

Nigel Thomas (Ed.)

LNCS 6977

Computer Performance Engineering

8th European Performance Engineering Workshop, EPEW 2011
Borrowdale, UK, October 2011
Proceedings



 Springer

Commenced Publication in 1973

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

David Hutchison

Lancaster University, UK

Takeo Kanade

Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler

University of Surrey, Guildford, UK

Jon M. Kleinberg

Cornell University, Ithaca, NY, USA

Alfred Kobsa

University of California, Irvine, CA, USA

Friedemann Mattern

ETH Zurich, Switzerland

John C. Mitchell

Stanford University, CA, USA

Moni Naor

Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz

University of Bern, Switzerland

C. Pandu Rangan

Indian Institute of Technology, Madras, India

Bernhard Steffen

TU Dortmund University, Germany

Madhu Sudan

Microsoft Research, Cambridge, MA, USA

Demetri Terzopoulos

University of California, Los Angeles, CA, USA

Doug Tygar

University of California, Berkeley, CA, USA

Gerhard Weikum

Max Planck Institute for Informatics, Saarbruecken, Germany

Nigel Thomas (Ed.)

Computer Performance Engineering

8th European Performance Engineering
Workshop, EPEW 2011
Borrowdale, UK, October 12-13, 2011
Proceedings



Springer

Volume Editor

Nigel Thomas
Newcastle University
School of Computing Science
Newcastle upon Tyne, NE1 7RU, UK
E-mail: nigel.thomas@ncl.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-24748-4 e-ISBN 978-3-642-24749-1
DOI 10.1007/978-3-642-24749-1
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011938124

CR Subject Classification (1998): D.2, C.2, H.3-4, F.3, D.4, C.2.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

© Springer-Verlag Berlin Heidelberg 2011

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

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

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

Printed on acid-free paper

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

Preface

This volume of LNCS contains papers presented at the 8th European Performance Engineering Workshop held at the Lodore Falls Hotel, Borrowdale in the English Lake District on 12th and 13th October 2011.

The accepted papers reflect the diversity of modern performance engineering. There were a number of papers presented which tackled modelling issues in stochastic process algebra, stochastic activity networks and Petri nets. Other papers addressed theoretical advances and applications, including those in communications networks, botnets, inventory systems and web services. There were also a number of papers dedicated to new software tools, showing the continued importance of this area within the wider performance community. As well as the main programme, the workshop also included a selected poster session of papers with considerable merit, but for which there was insufficient time in the main programme. The poster presentations focussed on software tools, model visualisation, analysis techniques and grid-based systems.

We were delighted to have keynote presentations from John Murphy from University College Dublin and Samuel Kounev from the Karlsruhe Institute of Technology. These talks reflected the state of performance engineering today. Samuel Kounev's talk discussed issues in the rigorous development of self-aware systems; systems that can adapt to meet performance (and other) goals. John Murphy's presentation tackled the issue of performance prediction in cloud-based systems, a rapidly developing area of interest.

As workshop chair I would like to thank everyone involved in making EPEW 2011 a success: Springer for their continued support of the workshop series, the programme committee and reviewers, and of course the authors of the papers submitted, without whom there could not be a workshop. We trust that you, the reader, find the papers in this volume interesting, useful and inspiring, and we hope to see you at future European Performance Engineering Workshops.

August 2011

Nigel Thomas

Organization

Programme Committee

Marco Bernardo	University of Urbino, Italy
Jeremy Bradley	Imperial College London, UK
Mario Bravetti	University of Bologna, Italy
Lucia Cloth	German University of Technology in Oman (GUtech), Oman
Vittorio Cortellessa	Università dell'Aquila, Italy
Michel Cukier	University of Maryland, USA
Tadeusz Czachórski	IITiS PAN, Polish Academy of Sciences, Poland
Nick Dingle	University of Manchester, UK
Karim Djemame	University of Leeds, UK
Paulo Fernandes	PUCRS, Brazil
Jean-Michel Fourneau	Université de Versailles, France
Stephen Gilmore	University of Edinburgh, UK
Marco Gribaudo	Politecnico di Milano, Italy
Helmut Hlavacs	University of Vienna, Austria
András Horváth	University of Turin, Italy
Stephen Jarvis	University of Warwick, UK
Carlos Juiz	University of Balearic Islands, Spain
Tomáš Kalibera	Purdue University, USA
Helen Karatza	Aristotle University of Thessaloniki, Greece
Leila Kloul	Université de Versailles, France
Samuel Kounev	Karlsruhe Institute of Technology, Germany
Geyong Min	University of Bradford, UK
John Murphy	University College Dublin, Ireland
Fernando Pelayo	University of Castilla - La Mancha, Spain
Marco Scarpa	University of Messina, Italy
Markus Siegle	Universität der Bundeswehr München, Germany
Mark Squillante	IBM Research, USA
Miklós Telek	Budapest University of Technology and Economics, Hungary
Nigel Thomas	Newcastle University, UK
Mirco Tribastone	Ludwig-Maximilians-Universität München, Germany
Sabine Wittevrongel	Ghent University, Belgium
Katinka Wolter	Freie Universität zu Berlin, Germany
Avelino Zorzo	PUCRS, Brazil

Additional Referees

Steffen Becker

Fabian Brosig

Ricardo Czekster

Tugrul Dayar

Tien Van Do

Michael Faber

Alexander Gouberman

Uli Harder

Rouven Krebs

Francesco Longo

Philipp Reinecke

Martin Riedl

Afonso Sales

Johann Schuster

Max Tschaikowski

Table of Contents

Invited Papers

Performance Engineering for Cloud Computing	1
<i>John Murphy</i>	
Engineering of Self-aware IT Systems and Services: State-of-the-Art and Research Challenges	10
<i>Samuel Kounev</i>	

Regular Papers

Accommodating Short and Long Web Traffic Flows over a DiffServ Architecture	14
<i>Salvador Alcaraz, Katja Gilly, Carlos Juiz, and Ramon Puigjaner</i>	
Automatic Synchronisation Detection in Petri Net Performance Models Derived from Location Tracking Data	29
<i>Nikolas Anastasiou, William Knottenbelt, and Andrea Marin</i>	
Performance Evaluation of Business Processes through a Formal Transformation to SAN	42
<i>Kelly Rosa Braghetto, João Eduardo Ferreira, and Jean-Marc Vincent</i>	
Monotonicity and Efficient Computation of Bounds with Time Parallel Simulation	57
<i>Jean-Michel Fourneau and Franck Quessette</i>	
Stochastic Restricted Broadcast Process Theory	72
<i>Fatemeh Ghassemi, Mahmoud Talebi, Ali Movaghar, and Wan Fokkink</i>	
Higher Moment Analysis of a Spatial Stochastic Process Algebra	87
<i>Marcel C. Guenther and Jeremy T. Bradley</i>	
Optimization for Multi-thread Data-Flow Software	102
<i>Helmut Hlavacs and Michael Nussbaumer</i>	
Formal Mapping of WSLA Contracts on Stochastic Models	117
<i>Rouaa Yassin Kassab and Aad van Moorsel</i>	
Comparison of the Mean-Field Approach and Simulation in a Peer-to-Peer Botnet Case Study	133
<i>Anna Kolesnichenko, Anne Remke, Pieter-Tjerk de Boer, and Boudewijn R. Haverkort</i>	

WMTTools - Assessing Parallel Application Memory Utilisation at Scale 148
Oliver Perks, Simon D. Hammond, Simon J. Pennycook, and Stephen A. Jarvis

On Stochastic Fault-Injection for IP-Packet Loss Emulation 163
Philipp Reinecke and Katinka Wolter

Analysis of Gossip-Based Information Propagation in Wireless Mesh Networks 174
Abolhassan Shamsaie, Wan Fokkink, and Jafar Habibi

Multi-class Network with Phase Type Service Time and Group Deletion Signal 189
Thu-Ha Dao-Thi, Jean-Michel Fournneau, and Minh-Anh Tran

Critical Level Policies in Lost Sales Inventory Systems with Different Demand Classes 204
Aleksander Wieczorek, Ana Bušić, and Emmanuel Hyon

Model-Based Evaluation and Improvement of PTP Syntonisation Accuracy in Packet-Switched Backhaul Networks for Mobile Applications 219
Katinka Wolter, Philipp Reinecke, and Alfons Mittermaier

Light-Weight Parallel I/O Analysis at Scale 235
Steven A. Wright, Simon D. Hammond, Simon J. Pennycook, and Stephen A. Jarvis

Poster Presentations

Can Linear Approximation Improve Performance Prediction ? 250
Vlastimil Babka and Petr Tůma

TWOEAGLES: A Model Transformation Tool from Architectural Descriptions to Queueing Networks 265
Marco Bernardo, Vittorio Cortellessa, and Mirko Flamminj

A Tool Suite for Modelling Spatial Interdependencies of Distributed Systems with Markovian Agents 280
Davide Cerotti, Enrico Barbierato, and Marco Gribaudo

A Grid Broker Pricing Mechanism for Temporal and Budget Guarantees 295
Richard Kavanagh and Karim Djemame

Visualisation for Stochastic Process Algebras: The Graphic Truth	310
<i>Michael J.A. Smith and Stephen Gilmore</i>	
Efficient Experiment Selection in Automated Software Performance Evaluations	325
<i>Dennis Westermann, Rowen Krebs, and Jens Happe</i>	
Author Index	341

Performance Engineering for Cloud Computing

John Murphy

Lero – The Irish Software Engineering Research Centre
School of Computer Science and Informatics, University College Dublin, Ireland
J.Murphy@ucd.ie

Abstract. Cloud computing potentially solves some of the major challenges in the engineering of large software systems. With the promise of infinite capacity coupled with the ability to scale at the same speed as the traffic changes, it may appear that performance engineering will become redundant. Organizations might believe that there is no need to plan for the future, to optimize applications, or to worry about efficient operation. This paper argues that cloud computing is an area where performance engineering must be applied and customized. It will not be possible to “cloud wash” performance engineering by just applying previous methods. Rather it is essential to both understand the differences between the cloud and previous systems, and the applicability of proposed performance engineering methods.

Keywords: Performance Engineering, Software Engineering, Cloud Computing.

1 Evolution of Performance Engineering

Performance engineering is typically a collection of techniques that help manage how well a system will operate or is operating. Performance engineering covers the full life cycle of a system, from choosing a candidate list of technologies, to the high level design, detailed design, modeling, unit testing, system testing, pre-production testing, live monitoring, capacity planning, upgrading and migration of the system. One of the major areas that performance engineering has been applied to is telecommunication systems, where there has been considerable research output in the last century since Erlang’s seminal work [1]. This mathematical treatment of the topic led to the development of teletraffic theory and was published in journals (e.g. The Post Office Electrical Engineers Journal¹ 1908-82, Bell System Technical Journal² 1922-83, and more recently Performance Evaluation³ 1981 on) and conferences (e.g. the major conference is the International Teletraffic Congress⁴ 1955 on). In the latter half of the 20th century, as computer networks were emerging, teletraffic theory was applied to these systems. In particular queueing theory [2] emerged as a leading tool to attempt to solve some of these complex computer networks performance problems. Many different issues arose from this translation and attracted considerable attention, such as

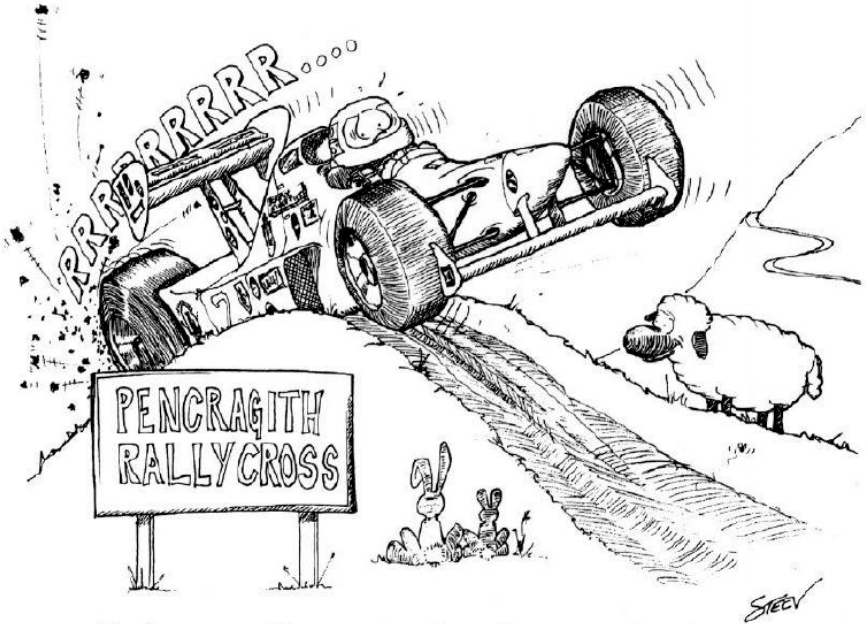
¹ Published by the Institution of Post Office Electrical Engineers

² <http://www.alcatel-lucent.com/bstj/>

³ http://www.elsevier.com/wps/find/journaldescription.cws_home/505618/description

⁴ <http://www.i-teletraffic.org/about-itc/>

self-similar traffic [3], effective bandwidths [4] and statistical multiplexing [5] which could be relevant when undertaking performance evaluation of the cloud. As computer systems became more complex, the issues surrounding performance were analyzed with similar tools and techniques [6] [7] and more recently there has been considerable effort to bring some of these theories to bear on component based software [8] and enterprise systems [9].



Bob soon discovers that having the fastest hardware doesn't guarantee success!

Fig. 1.⁵ A major challenge in applying performance engineering is to be aware of all aspects of the system and how it will be used and evaluated

The key to the success of transforming the tools and theories from one domain to another is in understanding the fundamental differences and limitations of both domains and the relevant theories involved. Cloud computing is a relatively new technology (the term was coined most likely in 2007 [10]) but is built on top of prior research in a number of areas. Cloud computing allows many of the aspects of traditional software systems to be ignored (or abstracted) and allows the scaling and growth of a system to take place automatically. There is a real danger that many of the tried and tested - and successfully implemented - methods and practices will not be put to use in the cloud. This could be either because cloud experts are not aware of them, or more critically that they will not be translated correctly to the cloud as the

⁵ Andrew Lee of Whitney Associates <http://www.1202performance.com/>

special characteristics of the cloud will not be taken into account. Success will involve not merely “cloud washing” performance engineering (where everything remains the same except the term “cloud” is added!) but rather to find which of the prior methods should be emphasized and possibly re-evaluated, and which of those methods might not be applicable.

2 Cloud Context

Cloud computing has been used to define applications delivered as services over the Internet (as well as the hardware and middleware that resides in data centers that are used to provide those services) [11]. It encompasses the concepts of Software as-a-service (SAAS), Platform (or Middleware) as-a-service (PAAS) and Infrastructure as-a-service (IAAS) which combine to make up the cloud. Public cloud refers to situations where the cloud, and in particular infrastructure as-a-service, is made available publicly to individuals and organizations and is charged using metered billing (i.e. pay for what you use). Public cloud allows different end users to share hardware resources and network infrastructure and examples include Amazon⁶ and Rackspace⁷. The private cloud is targeted at large organizations, and generally provides more flexible billing models as well as the ability for these users to define secure zones within which only their company has access to the hardware and network (e.g. Rackspace private cloud⁸, IBM⁹). Hybrid clouds often refer to situations where organizations are making use of both public and private cloud for their infrastructures. The concept of cloud computing also assumes that infinite resources are theoretically available on demand, whereby a user of the cloud can scale their cloud infrastructure immediately when the need arises, i.e. during a traffic surge.

The metered billing model applied by cloud service providers has significantly changed how organizations need to plan and finance their infrastructures. Infrastructure can be provided as-a-service and no longer requires large capital investments up front. This allows a more flexible and agile approach for many organizations when planning their infrastructure requirements. In particular it has reduced the barrier for startup companies entering the market, as no major capital investment is required to launch new services. Similarly large organizations do not need to make long term bets on their infrastructure and they thus can be more flexible and reactive to unplanned changes in company strategy.

Platform as-a-service is designed to support the entire application development lifecycle (development, testing, deployment, runtime, hosting and delivery). It allows organizations to quickly deploy and deliver live, scalable applications in a fraction of the time this has taken in the past. For example following the Google app engine tutorial¹⁰ will allow a developer to develop, design and deploy a live application with

⁶ Amazon EC2 Cloud, <http://www.amazon.com/ec2>

⁷ Rackspace Cloud, <http://www.rackspace.com/>

⁸ Rackspace Private Cloud,
http://www.rackspace.com/managed_hosting/private_cloud/

⁹ IBM Cloud Computing,
<http://www.ibm.com/cloud-computing/us/en/private-cloud.html>

¹⁰ Google App Engine, <http://code.google.com/appengine/>

a public facing dynamic web site in a matter of minutes. Traditionally this process may have taken a significantly longer amount of time: organizations would have needed to plan and allocate physical hardware resources, domain name and Internet Protocol (IP) addresses. Capacity planning and high load scalability issues (in terms of available resources) are largely handled by the PAAS provider.

Software as-a-service allows organizations to outsource the development, management and running of services. Increasingly organizations are making use of SAAS solutions for services that are common across their industry and the development of which is not their core competency. Typical examples include CRM systems (e.g. Salesforce¹¹), HR systems and accountancy systems (e.g. AccountsIQ¹²). Development environments and test and performance tools are also becoming available and popular as services. Examples include development environments (e.g. CloudBees¹³), cloud based monitoring systems (e.g. Cloudkick¹⁴), log management as-a-service technologies (e.g. Logentries¹⁵) and performance monitoring tools (e.g. New Relic¹⁶). The benefits of SAAS services is mainly in reduced management and running costs (compared to in-house systems), as well as the added benefits of using systems designed by specialists in the domain.

In summary, cloud computing gives the ability to design, develop and deploy large scale applications and it does so by abstracting away many of the complex issues. One major advantage is that it can scale with infinite demand for a particular application and this paradoxically presents new challenges for performance engineering.

3 Motivation of Performance Challenges

Performance has always been a major concern for software development and is a critical requirement for IT and software systems. With the immediate availability of theoretically infinite resources on demand, it may be reasonably asked whether performance is still a major concern as systems can "simply scale on demand when load increases". However performance, performance planning, and design are as important as ever and the availability of resources in the cloud has introduced new challenges along with benefits and opportunities. The new challenges are for all providers of as-a-service solutions, whether that is infrastructure, platform or software. The significant challenge is in providing horizontal scaling for their systems such that they can continue to grow and service new customers. Downtime for such providers is generally not acceptable as was recently witnessed with the Amazon outage in April 2011: unplanned downtime resulted in a large number of high profile organizations' systems also being down¹⁷ or in fact a more recent outage which was

¹¹ Salesforce, CRM, <http://www.salesforce.com>

¹² AccountsIQ, Online accountancy platform, <http://www.accountsiq.com/>

¹³ Cloudbees, The Cloudbees platform, <http://cloudbees.com/>

¹⁴ Cloudkick, Clou based monitoring, <http://cloudkick.com>

¹⁵ Logentries, Log management as-a-service, <https://logentries.com>

¹⁶ New Relic, Web app performance monitoring, <http://newrelic.com>

¹⁷ Pepitone, J.: CNN, Amazon EC2 outage downs Reddit, Quora (2011)
http://money.cnn.com/2011/04/21/technology/amazon_server_outage/index.htm

ongoing at the time of writing this paper¹⁸. As-a-service providers largely sell to other business users where downtime or poor performance is not acceptable as it can have a knock-on effect to their customers' business. Therefore performance and scalability is an important requirement for as-a-service providers.

Furthermore as-a-service systems tend to be larger in size than traditional in-house enterprise systems. This is due to the fact that they are often providing the in-house service on a mass scale to large numbers of enterprise customers. Thus as-a-service solutions are following new architectures and making use of new technologies to handle the massive volumes of data and user load. Examples include technologies associated with "Big Data" systems such as NOSQL data bases (e.g. Apache Cassandra¹⁹, MongoDB²⁰, Big Table [12]) or distributed file systems (such as Apache Hadoop [13]). The scale of as-a-service systems, in particular IAAS deployments, introduces a range of new problems for the performance community. IAAS organizations for example can manage tens of thousands of servers²¹.

New technologies and architectures require new performance monitoring and analysis techniques, algorithms and tools, to gather the required data for performance and system test teams, such that they can effectively assess the performance of systems during development and production. Skills are mainly lacking in these areas due to the fact that the technologies are at the cutting edge. There is an onus on educators and organizations to develop appropriate training schemes in the relevant areas and technologies, to cater for these new systems.

4 Classical Performance Engineering

The bulk of the research in the telecommunications domain is mathematical in nature and a considerable amount of it is based on queueing theory [1], [2]. However as the evolution of the domains changed to computer communications there was increased interest in the traffic profiles and the distributions associated with them. This included a re-examination of one of the main theories that traffic would aggregate when combined, and it was shown that for some traffic the opposite occurred. This effect, known as self-similar traffic, disrupted many previous assumptions and led to considerable scrutiny [3]. This area could be of particularly interest to cloud computing as one of the basic economic assumptions is that by combining many companies usage together, savings will follow.

Another breakthrough in performance engineering emerged when users were allowed to multiplex, or combine, their traffic together in a statistical manner (statistical multiplexing). This occurred in broadband networks and the theory of "effective bandwidths" was put forward as a way to deal with this new type of traffic

¹⁸ Wainwright, P.: ZDNet.com, Lightning Strike Zaps EC2 Ireland. 8th August 2011, <http://www.zdnet.com/blog/saas/lightning-strike-zaps-ec2-ireland/1382>

¹⁹ Apache Cassandra. <http://incubator.apache.org/cassandra/>

²⁰ mongodb. <http://www.mongodb.org/>

²¹ Rich Miller, "Who Has the Most Web Servers?" Datacenterknowledge.com, May 2009, <http://www.datacenterknowledge.com/archives/2009/05/14/whos-got-the-most-web-servers/>

planning [4], [5]. This allowed some of the older theories and methods to continue working, but changing the manner in which the resources were accounted for. Similarly this technique could have relevance in cloud computing, as one set of users could potentially affect other users, where they are sharing resources (in a statistical sense).

Performance engineering typically has to monitor (to collect the data), has to build models (to experiment with the system), and then has to be able to extract analysis from these models (to explore what-if type questions). The monitoring for pre-cloud enterprise systems is difficult with many layers of complexity; in cloud systems this becomes an increasingly more complex challenge. The modeling (either mathematical or simulation) has been extensively researched in software systems [6], [7], [8], [9] and many techniques can be employed to undertake to build useful models. However while there were many issues for enterprise systems, these will be exacerbated for cloud computing systems due to the scale and additional layers involved.

5 Cloud Specific Challenges

There are a number of areas where results from performance engineering of software systems could benefit the area of cloud computing. Examples include SAAS performance design; autonomics; performance monitoring; resource utilization; and data analysis.

5.1 SAAS Performance Design

Software as-a-service systems are centralized services typically designed to cater for large numbers of end users. Consumer services include social platforms (e.g. Facebook) or online email services (e.g. gmail). There are also increasing numbers of business services being delivered as-a-service. The nature in which these services are being implemented requires horizontal scalability and the ability to quickly scale up and down during times of different workloads [14]. Performance design for scalability and reliability is an important area for these systems. Performance design can be challenging on large scale systems with large numbers of components. These as-a-service systems will more than likely be even larger in scale than traditional in-house enterprise systems and thus performance design will be a major challenge, as it was for enterprise software [15], [16]. Furthermore, while hardware resources may be immediately available in abundance as part of using the cloud, there is an associated cost that is clearly measurable due to inefficient design. Software that inefficiently makes use of cloud resources (without due regard to the associated cost) may well result in a high financial cost. In the past the cost of running inefficient hardware was capped by the available hardware resources in-house (which was typically paid for through capital expenditure). In the cloud this is no longer the case and developers and designers are now closer to the financial costs associated with running their software. Thus responsible design of software with respect to performance is required such that efficient usage of the cloud is attained.

5.2 Autonomics

As systems grow in size, management from a performance perspective on a manual basis becomes more difficult. Autonomic management of systems has been a growing area of research over the past decade [17]. Automatic scaling based on alerting and user defined thresholds is something available today from as-a-service providers so that system will scale on demand. Automated automation and integration frameworks (e.g. Chef²²) are allowing this to happen currently for industry. Further advances in this area will be required such that performance monitoring can integrate with these frameworks for better autonomic performance management. In particular performance monitoring methodologies, real time analytics and decision making research will be required to drive the autonomic management process.

5.3 Performance Monitoring

With the abundance of new technologies and middleware (Distributed file systems, NoSQL databases, Search platforms²³) for large scale systems processing "Big Data" there is a need for performance monitoring and analysis techniques to be developed such that performance metrics can be obtained, analyzed and understood in the context of these new technologies. Traditional monitoring methods for enterprise systems may be applicable in certain cases, however new techniques will most likely be required specifically for these new platforms and architectures. Monitoring and management of the cloud are also starting to be delivered as-a-service^{14, 15, 16} which means that tool providers will centrally store monitoring data from large numbers of customers systems. This in itself will provide opportunities in terms of data analytics.

5.4 Resource Utilization

An emerging requirement exists in the area of measuring the utilization of large cloud deployments in an automated manner such that utilization metrics can be efficiently collected and properly understood. A view of how well hardware is being utilized in the context of different workloads is currently a major challenge for cloud providers. This understanding is required to maximize the efficiency of cloud infrastructures. Research is required in the area utilization analysis in the context of different software workloads. Such analysis can be applied, for example, to maximize the system utilization, to relocate workloads, to increase energy efficiency, or indeed to reduce costs.

