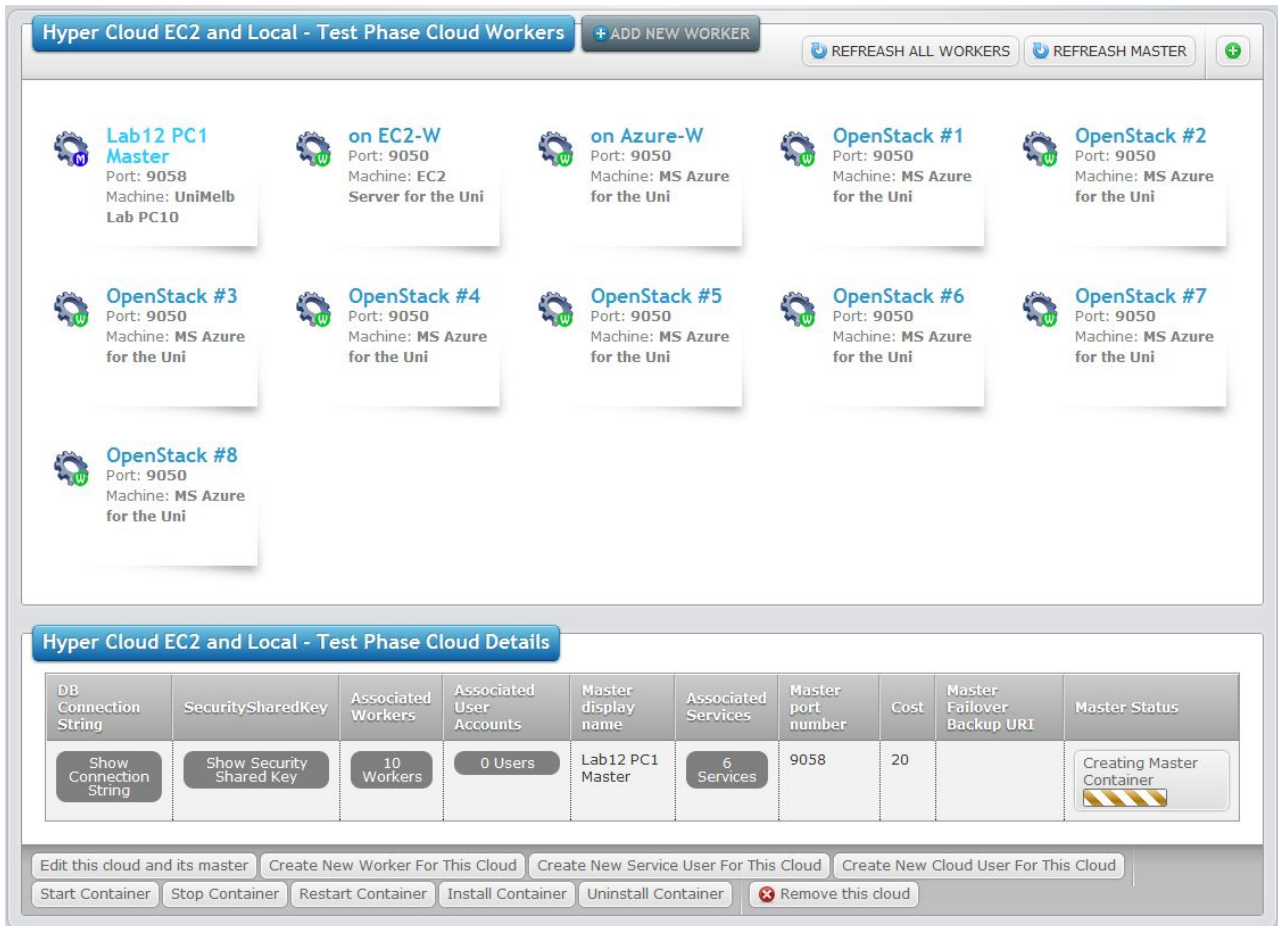


Figure 4: Cloud Details widget shows 10 workers and one master in a Cloud

HTML content to the view. The view repeats the same approach to get the HTML content from all the widgets. Finally, after the view wraps all the HTML contents, it sends the completed HTML page to the user.

Components in Figure 8 are decoupled to be replaced or extended. This gives the portal users and developers a flexibility to build adapters to suit their work or research.

7 Deployment Models

NIST defines four different Cloud deployment models (?), which CWP supports via Aneka along with the Desktop Grid model.