5.5 Data Analysis

While techniques and approaches for gathering monitoring data have become better appreciated through the development of performance tools for in-house enterprise systems [18], [19] the analysis of the large volume of data collected has been a major

²² Chef, Systems integration framework, <http://www.opscode.com/chef/>

²³ Apache Lucene, <http://lucene.apache.org/java/docs/index.html>

challenge. Work has been performed in this area to allow for domain knowledge to be applied for enterprise systems [16]. New challenges exist for the cloud however due to the larger scale of systems and the much larger volumes of data produced by these systems. Real time analytics is a growing area and provides challenges in the analysis of upwards of millions of events per second with real time constraints. An example of such systems today can be seen with the emergence of new technologies taking on these challenges such as log management as-a-service. For example, an individual enterprise may produce terabytes of log data per month²⁴ which can equate to 100,000s of events per second²⁵. A log management as-a-service technology handling log analysis for large numbers of enterprises must be able to manage millions of events per second, performing visualization, analysis and alerting in real time to allow for autonomic management of the system. Other similar technologies that are emerging include real time analytics for gaming platforms²⁶ and real time analytics for performance monitoring¹⁶. Further research in the area of data analytics with time constraints in the coming years will enhance the performance management of cloud based systems and this is probably going to be a rich area of research. Data mining, anomaly detection and machine learning techniques are possibly going to be applicable for as-a-service performance monitoring tools. Vendors of such technologies will be in control of large volumes of customer data compared to traditional performance monitoring tools (deployed in-house). As-a-service tools will therefore be exposed to this customer data and as such will provide opportunities for data mining techniques to be applied and patterns and trends identified that may prove beneficial to all customers.

6 Conclusions

Performance engineering has been applied to many domains over a long time period and with each new domain there is a translation of the most appropriate methods to successfully manage the system's performance. The traditional techniques can be reapplied, or methods and techniques that are most appropriate for the new systems may well undergo significant development. Typically there is a renewed focus on the most appropriate tools, and for the emerging cloud computing area this will most likely follow a similar pattern to previous transitions. Cloud washing will not work for performance engineering, but performance engineering will play a crucial role in the success of cloud computing.

Acknowledgments. The author appreciates the permission to use the cartoon (in Fig. 1) from Andrew Lee, and the many discussions about this paper (and the research) with Trevor Parsons and Liam Murphy. This research is supported, in part, by Science Foundation Ireland grant 10/CE/I1855.

²⁴ Oltsik, J.: The invisible log data explosion" Cnet.com (2007)

²⁵ Log Math, <http://chuvakin.blogspot.com/2010/08/log-math.html>

²⁶ Swrve, Real time gaming analytics, <http://swrve.com/>

References

1. Erlang, A.K.: Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *Elektroteknikeren* 13 (1917)
2. Kleinrock, L.: *Queueing Systems*. Wiley, Chichester (1975)
3. Leland, W.E., Taqqu, M.S., Willinger, W., Wilson, D.V.: On the self-similar nature of Ethernet traffic. In: *Communications Architectures, Protocols and Applications*, pp. 183–193. ACM, New York (1993)
4. Kelly, F.P.: Notes on effective bandwidths. *Stochastic Networks: Theory and Applications*, pp. 141–168. Oxford University Press, Oxford (1996)
5. Blondia, C., Casals, O.: Statistical multiplexing of VBR sources: A matrix-analytic approach. *Performance Evaluation* 16(1-3), 5–20 (1992)
6. Jain, R.: *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley- Interscience, New York (1991)
7. Smith, C.: *Performance Engineering of Software Systems*. Addison-Wesley Longman Publishing, Boston (1990)
8. Franks, G., Majumdar, S., Neilson, J., Petriu, D., Rolia, J., Woodside, M.: Performance analysis of distributed server systems. In: *6th International Conference on Software Quality*, pp. 15–26 (1996)
9. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering* 30(5), 295–310 (2004)
10. Boss, G., Malladi, P., Quan, D., Legregni, L., Hall, H.: *Cloud Computing*. IBM (2007)
11. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A View of Cloud Computing. *Communications of the ACM* 53(4), 50–58 (2010)
12. Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A., Gruber, R.E.: Bigtable: A Distributed Storage System for Structured Data. In: *Conference on Usenix Symposium on Operating Systems Design and Implementation*, pp. 205–218 (2006)
13. White, T.: *Hadoop: The Definitive Guide*. O’Reilly Media, Sebastopol (2010)
14. Borthakur, D., et al.: Apache Hadoop Goes Realtime at Facebook. In: *Proceedings of the International Conference on Management of Data* (2011)
15. Tate, B., Clarke, M., Lee, B., Linskey, P.: *Bitter EJB*. Manning (2003)
16. Parsons, T., Murphy, J.: Detecting Performance Antipatterns in Component Based Enterprise Systems. *Journal of Object Technology* 7(3), 55–90 (2008)
17. Dobson, S., Sterritt, R., Nixon, P., Hinchey, M.: Fulfilling the Vision of Autonomic Computing. *Computer* 43(1), 35–41 (2010)
18. Parsons, T., Mos, A., Trofin, M., Gschwind, T., Murphy, J.: Extracting Interactions in Component Based Systems. *IEEE Transactions on Software Engineering* 34(6), 783–799 (2008)
19. Kozioloka, H.: Performance evaluation of component-based software systems: A survey. *Performance Evaluation* 67(8), 634–658 (2010)

Engineering of Self-aware IT Systems and Services: State-of-the-Art and Research Challenges

Samuel Kounev

Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology (KIT)
76131 Karlsruhe, Germany
skounev@acm.org

Modern IT systems have highly distributed and dynamic architectures composed of loosely-coupled services typically deployed on virtualized infrastructures. Managing system resources in such environments to ensure acceptable end-to-end application Quality-of-Service (QoS) while at the same time optimizing resource utilization and energy efficiency is a challenge. The adoption of Cloud Computing technologies, including Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS), comes at the cost of increased system complexity and dynamicity. This makes it hard to provide QoS guarantees in terms of performance and availability, as well as resilience to attacks and operational failures [8]. Moreover, the consolidation of workloads translates into higher utilization of physical resources which makes the system much more vulnerable to threats resulting from unforeseen load fluctuations, hardware failures and network attacks.

We present an overview of our work-in-progress and long-term research agenda focusing on the development of novel methods, techniques and tools for the engineering of so-called *self-aware* IT systems and services¹ [6,4,7]. The latter are designed with built-in online QoS prediction and self-adaptation capabilities used to enforce QoS requirements in a cost- and energy-efficient manner. The current focus is on performance, availability and efficiency aspects, however, long-term we are planning to consider further QoS properties such as reliability and fault-tolerance. Self-awareness, in this context, is defined by the combination of three properties that IT systems and services should possess:

1. *Self-reflective*: i) aware of their software architecture, execution environment and the hardware infrastructure on which they are running, ii) aware of their operational goals in terms of QoS requirements, service-level agreements (SLAs) and cost- and energy-efficiency targets, iii) aware of dynamic changes in the above during operation,
2. *Self-predictive*: able to predict the effect of dynamic changes (e.g., changing service workloads or QoS requirements) as well as predict the effect of possible adaptation actions (e.g., changing service deployment and/or resource allocations),

¹ <http://www.descartes-research.net>

3. *Self-adaptive*: proactively adapting as the environment evolves in order to ensure that their QoS requirements and respective SLAs are continuously satisfied while at the same time operating costs and energy-efficiency are optimized.

Our approach to the realization of the above vision is based on the use of *online* service architecture models integrated into the system components and capturing all service aspects relevant to managing QoS and resource efficiency during operation [2,10,7]. In contrast to black-box models, the modeling techniques we are working on are designed to explicitly capture all relevant aspects of the underlying software architecture, execution environment, hardware infrastructure, and service usage profiles. In parallel to this, we are working on self-aware service platforms designed to automatically maintain models during operation to reflect the evolving system environment. The online models will serve as a “mind” to the running systems controlling their behavior, i.e., deployment configurations, resource allocations and scheduling decisions. To facilitate the initial model construction and continuous maintenance during operation, we are working on techniques for automatic model extraction based on monitoring data collected at run-time [1,5,3].

The online service architecture models are intended to be used during operation to answer QoS-related queries such as: What would be the effect on the QoS of running applications and on the resource consumption of the infrastructure if a new service is deployed in the virtualized environment or an existing service is migrated from one server to another? How much resources need to be allocated to a newly deployed service to ensure that SLAs are satisfied while maximizing energy efficiency? What QoS would a service exhibit after a period of time if the workload continues to develop according to the current trends? How should the system configuration be adapted to avoid QoS problems or inefficient resource usage arising from changing customer workloads? What operating costs does a service hosted on the infrastructure incur and how does the service workload and usage profile impact the costs? We refer to such queries as *online QoS queries*.

The ability to answer online QoS queries during operation provides the basis for implementing novel techniques for self-aware QoS and resource management [7,2,10]. Such techniques are triggered automatically during operation in response to observed or forecast changes in the environment (e.g., varying service workloads). The goal is to *proactively* adapt the system to such changes in order to avoid anticipated QoS problems, inefficient resource usage and/or high system operating costs. The adaptation is performed in an autonomic fashion by considering a set of possible system reconfiguration scenarios (e.g, changing VM placement and/or resource allocations) and exploiting the online QoS query mechanism to predict the effect of such reconfigurations before making a decision [2].

Each time an online QoS query is executed, it is processed by means of the online service architecture models which are composed dynamically after determining which specific parts of the system are relevant to answering the query. Given the wide range of possible contexts in which the online service models can

be used, automatic model-to-model transformation techniques (e.g., [9]) are used to generate tailored prediction models on-the-fly depending on the required accuracy and the time available for the analysis. Multiple prediction model types (e.g., queueing networks, stochastic Petri nets, stochastic process algebras and general-purpose simulation models) and model solution techniques (e.g., exact analytical techniques, numerical approximation techniques, simulation and bounding techniques) are employed here in order to provide flexibility in trading-off between prediction accuracy and analysis overhead.

Self-Aware Service Engineering [4,6] is a newly emerging research area at the intersection of several computer science disciplines including Software and Systems Engineering, Computer Systems Modeling, Autonomic Computing, Distributed Systems, Cluster and Grid Computing, and more recently, Cloud Computing and Green IT (see Figure 1). The realization of the described vision calls for an interdisciplinary approach considering not only technical but also business and economical challenges. The resolution of these challenges promises to reduce the costs of ICT and their environmental footprint while keeping a high growth rate of IT services.

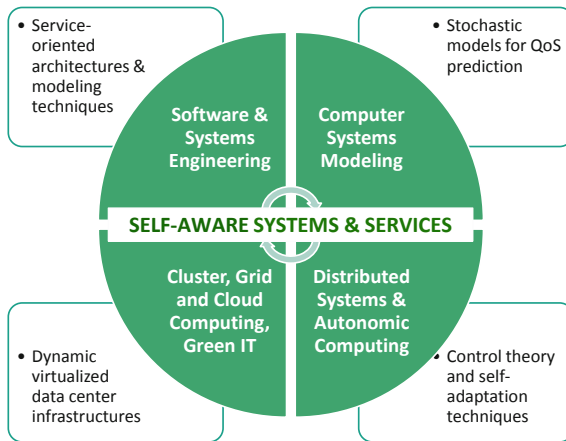


Fig. 1. Self-Aware Service Engineering

References

1. Brosig, F., Huber, N., Kounev, S.: Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems. In: 26th IEEE/ACM International Conference On Automated Software Engineering (ASE 2011), Oread, Lawrence, Kansas, November 6-11 (2011)
2. Huber, N., Brosig, F., Kounev, S.: Model-based Self-Adaptive Resource Allocation in Virtualized Environments. In: SEAMS 2011: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Waikiki, Honolulu, Hawaii, USA, May 23-24, ACM Press, New York (2011)

3. Huber, N., von Quast, M., Hauck, M., Kounev, S.: Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments. In: International Conference on Cloud Computing and Service Science (CLOSER 2011), Noordwijkerhout, The Netherlands, May 7-9 (2011)
4. Kounev, S.: Self-Aware Software and Systems Engineering: A Vision and Research Roadmap. In GI Softwaretechnik-Trends. In: Proceedings of Software Engineering 2011 (SE 2011), Nachwuchswissenschaftler-Symposium, Karlsruhe, Germany, February 21-25 (2011) ISSN 0720-8928
5. Kounev, S., Bender, K., Brosig, F., Huber, N., Okamoto, R.: Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics. In: 4th International ICST Conference on Simulation Tools and Techniques, Barcelona, Spain, March 21-25 (2011)
6. Kounev, S., Brosig, F., Huber, N.: Self-Aware QoS Management in Virtualized Infrastructures (Poster Paper). In: 8th International Conference on Autonomic Computing (ICAC 2011), Karlsruhe, Germany, June 14-18 (2011)
7. Kounev, S., Brosig, F., Huber, N., Reussner, R.: Towards self-aware performance and resource management in modern service-oriented systems. In: Proceedings of the 7th IEEE International Conference on Services Computing (SCC 2010), Miami, Florida, USA, July 5-10. IEEE Computer Society, Los Alamitos (2010)
8. Kounev, S., Reinecke, P., Joshi, K., Bradley, J., Brosig, F., Babka, V., Gilmore, S., Stefanek, A.: Providing Dependability and Resilience in the Cloud: Challenges and Opportunities. In: Avritzer, A., van Moorsel, A., Wolter, K., Vieira, M. (eds.) Resilience Assessment and Evaluation, Dagstuhl Seminar 10292. Springer, Heidelberg (2011)
9. Meier, P., Kounev, S., Koziolok, H.: Automated Transformation of Palladio Component Models to Queueing Petri Nets. In: 19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011), Singapore, July 25-27 (2011)
10. Nou, R., Kounev, S., Julia, F., Torres, J.: Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software* 82(3), 486-502 (2009)

Accommodating Short and Long Web Traffic Flows over a DiffServ Architecture

Salvador Alcaraz¹, Katja Gilly¹, Carlos Juiz², and Ramon Puigjaner²

¹ Miguel Hernández University,
Departamento de Física y Arquitectura de Computadores,
Avda. del Ferrocarril, 03202 Elche, Spain

{salcaraz, katya}@umh.es

² University of Balearic Islands,
Departament de Ciències Matemàtiques i Informàtica,
Carretera de Valldemossa, km 7.5, 07071 Palma de Mallorca, Spain
{cjuiz, putxi}@uib.es

Abstract. DiffServ architecture has been widely used to achieve QoS over the Internet. Taking into account that HTTP traffic is the most extended protocol over the Internet community, many solutions have been proposed to supply QoS to this protocol. Traditionally, DiffServ architectures have considered two-colour markings in order to distinguish between high and low priorities. We investigate the special treatment for web traffic, whose pattern is very close to mice and elephants distribution flows in Internet. We differentiate flows into short and long classes in order to ensure QoS for short flows, but we try to achieve certain QoS for some long flows. Metering, shapering and marking processes are used to classify the incoming flows at the DiffServ using three-colour marking. The final algorithm has been named Long Flow Promotions (LFP). The simulation tool used is *ns2* and the realistic synthetic web traffic has been generated with *PackMime-HTTP*. The results are compared to RED and DropTail queue management. LFP gets reasonably low latency values while providing high priority level to short flows and improving some performance parameters such as overhead and dropped packets.

Keywords: web traffic, DiffServ, token bucket, QoS, short and long flows, packet promotion.

1 Introduction

Since the World Wide Web (www) was developed by Tim Berners-Lee [1] working at CERN, in Geneve, the Hypertext Transfer Protocol (HTTP) has been the communications protocol most widely used in Internet [2]. Recently, new applications (kazaa, P2P, etc.) and features (web 2.0) have been added to the Internet traffic, nevertheless the latest studies [3] show that web traffic is still the most usual data flow in Internet. At the early stages, the web traffic was composed of static and small pages, that used to contain a few objects. Later, database queries, dynamic pages and some ad-hoc objects based on flash technology were

added to web traffic, that meant an increase in the web pages size and, hence, more packets per flow. Nowadays, web traffic implies that many technologies have to act together and interconnect the web around the world. Although HTTP does not provide any Quality of Service (QoS), different web users share the available bandwidth and the network resources of the Internet Service Provider (ISP). In this context, small web pages requested from web clients coexist with video streaming and database queries. The difference of size between each kind of flow can be considerable. If short flows are treated preferentially against long flows, some web flows could be excessively penalised or suffer considerable delay from server to client.

It is well documented that most of the Internet flows (around 80%) carry a short amount of traffic (around 20%), while the rest of flows (around 20%) represent most of the traffic (around 80%). These types of flows are named *mice* and *elephants*[4]. This fact sometimes leads to long response times for short browsing requests when the bandwidth is mostly used by long flows. Therefore, ISPs need to implement mechanisms to incorporate some enhanced QoS to their web sites in order to permit clients fast browsing without an excessive penalisation to rest of the flows. Regarding this subject, many solutions, environments and policies have been proposed [5].

Every application protocol in Internet generates a different traffic workload, but they might share the same First In Out (FIFO) queue at the switching and routing nodes. If queues are allowed to drop packets only during overflow conditions, then bursty traffic flows will face greater dropping probabilities than smooth traffic flows [6]. Random Early Detection (RED) [7] has been one of the most important solutions to detect and avoid the congestion in computer networks. With RED queue management, packets are dropped with a certain probability before the queue reaches an overflow state. The main advantage of RED over TailDrop queue was analysed by [8]. RED queue operates with a lower queue size, especially during peak load and congestion conditions. This feature allows bursts of packets to be accommodated into the available queue achieving an overall performance improvement.

The remainder of this paper is organised as follows: section 2 describes our proposal: the Long Flow Promotion algorithm (LFP). Section 3 presents a comparative of the simulation results of the LFP, RED and DropTail algorithms. Finally, some concluding remarks are presented.

2 Long Flow Promotion (LFP)

Traditional QoS strategies are mainly based on marking and differentiating flows over the Differentiated Services Model (DiffServ), which has two dedicated devices: *edge* and *core*. Edge nodes mark the packets by adding different labels to them. The information contained in these labels specifies the workload conditions related to the Service Level Agreement (SLA) contracted by the client. When the marked packets reach the *core* node, they are forwarded over different queues by applying the suitable Active Queue Management (AQM) or a

stochastic treatment in order to achieve the required QoS. This paper presents the LFP algorithm as an approach to improve the web traffic over a DiffServ architecture focused to permit the coexistence between short and long flows in the same shared network.

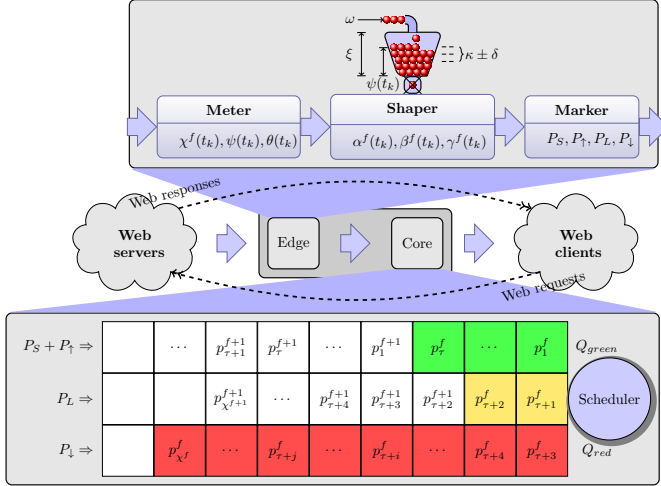


Fig. 1. LFP architecture. The DiffServ model is composed by the edge and core devices (in the middle). Edge device is composed by the meter, shaper and marker functions (at the top) and core device (at the bottom) is composed by the set of queues $Q = \{Q_{green}, Q_{yellow}, Q_{red}\}$ managed with priority queue scheduling algorithm.

Since web traffic has its own features, the LFP algorithm has been designed to improve the QoS of this type of traffic. The algorithm has been developed taking into account a couple of premises related to web traffic: a) End-users expectations; b) Mice and elephants paradigm.

a) The issue of *End-users expectations* has been widely discussed [9], and it is an important factor to determine the success of a website. When users are browsing the Internet, they want to go as fast as they can. Users can tolerate a long delay downloading heavy files such as multimedia streams, database queries or large documents, but they might not stand long delays while surfing the web; i.e. clicking into links or downloading small files such as images, sounds or any other small object. For these reasons, the end-users expectation should be strongly considered in a website development, and it is recommended to implement a suitable mechanism to improve the end-users perception about latencies and delays.

b) Web traffic flow size follows a well defined heavy-tailed [10] which is directly related to the *mice and elephants paradigm*. As we have commented previously, most of the traffic (around 80%) is carried out by a few flows (around 20%), that are denominated *elephants* in Internet. Therefore, the rest of flows, that are

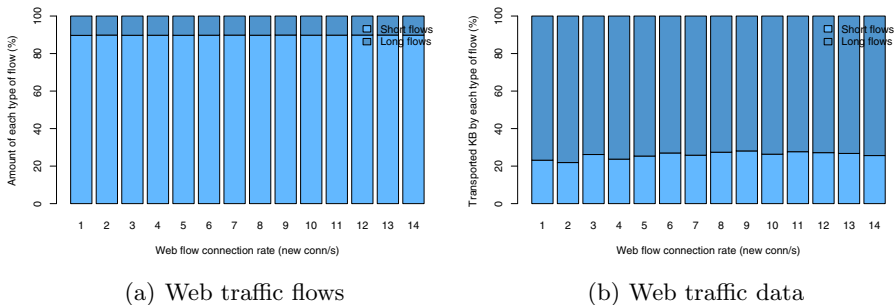


Fig. 2. Mice and elephants flows and amount of transported data

named *mice*, transport around 20% of data information. This means that the most usual flows in the Internet are short flows, although they only transport a few bytes. Several studies of Internet traffic and packet distributions of the most representative protocols in Internet have been undertaken striving to establish the *threshold* (τ) that would differentiate between *short flows* (*SF*) and *long flows* (*LF*) in web traffic. Chen et al. [11] proposed a table with five representative types of web pages. The average of the web page size varies in the range of [9, 12] Kbytes, from a minimum limit in the range of [1, 3] Kbytes, until maximum values in the range of [80, 90] Kbytes.

Considering an average size of 12 Kbytes, and using a Protocol Data Unit (PDU) of 1500 bytes, including an overhead of 60 bytes, and two ACK packets for the Transport Control Protocol (TCP) three-way handshake, we consider in our proposal a flow size average of 10 packets per flow as the threshold to differentiate between short and long flows. Since τ has been established, it is now possible to differentiate between short and long flows. *SF* will be those flows whose number of packets is lower than or equal to τ and therefore, *LF* are the rest of the flows. Fig. 2(a) shows a *mice/elephants* distribution for web response traffic. As it is depicted in the figure, around 90% of flows are classified as *mice/SF*, therefore, the rest of the flows, around 10% are classified as *elephants/LF*. According to the *mice and elephants paradigm*, the amount of Kbytes transported for each type of flow is shown in Fig. 2(b). Using the same threshold, $\tau = 10 \text{ packets}$, around 20% of the whole web traffic in Internet is transported by short flows (*mice/SF*), and the remainder of Kbytes, around 80%, are transported by the other type of flows (*elephants/LF*). We are going to describe now our proposal for promoting flows in order to avoid unnecessary delays for the end-users.

2.1 Preferential Treatment for Short Flows (S_1)

Following with the above premises, the overall incoming traffic that reaches the QoS system should be measured and classified in order to assign the desired QoS level to the most sensitive web traffic, that are the short flows. As it is illustrated in Fig. 1, LFP is developed over a DiffServ architecture composed of two types of devices: *edge* and *core*. LFP mechanism is defined as follows: web

traffic from web servers (HTTP responses) reaches the DiffServ area where it is firstly measured and classified at the *edge* device. In this device, after going through the *meter* and the *shaper* process, the incoming packets are marked with different labels. When the packets leave the *edge* device and reach the *core* device, they are sent over a specific queue, depending on the assigned label. Finally, the priority queueing scheduling strategy at the *core* device configures the QoS level in the system.

The overall incoming traffic that reaches the system is divided in n flows and defined as f_1, f_2, \dots, f_n . Each flow is composed of a sequence of p packets. Therefore, the flow f contains p packets and the packets sequence is defined as: $p_1^f, p_2^f, \dots, p_p^f$, where p_i^j defines the i -packet from the flow j , and that arrives to the system at the instant t_i^j , as it is depicted in Fig. 3.

Let us define $P^f = \{p_i^f, \forall i \in [0, t_k]\}$ as the set of packets from flow f . Therefore, $V(t_k) = \{P^f \mid \forall f\}$ is the overall traffic at the interval $[0, t_k]$. In order to simplify the expressions, several assumptions have been taken:

- The amount of data from each flow is quantified as packets instead of Kbytes.
- Only one packet reaches the system in the interval $[t_{k-1}, t_k)$.
- The web traffic considered corresponds only to responses. The request sizes are negligible compared to the response sizes.

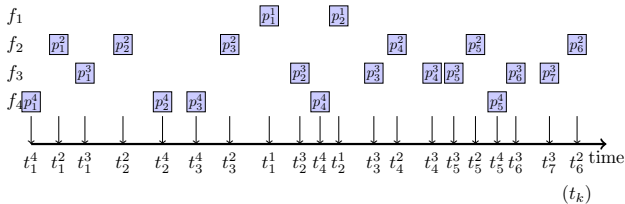


Fig. 3. Example of packet sequence that reaches the QoS system. Packets are labelled as follows: p_i^f is the k -th packet from flow f , and it arrives at instant t_i^f .