7.1 Desktop Grid

Aneka utilizes the unused computational power of desktop Personal Computers (PCs) connected on local area networks (LAN), virtual LAN, or over the Cloud. The main objective of a desktop grid is to accelerate application execution, especially distributed-aware applications that split a job into tasks. Aneka provides several .Net libraries for developers to develop this type of applications to be distributed among Aneka workers. Desktop grid allows organizations to utilize unused PCs resources without affecting the productivity of PC users.

7.2 Public Cloud

This is the most common model of the Cloud which allows several users to share the same infrastructure to reduce the cost and utilize the shared resources. The main features of the public Cloud are the resource availability, elasticity and cost efficiently. NIST defines public Cloud as "The Cloud infrastructure is provisioned for open use by the general public" (?). Aneka provisions public Cloud resources from Amazon EC2, Microsoft Windows Azure and GoGrid (Wei et al. 2011) to execute batch processing among them.

Multi-Public Cloud

Research on MultiCloud (Xiong et al. 2011), Inter-Cloud and Cloud Federation (Celesti et al. 2010) have emerged questioning the openness of the Cloud and discussing avoiding vendor lock-in. Also, deploying application on MultiCloud reduces the chance of outage by leveraging multiple running application servers on different providers, and switching to the most efficient one in case of any failure. In web applications, for example, MultiCloud application servers allow the system to redirect the users requests to where they can get the best experience by shortening the response time and increasing the availability¹². CWP uses Aneka to provision infrastructure Multi-Public Cloud resources from any of Amazon EC2, Microsoft Windows Azure and/or GoGrid (Wei et al. 2011). Those resources can be managed in a single resource pool or multiple resource pools.

¹²Cedexis: <http://www.cedexis.com/country-reports/>

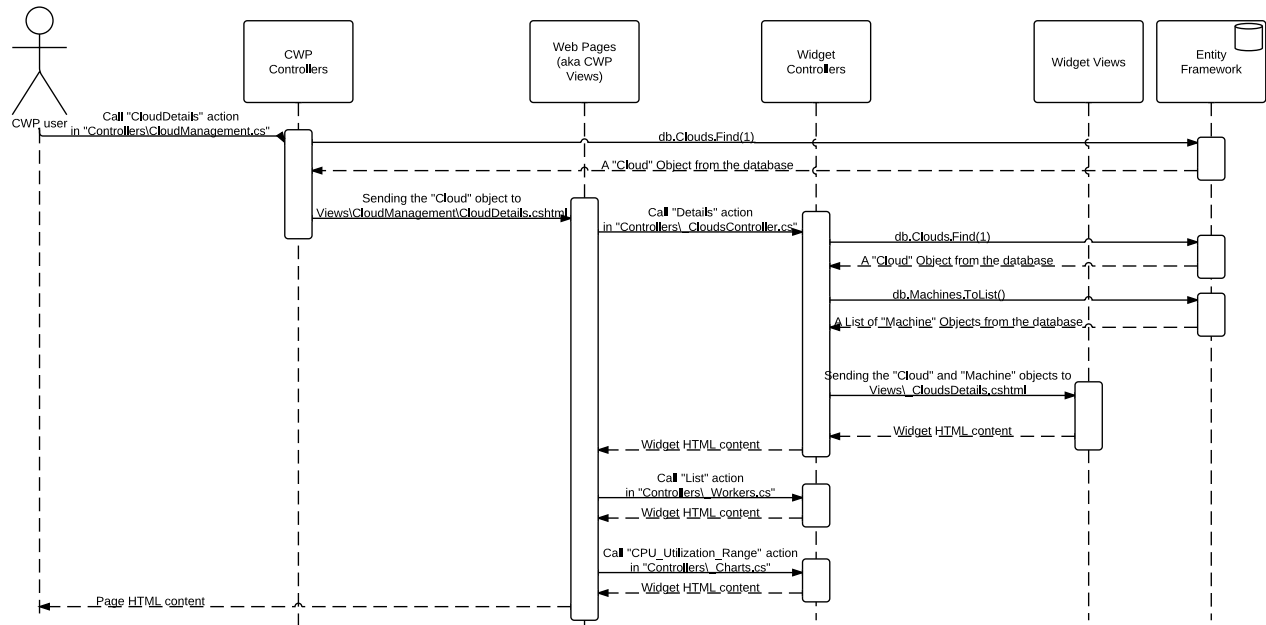


Figure 8: CWP Sequence Diagram Requests Page: "\CloudManagement\CloudDetails\1"

7.3 Private Cloud

Many open source projects (Cloud Stack¹³, Deltacloud, Eucalyptus (Nurmi et al. 2009), Open Nebula¹⁴, Open Stack, and Cloud Foundry) emerged giving organizations the opportunities to host a private Cloud. It provides more control on the data and increases security. NIST defines private Cloud as: "The Cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units)." (?). Almost any private Cloud project can be public when hosted and exposed to the general public as a service. Manjrasoft Aneka has an ongoing project to support automated provisioning for OpenStack private Cloud.

Outsourced Private Cloud

Private Cloud usually refers to the on-premise private Cloud where an organization hosts the head node locally and the rest of the resources on the public Cloud. However, some providers offer private Cloud service that can be almost 100% outsourced, which is basically a public Cloud service that has been adjusted to isolate the infrastructure of the service to be dedicated to single organization. This model offers the security strength of the private Cloud and the availability, elasticity and cost efficient of the public Cloud.

7.4 Hybrid Cloud

NIST defines hybrid Cloud as: "The [hybrid] Cloud infrastructure is a composition of two or more distinct Cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., Cloud bursting for load balancing between Clouds)" (?). Aneka has the ability to provision resources in hybrid Cloud (Vecchiola et al. 2012) where multiple tasks are

distributed between local desktop grid PCs and also Multi-Public Cloud.

8 Performance Evaluation

We evaluate Aneka performance, and the efficiency of its scheduling algorithm. Aneka schedules jobs that consist of groups of tasks among workers via a master node.

Seven small size Amazon EC2 instances were used, one master and six workers. All the machines have the same specification. Our evaluation method encompasses cases with up to 80 experiments using three different parameters: number of workers, programming models, and number of tasks. The parameter of interest is the time in seconds that Aneka takes to finish executing a job. Four different numbers of workers have been used: one (representing the sequential execution), two, four and six workers. Two programming models were used: Thread, using the Mandelbrot application, and Task using a distributed version of POV-Ray application. Both applications, Mandelbrot and POV-Ray, use Aneka APIs for scheduling. Two number of tasks have been used: 25 tasks (which is rendering an image with 5 columns and 5 rows) and 49 (which is 7x7 image rendering).

Comparing the sequential execution of a job (number of workers = 1) with the parallel execution (number of workers = 2, 3, and 6), Tukey simultaneous tests shows P-Values close to zero, which means statistically that there is a significant difference between the different number of workers. Also, ANOVA General Linear Model test for the time in second versus the number of workers shows $P\text{-Value} < 0.001$, which means that the time and number of workers are strongly related and each one of them affects the other, so having more workers means a huge reduction in time for executing a job.

Also, Two-Sample T-Test for the time and number of tasks shows a $P\text{-Value} < 0.001$ and 95% CI for difference (34.51, 61.35), which means that the number of the tasks effects significantly the time that Aneka needs to execute a job. As a result, both number of workers and number of tasks have a great impact

¹³Cloud Stack: <http://cloudstack.org/>

¹⁴Open Nebula: <http://opennebula.org>

For Task Programming Model:

$$\text{Time} = 60.1007 - 9.56779 * \text{Num of workers} + 0.780371 * \text{Num of tasks}$$

For Thread Programming Model:

$$\text{Time} = 12.1695 - 9.56779 * \text{Num of workers} + 0.780371 * \text{Num of tasks}$$

Figure 9: Regression analysis for the two programming models with the number of tasks and the number of workers

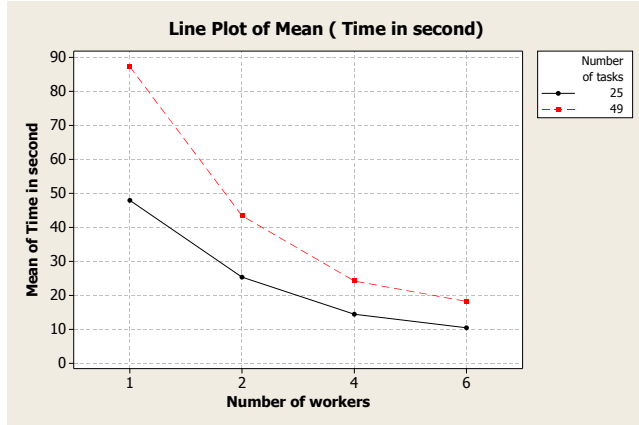


Figure 10: LinePlot of mean time in second

on the time for Aneka to finish executing a job, and a correlation analysis has been performed that supports this result.