Let us now describe the functions in the *edge* device. The *meter* is the first function that is computed, $\chi^f(t_k)$, and represents the number of packets of the flow f at time $[0, t_k]$ (i.e. from its birth until t_k). It is defined as follows:

$$\chi^f(t_k) = \text{Ord}(P^f) \quad (1)$$

After the *meter* process, the packets go through the *shaper* function, that defines the transition of a flow from short to long state, and it is defined by the threshold τ . First of all, the differentiation condition, S_1 :

$$S_1 \equiv \chi^f(t_k) \leq \tau \quad (2)$$

The *shaper* function at the *edge* node is defined for each incoming flow f at instant t_k as the discrete function $\alpha^f(t_k) \in \{0, 1\}$, defined as follows:

$$\alpha^f(t_k) = I(S_1), \quad (3)$$

where the discrete function $I(S)$ is:

$$I(S) = \begin{cases} 1 & \text{if } S \text{ is true} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

2.2 Packets Promotion Close to the Threshold (S_2)

Considering only the S_1 constraint to differentiate between short and long flows, the consequences for a flow f that arrives to the DiffServ system, will be the following:

- The range of packets $[1, \tau]$ receives always the highest priority level.
- The range of packets $[\tau + 1, \chi^f(t_k)]$ receives always a low priority level.

Hence, S_1 establishes a hard threshold between short and long flows. Therefore, flows of $\tau + i$ packets, where $i = 1, 2, 3, \dots$ are considered as long flows, despite their size being close to τ . Web traffic flow size distribution follows a heavy-tailed distribution, that is, there are many flow sizes close to τ . For this reason, the constraint that we define in this section tries to improve the QoS parameters for those flows with sizes close to the threshold τ , by introducing the *packets promotion* concept. This concept is related to those flows that are a bit longer than the threshold (τ), and that under particular system conditions, could be considered as short flows and, hence, receive a high priority QoS level. The most appropriate conditions to promote packets are low congestion and idle state of the system.

To deploy the S_2 constraint, the Token Bucket Model [12] has been used to detect the idle system state by counting the packet promotions. Although, the packet promotion is restricted in order to prevent either the increase of the system overhead or the promotion of the inappropriate packets.

The token bucket model is used to compute the amount of packets than can be promoted. The token bucket operation is defined as follows: the bucket emulates a depot of tokens, where each token indicates the possibility to send a packet over the high priority queue. The maximum capacity of the token bucket is defined by ζ tokens. New tokens are supplied at ω ratio in *tokens/s*. Tokens are always consumed with packets forwarded over the high priority queue, therefore, packets from short flows and promoted packets from long flows. The physical token bucket capacity is limited by a low and high level. If the tokens level is below the low level, it is considered that no tokens are available at the bucket. Otherwise, if the tokens level is higher than the high level, no more tokens can fill the bucket. The amount of available tokens in the bucket indicates the available bandwidth to offer a high priority service to incoming flows. If $V^*(t_k)$ defines the amount of promoted packets, then the state of the token bucket at instant t_k is defined by $\psi(t_k)$ as follows:

$$\psi(t_k) = \psi(t_{k-1}) + IN(t_k) - OUT(t_k) \quad (5)$$

where:

$$\begin{aligned} IN(t_k) &= \min(\omega * (t_k - t_{k-1}), \zeta - \psi(t_{k-1})) \\ OUT(t_k) &= V^*(t_k) I \{ V^*(t_k) \leq (\psi(t_{k-1}) + \min(\omega * (t_k - t_{k-1}), \zeta - \psi(t_{k-1}))) \} \end{aligned} \quad (6)$$

As the token bucket model is used to bound the quantity of promoted packets, $\theta(t_k)$ represents the normalised capacity of the bucket, defined as follows:

$$\theta(t_k) = \left[\frac{\psi(t_k)}{\zeta} \right]_0^{100} \quad (7)$$

However, it is desirable that the state of the token bucket, $\psi(t_k)$, remains close to a precise level or set point defined by $\kappa \in [0, 100]$ as it is depicted in Fig. 1. As the web traffic presents peaks of incoming traffic, $\psi(t_k)$ must range around a set point κ . In order to permit this working area, top and bottom limits are defined by the parameter $\delta \in [0, 100]$. Therefore, the working area, that is defined by $\kappa \pm \delta$ operates as follows:

- If $\psi(t_k)$ is close to κ , the packets promotion is at *open* state.
- If $\psi(t_k)$ reaches $\kappa - \delta$ level, the packets promotion will turn to *close* state. At this point, no packets are promoted. Hence, tokens are only consumed with packets from short flows. As the token bucket continues being filled with new tokens at ω ratio, then the token bucket level $\theta(t_k)$ will catch up $\kappa + \delta$ level again. At this point, the token bucket state will turn to *open* state again and the promotion process will be reactivated.

The token bucket operation bounds are defined by $[\kappa - \delta, \kappa + \delta] \mid 0 \leq \kappa - \delta \leq \kappa + \delta \leq 100$. Considering the above bounds, the differentiation function S_2 can be defined as follows:

$$S_2 \equiv \{ \theta(t_k) \geq \kappa + \delta \} \vee \{ (\kappa - \delta \leq \theta(t_k) \leq \kappa + \delta) \wedge \beta^f(t_{k-1}) \} \quad (8)$$

Once the metering process has concluded, the shaper process computes $\beta^f(t_k)$ function for each incoming flow:

$$\beta^f(t_k) = I(S_2) \quad (9)$$

S_2 alleviates S_1 weakness by using the packets promotion, but an adverse effect appears under particular circumstances of low congestion: when there are a few flows crossing the system, the token bucket state could be promoting some inappropriate flows. These flows can be extremely long, and they do not need any higher priority level because the end-users expect a long delay for these flows any way.

2.3 Detecting Elephant Flows (S_3)

The presence of *elephant* flows in the QoS system is always bad news, and it produces an overall system performance fall. For that reason, the main goal of

the constraint S_3 is the detection and isolation of extremely long flows. Such flows are classified as *elephant*, hence, they do not need some QoS requirements, and they can be treated with the lowest priority.

In order to detect and isolate those very long flows, the critical issue is to define the measurement to be applied. As the web traffic nature is very variable (heavy tail and self-similar distributions), setting a threshold as a fixed number of packets to differentiate flows as long or very long is not suitable, because an excessive low threshold could generate too many promoted packets and, by contrast, an excessive high threshold could generate too few promoted packets. None of both circumstances are desirable. For this reason, the value to determine when a flow is long or very long must be adaptive to the traffic conditions of each situation. Hence, we consider that it has to be calculated from the sizes of the last flows that have crossed the QoS system. Therefore, $X_{t_k, H}$ is the set of the last H flow sizes that have passed through the system, and is defined as follows:

$$X_{t_k, H} = \{\chi^f(i) \mid i \in [t_{k-H}, t_k], H \in \mathbb{N}, \forall f \mid P^f \in V(t_k)\} \quad (10)$$

Considering $F_{X_{t_k, H}}(x)$ as the distribution function of $X_{t_k, H}$, the u - *quantile* over (10) is defined by $Q_{X_{t_k, H}}(u)$ as follows:

$$Q_{X_{t_k, H}}(u) = \text{Inf}\{x \mid F_{X_{t_k, H}}(x) \geq u\} \quad (11)$$

$Q_{X_{t_k, H}}(u)$ is a dynamic and variable flow size measurement of the recent history of the flows crossing the system. As the S_3 goal is the detection and isolation of those flows whose sizes are extremely long, or longer than the last flows on the system, then the S_3 function is computed from (11) as follows:

$$S_3 \equiv \chi^f(t_k) \geq Q_{X_{t_k, H}}(u) \quad (12)$$

With the above definitions, S_3 is the differentiating condition between *elephants* and just long flows. In this way, flows longer than $Q_{X_{t_k, H}}(u)$ are considered as *elephants*, therefore, they should be considered with the lowest priority level. In the other case, they are considered as flows with medium priority level. By applying (4) to the differentiation function, the function $\gamma^f(t_k)$ is added to the *shaper* module:

$$\gamma^f(t_k) = I(S_3) \quad (13)$$

2.4 Scheduling the Packets

The last process at the edge device is the *marker*, which uses the set of labels $L = \{P_S, P_L, P_\uparrow, P_\downarrow\}$ for marking the incoming packets with a label $l \in L$ according to the following rules:

$$l = \begin{cases} P_S & \alpha^f(t_k) \\ P_L & \frac{\alpha^f(t_k)}{\alpha^f(t_k) \wedge \beta^f(t_k) \wedge \gamma^f(t_k)} \\ P_\uparrow & \frac{\alpha^f(t_k)}{\alpha^f(t_k) \wedge \beta^f(t_k) \wedge \gamma^f(t_k)} \\ P_\downarrow & \frac{\alpha^f(t_k)}{\alpha^f(t_k) \wedge \beta^f(t_k) \wedge \gamma^f(t_k)} \end{cases} \quad (14)$$

From the above expressions, the LFP algorithm is modelled as the finite state machine depicted in Fig. 4. LFP is composed by the set of states $Q = \{q_0, q_1, q_2\}$ and the set of transitions $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$, where $t_1 = \alpha^f(t_k)$, $t_2 = \alpha^f(t_k)$, $t_3 = \gamma^f(t_k) \wedge \beta^f(t_k)$, $t_4 = \gamma^f(t_k) \wedge \alpha^f(t_k)$, $t_5 = \gamma^f(t_k)$ and $t_6 = 1$.

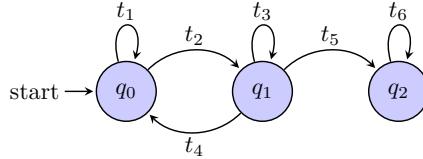


Fig. 4. Finite State Machine of LFP algorithm

When the packets have left the *edge* device, they reach the next device at the DiffServ architecture, that is the *core* device as it is illustrated in Fig. 1. This device is defined with the set of queues $Q = \{Q_{green}, Q_{yellow}, Q_{red}\}$. The packets marked as P_S or P_{\uparrow} are forwarded over Q_{green} ; packets marked with P_L are forwarded over Q_{yellow} and finally, packets marked with P_{\downarrow} are forwarded over the lower priority queue Q_{red} . The dispatching algorithm based on priority queuing where Q_{green} is the highest priority queue, Q_{yellow} is the intermediate priority queue and finally, Q_{red} as the lowest priority queue. Hence, the highest QoS is assured for P_S and P_{\uparrow} packets. The penalisation is for P_{\downarrow} packets because they are always forwarded over Q_{red} . And the rest of them, P_L packets, receive intermediate QoS level, as they neither are *elephants*, nor have received a high priority QoS level due to incoming traffic conditions, token bucket configuration or flow length.

The election of a suitable u -quantile over $X_{t_k, H}$ is important for establishing the threshold of *elephants* detection. Experimental values are obtained from the simulation results that are shown in Fig. 5. Let us consider $u = 99$, then we get a value of $v = 123$ packets and only 20% of the long flows are marked as *elephants*. A 90-quantile of $X_{t_k, H}$ gets a value of $v = 17$ packets, and 35% of long flows are marked as *elephants*. We have decided to select an intermediate value, the 95-quantile, that means $v = 30$ packets. In this case, 30% of long flows are marked as *elephants*.

3 Simulation Results

The simulation has been driven using ns2 [13] in order to improve the end-to-end web traffic latency and analyse the effect over other performance parameters such as dropped packets, throughput and overhead. The network architecture used is based on a single bottleneck dumbbell topology where the DiffServ model has been implemented (see Fig. 1). The QoS system has been implemented as a bottleneck link of 2 Mbps in order to appreciate the congestion level produced by the incoming traffic. Some authors [14,15] have recommended the use of small

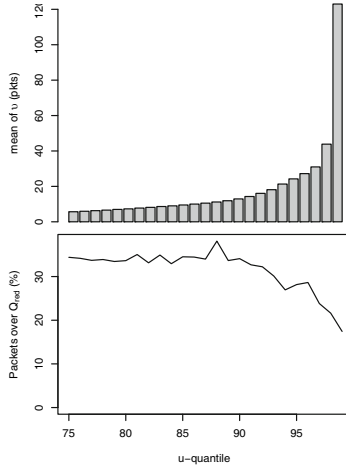


Fig. 5. Effect of u-quantile over the *elephants* detection and v value

buffers in internetwork devices, therefore, according to these recommendations, every device buffer has been configured with a capacity of 50 packets. Parameters related to the buffer size have been modified ($min_{th} = 10, max_{th} = 40$). Both *edge* and *core* devices at the DiffServ architecture have been configured as the RED management with the original values ($max_p = 0.02, w_q = 0.001$). Web traffic is the only traffic that goes through the system. Web clients are modelled as a cloud where the web requests go through the QoS system and reach the web server cloud. Web server responses return from the web server cloud and arrive to the clients after going through the system. For our purposes, web traffic requests are negligible and the analysis is focused only in web traffic responses. The incoming synthetic traffic has been generated by using HTTP PackMime [16], where the web traffic is modelled as stochastic models obtained from the traffic analysis of a real link. The web traffic intensity is modulated with the R parameter, that sets up the incoming traffic as *new conn/s* in the system. We run 10 simulations with the same parameter values but varying the seed, and the results obtained are averaged in order to achieve a higher degree of confidence. In order to analyse the effect of the congestion, three congestion levels have been considered and summarised in the Table 1: *low*, *medium* and *heavy*.

After the marking process at the *edge* node and the scheduling packets process at the *core* node, the packets distribution among Q_{green} , Q_{yellow} and Q_{red} queues is depicted in Fig. 6. As it has been mentioned above, the first τ packets from every flow are marked with label P_S , and therefore, are forwarded over Q_{green} . From $\tau + 1$ packets onward every flow is considered as a long flow. Depending on $\beta^f(t_k)$, some packets are marked with P_\uparrow and therefore, they are forwarded over Q_{green} as well. In the same way, after applying $\gamma^f(t_k)$, elephant flows are detected, and therefore, their packets are marked with P_\downarrow and forwarded over Q_{red} . Finally, the rest of packets are marked as P_L and forwarded over Q_{yellow} .

Table 1. The R parameter has been used to establish each congestion level. The parameters related to the incoming traffic are been calculated : C , mean of simultaneous connections; F , mean simultaneous flows. Performance parameters related to the shared resource are: U , utilisation (%); $Throughput$ in Mbps, is the amount of TCP data transmitted per time; $Goodput$ in Mbps, is the amount of HTTP data transmitted per time; $Overhead$ produced to transmit the required HTTP data; P^{drop} packet drop probability and F^{drop} is the probability that a flow has dropped packets.

Level			Alg	U	$Throughput$	$Goodput$	$Overhead$	P^{drop}	F^{drop}
R	C	F							
Low			DropTail	17.8	0.371	0.356	4.12	1.2e-03	4.08e-03
6	3.22	32.73	RED	18.1	0.383	0.361	5.66	1.56e-02	3.89e-02
			LFP	17.8	0.371	0.356	4.22	1.59e-03	3.33e-03
			Medium			DropTail	31.4	0.657	0.628
10	4.81	52.84	RED	32.9	0.706	0.658	6.79	2.77e-02	7.15e-02
			LFP	30.9	0.647	0.618	4.51	4.82e-03	1.05e-02
			Heavy			DropTail	46.8	0.986	0.936
14	6.91	73.68	RED	43.6	0.947	0.872	7.85	3.85e-02	9.77e-02
			LFP	43.3	0.912	0.867	4.94	9.43e-03	2.24e-02

Focusing on the end-to-end latency metric of a particular traffic level fixed by $R = 10\text{ conn/s}$, the evolution of the latency over the flow size, is shown in Fig. 7. This figure shows that DropTail always gets the worst latency for every flow size. For the shortest flows, RED shows a slightly better behaviour than LFP, but from τ onward, the latency trend changes and its values for LFP are normally lower than RED. This can be explained because from the minimum flow size to τ , every flow is treated with the highest quality of service level in LFP, that clearly improves the latency versus DropTail. Meanwhile, the latency in this range compared to RED suffers a minimal penalisation because of the packets promotion. Close to the τ , all the packets from long flows are treated as short flows packets, and hence, obtain the same latency.

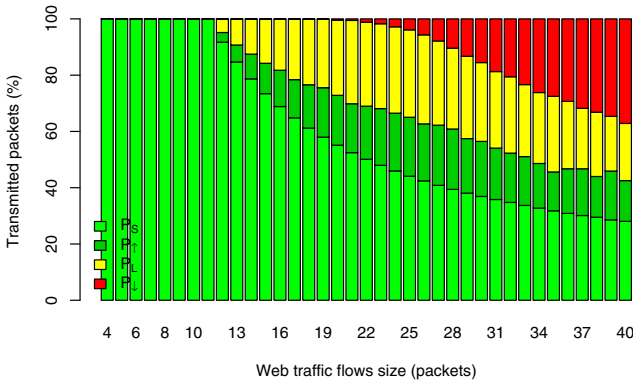


Fig. 6. Packet label classification

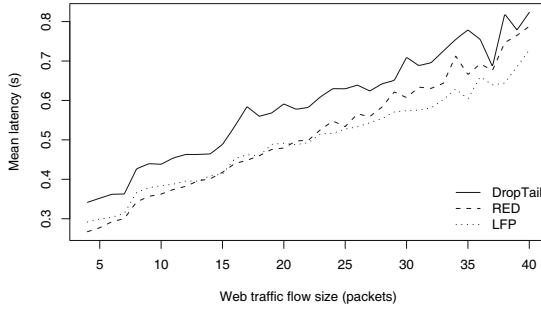


Fig. 7. End-to-end web traffic latency

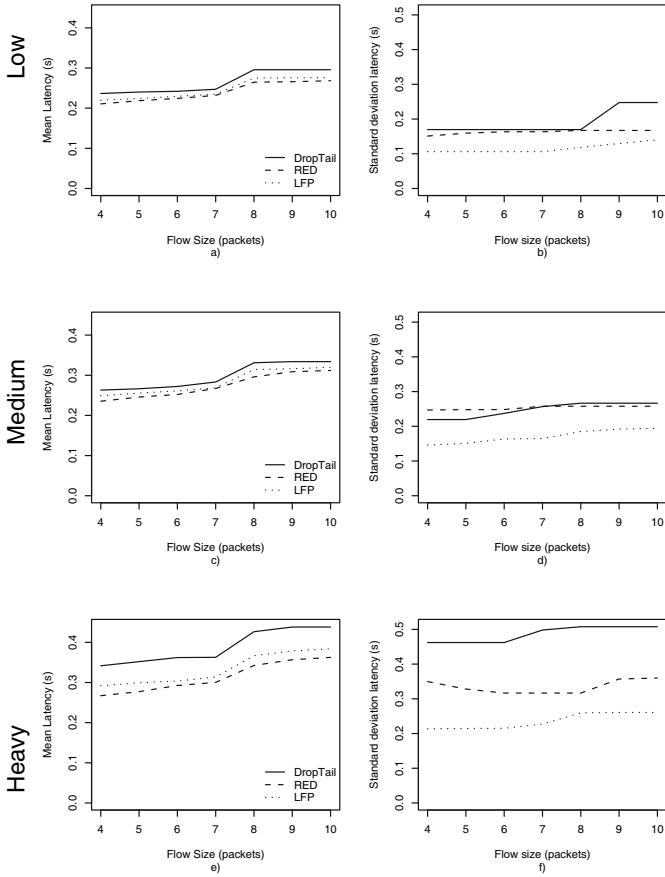


Fig. 8. Mean and standard deviation latency

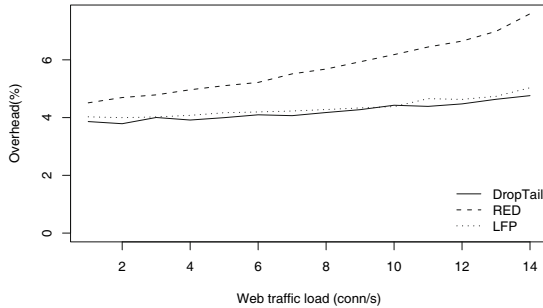


Fig. 9. Overhead

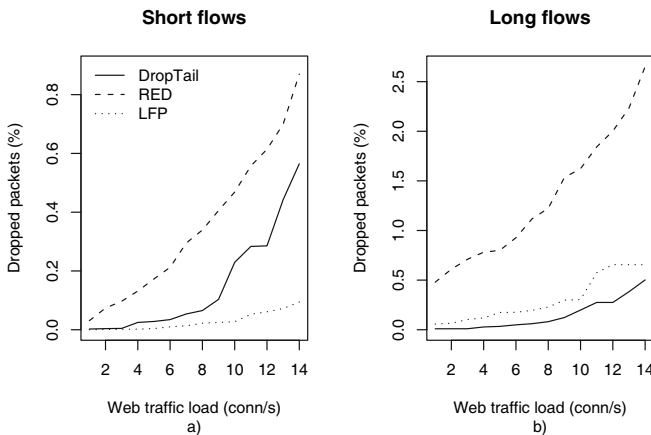


Fig. 10. Dropped packets

The latency of short flows has been isolated and plotted in Fig. 8. The mean latency and standard deviation for DropTail, RED and LFP, for each congestion level have been drawn. Regarding the latency, there are not substantial differences between them. The latency for LFP always remains between DropTail and RED. The standard deviation for LFP is always lower than the other two proposals.

The end-to-end final latency must be analysed with other performance parameters. As it has been summarised in Table 1, LFP obtains a considerably lower *overhead* than RED and very similar *goodput*, for every congestion level. Related to dropped packets, LFP gets the lowest P^{drop} and F^{drop} for each congestion level as well. The *overhead* evolution for each algorithm is clearly depicted in Fig. 9. While DropTail and LFP get almost constant *overhead* when increasing the web traffic load, RED shows a growing trend. Even for heavy congestion level, RED overhead reaches up to 8%, while DropTail and LFP remain slightly over 4%.

Dropped packets effect for DropTail, RED and LFP is shown in Fig. 10. For short flows, LFP is always the proposal with the fewest dropped packets for every web traffic scenario. Obviously, the short flows preferential treatment produces a growth in the long flow queue and therefore there are more dropped packets at heavy congestion level. However, the dropped packets phenomena for long flows using LFP is higher than DropTail, but it is lower than RED for every web traffic scenario.

4 Conclusions

DiffServ architecture has been placed as the most suitable environment to deploy QoS issues in the ISPs. As QoS is not implemented in HTTP protocol, we propose an algorithm named LFP to get preferential treatment for short flows. We also consider the promotion of some long flows under some circumstances and, finally, the penalisation of the extremely long flows.

RED and DropTail are popular algorithms that also implement QoS in a Diff-Serv environment. We have compared the ns2 simulation results obtained with ns2 for LFP, RED and DropTail. We have observed that LFP clearly outperforms DropTail and obtains similar results than RED in terms of mean latency, but improves its standard deviation. Considering the *overhead*, LFP shows an important improvement compared to RED for all congestion levels. There are also benefits in the packet drop probability as LFP always drops less packets than RED. Therefore we consider that LFP algorithm described in this paper is a suitable solution to be used in a QoS Diffserv architecture.

References

1. Berners-Lee, T., Fielding, R., Frystyk, H.: Hypertext transfer protocol – HTTP/1.0 (1996)
2. Cleveland, W.S., Sun, D.X.: Internet traffic data. *Journal of the American Statistical Association* 95, 979–985 (2000); reprinted in Raftery, A.E., Tanner, M.A., Wells, M.T. (eds.) *Statistics in the 21st Century*, pp. 214–228. Chapman & Hall/CRC, New York (2002)
3. Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.: Internet traffic classification demystified: myths, caveats, and the best practices. In: *CONEXT 2008: Proceedings of the 2008 ACM CoNEXT Conference*, pp. 1–12. ACM, New York (2008)
4. Guo, L., Matta, I.: The war between mice and elephants. In: *Ninth International Conference on Network Protocols*, pp. 180–188 (2001)
5. Firoiu, V., Le Boudec, J.Y., Towsley, D., Zhang, Z.L.: Theories and models for Internet quality of service. *Proceedings of the IEEE* 90, 1565–1591 (2002)
6. Crovella, M.E., Bestavros, A.: Self-similarity in world wide web traffic: evidence and possible causes. *IEEE/ACM Trans. Netw.* 5, 835–846 (1997)
7. Floyd, S., Jacobson, V.: Random early detection gateways for congestion avoidance. *IEEE/ACM Trans. Netw.* 1, 397–413 (1993)

8. Brandauer, C., Iannaccone, G., Diot, C., Ziegler, T., Fdida, S., May, M.: Comparison of tail drop and active queue management performance for bulk-data and web-like Internet traffic, p. 0122. IEEE Computer Society, Los Alamitos (2001)
9. Jun, C., Shun-Zheng, Y.: The structure analysis of user behaviors for web traffic. In: ISECS International Colloquium on Computing, Communication, Control, and Management, CCCM 2009, vol. 4, pp. 501–506 (2009)
10. Zhu, X., Yu, J., Doyle, J.: Heavy tails, generalized coding, and optimal web layout. In: Proceedings of Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2001, vol. 3, pp. 1617–1626. IEEE, Los Alamitos (2001)
11. Chen, X., Heidemann, J.: Preferential treatment for short flows to reduce web latency. *Comput. Networks* 41, 779–794 (2003)
12. Ahmed, N.U., Wang, Q., Barbosa, L.O.: Systems approach to modeling the token bucket algorithm in computer networks. *Mathematical Problems in Engineering* 8 (3), 265–279 (2002)
13. The network simulator NS-2, <http://www.isi.edu/nsnam/ns/>
14. Kelly, F., Raina, G., Voice, T.: Stability and fairness of explicit congestion control with small buffers. *SIGCOMM Comput. Commun. Rev.* 38, 51–62 (2008)
15. Shifrin, M., Keslassy, I.: Modeling TCP in small-buffer networks. In: Das, A., Pung, H.K., Lee, F.B.S., Wong, L.W.C. (eds.) *NETWORKING 2008*. LNCS, vol. 4982, pp. 667–678. Springer, Heidelberg (2008)
16. Cao, J., Cleveland, W., Gao, Y., Jeffay, K., Smith, F., Weigle, M.: Stochastic models for generating synthetic HTTP source traffic. In: *INFOCOM 2004*. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 3, pp. 1546–1557 (2004)

Automatic Synchronisation Detection in Petri Net Performance Models Derived from Location Tracking Data

Nikolas Anastasiou¹, William Knottenbelt¹, and Andrea Marin²

¹ Department of Computing, Imperial College London
South Kensington Campus, London SW7 2AZ

{na405,wjk}@doc.ic.ac.uk

² D.A.I.S., Università Ca' Foscari Venezia
via Torino, 155, Venice, Italy
marin@dsi.unive.it

Abstract. The inference of performance models from low-level location tracking traces provides a means to gain high-level insight into customer and/or resource flow in complex systems. In this context our earlier work presented a methodology for automatically constructing Petri Net performance models from location tracking data. However, the capturing of synchronisation between service centres – the natural expression of which is one of the most fundamental advantages of Petri nets as a modelling formalism – was not explicitly supported. In this paper, we introduce mechanisms for automatically detecting and incorporating synchronisation into our existing methodology. We present a case study based on synthetic location tracking data where the derived synchronisation detection mechanism is applied.