Looking at the two programming models, the regression analysis shows the equations in Figure 9. That means both models have the same spread but the thread programming model (60.1007) always executes faster than the task programming model (12.1695).

The line plot (Figure 10) shows how the increase of the number of workers to execute the tasks in parallel reduces the time that Aneka took to execute a job. Also, the sequence execution (number of workers = 1) has a wide gap between 25 and 49 tasks, while it is the opposite on parallel execution when we have 2, 4 or 6 workers, and the gap get closer when we have more workers.

The box plot (Figure 11) of the time Aneka takes to execute a job grouped in the number of tasks and the number of workers - shows how the number of workers decreased the time the Aneka takes to execute a job significantly. Also, the difference between the two box plots within the same number of workers is large in the sequence execution (number of workers = 1) while it becomes narrower when we increase the number of workers.

As a result of this experiment, Aneka scheduling algorithm has been proven to perform efficiently for executing tasks in distributed machines, especially when the number of workers is increased. Surprisingly, with all the network latency and overhead to send and receive data, the 49 image rendering tasks does not have significant effect on Aneka performance compared to 25 tasks as shown in the box plot Figure 11.

9 Conclusion and Future Work

Cloud Web Portal (CWP) is an open source Cloud management portal for researchers and developers to

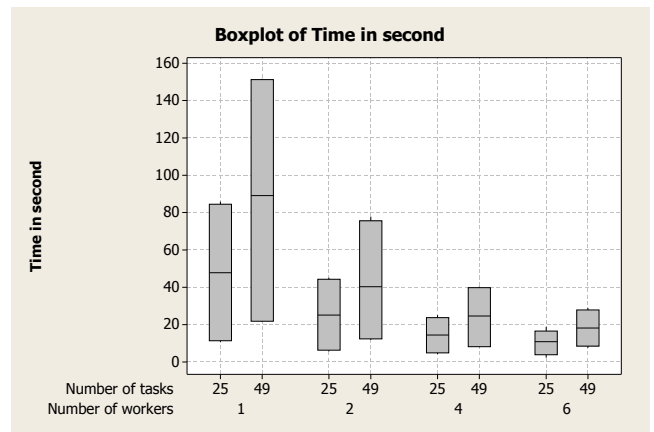


Figure 11: Boxplot of time in second

prove a research concept, test a code or deliver a product. CWP uses Aneka as it is framework, which gives CWP several features especially in monitoring, billing/accounting, scheduling, and provisioning of local desktop PCs, private, public or hybrid Cloud. Aneka can be used by the developers using its APIs. Our evaluation method encompassing cases with up to 80 experiments using three different parameters show that Aneka scheduling algorithm perform efficiently for executing tasks in distributed machines, especially when the number of workers is increased. Surprisingly, with all the network latency and overhead to send and receive data, the 49 image rendering tasks do not have significant effect on Aneka performance compared to the 25 tasks.

As CWP is extendible, it is possible to build adapters for mapping its capability to other Cloud platforms. One of the undergoing and future works is the support for Multi-Public Cloud service measurement and keep tracking of the Cloud condition to select and allocate Multi-Public Cloud resources more accurately based on the users QoS and budget – by implementing SMICloud (Garg et al. 2011) on CWP.

Software Availability

A software of Cloud Web Portal (CWP) represented in this paper can be downloaded from <http://www.cloudbus.org/cwp>.

Acknowledgements

The authors would like to thank Dr. Rodrigo Calheiros and Nikolay Grozev for their valuable comments for improving the paper.

References

- Buyya, R., Garg, S. & Calheiros, R. (2011), SLA-Oriented Resource Provisioning for Cloud Computing: Challenges, Architecture, and Solutions, *in* 'Cloud and Service Computing (CSC), 2011 International Conference on', pp. 1–10.
- Celesti, A., Tusa, F., Villari, M. & Puliafito, A. (2010), How to Enhance Cloud Architectures to Enable Cross-Federation, *in* 'Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on', pp. 337–345.