Keywords: Location Tracking, Performance Modelling, Data Mining, Generalised Stochastic Petri Nets.

1 Introduction

The proliferation of GPS-enabled mobile devices, RFID tags and high-precision indoor location tracking systems has led to the widespread availability of detailed low-level data describing the movements of customers and/or resources. One way to practically exploit this data in order to gain insight into high-level system operation is to automatically derive a performance model in the form of a queueing network [8] or Stochastic Petri Net [1].

Such models enable us to extract useful information about customer and resource flow and to identify possible bottlenecks in the underlying system. Once a realistic and accurate model has been constructed it can be also used as a predictive tool; for example, the model can be modified to examine the system's performance under hypothetical scenarios, i.e. addition/removal of resources or increased workload.

Our earlier work [1] has presented a methodology which is able to automatically construct Generalised Stochastic Petri Net [2,9] performance models from raw location tracking data. However, synchronisation between service centres was not explicitly captured. This is a serious deficiency since many physical customer processing systems such as hospitals, airports and car assembly lines exhibit many instances of synchronisation. For example, if we consider a treatment room in a hospital, the examination of a patient cannot occur without the presence of a doctor. Thus, the purpose of the present paper is to introduce a mechanism for automatically detecting and incorporating synchronisation into our existing methodology.

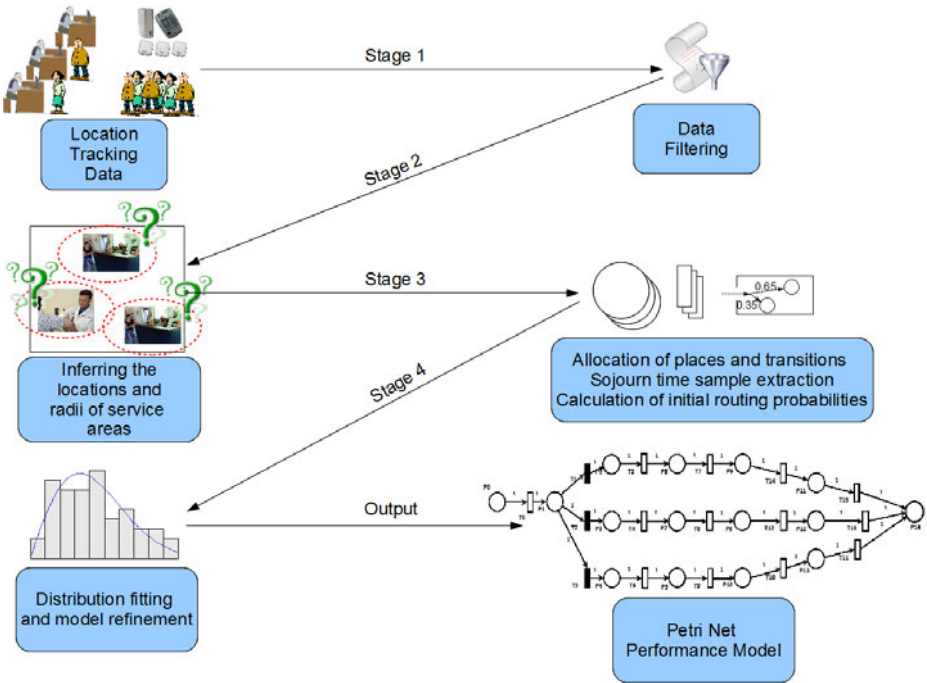


Fig. 1. The four-stage data processing pipeline as in [1]

Our existing methodology is based on a four-stage data processing pipeline (see Figure 1). The first stage of the pipeline performs some basic data filtering. The second stage is responsible for inferring the locations and radii of service areas in the system. The third stage constructs a set of places and transitions to represent customer flow in the system. Sojourn time samples for each customer processed within each service area are also extracted, and separated into waiting time and service time. Likewise, travelling time samples between pairs of service areas are extracted. Finally, a simple counting mechanism is used to calculate the initial routing probabilities of the customers in the system. The fourth stage

uses the G-FIT tool [10] to fit a Hyper-Erlang distribution (HErD) [6] to the extracted service time and travelling time samples and refines both the structure and parameters of the model accordingly. The inferred model is stored in the portable PNML format [3] and can be visualised using PIPE2, an open source platform independent Petri Net editor [4], amongst other tools.

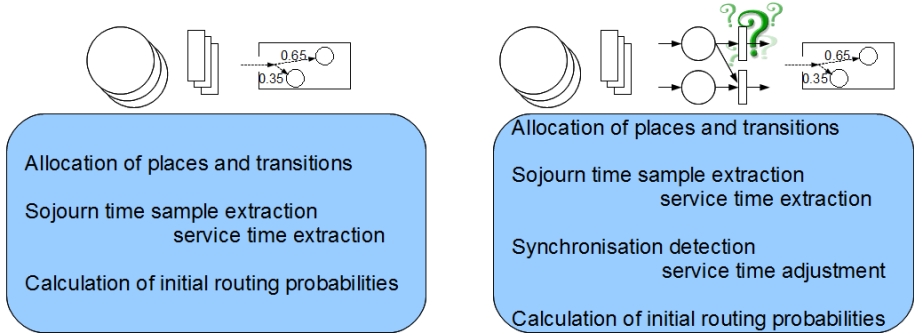


Fig. 2. A high-level description of the third stage of the processing pipeline before (left) and after (right) the incorporation of the synchronisation detection mechanism

Here we focus on the third stage of the data processing pipeline. As shown in overview in Figure 2, and as described in detail in Section 2, we expand this stage by specifying an additional task which detects synchronisation between service areas. Section 3 presents a case study based on synthetic location tracking data generated using an extended version of LocTrackJINQS [7]. Section 4 concludes and considers future work.

2 Synchronisation Detection Mechanism

Our aim is to construct a conservative scheme to determine whether the processing of customers at each service area is subject to some synchronisation conditions (expressed as conditions on the number of customers present within other service areas) at certain time points. To this end, we have designed three functions (see Algorithms 1, 2 and 3) that can be applied together to perform the synchronisation detection task.

To formalise our approach, we introduce some notation:

- N is the total number of service areas inferred from the second stage of the processing pipeline,
- $P = \{P_1, P_2, \dots, P_N\}$ is the set of inferred service areas (subsequently represented by places in the derived Petri net model),
- $C_i = \{c_i^{(1)}, c_i^{(2)}, \dots, c_i^{(n_i)}\}$ is the multiset¹ of all customers processed by P_i , with $n_i = |C_i|$,

¹ Thus supporting the possibility of multiple service periods for the same customer.

- $e_i^{(j)}$ is the timestamp of the entry of $c_i^{(j)}$ into P_i ,
- $s_i^{(j)}$ is the service initiation timestamp of $c_i^{(j)}$, and
- $f_i^{(j)}$ is the service termination timestamp of $c_i^{(j)}$.
- $M_i(t)$ is the number of customers present on service area P_i at timestamp t . This is also referred to as the *marking* of P_i at time t .
- $M_i(t_1, t_2)$ is the maximum number of customers observed on service area P_i during the time interval $[t_1, t_2)$. This is also referred to as the *maximum marking* of P_i during the time interval $[t_1, t_2)$.

Given a service area P_i , we consider the processing of each customer that receives service there in turn. Our approach is based on finding evidence – for each customer – that its processing may have been dependent on the presence of customers on other service areas. This evidence has two components: one is the maximum marking observed on each of the other service areas during the interval during which the customer was serviced; the other is the marking observed on each of the other service areas at the instant of termination of the customer’s service. These two components are combined across all customers processed by P_i – taking into account of the possibility of error and noise – to yield the likely synchronisation conditions of service at P_i^2 . Formally:

Definition 1. *The j th customer $c_i^{(j)} \in C_i$ is said to receive service at P_i with possible synchronisation from each service area $P_k, k = 1, \dots, N$ and $k \neq i$, if:*

$$M_k(s_i^{(j)}, f_i^{(j)}) > 0, \quad (1)$$

and

$$M_k(f_i^{(j)}) > 0 \quad (2)$$

Definition 2. *Synchronisation between server P_i and server(s) $P_k, k = 1, \dots, N$ and $k \neq i$, is inferred if the synchronisation percentage s_p defined as,*

$$s_p(i, k) = \frac{|\{c_i^{(j)} \mid M_k(s_i^{(j)}, f_i^{(j)}) > 0, M_k(f_i^{(j)}) > 0, j = 1, \dots, n_i\}|}{n_i} \quad (3)$$

satisfies

$$s_p \geq s_{thresh} \quad (4)$$

where s_{thresh} is the hypothesis acceptance threshold.

The value of the acceptance threshold (typically in the range $[0.8, 1]$) can be chosen according to factors such as the precision of the location tracking system used, tag update rate and topology of the system being modelled.

² Here we assume a single class of customers. It is straightforward to apply the combination across each customer class in a scenario with multiple customer classes.

2.1 Algorithm Description

Algorithm 1 and Algorithm 2 present auxiliary functions that straightforwardly compute $M_k(t)$ and $M_k(t_s, t_f)$ respectively.

The main function, `computeSynchronisation` (see Algorithm 3), is applied in turn to every service area $P_i \in P$. There are two phases in this algorithm. In the first phase (see lines 4 to 12) we construct the `csmMatrix`, which describes the possible set of synchronisation dependencies between the service of customer j at P_i , and the markings of the other service areas. In this phase, for each P_k , $i \neq k$, we also count the number of customers for which a potential synchronisation was observed, during their service at P_i . In the second phase (see lines 13 to 18) we calculate $s_p(i, k)$ and if its value exceeds the value of s_{thresh} we compute the synchronisation marking on P_k required to support service at P_i . This is computed as a low percentile of the set of synchronisation markings; this is preferred to simply taking the minimum because it is more robust to measurement errors inherent in location tracking systems. This percentile is determined by the function `percentile(M, α)` (see line 16, Algorithm 3) which computes the α th percentile of the set M .

Algorithm 1. $M(k, t) : \text{int}$

```

1: marking  $\leftarrow$  0
2: for  $j = 1$  to  $n_k$  do
3:   if  $c_k^{(j)}$  was present in  $P_k$  at  $t$  then
4:     marking  $\leftarrow$  marking + 1
5:   end if
6: end for
7: return marking

```

Algorithm 2. $M(k, t_s, t_f) : \text{int}$

```

1: maxMarking  $\leftarrow$   $M(k, t_s)$ 
2: for  $j = 1$  to  $n_k$  do
3:   if  $t_s \leq e_k^{(j)} < t_f$  then
4:     instMarking  $\leftarrow$   $M(k, e_k^{(j)})$ 
5:     if instMarking  $>$  maxMarking then
6:       maxMarking  $\leftarrow$  instMarking
7:     end if
8:   end if
9: end for
10: return maxMarking

```

If we assume that $\forall i, n_i = n$, then the worst-case time complexity of the function `computeSynchronisation` and synchronisation detection for the entire network are $O(N \cdot n^3)$ and $O(N^2 \cdot n^3)$ respectively. Based on the same assumption, space complexity is bounded above by the size of `csmMatrix` (see Algorithm 3) and is $O(N \cdot n)$.

Algorithm 3. `computeSynchronisation(i, s_{thresh}) : int[N]`

```

1: customersWithSynch ← new int[N] = {0,0,...,0}
2: synchMarking ← new int[N] = {0,0,...,0}
3: csmMatrix ← new int[ni][N] = { {0,0,...,0}, {0,0,...,0}, ..., {0,0,...,0} }
4: for j = 1 to ni do
5:   for k = 1 to N do
6:     if k == i then continue
7:     csmMatrix[j][k] ← min(M(k, si(j), fi(j)), M(k, fi(j)))
8:     if csmMatrix[j][k] > 0 then
9:       customersWithSynch[k] ← customersWithSynch[k] + 1
10:    end if
11:  end for
12: end for
13: for k = 1 to N do
14:   if k == i then continue
15:   if customersWithSynch[k] / ni ≥ sthresh then
16:     synchMarking[k] ← percentile({csmMatrix[1][k], ..., csmMatrix[ni][k]}, 5)
17:   end if
18: end for
19: return synchMarking

```

Whenever synchronisation is detected involving the processing of customers at P_i , the corresponding service time samples of those customers need to be adjusted to take into account the proportion of time during which the synchronisation condition(s) are satisfied. This is because we assume that service only progresses when the synchronisation condition(s) are met.

2.2 Synchronisation Representation in Our Models

After synchronisation between service areas is detected, it needs to be incorporated into the GSPN performance model that is constructed during stage four of the data processing pipeline.

Considering the place representing P_i and its outgoing service transition t_i , then for every place representing P_k such that `synchMarking[k] > 0`, we connect P_k to t_i using a double-headed arc between P_k and t_i with weight equal to `synchMarking[k]`. We use this representation since we are dealing with location tracking environments where customer entities are preserved.

In [1] we described how we fit a HErD to the extracted service time samples of each service area. We represent each HErD by substituting each transition created in the first task of the third stage by a GSPN subnet, as shown in Figure 3. Now this representation is slightly altered to incorporate synchronisation, when detected.

Let us consider just the transition T_3 in Figure 4 to demonstrate how the synchronisation between P_2 and P_1 is constructed.

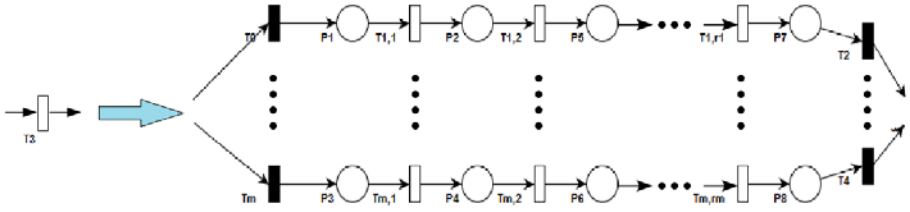


Fig. 3. Replacement of a transition by a GSPN subnet reflecting the fitted HErD (general form)

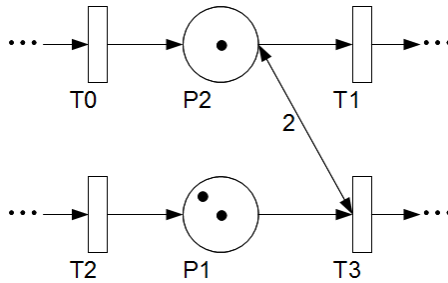


Fig. 4. Modelling synchronisation between server places P_1 and P_2 with P_2 being the synchronising place with a synchronisation marking equal to two

In order to preserve the synchronisation condition on T_3 as shown in Figure 4, we need to connect P_2 to every immediate transition on the left-hand side of the subnet with arcs – one for each immediate transition – of weight two. Similarly, in order to preserve the number of tokens in P_2 we need to connect every immediate transition on the right-hand side of the subnet to P_2 with arcs of weight two. Assuming that T_3 was replaced by a subnet reflecting a four-state HErD with two Erlang branches, each with two states, the resulting model is shown in Figure 5.

3 Case Study

In this section we conduct a case study to test and demonstrate the developed synchronisation detection mechanism. For this case study we have generated location tracking data using an extended version of LocTrackJINQS [7], which supports synchronisation between pairs of service areas.

We present the results for the server location and service radii inference, the synchronisation detection mechanism and the service time distribution fitting (adjusted for synchronisation). Figure 6 shows the experimental setup for the case study and the flow of customers in the system (indicated by arrows).

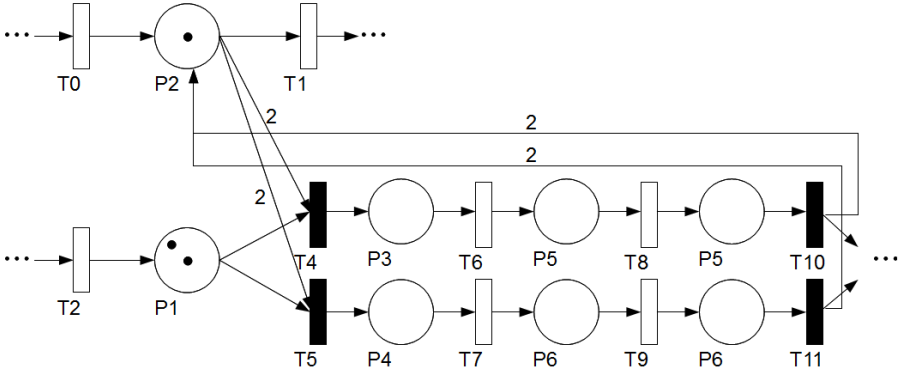


Fig. 5. Modelling synchronisation between places P_1 and P_2 , using a GSPN subnet. Service initiates at P_1 only if P_2 is marked with at least two tokens and P_1 with one.

The simulation takes place in a virtual $25\text{m} \times 25\text{m}$ environment and the customers are assumed to travel within the system with speed drawn from a normal distribution with mean 0.5 m/s and standard deviation 0.15 m/s . The location update error, which emulates the standard error of a real-life location tracking system, is also normally distributed with mean 0.15m and standard deviation 0.2m .

Table 1. The parameters for each service area in the system, for this case study. The parameters of the HErD represent the phase lengths, weights and rates for each branch respectively, separated by a semi-colon.

	Server Location	Service Radius	Service Time Density
S1	(8.0,5.0)	0.5	Erlang(2, 0.1)
S2	(8.0,15.0)	0.7	Exp(0.1)
S3	(16.0,5.0)	0.6	Erlang(4, 0.3)
S4	(16.0,15.0)	0.5	HErD(2, 3; 0.5, 0.5; 0.08, 0.12)

Each service area consists of a single customer processing server and has a random customer service discipline. Service areas S_2 and S_3 require at least one customer to be present in service areas S_1 and S_4 respectively in order to service their customers. The service time for each server follows a different density function. The actual location and service radius of each service area as well as its service time density can be seen in Table 1.

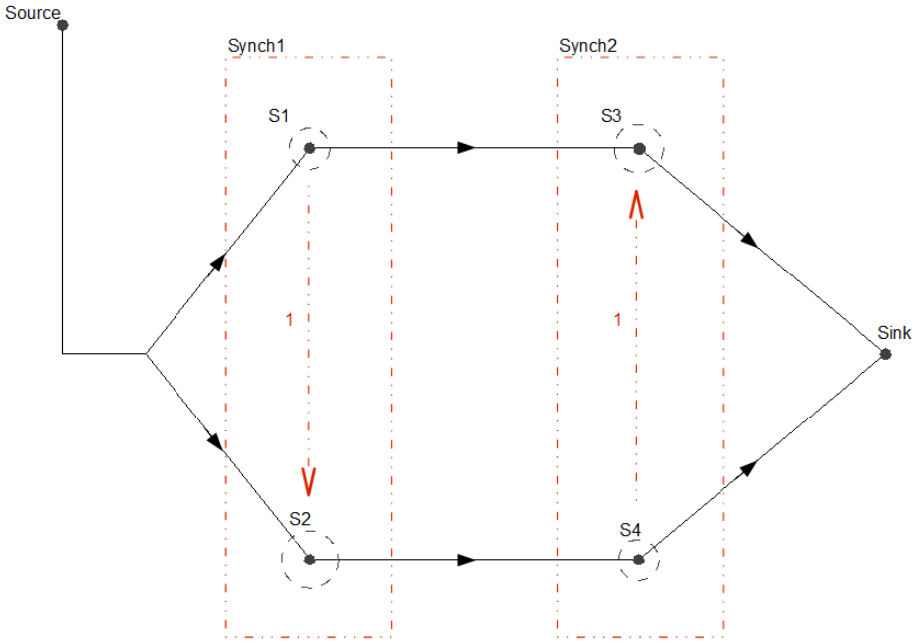


Fig. 6. The experimental setup in terms of abstract system structure. The arrows represent the customer flow in the system and the branching to $S1$ and $S2$ occurs with equal probabilities. The service areas contained in the dotted red rectangles Synch1 and Synch2 are subject to synchronisation. The synchronisation condition is represented by the dotted red arrow. Its source indicates the synchronising service area and its target the service area to be synchronised. The number of customers required to be present in the synchronising service area so that service can be supported in the synchronised service area is denoted by the weight of the dotted red arrow.

3.1 Results

The inferred locations and service radii of the service areas as well as the error between these and their actual values, are depicted in Table 2. From these results we can see that the location and radii of the service areas are approximated very well. The maximum error for the location inference is 0.214 metres and for the service radius approximation is 0.175 metres.

For the evaluation of the sample extraction and HERD fitting process we conduct a Kolmogorov–Smirnov test, to examine the compatibility of the extracted service time samples for each service area with its best-fit HERD (see Tables 3 and 4). Similarly to [1], we enumerate all possible HERDs up to a maximum number of states. This number is set equal to ten, i.e. $N = 10$, for all cases where the coefficient of variation of the extracted sample is greater than 0.4 and twenty five, i.e. $N = 25$, when it is less. The best-fit HERD is chosen using the Akaike Information Criterion (AIC) [5].

Table 2. The inferred location and service radius for each server in the system accompanied with the absolute error, for each case study

	Server Location			Service Radius		
	Real	Inferred	Error	Real	Inferred	Absolute Error
S1	(8.0,5.0)	(7.856,4.989)	0.144	0.5	0.566	0.066
S2	(8.0,15.0)	(7.963,14.947)	0.199	0.7	0.839	0.139
S3	(16.0,5.0)	(16.124,4.964)	0.129	0.6	0.759	0.159
S4	(16.0,15.0)	(16.021,14.961)	0.214	0.5	0.675	0.175

Table 3. The parameters of the HErD fitted for each server's service time density. The parameters of the HErD represent the phase lengths, weights and rate for each branch respectively, separated by a semi-colon.

	Service Time Density	Fitted HErD Parameters		
		Phase Lengths	Rate (3 d.p.)	Weights (3 d.p.)
S1	Erlang(2, 0.1)	4	0.222	1.0
S2	Exp(0.1)	1	0.118	1.0
S3	Erlang(4, 0.3)	2, 4, 4	0.052,0.311,23.232	0.153,0.780,0.067
S4	HErD(2,3;0.5,0.5;0.08,0.12)	3	0.136	1.0

Figure 8 shows the constructed GSPN performance model in compact transition form. We observe that the structure of the inferred model matches the structure of the abstract simulated system. The transitions between pairs of server places (places that correspond to the service areas) represent the travelling time for each particular pair. The weights of the immediate transitions T_1 and T_0 are 0.457 and 0.543 respectively, approximately matching the simulated routing probabilities of the customer flow which are 0.5 and 0.5. In the model we can also see the constructed synchronisation between S2 and S1 (synchronising service area) as well as between S3 and S4 (synchronising service area).