- Galloway, J., Haack, P., Wilson, B. & Allen, K. S. (2011), *Professional ASP.NET MVC 3*, 1 edn, Wrox. The book in Kindle.
- Garg, S., Versteeg, S. & Buyya, R. (2011), SMICloud: A Framework for Comparing and Ranking Cloud Services, in 'Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on', pp. 210 –218.
- Nurmi, D., Wolski, R., Grzegorzczak, C., Obertelli, G., Soman, S., Youseff, L. & Zagorodnov, D. (2009), The Eucalyptus Open-Source Cloud-Computing System, in 'Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on', pp. 124 –131.
- Vecchiola, C., Calheiros, R. N., Karunamoorthy, D. & Buyya, R. (2012), 'Deadline-Driven Provisioning of Resources for Scientific Applications in Hybrid Clouds with Aneka', *Future Generation Computer Systems* **28**, 58 – 65.
- Wei, Y., Sukumar, K., Vecchiola, C., Karunamoorthy, D. & Buyya, R. (2011), 'Aneka Cloud Application Platform and Its Integration with Windows Azure', *CoRR* **abs/1103.2590**.
- Xiong, N., Rindos, A., Russell, M. L., Robinson, K. P., Vandenberg, A. & Pan, Y. (2011), 'Sharing Computing Resources to Satisfy Multi-Cloud User Requirements', *International Journal of Cloud Computing* **1**, 81–100.

Author Index

Alrokayan, Mohammed, 49

Bayliss, Christopher, 39

Berretta, Regina, 3

Burgstaller, Bernd, 13

Buyya, Rajkumar, 49

Cesare, Silvio, 21

Choi, Hyewon, 13

Choi, Seungmi, 31

Garg, Saurabh Kumar, iii

Han, Hyuck, 31

Javadi, Bahman, iii

Lee, Young Choon, 31

Morandini, Luca, 39

Moscato, Pablo, 3

Shamsul Arefin, Ahmed, 3

Sinnott, Richard, 39

Tomko, Martin, 39

Xiang, Yang, 21

Yeom, Heon Y., 31

Zomaya, Albert Y., 31

Recent Volumes in the CRPIT Series

ISSN 1445-1336

Listed below are some of the latest volumes published in the ACS Series *Conferences in Research and Practice in Information Technology*. The full text of most papers (in either PDF or Postscript format) is available at the series website <http://crpit.com>.