4 Conclusion

This paper has presented a mechanism for synchronisation detection between customer-processing service areas in a system. This mechanism has been implemented as a new component of our existing methodology which automatically constructs GSPN performance models from location tracking data. We conjecture

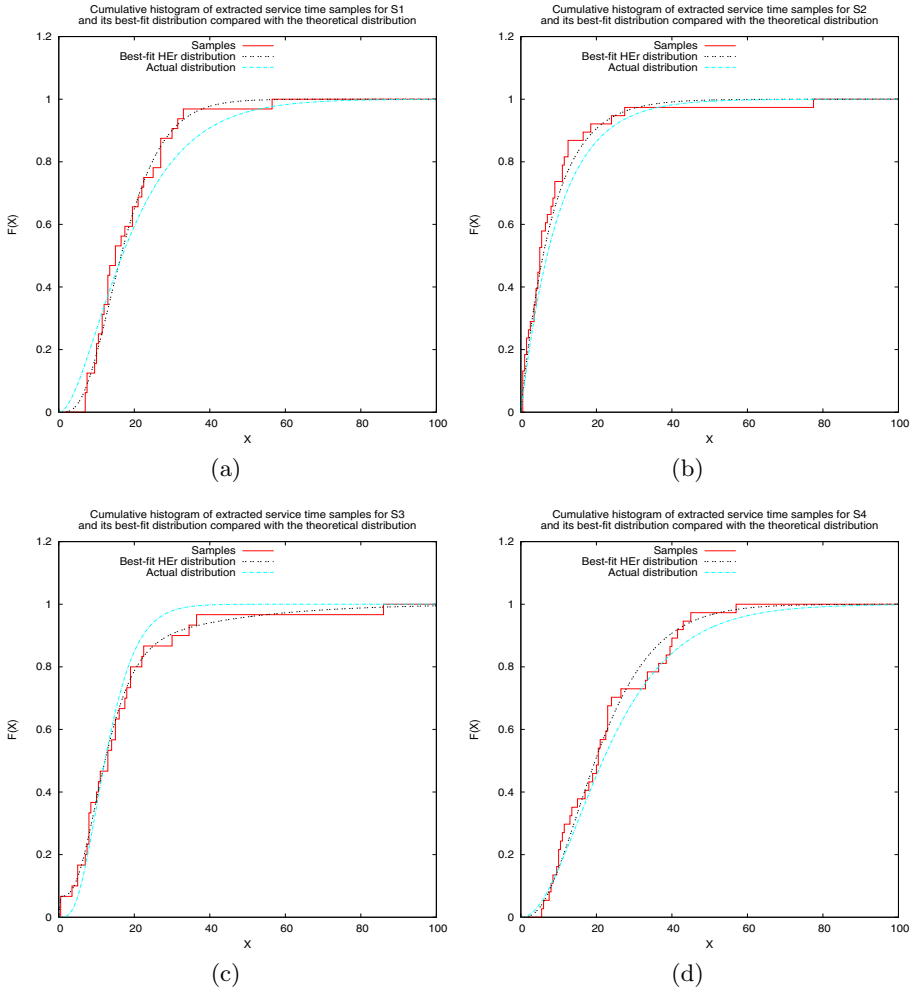


Fig. 7. Case Study 2: Graphs 7(a), 7(b), 7(c) and 7(d) show the cumulative histogram of the extracted service time samples (adjusted for synchronisation for S2 and S3) and its best-fit hyper-Erlang distribution compared with the theoretical distribution for S1, S2, S3 and S4 respectively

that our methodology can be applied to a large variety of systems whose underlying GSPN structure includes extended free choice (EFC) nets. An example of a real-life situation where this methodology could be successfully applied is a Magnetic Resonance Imaging (MRI) unit of a hospital. The MRI control room is physically separate from the MRI chamber and requires a radiologist to operate it; the patient screening process cannot initiate if a radiologist is not present in the control room.

Table 4. Kolmogorov-Smirnov test at significance levels 0.1 and 0.05 applied to the extracted service time samples (adjusted for synchronisation) for each service area in the case study. The null hypothesis is that the extracted samples belong to the corresponding best-fit HErD.

S1	Test Statistic	0.1268	
	α	0.1	0.05
	Critical Values	0.2076	0.2307
	Compatible ?	Yes	Yes
S2	Test Statistic	0.1074	
	α	0.1	0.05
	Critical Values	0.1914	0.2127
	Compatible ?	Yes	Yes
S3	Test Statistic	0.0861	
	α	0.1	0.05
	Critical Values	0.0.2141	0.2378
	Compatible ?	Yes	Yes
S4	Test Statistic	0.0921	
	α	0.1	0.05
	Critical Values	0.1938	0.2153
	Compatible ?	Yes	Yes

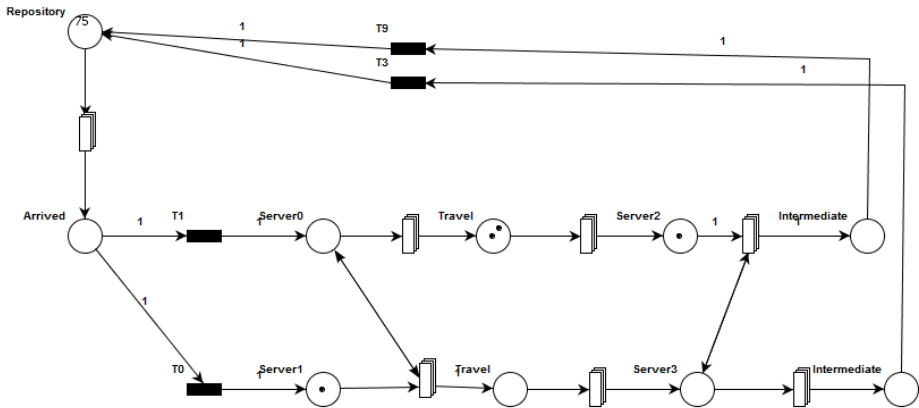


Fig. 8. Visualisation of the inferred GSPN performance model for the case case study (in compact transition form)

The case study results indicate that the developed methodology can infer the stochastic features and the presence of synchronisation in simple systems accurately, at least when synthetically-generated location tracking data is used.

Our current work has made several assumptions, i.e. one customer class, single-server semantics and random service discipline. In our future work we wish to relax them. We aim to use Coloured Generalised Stochastic Petri Nets (CGSPNs) to enable the support of multiple customer classes as well as prioritised service

disciplines. Using CGSPNs we could also improve the accuracy of our models since we can use transitions that change the colour of the token (upon their firing) and thus control the routing of customers as they pass through various processing stages.

References

1. Anastasiou, N., Horng, T.-C., Knottenbelt, W.: Deriving Generalised Stochastic Petri Net performance models from High-Precision Location Tracking Data. In: Proc. 5th International ICST Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2011), Paris, France (May 2011)
2. Bause, F., Kritzinger, P.: Stochastic Petri Nets. Friedrich Vieweg & Sohn Verlag (2002)
3. Billington, J., Christensen, S., van Hee, K.M., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, technology, and tools. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 483–505. Springer, Heidelberg (2003)
4. Bonet, P., Llado, C.M., Puijaner, R., Knottenbelt, W.: PIPE v2.5: A Petri Net Tool for Performance Modelling. In: Proceedings of 23rd Latin American Conference on Informatics (CLEI 2007), San Jose, Costa Rica (October 2007)
5. Bozdogan, H.: Model selection and Akaike's Information Criterion (AIC): The general theory and its analytical extensions. *Psychometrika* 52, 345–370 (1987)
6. Fang, Y.: Hyper-Erlang Distribution Model and its Application in Wireless Mobile Networks. *Wireless Networks* 7, 211–219 (2001)
7. Horng, T.-C., Anastasiou, N., Knottenbelt, W.: LocTrackJINQS: An Extensible Location-aware Simulation Tool for Multiclass Queueing Networks. In: Proc. 5th International Workshop on Practical Applications of Stochastic Modelling (PASM 2011), Karlsruhe, Germany (March 2011)
8. Horng, T.-C., Dingle, N., Jackson, A., Knottenbelt, W.: Towards the Automated Inference of Queueing Network Models from High-Precision Location Tracking Data. In: Proc. 23rd European Conference on Modelling and Simulation (ECMS 2009), pp. 664–674 (May 2009)
9. Marsan, M., Conte, G., Balbo, G.: A Class of Generalized Stochastic Petri Nets for Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems* 2(2), 93–122 (1984)
10. Thümmler, A., Buchholz, P., Telek, M.: A Novel Approach for Phase-Type Fitting with the EM Algorithm. *IEEE Transactions on Dependable and Secure Computing* 3, 245–258 (2005)

Performance Evaluation of Business Processes through a Formal Transformation to SAN

Kelly Rosa Braghetto^{1,*,**}, João Eduardo Ferreira¹, and Jean-Marc Vincent²

¹ Department of Computer Science, University of São Paulo
Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, Brasil
{kellyrb, jef}@ime.usp.br

² LIG Laboratory – INRIA MESCAL Project, Joseph Fourier University
51, avenue Jean Kuntzmann, F-38330, Montbonnot, France
Jean-Marc.Vincent@imag.fr

Abstract. The qualitative and quantitative analysis of operational processes recently started to receive special attention with the *business process management* systems. But the *Business Process Model and Notation* (BPMN), the standard representation of business processes, is not the most appropriate kind of model to support the analysis phase. Most of the works proposing mappings from BPMN to formal languages aim model verification, but few are directed to quantitative analysis. In this work, we state that a well-defined BPMN Process diagram can originate a *Stochastic Automata Network* (SAN) – a compositionally built stochastic model. More than support verification, SAN provides a numerical evaluation of processes’ performance. SAN attenuates the state-space explosion problem associated with other Markovian formalisms and is used to model large systems. We defined an algorithm that automatically converts BPMN diagrams to SAN models. With these SAN models, we make analytical performance evaluations of business processes.

Keywords: Business Processes, BPMN, Performance Evaluation, Stochastic Automata Network.

1 Introduction

Significant efforts have been made to standardize modeling and execution languages, in order to improve the interoperability of tools developed to Business Process Management (BPM). The most important result of these efforts is the *Business Process Model and Notation* (BPMN) [7], a standard notation for graphical representation of business processes. Despite being able to support business users in different phases of the business process life cycle, BPMN models are not the most appropriate models to support the analysis phase. Since BPMN models have no formal semantics, they are not well suited to qualitative analysis. Furthermore, BPMN models do not provide mechanisms to quantify the effort required to perform the activities, nor the capacity of work of shared resources and their access policies. This deficiency hinders the use of BPMN for performance

* The student was supported by the Brazilian government (CAPES and FAPESP).

** Contact author. The order of authors is merely alphabetical.

evaluation (quantitative analysis). In BPM domain, the objective of performance analysis is to evaluate performance indices – service time, waiting time, queue size, and resource utilization – that enable us to improve the business processes by identifying inefficiencies, such as bottlenecks and idle resources.

In this work, we state that a well-defined BPMN Process diagram can originate a *Stochastic Automata Network* (SAN) [8]. More than support verification, SAN models are able to provide a numerical performance analysis of business processes. SAN is a structured Markovian formalism that enables us to build stochastic models in a compositional approach. Created to attenuate the well-known state-space explosion problem associated with the Markovian formalisms, SAN can be applied in the modeling of large/complex systems. It is very efficient regarding the memory consumption, in addition to provide the concept of functional rates – a feature that improve the expressiveness of the stochastic formalism and facilitate the modeling of the dependencies that may exist between the rate/probability of components and the global state of the system.

Our main contribution is an algorithm that automatically converts BPMN Process diagrams to SAN models by means of a set of mappings and simple operations that we defined over the models. The algorithm was implemented as part of a software tool called *BP2SAN*. With the support of a SAN solver (such as the software tool *PEPS* [2]), we extract from the automatically generated models variated performance indices of the business processes.

The automatically generated SAN model represents the behavior of one instance of the business process. To analyze the behavior of the system when n process instances are being executed in parallel (sharing finite resources), we just need to create n replicas of the initial SAN model. When we consider multiple parallel instances we easily obtain models with more than one million of states, considered intractable in several analysis techniques. However, with SAN we can treat such large state spaces more frequently than with other techniques.

The paper's remainder is organized as follows. Section 2 discusses related works, while Section 3 briefly presents SAN and BPMN. Section 4 formalizes the structure and the properties of the models we use in this work. Our algorithm for the conversion of BPMN models to SAN is defined in Section 5. Section 6 provides an example to illustrate the conversion of a business process to SAN and presents some performance results obtained from the analytical model. Finally, concluding remarks are made in Section 7.

2 Related Works

Several works such as [4,11] proposed mappings from business process models to formal languages aiming validation and verification. Other approaches such as [5,3,10,1] are devoted to the conversion of business process models to stochastic formalisms (e.g., stochastic Petri nets and stochastic process algebras) aiming quantitative analysis. In this section, we will restrict ourselves to the latter.

The work of Canevet *et al.* [3] proposed an automated mapping from the *Unified Modeling Language* (UML) state diagrams enhanced with performance information to *Performance Evaluation Process Algebra* (PEPA). The performance information they refer are probabilities attached to the states and rates

attached to the transitions of the UML state model. One important advantage of the approach proposed by the authors is that the performance results obtained from the solution of the PEPA model can be reflected back to the UML level. However, the approach does not support functional rates, preventing some important aspects related to performance from being contemplated in the modeling.

The proposal of Prandi *et al.* [10] was a mapping from BPMN to *Calculus for Orchestration of Web Services* (COWS), a process calculus inspired by the *Business Process Execution Language*. The authors made a brief discussion about the use of a stochastic extension of COWS in the quantitative analysis of business processes. Despite being based in a compositional formalism, Stochastic COWS does not explore this feature in analysis, thus suffering of the same state-space explosion problem that limits the use of other Markovian formalisms in practice.

Oliveira *et al.* [5] proposed an approach to model resource-constrained business processes using *Generalized Stochastic Petri Nets* (GSPN). In their approach, the resulted nets can be unbounded and, for this reason, they need to use simulation to obtain performance indices over the process models.

Braghetto *et al.* [1] compared the viability of applying three stochastic formalisms – GSPN, PEPA and SAN –, in the analytical modeling of business processes. They verified that the formalisms are able to express with equivalent facilities basic business process scenarios, but more advanced scenarios evidenced their pros et cons. Since SAN and PEPA are intrinsically compositional, they enable a structured analysis, in addition to the facility to extend a model without impacting the previous modeled behavior. SAN and GSPN have the explicit notion of state and the concept of functional rates, what helps to model functional dependencies between the process components. The study made in [1] was the first to consider the use of SAN to model business processes. But the translations presented from BPMN to SAN were handmade, in small examples, and indicated that a more automatic approach should take place if we want to use these models to analyze business processes in practice.

3 Fundamentals

Two topics are required for the understanding of this work: SAN and the main structures of the BPMN Process diagrams. We present them in this section.

3.1 Stochastic Automata Network

The *Stochastic Automata Network* (SAN) is a technique used to model systems with large state spaces, introduced by Plateau in 1985 [8, 9]. SAN has been successfully applied to model parallel and distributed systems that can be viewed as collections of components that operate more or less independently, requiring only infrequent interaction such as synchronizing their actions, or operating at different rates depending on the state of parts of the overall system.

A system is described in SAN as a set of N subsystems modeled as a set of stochastic automata $A^{(i)}$, $1 \leq i \leq N$, each one containing n_i local states and transitions among them. The global state of a SAN is defined by the combinations of the internal state of each automaton. A change in the state of a SAN is caused

by the occurrence of an *event*. *Local events* cause a state transition in only one automaton (*local transition*), while *synchronizing events* cause simultaneous state transitions in more than one automaton (*synchronizing transitions*). A transition is labeled with the list of events that may trigger it.

All event transitions in the model are associated with rates (the inverse of the average execution time) in which the transitions occur. The rate is the parameter of an exponential distribution that governs the behavior of the transition. The rate of an event may be constant (a nonnegative real number) or may depend upon the state in which it takes place. In this last case, the rate is a function from the global state space to the nonnegative real numbers and is called *functional transition rate*. For example, one can use functional transition rate to model how the execution time of an activity in the system is affected by the variation of the workload, or to model dependency existent between the probability of the execution of an activity and the current state of the process.

The expression of the *infinitesimal generator* (transition rate matrix) of the underlying Markov chain of a well defined SAN is given by the generators on these smaller spaces and by operators from the *Generalized Tensor Algebra* (GTA), an extension of the *Classical Tensor Algebra* (CTA). The tensor formula that gives the infinitesimal generator of a SAN model is called *Markovian Descriptor*.

Each automaton $A^{(i)}$ of a SAN model is described by $n_i \times n_i$ square matrices. In the case of SAN models with synchronizing events, the descriptor is expressed in two parts: a *local* part (to group the local events), and a *synchronizing* part (to group the synchronizing events). The local part is defined by the tensor sum of $Q_l^{(i)}$ – the infinitesimal generator matrices of the local transitions of each $A^{(i)}$. In the synchronizing part, each event corresponds to two tensor products: one for the occurrence matrices $Q_{s^+}^{(i)}$ (expressing the positive rates) and the other for the adjusting matrices $Q_{s^-}^{(i)}$ (expressing the negative rates). The descriptor is the sum of the local and the synchronizing parts, expressed as:

$$Q = \bigoplus_{i=1}^N \bigotimes_g Q_l^{(i)} + \sum_{s \in \varepsilon} \left(\bigotimes_{i=1}^N \bigotimes_g Q_{s^+}^{(i)} + \bigotimes_{i=1}^N \bigotimes_g Q_{s^-}^{(i)} \right) \quad (1)$$

where $\begin{cases} N \text{ is the number of automata of the SAN model} \\ \varepsilon \text{ is the set of identifiers of synchronizing events} \end{cases}$

The state-space explosion problem associated with Markov chain models is attenuated by the fact that the state transition matrix is stored in a compact form, since it is represented by smaller matrices. All relevant information can be recovered from these matrices without explicitly build the global matrix.







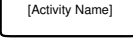


A SAN model can be numerically solved using the PEPS tool [2]. PEPS includes several numerical iterative methods to solve SAN models and implements strategies to improve the time/space trade-off in the computation of the solutions.

3.2 Business Process Model and Notation

The *Business Process Model and Notation* (BPMN) is a standard for graphical representation of business processes. According to the specification document of

BPMN [7], “a process describes a sequence or flow of activities in an organization with the objective of carrying out work”. A Process diagram is a graph of elements – activities, events, gateways, and sequence flows – that define a finite execution semantics. In this work we deal with a subclass of the models that can be represented as a BPMN Process diagram – the well-defined ones (Section 4.1, Definition 5) –, in order to guarantee that the conversion to a SAN model can be made. Table 1 introduces the BPMN objects accepted as input by our conversion algorithm. These objects are very important to process modeling and with them we are able to express a rich class of business process models.

Table 1. Basic flow and connecting objects of BPMN

	Start Event		Exclusive Gateway		Sequence Flow
	End Event		Parallel Gateway		Association
	Atomic Activity		Inclusive Gateway		Annotation

The *start event* indicates where a process will start, and *end event* indicates where a process will end. An *activity* is an atomic work performed in the process. A *sequence flow* shows the order in which activities will be performed. A *gateway* is used to control the divergence and convergence of sequence flows. In this work, we deal with the following gateway types:

- *Exclusive Gateways.* A *diverging exclusive gateway* (XOR-split) is used to create alternative paths within a process flow (only one path can be taken). A *converging exclusive gateway* (XOR-join) is used to merge alternative paths;
- *Parallel Gateways.* A *diverging parallel gateway* (AND-split) creates parallel paths. The *converging parallel gateway* (AND-join) will wait for all incoming flows before triggering the flow through its outgoing sequence flows;
- *Inclusive Gateways.* A *diverging inclusive gateway* (OR-split) can be used to create alternative but also parallel paths within a process flow. Unlike the exclusive gateway, all condition expressions are evaluated. All sequence flow with a true evaluation will be taken. A *converging inclusive gateway* (OR-join) is used to merge a combination of alternative and parallel paths.

An *association* is used to link information with graphical elements. *Text annotations* provide additional information for readers of the BPMN diagrams.

A process is instantiated when one of its start events occurs. Each start event that occurs creates a token on its outgoing sequence flow, which is followed as described by the semantics of the other process elements. For example, the parallel gateway is activated if there is at least one token on each incoming sequence flow; the parallel gateway consumes exactly one token from each incoming sequence flow and produces exactly one token at each outgoing sequence flow. We can consider that a process instance is completed if and only there is no token remaining within the process instance.

4 Definitions for the BPMN to SAN Conversion

Section 4.1 introduces the properties of a BPMN graph that we consider as a valid input for our conversion algorithm. In Section 4.2 we formally define the structure of the SAN graph resulted from a conversion, and the operations over SAN graphs that support the algorithm (described in Section 5).

4.1 BPMN Graph Definitions

As we briefly discussed in Section 3.2, a BPMN Process diagram is a directed graph constituted of vertices of events, activities and gateways. In this work, we restricted ourselves to a subclass of all process diagrams that can be formed from BPMN objects. Definitions 1 to 5 formally describe this subclass.

Definition 1. *BPMN Process Graph*

A BPMN Process graph BG is a directed graph represented by the tuple $BG = (V, E, L, \ell, p)$, with $V = S \cup A \cup G \cup F$, where:

- S is a set of vertices representing the start events
- A is a set of vertices representing the atomic activities
- G is a set of vertices representing the gateways
- F is a set of vertices representing the end events
- $E \subseteq (V \times V)$ is a set of directed edges
- L is a set of vertex labels
- $\ell : V \rightarrow L$ is a labeling function of vertices
- $p : E' \rightarrow [0, 1]$, where $E' \subseteq E$, is a partial probability function that associates edges with probability values

A directed edge in BG is a pair (v, w) – where $v, w \in V$, indicating that v is an input vertex of w , and w is an output vertex of v . A label is used to denote the name of the event or activity being modeled, but in the case of a gateway vertex, it indicates the gateway type.

Definition 2. *Path in a BPMN Process Graph*

Let $BG = (V, E, L, \ell, p)$ be a BPMN Process graph and $v_1, v_n \in V$.

A path from v_1 to v_n (represented by $v_1 \rightsquigarrow v_n$) is a sequence of vertices v_1, v_2, \dots, v_n (with $v_i \in V$) such that, for $0 < i < n$, $(v_i, v_{i+1}) \in E$.

Definition 3. *Input Vertices and Output Vertices*

Let $BG = (V, E, L, \ell, p)$ be a BPMN Process graph.

The functions $\text{inputs} : V \rightarrow 2^V$ and $\text{outputs} : V \rightarrow 2^V$ that give the input vertices and the output vertices, respectively, of a vertex in BP are defined as $\forall v \in V, \text{inputs}(v) = \{u \in V \mid (u, v) \in E\}$ and $\text{outputs}(v) = \{w \in V \mid (v, w) \in E\}$.

In order to minimize the semantic ambiguities that BPMN constructors can introduce in the model and to guarantee some properties that a well-formed business process model must respect, our conversion method makes some assumptions about the BPMN model provided as input (formalized in Definition 4). These assumptions facilitate the conversion and were already used (mainly the first 5 ones) in other related works, such as in [10].

Definition 4. *Well-Formed BPMN Process Graph*

Let $BG = (V, E, L, \ell, p)$ be a BPMN Process graph, with $V = S \cup A \cup G \cup F$. BG is a well-formed BPMN Process graph if and only if:

- $\forall v \in S, (|\text{inputs}(v)| = 0) \wedge (|\text{outputs}(v)| = 1)$
- $\forall v \in F, (|\text{inputs}(v)| = 1) \wedge (|\text{outputs}(v)| = 0)$
- $\forall v \in A, (|\text{inputs}(v)| = 1) \wedge (|\text{outputs}(v)| = 1)$
- $\forall v \in G, (\ell(v) = "+") \vee (\ell(v) = "O") \vee (\ell(v) = "X")$
- $\forall v \in G, ((|\text{inputs}(v)| > 1) \wedge (|\text{outputs}(v)| = 1)) \vee ((|\text{inputs}(v)| = 1) \wedge (|\text{outputs}(v)| > 1))$
- $\forall v \in A \cup G, \exists s \in S$ such that \exists a path $s \rightsquigarrow v$
- $\forall v \in A \cup G, \exists f \in F$ such that \exists a path $v \rightsquigarrow f$
- $\forall v \in G$ such as $(\ell(v) = "X") \vee (\ell(v) = "O")$, $\forall w \in \text{outputs}(v)$, $p((v, w))$ must be defined
- $\forall v \in G$ such as $\ell(v) = "X"$, $\sum_{w \in \text{outputs}(v)} p((v, w)) = 1$

Therefore, a well-formed BPMN Process graph is a graph in which:

- a start event vertex can have only one output vertex and no input vertices;
- an end event vertex can have only one input vertex and no output vertices;
- an activity vertex can have only one input vertex and only one output vertex;
- each gateway vertex has one of the following labels: "+", for a parallel gateway; "O", for an inclusive gateway; and "X", for an exclusive gateway;
- a gateway vertex can perform a role of divergence or a role of convergence (but not the two roles at the same time). As a consequence, a gateway can have only one input vertex and more output vertices (case of divergence), or can have only one output vertex and more input vertices (case of convergence);
- for all vertex v of activity or gateway: (i) there exists a path from one start event vertex to v , and (ii) there exists a path from v to one end event vertex;
- each output edge of an exclusive/inclusive gateway vertex must have an associated probability value;
- the sum of the probabilities of the output edges of an exclusive gateway vertex must be 1.

It is important to mention that the association of edges with probabilities does not exist in the specification of BPMN. We introduced this feature in our definition because the dynamic of a business process is probabilistic by essence and probabilities are quantifications of the business process behavior.

The assumptions above are all related to syntactical properties of the BPMN model, i.e. structural restrictions over the BPMN graph. But there exist also some important semantical properties that we need to assume to have a well-defined BPMN Process model (specified by Definition 5). In a well-defined BPMN Process model, two important properties for a business process are granted: (i) it does not contain unreachable activities, and (ii) it can always terminate.

Definition 5. *Well-Defined BPMN Process Model*

A well-defined BPMN Process model is a well-formed BPMN Process graph in which:

- an exclusive gateway does not converge (join) parallel sequence flows;

- a parallel gateway does not converge (synchronize) alternative sequence flows;
- an inclusive gateway only converges (merges) sequence flows originated by another inclusive gateway. In addition, there is an one-to-one correspondence between the diverging and the converging inclusive gateways.

4.2 SAN Model Definitions

Definitions from 6 to 11 formally specify the structure of a SAN model and its operations such as they are used in our conversion algorithm.

Definition 6. SAN Model and SAN Automaton

A SAN model \mathcal{S} is a set $\mathcal{S} = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_n\}$ of n SAN automata.

A SAN automaton \mathcal{A} is given by a tuple $\mathcal{A} = (Q, E, T, L, \ell, p)$, where:

- Q is a set of states
- E is a set of events
- $T \subseteq (Q \times Q \times E)$ is a set of state transitions labeled by events
- L is a set of state labels
- $\ell : Q \longrightarrow L$ is a labeling function of states
- $p : T' \longrightarrow [0, 1]$, where $T' \subseteq T$, is a partial probability function that associates a transition with a probability

Definition 7. Input Transitions and Output Transitions

Let $\mathcal{A} = (Q, E, T, L, \ell, p)$ be a SAN automaton.

The functions $\text{inputs} : Q \rightarrow 2^T$ and $\text{outputs} : Q \rightarrow 2^T$ that give the input transitions and output transitions, respectively, of a state of \mathcal{A} are defined as

$$\begin{aligned} \forall q \in Q, \text{inputs}(q) &= \{(p, q, e) \mid (p, q, e) \in T\} \text{ and} \\ \text{outputs}(q) &= \{(q, r, e) \mid (q, r, e) \in T\}. \end{aligned}$$

Definition 8. Source State and Absorbing State

Let $\mathcal{A} = (Q, E, T, L, \ell, p)$ be a SAN automaton and $q \in Q$ a state of \mathcal{A} .

If $\text{inputs}(q) = \emptyset$, then q is a source state.

If $\text{outputs}(q) = \emptyset$, then q is an absorbing state.

In this work, we propose an algorithm to automatically convert a well-defined BPMN Process model (Definition 4) to a SAN model in the format of Definition 6. This conversion starts with the individual mapping of the objects of the BPMN graph into SAN objects, according with the illustrations in Table 2. In a general way, we can say that the mapping of each vertex of the BPMN graph originates in the SAN model at least one new automaton, with at least two states and a transition between them (associated with a new event labeled with the identifier of the BPMN vertex). Event, activity, and exclusive gateway vertices generate only local events. Parallel gateways originates only synchronizing events. Inclusive gateways (that must appears in pairs, delimiting closed blocks) generate both local and synchronizing events.

Parallel sequence flows are mapped to sets of synchronized automata. A choice is expressed by a state that has more than one output transition (remembering that, in this case, each output transition must be weighted by a probability value). An atomic activity a is mapped into a 3-state sequential automaton, where the

first state indicates that the activity is waiting for the availability of its required resources, the second state indicates that it had already obtained the access to the resources (by event r_a), and the third state indicates that the execution of the activity is finished (with the occurrence of event a).

In order to obtain the SAN model correspondent to a well-defined BPMN Process model, we must compound the simple automata generated from the mappings of Table 2. This composition is made by means of a set of *simplification operations* applicable over SAN models, that we define in the following.

Definition 9. *State Merging Operation* (\triangleright)

Let $\mathcal{A} = (Q, E, T, L, \ell, \mathfrak{p})$ be a SAN automaton and $q_1, q_2 \in Q$ be states of \mathcal{A} .

The state merging of q_1 and q_2 in \mathcal{A} (represented by $\mathcal{A}[q_1 \triangleright q_2]$) results in a SAN automaton $\mathcal{A}_M = (Q_M, E_M, T_M, L_M, \ell_M, \mathfrak{p}_M)$ such that:

- $Q_M = Q \setminus \{q_2\}$
- $E_M = E$
- $T_M = \{(p, q, e) \in T \mid (p \neq q_2) \wedge (q \neq q_2)\} \cup \{(p, q_1, e) \mid (p, q_2, e) \in T\} \cup \{(q_1, q, e) \mid (q_2, q, e) \in T\}$
- $L_M = L$
- $\forall q \in Q_M, \ell_M(q) = \ell(q)$
- $\forall (p, q, e) \in T_M, \mathfrak{p}_M((p, q, e)) = \begin{cases} \mathfrak{p}((p, q, e)), & \text{if } (p, q, e) \in T \\ \mathfrak{p}((p, q_2, e)), & \text{if } (q = q_1) \wedge ((p, q_2, e) \in T) \\ \mathfrak{p}((q_2, q, e)), & \text{if } (p = q_1) \wedge ((q_2, q, e) \in T) \end{cases}$

This operation eliminates q_2 from \mathcal{A} , transforming all the input/output transitions of q_2 in input/output transitions of q_1 (keeping the label of q_1 unchanged).

Definition 10. *State Suppression Operation* (\blacktriangleright)

Let $\mathcal{A} = (Q, E, T, L, \ell, \mathfrak{p})$ be a SAN automaton and $q \in Q$ be a state of \mathcal{A} such that $|\text{outputs}(q)| = 1$. Let t_o be the output transition of q and o be its output state.

The state suppression of q in \mathcal{A} (represented by $\mathcal{A}[q \blacktriangleright]$) results in a SAN automaton $\mathcal{A}_S = (Q_S, E_S, T_S, L_S, \ell_S, \mathfrak{p}_S)$ such that:

- $Q_S = Q \setminus \{q\}$
- $E_S = \{e \in E \mid (p, r, e) \in (T \setminus \{t_o\})\}$
- $T_S = \{(p, r, e) \in T \mid (p \neq q) \wedge (r \neq q)\} \cup \{(p, o, e) \mid (p, q, e) \in T\}$;
- $L_S = \{l \in L \mid \exists p \in Q, ((\ell(p) = l) \wedge (p \neq q))\}$
- $\forall q \in Q_S, \ell_S(q) = \ell(q)$
- $\forall (p, r, e) \in T_S, \mathfrak{p}_S((p, r, e)) = \begin{cases} \mathfrak{p}((p, r, e)), & \text{if } (p, r, e) \in T \\ \mathfrak{p}((p, q, e)), & \text{in the other cases} \end{cases}$

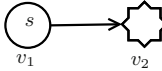
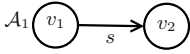
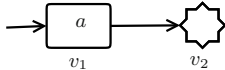

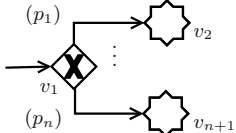
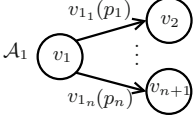
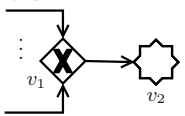
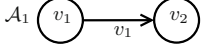
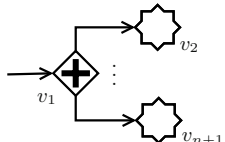
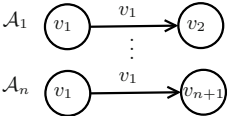
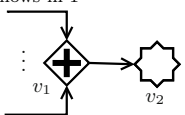
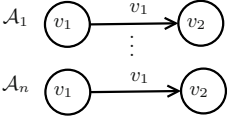
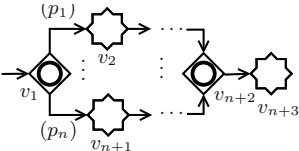
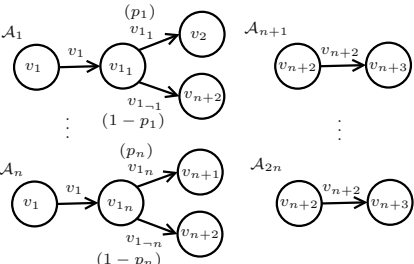
This operation eliminates q and its output transition from \mathcal{A} , transforming all the input transitions of q in input transitions of its only output state o .


Definition 11. *Automata Concatenation Operation* (\boxplus)

Let $\mathcal{A}_1 = (Q_1, E_1, T_1, L_1, \ell_1, \mathfrak{p}_1)$ and $\mathcal{A}_2 = (Q_2, E_2, T_2, L_2, \ell_2, \mathfrak{p}_2)$ be two SAN automata. Let $q_1 \in Q_1$ be an absorbing state and $q_2 \in Q_2$ be a source state.

The concatenation of \mathcal{A}_1 and \mathcal{A}_2 via the states q_1 and q_2 (represented by $\mathcal{A}_1 \boxplus_{q_1, q_2} \mathcal{A}_2$) is the SAN automaton $\mathcal{A}_C = (Q_C, E_C, T_C, L_C, \ell_C, \mathfrak{p}_C)$, where:

Table 2. Mapping of the BPMN objects into SAN

BPMN Object ^{1,2}	SAN Mapping
Start event labeled s 	
Atomic activity labeled a 	
Exclusive gateway diverging 1 sequence flow in n 	
Exclusive gateway converging n sequence flows in 1 	
Parallel gateway diverging 1 sequence flow in n 	
Parallel gateway converging n sequence flows in 1 	
Inclusive gateway block diverging 1 sequence flow in n and re-converging them in 1 again 	

¹ We are using the symbol  to express any valid BPMN vertex.² We included two textual annotations in the vertices: a label (inside the vertex), and an identifier v_i (at the bottom of the vertex).

- $Q_C = (Q_1 \cup Q_2) \setminus \{q_2\}$
- $E_C = E_1 \cup E_2$
- $T_C = T_1 \cup \{(q_1, p, e) \mid (q_2, p, e) \in T_2\} \cup (T_2 \setminus \{(q_2, p, e) \mid (q_2, p, e) \in T_2\})$
- $L_C = L_1 \cup L_2$
- $\forall q \in Q_C, \ell_C(q) = \begin{cases} \ell_1(q), & \text{if } q \in Q_1 \\ \ell_2(q), & \text{in the other cases} \end{cases}$
- $\forall t = (p, q, e) \in T_C, p_C(t) = \begin{cases} p_1(t), & \text{if } t \in T_1 \\ p_2((q_2, q, e)), & \text{if } (p = q_1) \wedge ((q_2, q, e) \in T_2) \\ p_2(t), & \text{in the other cases} \end{cases}$

5 The Conversion Algorithm

Algorithm 1 shows the main steps involved in the conversion of a well-defined BPMN Process model to a SAN model. It first creates a SAN model composed of all automata generated from the individual conversion of the vertices of the input BPMN graph. This individual conversion, made by function “ConvertVertexInAutomata”, is the implementation of the mappings described in Table 2. In the sequence, this SAN model is reduced using Procedure 1, which applies the operations defined in Section 4.2 to create the final SAN model.

Algorithm 1. ConvertBPtoSAN(BP)

Input: BP – a well-formed BPMN graph that is also a well-defined Process model

Output: S – a SAN model

- 1: $S \leftarrow \emptyset$
 - 2: $\mathcal{V} \leftarrow S_{BP} \cup A_{BP} \cup G_{BP} \cup F_{BP}$ {All vertices of BP }
 - 3: **for all** $v \in \mathcal{V}$ **do**
 - 4: $S \leftarrow S \cup \text{ConvertVertexInAutomata}(BP, \text{vertex}, S)$
 - 5: **end for**
 - 6: $\text{ReduceSANModel}(BP, S)$
 - 7: **return** S
-

The final number of automata in a SAN model generated by our conversion method from a well-defined BPMN model BP is given by:

$$|S_{BP}| + \sum_{g \in G'} (|\text{outputs}(g)| - 1)$$

where $G' = \{g \in G_{BP} \mid (|\text{outputs}(g)| > 1) \wedge (\ell_{BP}(g) \in \{\text{“}\times\text{”}, \text{“}\bigcirc\text{”}\})\}$.

We can see each automaton as an independent sequence flow of the business process. For that, we have at least as many automata as the number of start events in BP ($|S_{BP}|$). In addition, for each divergent parallel or inclusive gateway, a new set of automata is required. The size of this set is given by the number of branches (outputs) of the divergent gateway less 1 (because one of the branches is treated as the continuation of the automaton that originates the divergence).

Procedure 1. ReduceSANModel(BP, \mathcal{S})**Input:** BP – a well-formed BPMN graph**Input/Output:** \mathcal{S} – a SAN model

```

1: { Concatenate the “sequential” automata }
2: while there is an absorbing state  $q_1 \in \mathcal{A}_1$  and a source state  $q_2 \in \mathcal{A}_2$  (with
    $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{S}$ ) such that  $\ell_{\mathcal{A}_1}(q_1) = \ell_{\mathcal{A}_2}(q_2)$  do
3:    $\mathcal{A}_C \leftarrow \mathcal{A}_1 \overset{q_1}{\boxplus} \overset{q_2}{\mathcal{A}_2}$  { Concatenate the two automata }
4:   { Merge the equivalent states correspondent to the converging exclusive gate-
     ways }
5:   while  $\exists q_1, q_2 \in Q_{\mathcal{A}_C}$  such that  $\ell_{\mathcal{A}_C}(q_1) = \ell_{\mathcal{A}_C}(q_2)$  do
6:      $\mathcal{A}_C \leftarrow \mathcal{A}_C[q_1 \triangleright q_2]$ 
7:   end while
8:    $\mathcal{S} \leftarrow (\mathcal{S} \setminus \{\mathcal{A}_1, \mathcal{A}_2\}) \cup \mathcal{A}_C$ 
9: end while
10: for all  $\mathcal{A} \in \mathcal{S}$  do
11:   { Remove the states created to join alternative sequence flows }
12:   while  $\exists q \in Q_{\mathcal{A}}$  and  $\exists v \in G_{BP}$  such that:  $(\ell_{\mathcal{A}}(q) = v \wedge \ell_{BP}(v) = \text{“} \times \text{”} \wedge$ 
      $|\text{outputs}(v)| = 1)$  do
13:      $\mathcal{A} \leftarrow \mathcal{A}[q \blacktriangleright]$  { Suppress state  $q$  }
14:   end while
15:   { Merge the source state with the absorbing states }
16:   while  $\exists q_2 \in Q_{\mathcal{A}}$  such that  $q_2$  is an absorbing state do
17:      $q_1 \leftarrow$  the only source state of  $\mathcal{A}$ 
18:      $\mathcal{A} \leftarrow \mathcal{A}[q_1 \triangleright q_2]$  { Merge the states  $q_1$  and  $q_2$  }
19:   end while
20: end for

```

One remark must be made about our conversion algorithm: the order in which the pair of automata are selected to be concatenated (lines 2 and 3 of Procedure 1) impacts the final models. Different orders may generate structurally different SAN models that are equivalent in terms of modeled behavior.

The SAN model generated by our conversion algorithm reflects the behavior of one instance of the business process, disregarding the resource usage. To analyze the behavior of the system when several instances are being executed in parallel, we need to replicate the automata of the SAN model – each replica represents an instance of the process. SAN counts on the concept of replication and techniques to aggregate similar components in order to reduce the state space.

The conversion method was implemented as part of a software tool – BP2SAN¹. This tool receives as input a BPMN model (textually described) and generates behaviorally equivalent SAN models corresponding to the given business process. The resulted SAN models are textually expressed using the syntax accepted by the PEPS tool. The generated SAN models can be further enriched, for example, with information about the rates associated with the events, or with additional automata and functional rates expressing resource constraints. With the support of PEPS, performance indices can be extracted from the generated SAN models by defining numerical functions over the global state space of the system and integrating them with the stationary probability distribution of the model.

¹ BP2SAN is publicly available at <http://www.ime.usp.br/~kellyrb/bp2san/>.

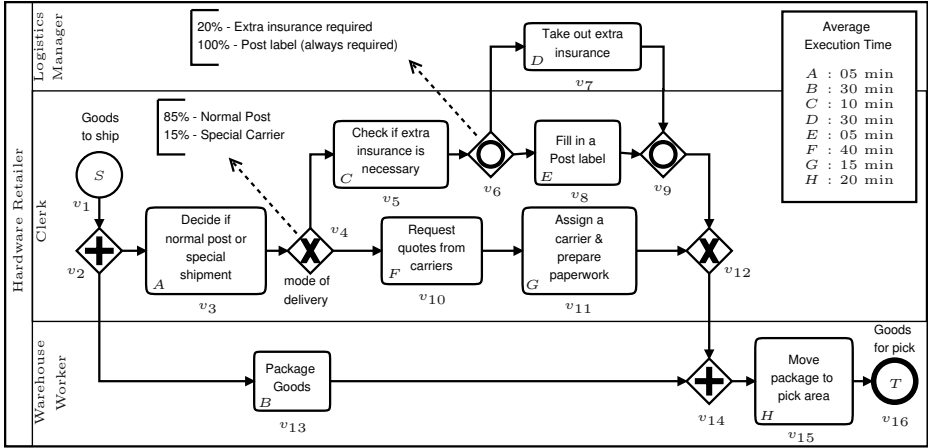


Fig. 1. BPMN model of the shipment process of a hardware retailer (extracted from [6])

6 Example – Shipment Process of a Hardware Retailer

Fig. 1 shows a model in BPMN extracted from the examples indicated in the specification document of BPMN 2.0 [6]. It represents the steps a hardware retailer has to fulfill before the order can be shipped to the customer. The model is divided in three lanes, each one labeled with the employee responsible for the execution of the activities in it: *Logistics Manager*, *Clerk* and *Warehouse Worker*.

The SAN model generated from the conversion of the vertices in Fig. 1 is shown in Fig. 2. After the conversion of the vertices, the SAN model is reduced according the steps defined in Procedure 1. Fig. 3 shows one of the possible SAN models that can be resulted and the sequences of operations that originated it.

We solved the model of Fig. 3 in PEPS, considering from one to three parallels process instances. For events representing activities, we used rates defined over the average execution times indicated in Fig. 1. For the other events, we used a constant high rate. Each employee can only perform an activity at a time. When an instance needs an employee to perform an activity and he/she is busy, the instance will wait for his/her availability. Functional rates were used to prevent the execution of an activity when the resource it depends on is not available. Table 3 shows the results obtained in the analysis. With three parallel instances, the service time duplicated if compared to the sequential case.

Table 3. The results obtained through the solution of the SAN model in Fig. 3

Parallel Instances	State Space	Reachable State Space	Service Time (h)	Utilization . Manager	Utilization Clerk	Utilization W. Worker
1	380	85	1.106	0.052	0.269	0.518
2	144,400	5,809	1.599	0.080	0.412	0.793
3	54,872,000	349,013	2.228	0.093	0.476	0.916

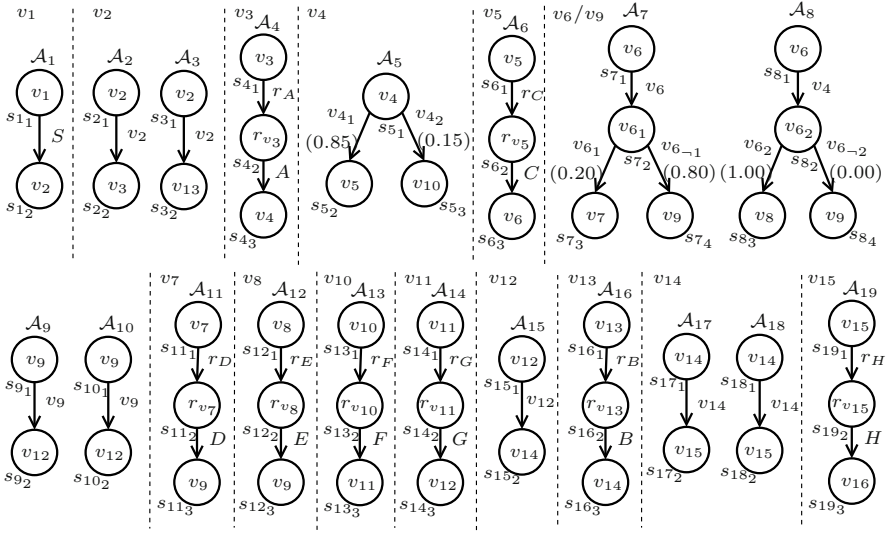


Fig. 2. SAN model obtained after the conversion of the BPMN vertices

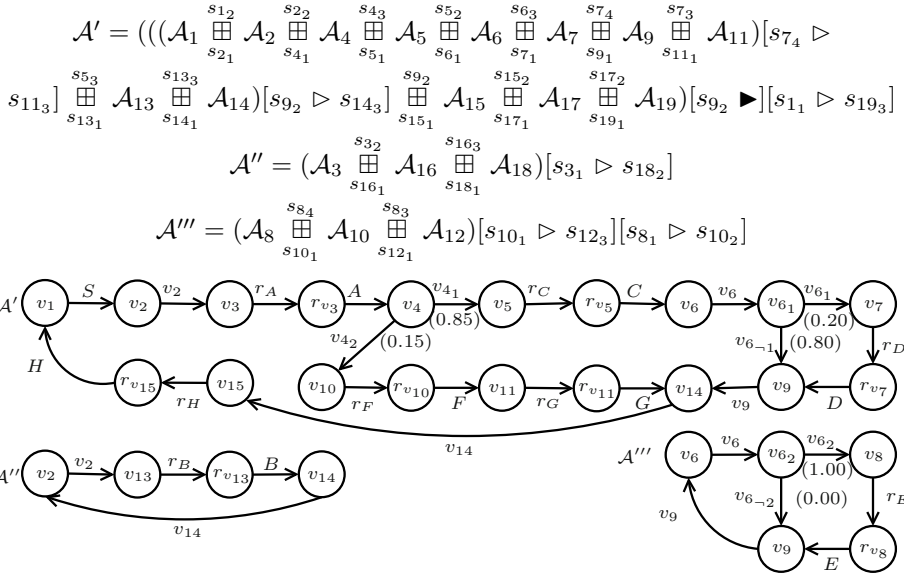


Fig. 3. SAN model that we can obtain from the reduction of the models in Fig. 2

7 Concluding Remarks

In this work, we presented an algorithm to automatically convert a subclass of the BPMN Process diagrams into SAN models. First, we formally specified

the characteristics of a well-defined BPMN Process model. After, we defined mappings of BPMN objects into elementary SAN models. Finally, operations over SAN models were defined to transform elementary models in the SAN model of a business process.

We exemplified the functioning of our algorithm with a typical example of business process. The corresponding SAN model generated by our tool **BP2SAN** was then solved using **PEPS**, for different numbers of parallel instances (what gave us large state spaces). From the extracted performance indices, we were able to evaluate how the service time of the process and the utilization of the human resources were impacted by the system workload.

Our ongoing work focuses in the modeling of business processes for performance analysis considering resource management information. Business process activities usually depend on different resources to be executed. The expected performance of a business process depends on how these resources are provisioned and used. We are currently working in: (i) the definition of annotations over BPMN models to specify the resources requirements of each activity and how the resources are shared between activities executed parallelly; and (ii) an automated method to extend the SAN model with this information.

References

1. Braghetto, K.R., Ferreira, J.a.E., Vincent, J.M.: Performance analysis modeling applied to business processes. In: 2010 Spring Simulation Multiconference, pp. 122:1–122:8. SCS/ACM, New York (2010)
2. Brenner, L., Fernandes, P., Plateau, B., Sbeity, I.: PEPS2007 - stochastic automata networks software tool. In: 4th International Conference on Quantitative Evaluation of Systems, pp. 163–164. IEEE Computer Society, Washington (2007)
3. Canevet, C., Gilmore, S., Hillston, J., Prowse, M., Stevens, P.: Performance modelling with the unified modelling language and stochastic process algebras. *IEE Proceedings Computers and Digital Techniques* 150(2), 107–120 (2003)
4. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. *Information and Software Technology* 50(12), 1281–1294 (2008)
5. Oliveira, C., Lima, R., Andre, T., Reijers, H.A.: Modeling and analyzing resource-constrained business processes. In: 2009 IEEE International Conference on Systems, Man and Cybernetics, pp. 2824–2830. IEEE Press, Los Alamitos (2009)
6. OMG: BPMN 2.0 by example, version 1.0 (non-normative) (2011)
7. OMG: Business process model and notation (BPMN), version 2.0 (2011)
8. Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. *SIGMETRICS Perform. Eval. Rev.* 13(2), 147–154 (1985)
9. Plateau, B., Atif, K.: Stochastic automata network for modeling parallel systems. *IEEE Transactions on Software Engineering* 17(10), 1093–1108 (1991)
10. Prandi, D., Quaglia, P., Zannone, N.: Formal analysis of BPMN via a translation into COWS. In: Wang, A.H., Tennenholtz, M. (eds.) *COORDINATION 2008*. LNCS, vol. 5052, pp. 249–263. Springer, Heidelberg (2008)
11. Wong, P.Y.H., Gibbons, J.: A process semantics for BPMN. In: Liu, S., Araki, K. (eds.) *ICFEM 2008*. LNCS, vol. 5256, pp. 355–374. Springer, Heidelberg (2008)

Monotonicity and Efficient Computation of Bounds with Time Parallel Simulation

Jean-Michel Fourneau and Franck Quessette

PRiSM, Université de Versailles-Saint-Quentin, CNRS UMR 8144, France

Abstract. We adapt Nicol's approach for the time parallel simulation with fix-up computations. We use the concept of monotonicity of a model related to the initial state of the simulation to derive bounds of the sample-paths. We prove several algorithms with fix-up computations which minimises the number of runs before we get a consistent sample-path. We obtain proved upper or lower bounds of the sample-path of the simulation and bounds of some estimates as well.