- Volume 113 - Computer Science 2011**
Edited by Mark Reynolds, The University of Western Australia, Australia. January 2011. 978-1-920682-93-4.
Contains the proceedings of the Thirty-Fourth Australasian Computer Science Conference (ACSC 2011), Perth, Australia, 17-20 January 2011.
- Volume 114 - Computing Education 2011**
Edited by John Hamer, University of Auckland, New Zealand and Michael de Raadt, University of Southern Queensland, Australia. January 2011. 978-1-920682-94-1.
Contains the proceedings of the Thirteenth Australasian Computing Education Conference (ACE 2011), Perth, Australia, 17-20 January 2011.
- Volume 115 - Database Technologies 2011**
Edited by Heng Tao Shen, The University of Queensland, Australia and Yanchun Zhang, Victoria University, Australia. January 2011. 978-1-920682-95-8.
Contains the proceedings of the Twenty-Second Australasian Database Conference (ADC 2011), Perth, Australia, 17-20 January 2011.
- Volume 116 - Information Security 2011**
Edited by Colin Boyd, Queensland University of Technology, Australia and Josef Pieprzyk, Macquarie University, Australia. January 2011. 978-1-920682-96-5.
Contains the proceedings of the Ninth Australasian Information Security Conference (AISC 2011), Perth, Australia, 17-20 January 2011.
- Volume 117 - User Interfaces 2011**
Edited by Christof Lutteroth, University of Auckland, New Zealand and Haifeng Shen, Flinders University, Australia. January 2011. 978-1-920682-97-2.
Contains the proceedings of the Twelfth Australasian User Interface Conference (AUIC2011), Perth, Australia, 17-20 January 2011.
- Volume 118 - Parallel and Distributed Computing 2011**
Edited by Jinjun Chen, Swinburne University of Technology, Australia and Rajiv Ranjan, University of New South Wales, Australia. January 2011. 978-1-920682-98-9.
Contains the proceedings of the Ninth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2011), Perth, Australia, 17-20 January 2011.
- Volume 119 - Theory of Computing 2011**
Edited by Alex Potanin, Victoria University of Wellington, New Zealand and Taso Viglas, University of Sydney, Australia. January 2011. 978-1-920682-99-6.
Contains the proceedings of the Seventeenth Computing: The Australasian Theory Symposium (CATS 2011), Perth, Australia, 17-20 January 2011.
- Volume 120 - Health Informatics and Knowledge Management 2011**
Edited by Kerry Butler-Henderson, Curtin University, Australia and Tony Sahama, Queensland University of Technology, Australia. January 2011. 978-1-921770-00-5.
Contains the proceedings of the Fifth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2011), Perth, Australia, 17-20 January 2011.
- Volume 121 - Data Mining and Analytics 2011**
Edited by Peter Vamplew, University of Ballarat, Australia, Andrew Stranieri, University of Ballarat, Australia, Kok-Leong Ong, Deakin University, Australia, Peter Christen, Australian National University, Australia and Paul J. Kennedy, University of Technology, Sydney, Australia. December 2011. 978-1-921770-02-9.
Contains the proceedings of the Ninth Australasian Data Mining Conference (AusDM'11), Ballarat, Australia, 1-2 December 2011.
- Volume 122 - Computer Science 2012**
Edited by Mark Reynolds, The University of Western Australia, Australia and Bruce Thomas, University of South Australia, Australia. January 2012. 978-1-921770-03-6.
Contains the proceedings of the Thirty-Fifth Australasian Computer Science Conference (ACSC 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 123 - Computing Education 2012**
Edited by Michael de Raadt, Moodle Pty Ltd and Angela Carbone, Monash University, Australia. January 2012. 978-1-921770-04-3.
Contains the proceedings of the Fourteenth Australasian Computing Education Conference (ACE 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 124 - Database Technologies 2012**
Edited by Rui Zhang, The University of Melbourne, Australia and Yanchun Zhang, Victoria University, Australia. January 2012. 978-1-920682-95-8.
Contains the proceedings of the Twenty-Third Australasian Database Conference (ADC 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 125 - Information Security 2012**
Edited by Josef Pieprzyk, Macquarie University, Australia and Clark Thomborson, The University of Auckland, New Zealand. January 2012. 978-1-921770-06-7.
Contains the proceedings of the Tenth Australasian Information Security Conference (AISC 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 126 - User Interfaces 2012**
Edited by Haifeng Shen, Flinders University, Australia and Ross T. Smith, University of South Australia, Australia. January 2012. 978-1-921770-07-4.
Contains the proceedings of the Thirteenth Australasian User Interface Conference (AUIC2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 127 - Parallel and Distributed Computing 2012**
Edited by Jinjun Chen, University of Technology, Sydney, Australia and Rajiv Ranjan, CSIRO ICT Centre, Australia. January 2012. 978-1-921770-08-1.
Contains the proceedings of the Tenth Australasian Symposium on Parallel and Distributed Computing (AusPDC 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 128 - Theory of Computing 2012**
Edited by Julián Mestre, University of Sydney, Australia. January 2012. 978-1-921770-09-8.
Contains the proceedings of the Eighteenth Computing: The Australasian Theory Symposium (CATS 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 129 - Health Informatics and Knowledge Management 2012**
Edited by Kerry Butler-Henderson, Curtin University, Australia and Kathleen Gray, University of Melbourne, Australia. January 2012. 978-1-921770-10-4.
Contains the proceedings of the Fifth Australasian Workshop on Health Informatics and Knowledge Management (HIKM 2012), Melbourne, Australia, 30 January – 3 February 2012.
- Volume 130 - Conceptual Modelling 2012**
Edited by Aditya Ghose, University of Wollongong, Australia and Flavio Ferrarotti, Victoria University of Wellington, New Zealand. January 2012. 978-1-921770-11-1.
Contains the proceedings of the Eighth Asia-Pacific Conference on Conceptual Modelling (APCCM 2012), Melbourne, Australia, 31 January – 3 February 2012.
- Volume 134 - Data Mining and Analytics 2012**
Edited by Yanchang Zhao, Department of Immigration and Citizenship, Australia, Jiyong Li, University of South Australia, Paul J. Kennedy, University of Technology, Sydney, Australia and Peter Christen, Australian National University, Australia. December 2012. 978-1-921770-14-2.
Contains the proceedings of the Tenth Australasian Data Mining Conference (AusDM'12), Sydney, Australia, 5-7 December 2012.