1 Introduction

A parallel discrete event simulation is the construction of the slices of the sample-path on a set of processors. These slices can be obtained by a state decomposition or a decomposition of the time interval. The most common approach is the space decomposition, i.e. a grouping of state variables into subsets which are affected to parallel processors. These processors exchange messages about the scheduling of the future events to avoid temporal faults or to correct them. Unfortunately the spatial decomposition approach has a limited parallelism and has in general an important overhead due to this synchronisation of future events.

Time Parallel Simulation (TPS in the following) follows a different approach considering a decomposition of the time axis and performing the simulations on time intervals in parallel (see [11] chap. 6 and references therein). Afterwards the simulation results are combined to build the overall sample-path. TPS has a potential to massive parallelism [18] as the number of logical processes is only limited by the number of times intervals which is a direct consequence of the time granularity and the simulation length. But the final and initial states of adjacent time intervals do not necessarily coincide at interval boundaries, possibly resulting in incorrect state changes. The efficiency of TPS depends on our ability to guess the state of the system at the beginning of the simulation intervals or to efficiently correct the guessed states to finally obtain a consistent sample-path after a small number of trials. Several properties had already been studied: regeneration [17], efficient forecasting of some points of the sample-path [12], parallel prefix computation [13], a guessed initial state followed up by some fix-up computations when the first state has a weak influence on the sample-path [18]. Another approach consists in relaxing these assumptions to obtain an approximation of the results [3,16,21].

We have previously introduced two properties of the model both related to monotonicity (inseq-monotonicity in [7] and hv-monotonicity in [6,8]), which can

be used to improve the efficiency of TPS. We have developed in these publications the basic theory needed to compare the sample-paths of simulations and to improve the speed of the TPS. In our approach, the simulation model is seen as a deterministic black box with an initial state and one input sequence which computes one output sequence. All the randomness of the model is in the input sequence. We define some orderings on the sequences and the states. A model is monotone when it preserves the ordering. If this property holds, we have proposed new approaches to improve the efficiency of TPS. We obtain an upper or a lower bound of the output sequence. Then we compute a reward on the output sequence. The ordering on the sequences are defined in such a way that typical rewards such as the moments or the tail of the distributions are consistently ordered when the sequences are ordered. In performance evaluation or reliability studies, we must prove in general that the systems we analyze satisfy some quality of service requirements. Thus it is sufficient to prove that the upper bound (or lower bound, depending on the measure of interest) of the output sequence satisfies the constraint. We have already proved in [10] that many queueing network models are hv-monotone.

Here we develop the ideas presented in [6,8] giving much more details, some new algorithms for speculative computations and some numerical results. We show how we can build a sample-path for a bound of the exact simulation using some simulated parts and ordering relations without some of the fix-up phases proposed in [18]. The inseq-monotonicity and hv-monotonicity concepts are both related to the stochastic monotonicity used to compare random variables and stochastic processes [9,20] and the event monotonicity which is the main assumption for the monotone Coupling From The Past algorithm for perfect simulation [19]. Hv-monotonicity is related to the non crossing property used in [1]. For these various notions of monotonicity of stochastic processes, one can also refer to [15] for a comparison.

The rest of the paper is as follows. In section 2, we give a detailed presentation of Nicol's approach of TPS with fix-up computation phases. Then in section 3, we define the comparison of sequences, the inseq-monotonicity and the hv-monotonicity and their relations with stochastic ordering. The approach is illustrated with examples to clarify the concepts. In Section 4 we first present the initial algorithm proposed in [8] and we extend it in several directions proving the convergence of some algorithms in any number of runs smaller than the number of processors. Section 5 is devoted to a simple model of a Web server. We prove that the system is hv-monotone and we provide some numerical results for the speed up and the efficiency of the approach.

2 Time Parallel Simulation with Fix-Up Phases

Let us now detail the classical fixed up approach [18] before we introduce the first modification presented in [8] and the new extensions which are the main results of this paper. For the sake of completeness we begin by a short introduction to the fixed up approach proposed by Nicol. In a first step, the interval $[0, T)$ is divided

into K equal intervals $[t_i, t_{i+1})$. K is the number of processors. Let $X(t)$ be the state at time t during the exact simulation obtained in a centralised manner. The aim is to build $X(t)$ for t in $[0, T)$ through an iterative distributed algorithm. For the sake of simplicity, we assume that for all i between 1 and K , logical process LP_i simulates the i -th time interval. We denote as $Y_j^i(t)$ the state of the system at time t obtained during the j -th iteration of logical process LP_i . The initial state of the simulation is known and is used to initialise LP_1 . During the first run, the other initial states (say $Y_1^i(t_i)$) for the initial state of simulation at LP_i are chosen at random or with some heuristics. Simulations are ran in parallel. The ending states of each simulation (i.e. $Y_1^i(t_{i+1})$) are computed at the end of the simulation of the time intervals. They are compared to the initial state we have previously used (i.e. $Y_1^{i+1}(t_{i+1})$). The equality of both states is a necessary condition for consistency of the path. Otherwise one must run a new set of parallel simulations for the inconsistent parts using $Y_1^i(t_{i+1})$ as starting point (i.e. $Y_2^{i+1}(t_{i+1}) \leftarrow Y_1^i(t_{i+1})$) of the next run on logical process LP_{i+1} . These new runs are performed with the same sequence of random inputs. The simulations are ran until all the parts are consistent. The number of rounds before the whole simulation is consistent is smaller than the number of LP . It is clear that at the end of the first run, the simulation performed at LP_1 is consistent. Similarly by induction on i , at the end of round i , LP_i is consistent. It is the worst case we may obtain, and in that case the time to perform the simulation in parallel is equivalent to the time in sequential.

Note that performing the simulation with the same input sequence may speed up the simulation due to coupling. Suppose that we have stored the previous sample-paths computed by LP_i . Suppose now that for some t , we find that the new point $Y_k^i(t)$ is equal to a formerly computed point $Y_m^i(t)$. As the input sequence is the same for both runs, both sample-paths have now merged:

$$Y_m^i(u) = Y_k^i(u) \quad \forall u \geq t.$$

Thus it is not necessary to build the new sample-path. Such a phenomenon is defined as the coupling of sample-paths. Note that it is not proved that the sample-paths couple and this is not necessary for the proof of the TPS that it happens. Clearly, coupling allows to speed up the computations performed by some LP and also reduces the number of rounds before the whole simulation becomes coherent. Indeed a coupling means that the state reached at the end of a time interval is not that dependent of the initial state of the simulation.

Consider an example of TPS with one fix-up step in Figure 1. During the first step, five parts of the simulation are ran in parallel. Part 1 and Part 2 are consistent as the initial point of LP2 is the final point of LP1 due to a correct guess. Other parts are not consistent and the simulations are ran again with a correct initialisation (A instead of B for the second run of part 4 for instance) and the same random inputs to obtain new sample-paths. The former sample-paths are compared to the new ones and as soon that the simulations couple, they are stopped as simulations after coupling will follow the same paths. If the simulations do not couple, we only keep the new sample-path. After each

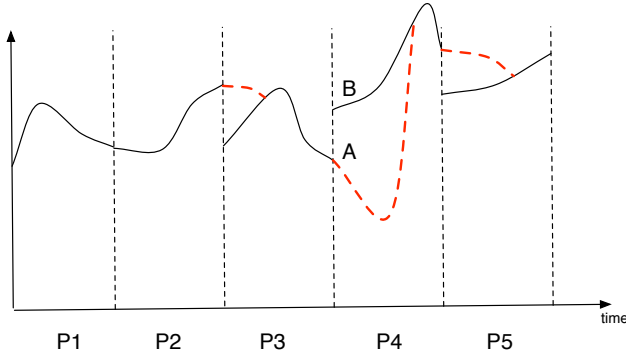


Fig. 1. TPS, coupling and fixing the sample-path

step, the number of consistent sample-paths will increase. The efficiency of the approach is related to the number of consistent parts at each step. In Fig. 1 the new runs are represented in dotted red lines and we see that all the simulations couple during the first fix-up step.

We assume that the logical processes LP_i ($i = 1..K$) perform simulations and exchange information with a Master process which must check the consistency of the partial sample-paths. Thus the simulation effort is distributed among $K + 1$ processes. The whole approach is summarised in the following algorithm for LP_i ($K \geq i > 1$) and the Master process. The first logical process in charge of the simulation of the first time interval slightly differs for this general pseudo-code: $Y_1^1(t_1)$ is equal to $X(t_1)$ the initial state of the whole simulation.

Algorithm LP_i

1. $k \leftarrow 0$. Read in shared memory input sequence $I(t)$ for all t in $[t_i, t_{i+1}[$.
2. $Y_1^i(t_i) \leftarrow Random$.
3. Loop
 - (a) $k++$.
 - (b) Perform run k of Simulation with Coupling on the time interval $[t_i, t_{i+1}[$ to build $Y_k^i(t)$ using input sequence $I(t)$.
 - (c) Send to the Master: the state $Y_k^i(t_{i+1})$.
 - (d) Receive from the Master: Consistent(i) and a new initial state U .
 - (e) If not Consistent(i) $Y_{k+1}^i(t_i) \leftarrow U$.
4. Until Consistent(i).
5. $\forall t \in [t_i, t_{i+1}[$, $X(t) \leftarrow Y_k^i(t)$ and write $X(t)$ in shared memory.

Algorithm Master

1. For all i Consistent(i) $\leftarrow False$.
2. Consistent(0) $\leftarrow True$; LastConsistent $\leftarrow 0$; $k \leftarrow 0$.
3. Loop
 - (a) $k++$.

- (b) For all i , if not Consistent(i) Receive from LP_i the state $Y_k^i(t_{i+1})$.
 - (c) $Y_1^0(t_1) \leftarrow Y_1^1(t_1)$.
 - (d) $i \leftarrow \text{LastConsistent}$.
 - (e) Loop
 - i. $i++$.
 - ii. If $(Y_k^i(t_i) = Y_k^{i-1}(t_i))$ and Consistent($i-1$) then Consistent(i) \leftarrow True.
 - (f) Until (not Consistent(i)) or ($i > K$).
 - (g) $\text{LastConsistent} \leftarrow i - 1$.
 - (h) For all i send to LP_i Consistent(i) and the state $Y_k^{i-1}(t_i)$.
4. Until $\text{LastConsistent} = K$.
-

3 Monotone Systems, Comparison of States and Comparison of Sequences

A simulation model is defined as an operator on a sequence of parameters (typically the initial state) and an input sequence (typically inter arrival times and service times) and produces an output sequence (typically the state of the system or a reward). Let \mathcal{M} be a simulation model. We denote by $\mathcal{M}(a, I)$ the output of model \mathcal{M} when the parameter sequence is a and the input sequence I . As usual an operator is monotone iff its application on two comparable sequences provides two output sequences which are also comparable. We have used the following point ordering (denotes as \preceq_p in [7], but various orders are possible. This order is interesting because it implies a comparison on the rewards computed on the output sequence.

Definition 1. Let I_1 and I_2 be two sequences with length n . $I(m)$ is the m -th element of sequence I . $I_1 \preceq_p I_2$ if and only if $I_1(t) \leq I_2(t)$ for all index $t \leq n$.

Property 1. Assume that the rewards are computed with function r applied on state $O(t)$ at time t . If the rewards are non negative, then:

$$O_1 \preceq_p O_2 \implies R(O_1) = \sum_t r(t, O_1(t)) \leq R(O_2) = \sum_t r(t, O_2(t)).$$

Many rewards such as moments and tails of distribution are non negative.

Based on this simple idea, we can define two properties which allows to compare the outputs of a simulation model when the change the input sequence (inseq-monotone) or the initial state of the simulation (hv-monotone). In the context of queueing networks for instance, this will describe how evolve the sample-path of the population when we change the arrivals or the initial population in the queue. We consider two arbitrary orderings \preceq_α and \preceq_β .

Definition 2 (inseq-monotone Model). Let \mathcal{M} be a simulation model, \mathcal{M} is input sequence monotone (or inseq-monotone in the following) with respect to orderings \preceq_α and \preceq_β if and only if for all parameter sequence a and input sequences I and J such that $I \preceq_\alpha J$, then $\mathcal{M}(a, I) \preceq_\beta \mathcal{M}(a, J)$.

Definition 3 (hv-monotone Model). Let \mathcal{M} be a simulation model, \mathcal{M} is monotone according to the input state or hidden variable (hv-monotone in the following) with respect to orderings \preceq_α and \preceq_β if and only if for all parameter sets a and b such that $a \preceq_\alpha b$, then $\mathcal{M}(a, I) \preceq_\beta \mathcal{M}(b, I)$ for all input sequence I .

The inseq-monotonicity and the hv-monotonicity are related to the stochastic monotonicity used to compare Markov chains [20].

Definition 4 (Stochastic comparison). Let X and Y be random variables taking values on a totally ordered finite space $\{1, 2, \dots, n\}$ with p and q as probability distribution vectors, then X is said to be less than Y in the strong stochastic sense (denoted as $X \preceq_{st} Y$), if and only if $\sum_{j=k}^n p_j \leq \sum_{j=k}^n q_j$ for $k = 1, 2, \dots, n$. The stochastic ordering is also used for distribution and we note that $p \preceq_{st} q$.

Example 1. One can easily check that: $(0.1, 0.3, 0.2, 0.4) \preceq_{st} (0., 0.4, 0., 0.6)$.

The relation between inseq-monotonicity and the strong stochastic ordering is described in [7]. We now prove that stochastic monotonicity of Discrete Time Markov Chains (DTMC in the following) implies hv-monotonicity of the simulation model when the state space is fully ordered. Let us first define stochastic monotone Markov chains using the matrix formulation presented in [9]. It is known for a long time [20] that monotonicity and comparability of the one step transition probability matrices of time-homogeneous MCs yield sufficient conditions for their stochastic comparison.

Definition 5 (Monotone DTMC). Let M_Z be the stochastic matrix associated to Markov chain Z_t on a totally ordered state space \mathcal{S} , M_Z is stochastically monotone if and only if for all probability distributions u and v such that $u \preceq_{st} v$, then $uM_Z \preceq_{st} vM_Z$.

Property 2. Let M_Z be a stochastic matrix, let $M(i, *)$ be the i -th row of M and may be seen as a probability distribution, M_Z is stochastically monotone iff $M_Z(i, *) \preceq_{st} M_Z(i + 1, *)$.

Example 2. $M1 = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.2 & 0.6 \\ 0.0 & 0.1 & 0.3 & 0.6 \\ 0.0 & 0.0 & 0.1 & 0.9 \end{bmatrix}$ is monotone while $M2 = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.1 \\ 0.2 & 0.1 & 0.1 & 0.6 \\ 0.0 & 0.1 & 0.3 & 0.6 \\ 0.2 & 0.0 & 0.0 & 0.8 \end{bmatrix}$ is not.

We may use an event representation of the simulation model. Events are associated with transitions. Let ev be an event in this model and let x be a state, we denote by $ev(x)$ the state obtained by application of event ev on state x . It is more convenient that some events do not have any effect (for instance the end of service event on an empty queue will be a loop). Monotonicity property has already be defined for events and their links with stochastic monotonicity are now understood (see [15]).

Definition 6 (event -monotone). *An event ev is monotone if its probability does not depend on the state and for all states x and y , $x \preceq y$ implies that $ev(x) \preceq ev(y)$.*

It is now possible to show the links between strong stochastic ordering of DTMCs and hv-monotonicity of their simulations.

Property 3. *Let M_Z be a monotone stochastic matrix on a totally ordered state space (denoted by \leq to emphasis the total ordering) and let $\mathcal{M}()$ be the simulation of this chain. a is the initial state and I is a sequence of uniform random values in $[0, 1]$. Assume that the output $O = \mathcal{M}(a, I)$ is the state of the chain at each time. Thus, $\mathcal{M}()$ is hv-monotone w.r.t. the total ordering on the states (i.e. \leq) and the point ordering \preceq_p on the output sequence.*

Proof: It is proved in [15] that a stochastically monotone DTMC on a totally ordered state space admits a set of event (e_1, \dots, e_m) which is monotone. Suppose that $a \geq b$ and consider $O1 = \mathcal{M}(a, I)$ and $O2 = \mathcal{M}(b, I)$. As we use the same $I(t)$, both simulations perform the same event at time t . Let e_k be this event. By construction $O1(t+1) = e_k(O1(t))$ and $O2(t+1) = e_k(O2(t))$. The events are all monotone. Thus is if $O1(t) \leq O2(t)$, we get $O1(t+1) \leq O2(t+1)$. By induction the model is monotone for the point ordering on the output sequence. \square

Note that hv-monotonicity or inseq-monotonicity do not imply strong stochastic monotonicity in general. Let us now consider one basic example to illustrate the approach.

Example 3 (DTMC). *Consider again matrix $M1$ defined in Example 2. Assume that the input sequence consists in uniform random values in $[0, 1]$ used in an inverse transform method to find the transitions according to matrix $M1$. Assume that this sequence I is equal to $(0.5, 0.05, 0.15, 0.06, 0.25, 0.45)$. Assume that the output of the model is the state of the Markov chain at each step and that the hidden variable is the initial state of the chain. As $M1$ is stochastically monotone, the simulation $\mathcal{M}_1()$ of matrix $M1$ is hv-monotone w.r.t. natural ordering on the states and point ordering on the output sequence. Therefore:*

$$\mathcal{M}_1(1, I) \preceq_p \mathcal{M}_1(2, I).$$

Indeed, one can easily check that the output of $\mathcal{M}_1(1, I)$ is the following sequence of states $(1, 3, 2, 2, 1, 2, 4)$ while the output of $\mathcal{M}_1(2, I)$ is $(2, 4, 3, 3, 2, 3, 4)$; the verification of the point ordering is readily made. Note also that both simulations have coupled at time 6 on state 4.

Remark 1. *Note that increasing the initial state does not in general results in an upper bounding sample-path as shown in the following example.*

Example 4 (DTMC 2). *Consider now matrix $M2$ defined in Example 2. We make the same assumptions on the input and output sequences as in the previous example. Let $I = (0.15, 0.5, 0.15)$. The sample-path beginning with state 1 is $(1, 2, 4, 1)$ while beginning with state 2 it is $(2, 1, 3, 3)$. Clearly the two vectors cannot be compared with the \preceq_p ordering.*

4 Fast Parallel Computation of Bounds with TPS

We assume in this section that we consider an arbitrary hv-monotone model and we develop the approach introduced in [6,8]. We show how to modify the TPS to converge faster on a bound of the sample-path. We have already proved in [10] that many queueing networks model are hv-monotone. To speed up the computation we change the rules about the consistency of the simulations performed by the logical processes. We now say that two parts are consistent if they are a proved bound of the sample-path that we do not have computed. To illustrate the approach let us suppose that we want to compute faster a lower bound of the sample-path. We modify the TPS as follows: the simulation processes LP_i ($i = 1..N$) are not modified but the Master process performs a slightly different computation. The first assumption is, as before, that all parts except the first one are marked as not consistent while the first one is consistent. Suppose that at the end of round k , the points $Y_k^i(t_{i+1})$ have been computed using $Y_k^i(t_i)$ as starting points. If $Y_k^i(t_{i+1}) = Y_k^{i+1}(t_{i+1})$ and part i is consistent, mark part $i + 1$ as consistent. Furthermore if $Y_k^i(t_{i+1}) > Y_k^{i+1}(t_{i+1})$ and part i is consistent, the concatenation of the parts provide a sample-path of a lower bounding sequence. Thus mark part $i + 1$ as consistent. Two consecutive parts are consistent if the first one is consistent and if the second one is a proved lower bound. Let us more precisely describe the new version of the Master Process, we only report the inner loop.

— Lower Bounding sample-path —

3.e Loop

i) $i++$;

ii) if $(Y_k^i(t_i) = Y_k^{i-1}(t_i))$ and Consistent(i-1) then Consistent(i) \leftarrow True;

iii) elsif $(Y_k^i(t_i) < Y_k^{i-1}(t_i))$ and Consistent(i-1) then Consistent(i) \leftarrow True;

3.f Until (not Consistent(i)) or ($i > K$);

In the original method [18], instruction 3.e.iii) of our approach is not included. This version of the method is proved to provide a lower bound of the sample-path (see [8] for the proof of this first algorithm).

Theorem 1. *Assume that the simulation model is hv-monotone, the new version of the Master for the TPS algorithm makes the logical simulation processes build a point-wise lower bound of the sample-path faster than the original approach. Thus the number of runs is smaller than K .*

For instance, Fig. 1 shows that after step 2, we have coupled and we have obtain an exact solution. But at the end of step 1 we have obtained a proved lower bound if the system is hv-monotone. Thus we can immediately stop the computation and consider the sample-path after the first run. This sample-path is not an exact one but it is a point-wise lower bound of the exact sample-path. Due to

Property 1 the expectations of non negative rewards on this sample-path are lower bounds of the exact expected rewards. Note that the bound is proved while the approximations provided by other methods [3,21] do not provide such an information (see also Remark 1 to show that changing the initial state does not always result in a bound of the sample-path).

Designing an algorithm for computing an upper bound is straightforward. But instead we show how to improve the method and obtain a trade-off between the number of runs and the accuracy. We can again improve the speed of convergence with a clever choice of the input points sent by the Master to the simulation processes LP_i .

In the original approach when process LP_{i-1} is not consistent, then LP_i is not consistent either, even if the condition $(Y_k^i(t_i) = Y_k^{i-1}(t_i))$ holds. However it is useless to compute again a new part of the simple path using $(Y_k^i(t_i))$ as a starting point. Indeed the former run has already computed this particular part and beginning with the same state and using the same sequence of inputs, we will again obtain exactly the same results. Thus the processor is idle and it can be used instead for some speculative computations: it may typically compute a new part for the same sequence of inputs using a speculative initial point. Thus one must assume that the Master process stores several parts of the sample-path using the same sequence of inputs but initialised with several starting points. It checks the consistency of the parts and it asks to create new speculative parts when needed.

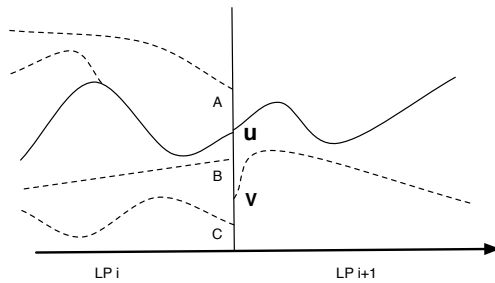


Fig. 2. How to chose a new initial point for a speculative computation

It remains to describe how to chose a new starting point for the next run. The configuration is depicted in Fig. 2. Assume that we want to compute a lower bound and that Min is the smallest state. Consider the results of run k . Assume that $(Y_k^{i+1}(t_{i+1}) = Y_k^i(t_{i+1})) = \mathbf{u}$. The next initial value is chosen at random between Min and \mathbf{u} (\mathbf{u} excluded). Let \mathbf{v} be this random value. We will run a new simulation with the same input sequence $I(t)$ starting form state \mathbf{v} . At the end of the run we have at least two speculative parts (one beginning with \mathbf{u} and one beginning with \mathbf{v} , and maybe some other ones previously computed). Assume that the state space is totally ordered. Now consider the four cases for

the simulation performed during that run on LP_i . Assume that at the end of that run, part i is found consistent. This simulation may end in:

- $A > \mathbf{u}$. In that case, part $i + 1$ beginning with \mathbf{u} is consistent.
- \mathbf{u} because of coupling. Again part $i + 1$ beginning with \mathbf{u} is consistent.
- B such that $\mathbf{u} > B \geq \mathbf{v}$. In that case part $i + 1$ beginning with \mathbf{v} is consistent.
- $C < \mathbf{v}$. None of the two speculative simulations of part $i + 1$ is consistent. One needs another run.

Clearly choosing a new speculative part beginning with a state smaller than \mathbf{u} will help to find a lower bound of the sample-path while a simulation initialised with a state larger than \mathbf{u} will not.

When the state space is only partially ordered, we have one more case where the states are not comparable and one more run is needed. Thus we use a similar set of rules to decrease the number of runs before the whole system has converged to a bound of the sample-path. We present in the next algorithm how such a speculative computation can be organised by the Master process. Again we only present the inner loop computation. The remaining part of the Master algorithm is kept unchanged.

– Speculative Runs when idle –

3.e Loop

- i) $i++$; Let k be the last run index
- ii) if Consistent($i-1$) then
 - Let l be the run index for which part $i - 1$ is consistent.
 - If $\exists m$ such that $Y_m^i(t_i) = Y_l^{i-1}(t_i)$ then Consistent(i) \leftarrow True and the m -th run is used for consistency of part i .
 - Elseif $\exists m$ such that $Y_m^i(t_i) < Y_l^{i-1}(t_i)$ then Consistent(i) \leftarrow True and the m -th run is used for consistency of part i .
 - Else $Y_{k+1}^i(t_i) \leftarrow Y_l^{i-1}(t_i)$.
- iii) Else
 - If $\exists m$ such that $Y_m^i(t_i) = Y_k^{i-1}(t_i)$ then $Y_{k+1}^i(t_i) \leftarrow$ Random state between Min and $Y_k^{i-1}(t_i)$.
 - Else $Y_{k+1}^i(t_i) \leftarrow Y_k^{i-1}(t_i)$.

3.f Until ($i > K$);

3.g LastConsistent $\leftarrow i - 1$.

3.h For all i send to LP_i Consistent(i) and the state $Y_{k+1}^i(t_i)$.

Finally we also prove an algorithm to compute a lower bound path with a convergence in any fixed number of round (say R) under the same assumptions on state Min . During the first $R - 1$ runs, the Master Process acts as before but if the simulation has not converged to a consistent sample-path (i.e. a bound of the exact one) at the end of run $R - 1$, the Master sends to the simulation processes LP_i state Min as a starting point for the next run.

 Convergence in R runs

3.e Loop

i) $i++$; Let k be the last run indexii) if Consistent($i-1$) then– Let l be the run index for which part $i-1$ is consistent.– If $\exists m$ such that $Y_m^i(t_i) = Y_l^{i-1}(t_i)$ then Consistent(i) \leftarrow True and the m -th run is used for consistency of part i .– Elseif $\exists m$ such that $Y_m^i(t_i) < Y_l^{i-1}(t_i)$ then Consistent(i) \leftarrow True and the m -th run is used for consistency of part i .– Else $Y_{k+1}^i(t_i) \leftarrow Y_l^{i-1}(t_i)$.

iii) Else

– if $k = R - 1$ then $Y_{k+1}^i(t_i) \leftarrow Min$.– Elseif $\exists m$ such that $Y_m^i(t_i) = Y_k^{i-1}(t_i)$ then $Y_{k+1}^i(t_i) \leftarrow$ Random state between Min and $Y_k^{i-1}(t_i)$.– Else $Y_{k+1}^i(t_i) \leftarrow Y_k^{i-1}(t_i)$.3.f Until ($i > K$);3.g LastConsistent $\leftarrow i - 1$.3.h For all i send to LP_i Consistent(i) and the state $Y_{k+1}^i(t_i)$.

Property 4. *The Master algorithm with inner loop "Convergence in R runs" needs less than R runs to build a consistent lower bound of the sample-path.*

Proof: due to computation at the end of run $R - 1$, the logical processes are initialised with the minimal state. Thus the consistency condition holds after one more run and all the parts are consistent at the end of run R . \square

We now describe briefly more improvements that we cannot detail for the sake of conciseness.

1. When a part is consistent, the process is idle and it can be used after reading in shared memory a new input sequence to build other speculative paths for a remaining part of the simulation which is not consistent at that time.
2. Remember that the initial approach provides an exact solution. Thus we can obtain a time versus accuracy trade-off by using the traditional approach during the first $R1$ runs and our approach with bounds during the next $R2$ runs and completing the simulation in one last run where the initial states for non consistent parts are initialised with minimal or maximal states. We obtain a proved convergence in $R1 + R2 + 1$ steps and the parts computed and found consistent at the end of run $R1$ are exact.

5 Modelling a Web Server

Some numerical results have already been published for monotone queueing networks such as Jackson networks [10]. To illustrate the approach with a new example, we analyse now a simple model of a web server. The system consists in a scheduler and a set of stations (see Figure 3). All these components are modelled by FIFO queues with infinite capacity. Web servers have receive

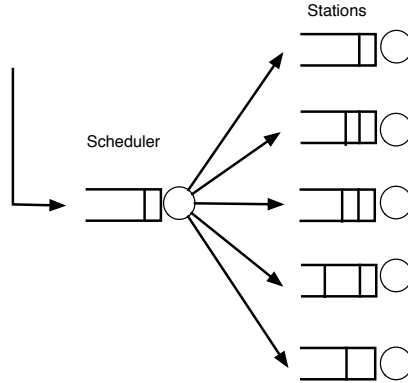


Fig. 3. Web Server as a queueing network

considerable attention (see for instance [14]) to optimise the impact of the scheduler on the performance of the system taking into account some partial information on the load of the queues and the work requested by the customers.

We first describe the model, we prove that the system is hv-monotone and we report some numerical results for the speed-up. Here we consider a discrete time model where the slot time is equal to the duration of the service time at the scheduler. The page arrivals follow a Poisson distribution with rate λ . Pages contain an HTTP GET for a file. Thus the important features of a page is the sum of the sizes of the objects included in the page. The sizes are estimated by the scheduler. The service times in the scheduler are constant and all equal to 1. Let T_0, T_1, \dots, T_N be $N + 1$ increasing number where T_N (resp. T_0) is the biggest (resp. smallest) size estimation. We assume that $T_0 > 0$. The pages are sent to the stations according to the estimation of their size. Station S_1 serves pages whose size estimation is in the interval $[T_0, T_1)$. Similarly, station S_i receives the pages with size in the interval $[T_{i-1}, T_i)$. The sizes of the page are independent and identically distributed. They follow a simple distribution with a large variance: $[\frac{1}{u}]$ with $0 < u \leq 1$. The services are supposed to be geometric with rate μ_i at queue S_i . Due to these assumptions, the state of the system is the number of customers at each queue (i.e. a $N + 1$ tuple of integers). Let u be a state; $u(0)$ will denote the size of the scheduler queue while $u(i)$, $i = 1..N$ will be the size of the queue associated with server S_i . The output of the model is the state of the system at time t . We use the following ordering on the initial states.

Definition 7. We define the \preceq_β ordering on states as the product of the natural orderings on each component of the tuple: $u \preceq_\beta v$ iff $u(i) \leq v(i)$ for all i .

Property 5. This model of a web server is hv-monotone with \preceq_β ordering on the parameter sequence and \preceq_p on the output sequence.

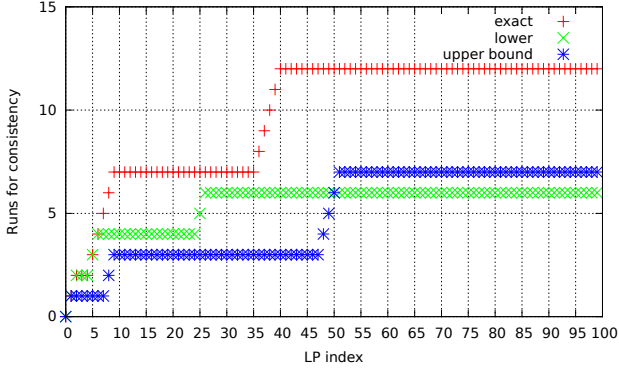


Fig. 4. Number of runs before global consistency, $\mu = 0.56$, $\lambda = 0.8$, uniform initial state

Proof: Let u_0 and v_0 be two states such that $u_0 \preceq_{\beta} v_0$ and let us denote by u_t and v_t the states at time t . Let us first consider $u_0(0)$ and $v_0(0)$. As $u_0(0) \leq v_0(0)$ and as the service of the scheduler is constant and the arrivals are the same because we use the same input sequence for both simulations, we have $u_t(0) \leq v_t(0)$ for all t . And in the simulation beginning with u_0 , all the customers leave the scheduler at the same time they leave in the simulation beginning with v_0 . Furthermore the routing is the same in both simulations. Note however that we must use the same amount of random values in the input sequence in both simulations. Thus we assume that the input sequence is slotted and all values produced at time t are only available for transitions at time t . If they are not used, they will be deleted and a new set of random values will be provided for time slot $t + 1$.

Now consider $u_0(i)$ and $v_0(i)$ for an arbitrary i . Due to the previous remarks, the arrivals of customers in S_i in simulation beginning with u_0 also happen at the same time and at the same queue in simulation beginning with v_0 . The service rates are the same in both simulations. Therefore $u_0(i) \leq v_0(i)$ implies that $u_t(i) \leq v_t(i)$. Finally we get: $u_t \preceq_{\beta} v_t$ for all t . \square

We present in the next figures some numerical results obtained for some parameters for λ and μ_i . The number of simulation processes K is equal to 100. We give for some typical simulations, the time necessary for simulation processes LP_i to be consistent for all i . We report the results for the usual algorithm and for the bounding algorithms (upper and lower bounds) without the improvements on the computation of speculative parts by idle processors. We also used two distributions for the Random states used in the initialisation part. Clearly both bounding algorithms are more efficient than the usual approach. They provide a bound of the paths and the rewards while exact methods are slower and simple approximations [3] do not give a guarantee on the performance. We only need 7 runs for obtaining 100 consistent parts of the sample path.

6 Conclusion

Many queueing systems are known to be monotone and such a property has not been used for simulation except within some Coupling From The Past algorithms. We think that monotonicity of the model has many applications in simulation which remain to be studied, especially for TPS or the space decomposition approaches. We will apply this technique for stochastic model checking by simulation and for the analysis of queueing elements with sophisticated service or access discipline. Indeed in stochastic model checking, we often have to compute bounds of probability for long paths [22,5] and TPS looks like a very efficient solution. Similarly, queues with complex discipline designed for service differentiation are difficult to analyse exactly. Improved TPS may be an alternative to stochastic bound and numerical analysis or fluid methods, see for instance [4] for a diffusion model of the Pushout mechanism and [2] for the analysis of the delays in WFQ queues.

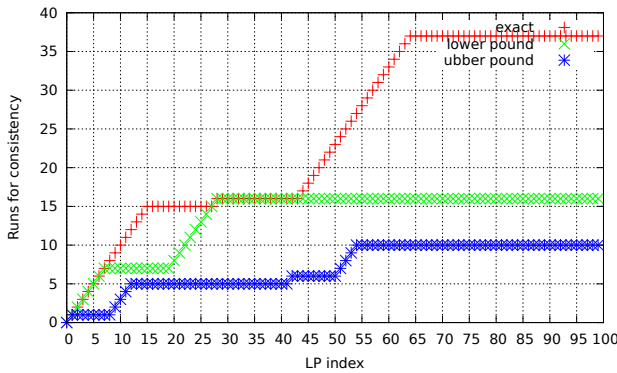


Fig. 5. Number of runs before each process is consistent, $\mu = 0.55$, $\lambda = 0.8$, second distribution for initial state

References

1. Andradottir, S., Hosseini-Nasab, M.: Parallel simulation of transfer lines by time segmentation. *European Journal of Operational Research* 159(2), 449–469 (2004)
2. Ben Mamoun, M., Pekergin, N.: Stochastic delay bounds of fair queueing policies by analyzing weighted round robin-related policies. *Performance Evaluation* 47(2), 181–196 (2002)
3. Bölöni, L., Turgut, D., Wang, G., Marinescu, D.C.: Challenges and benefits of time-parallel simulation of wireless ad hoc networks. In: *Valuetools 2006: 1st International Conference on Performance Evaluation Methodologies and Tools*, p. 31. ACM, New York (2006)
4. Czachórski, T., Fourneau, J.-M., Pekergin, F.: Diffusion model of the push-out buffer management policy. In: *INFOCOM*, pp. 252–261 (1992)
5. El Rabih, D., Pekergin, N.: Statistical model checking using perfect simulation. In: Liu, Z., Ravn, A.P. (eds.) *ATVA 2009*. LNCS, vol. 5799, pp. 120–134. Springer, Heidelberg (2009)

6. Fourneau, J.-M., Kadi, I.: Time parallel simulation of monotone systems. In: Poster Session, IFIP Performance Conference, Namur, Belgium (2010)
7. Fourneau, J.-M., Kadi, I., Pekergin, N.: Improving time parallel simulation for monotone systems. In: Turner, S.J., Roberts, D., Cai, W., El-Saddik, A. (eds.) DS-RT, pp. 231–234. IEEE Computer Society, Los Alamitos (2009)
8. Fourneau, J.-M., Kadi, I., Quesette, F.: Time parallel simulation and hv-monotonicity. In: Proceedings of the 26th International Symposium on Computer and Information Sciences. LNEE. Springer, Heidelberg (2011)
9. Fourneau, J.-M., Pekergin, N.: An algorithmic approach to stochastic bounds. In: Calzarossa, M.C., Tucci, S. (eds.) Performance 2002. LNCS, vol. 2459, pp. 64–88. Springer, Heidelberg (2002)
10. Fourneau, J.-M., Quesette, F.: Monotone queuing networks and time parallel simulation. In: Al-Begain, K., Balsamo, S., Fiems, D., Marin, A. (eds.) ASMTA 2011. LNCS, vol. 6751, pp. 204–218. Springer, Heidelberg (2011)
11. Fujimoto, R.M.: Parallel and Distributed Simulation Systems. Wiley Series on Parallel and Distributed Computing (2000)
12. Fujimoto, R.M., Cooper, C.A., Nikolaidis, I.: Parallel simulation of statistical multiplexers. *J. of Discrete Event Dynamic Systems* 5, 115–140 (1994)
13. Greenberg, A.G., Lubachevsky, B.D., Mitrani, I.: Algorithms for unboundedly parallel simulations. *ACM Trans. Comput. Syst.* 9(3), 201–221 (1991)
14. Harchol-Balter, M., Schroeder, B., Bansal, N., Agrawal, M.: Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.* 21, 207–233 (2003)
15. Kadi, I., Pekergin, N., Vincent, J.-M.: Different monotonicity definitions in stochastic modelling. In: Al-Begain, K., Fiems, D., Horváth, G. (eds.) ASMTA 2009. LNCS, vol. 5513, pp. 144–158. Springer, Heidelberg (2009)
16. Kiesling, T.: Using approximation with time-parallel simulation. *Simulation* 81, 255–266 (2005)
17. Lin, Y., Lazowska, E.: A time-division algorithm for parallel simulation. *ACM Transactions on Modeling and Computer Simulation* 1(1), 73–83 (1991)
18. Nicol, D., Greenberg, A., Lubachevsky, B.: Massively parallel algorithms for trace-driven cache simulations. *IEEE Trans. Parallel Distrib. Syst.* 5(8), 849–859 (1994)
19. Propp, J., Wilson, D.: Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures and Algorithms* 9(1&2), 223–252 (1996)
20. Stoyan, D.: Comparison Methods for Queues and Other Stochastic Models. John Wiley and Sons, Berlin (1983)
21. Turgut, D., Wang, G., Boloni, L., Marinescu, D.C.: Speedup-precision tradeoffs in time-parallel simulation of wireless ad hoc networks. In: DS-RT 2006: Proceedings of the 10th IEEE International Symposium on Distributed Simulation and Real-Time Applications, pp. 265–268. IEEE Computer Society Press, Los Alamitos (2006)
22. Younes, H.L., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation* 204(9), 1368–1409 (2006)

Stochastic Restricted Broadcast Process Theory

Fatemeh Ghassemi^{1,2}, Mahmoud Talebi¹, Ali Movaghar^{1,2}, and Wan Fokkink³

¹ Sharif University of Technology, Tehran, Iran

² Institute for studies in Theoretical Physics and Mathematics, Tehran, Iran

³ VU University Amsterdam, The Netherlands

Abstract. We provide a framework for modeling and analyzing both qualitative and quantitative aspects of mobile ad hoc network (MANET) protocols above the data-link layer. We extend Restricted Broadcast Process Theory [11,9]: delay functions are assigned to actions, while the semantics captures the interplay of a MANET protocol with stochastic behavior of the data-link and physical layer, and the dynamic topology. A continuous-time Markov chain is derived from our semantic model by resolving non-determinism, using the notion of weak Markovian network bisimilarity. The framework is applied to a leader election algorithm.

Keywords: Ad hoc protocol, Cross layer performance evaluation, Stochastic process algebra, Markovian model.

1 Introduction

In mobile ad hoc networks (MANETs), nodes communicate with each other using wireless transceivers which are unreliable. Nodes move arbitrarily and the topology of the network changes *dynamically*. MANET protocols should be able to tolerate faults that may arise due to unreliable wireless communication and changes in the underlying topology, while quality of service metrics in benchmarks should be satisfied. Therefore a unified framework for the verification and evaluation of MANET protocols can alleviate the complexity in the design process of such protocols.

We introduced *Restricted Broadcast Process Theory (RBPT)* in [11,9] to specify and verify MANET protocols, taking into account mobility. Here we extend this framework to *Stochastic RBPT (SRBPT)*, to evaluate properties of MANET protocols above the data-link layer. Performance evaluation of MANET protocols depends on physical characteristics of the nodes, the underlying dynamic topology of the network, the protocol behavior itself, and its collaboration with data-link layer protocols. The physical characteristics of nodes and their underlying topology define whether two nodes can communicate, while data-link layer protocols define how fast nodes can communicate.

To study the cross-layer performance of protocols above the data-link layer, an abstract model of the MAC protocol is used. The MAC protocol at the data-link layer manages transmissions of a node to reduce their collisions with other possible ones occurring in the vicinity. This abstract model may specify the aggregated behavior of the MAC protocols of an arbitrary number of nodes in terms of some delay functions like [19] or the behavior of a single MAC protocol from the point view of upper-layer protocols as a queue with a limited capacity of K and service time equal to MAC response

time T_{Mac} like [24]. To provide a compositional framework for evaluating MANETs, we choose the latter approach and then unfold the behavior of a MANET protocol deployed at a node, at the semantic level in terms of its interaction with its underlying MAC protocol [12]. To enhance the non-compositional framework of [12] with modular specification and analysis of MANETs, we provide a process algebra with more expressive power in defining the timing behavior of protocols. This process algebra follows the lines of *RBPT* [10] in syntax and the model of [12] in semantics.

The effect of physical and the dynamic underlying topology can be captured through the probability that a node receives a message successfully (P_{rcv}), and the probability that a communication link exists between two nodes (P_{UP}) with an identical mobility behavior. To remedy the effect of mobility on MAC layer performance factors like response time, we assume networks of nodes with an identical MAC layer and mobility model. Therefore, *SRBPT* semantic model is parameterized by mentioned four parameters: K , T_{Mac} , P_{rcv} and P_{UP} . We capture the interplay of all these parameters in the operational semantics. Due to the existence of internal immediate actions (with zero delay), probabilistic and non-deterministic choice coexist in the semantics. Non-deterministic choices of a model can be resolved by means of our weak Markovian network bisimilarity, a congruence relation on labeled multi-transition systems, while the performance measures of the model are preserved. We explain when and how a Markov chain can be derived from the specification of a MANET.

With this unified framework, one can first verify the correctness of a MANET specified in *RBPT*. Then one can specify appropriate action delays and semantics parameters to evaluate the quality of service metrics of a MANET protocol deployed on a specific data-link layer in a dynamic network. We illustrate the applicability of our framework by the analysis of a leader election protocol for MANETs [27].

Related works. PEPA, a stochastic process algebra, has been exploited to investigate the performance of backoff algorithms in ad hoc protocols [25] and the performance of WiFi protocol in configurations that the IEEE 802.11i does not guarantee the fairness for channel access [20]. PRISM have been used in the verification of MANET protocols like the Bluetooth device discovery [5], the MAC protocol of IEEE 802.11 [22], the IEEE 1394 root contention protocol [4], and the ZeroConf dynamic configuration protocol for IPv4 link-local addresses [21]. These protocols (except ZeroConf) mainly belong to the data-link layer, so the effect of the data-link layer and the dynamic topology are not considered. The wireless communication modeled in these works is either point-to-point or unreliable, while collisions that occur during a transmission are modeled and the underlying topology is considered fixed. However, modeling collisions (which are handled by MAC protocols) or point-to-point communication is not appealing for performance evaluation of MANET protocols (above the data-link layer). On the other hand, *non-blocking* and *topology-dependent* behaviors intrinsic in wireless communication, called local broadcast, cannot be modeled by *CSP*-like parallel composition in PEPA and PRISM in a modular way [7]. The only framework tailored to performance evaluation of wireless protocols (which can be used for MANETs) is [8]. That framework is more suitable for protocols beneath the data-link layer, since it only considers the physical characteristics of nodes and their underlying topology (which is static), and not the effect of the data-link layer.

2 Evaluation Factors

As explained in Sect. 1, performance evaluation of protocols (above the data-link layer) depends on data-link and physical layer protocols and the underlying dynamic topology of the network. These factors are taken into account in our semantic model in [12]:

- (T_{Mac}, K) specifying the abstract data-link layer model: when a protocol has data to be transmitted, it delivers its message to its underlying data-link layer. The message is inserted in the queue of the node's data-link layer if the queue is not full, and messages in this queue are transmitted with an average delay equal to the average response time T_{Mac} of the data-link layer, i.e., the average time a message spends in a data-link layer queue, waiting to be transmitted or being transmitted.
- P_{UP} abstracting the dynamic underlying topology: we assume nodes move under an identical mobility model. Therefore in the steady-state, the probability that a link exists between two arbitrary nodes can be computed as explained in [23] for a mobility model (see also [12]). Generally speaking, the higher P_{UP} , the more successful communication.
- P_{rcv} abstracting physical layer protocols: a node located in the transmission range of a sender will receive its messages successfully with a probability P_{rcv} . In [8], this probability is computed by taking distance, signal strength and interference of other nodes into account, while [28] incorporates channel and radio parameters such as the path loss exponent and shadowing variance of the channel, and modulation and encoding of the radio. So this parameter provides an abstraction from physical characteristics of nodes, physical layer protocols and noise from the environment.

It should be noted that when the data-link layer of a node like ℓ broadcasts, it communicates to ℓ' with probability $P_{rcv} \times P_{UP}$ if there is a communication link between them (the link is UP) and ℓ' successfully receives. Otherwise either despite the readiness of ℓ' to receive, there is no communication link between them (the link was DOWN), or the link was UP but ℓ' received noisy data. Therefore ℓ does not communicate to ℓ' with probability $1 - P_{rcv} \times P_{UP}$. If ℓ' was not ready to receive, ℓ does not communicate to ℓ' with probability 1. We encode these concerns in the semantics: to each transition a probability is assigned (which is multiplied with the rate of the transition).

3 Stochastic RBPT

Network protocols (in particular MANET protocols) rely on data. To separate the manipulation of data from processes, we make use of abstract data types [6]. Data is specified by equational specifications: one can declare data types (so-called *sorts*) and functions working upon these data types, and describe the meaning of these functions by equational axioms. Following of μ CRL [15], *Stochastic Restricted Broadcast Process Theory (SRBPT)* is provided with equational abstract data types. The semantics of the data part (of a specification), denoted by \mathcal{ID} , is defined as in [15]. It should contain *Bool*, the booleans, with distinct T and F constants. We assume the data sorts *Bool* and *Nat*, the natural numbers, with the standard operations on them.

We mention some notations used in the definition of the formal syntax of *SRBPT*. Let D denote a data sort; u, v and d range over closed and open data terms of sort D , respectively. The functions $eq : D \times D$ and $if : Bool \times D \times D$ are defined for all data sorts D . The former only returns T if its two data parameters are semantically equal. The latter returns the first argument if the boolean parameter equals T , and otherwise the second argument. $d[d_1/d_2]$ denotes substitution of d_2 by d_1 in data term d ; this extends to substitution in process terms. Loc denotes a finite set of addresses, ranged over by ℓ , which represent hardware addresses of nodes. Msg denotes a set of message types communicated over a network and ranged over by m . Messages are parameterized with data; w.l.o.g. we let all messages have exactly one such parameter. \mathcal{A}_p denotes a countably infinite set of protocol names which are used in recursive process specifications.

3.1 Actions: Types and Rates

Each action in *SRBPT* is a pair (α, r) where α is the action name and r the rate. A process performs two types of actions: sending and receiving a message. The rate indicates the speed at which the action occurs from the viewpoint of the data-link layer. According to their rates, actions are classified as *active* and *passive*. *Active* actions have as rate either a positive real number or \top . A positive real number denotes the parameter of the exponentially distributed random variable specifying the duration of the action. Such actions are called *delayable*. The \top rate denotes *immediate* actions; either they are irrelevant from a performance point of view, or they have duration zero. *Passive* actions have an undefined rate, denoted by \perp . The duration of a passive action is fixed only by synchronizing it with an active action. Send actions are active while receive actions are passive. Restriction of the duration of active actions to exponential distributions enables us to define in *SRBPT* the classical interleaving semantics for the parallel composition operator, i.e. local broadcast, and to derive a Markov chain from the semantic model.

3.2 Syntax

The syntax of *SRBPT* is:

$$t ::= 0 \mid (\alpha, r).t \mid t + t \mid [b]t \diamond t \mid \sum_{d:D} t \mid A(d), A(v : D) \stackrel{def}{=} t \mid \llbracket t \rrbracket_\ell \mid t \parallel t.$$

A process can be a deadlock, modeled by 0. A process $(\alpha, r).t$ performs action α with rate $r \in \{\top, \perp\} \cup \mathbb{R}^{>0}$ and then behaves as t . An action α can be a send $snd(m(d))$ or a receive $rcv(m(d))$. A process $t_1 + t_2$ behaves non-deterministically as t_1 or t_2 . A conditional command $[b]t_1 \diamond t_2$ defines process behavior based on the data term b of sort *Bool*; if it evaluates to T in the data semantics the process behaves as t_1 , and otherwise as t_2 . Summation $\sum_{d:D} t$, which binds the name d in t , defines a non-deterministic choice among $t[u/d]$ for all closed $u \in D$. A process is specified by $A(d : D) \stackrel{def}{=} t$ where $A \in \mathcal{A}_p$ is a protocol name and d a variable name that appears free in t , meaning that it is not within the scope of a sum operator in t . We restrict to process specifications in which each occurrence of an $A(d)$ in t is within the scope of an action prefix. The simplest form of a MANET is a node, represented by the deployment operator $\llbracket t \rrbracket_\ell$; it

denotes process t deployed at network address ℓ . A MANET can be composed from MANETs using \parallel ; the MANETs communicate with each other by restricted broadcast.

We only consider well-defined *SRBPT* terms, meaning that processes deployed at a network address, called protocols, are defined by action prefix, choice, summation, conditional, and protocol names:

- If $t \equiv rcv(m(d)).t'$, then it is in the context of a $\sum_{d:D}$, which is in the context of a deployment operator. Furthermore, t' should be well-defined.
- If $t \equiv snd(m(d)).t_1$ or $t \equiv t_1 + t_2$ or $t \equiv [b]t_1 \diamond t_2$ or $t \equiv \sum_{d:D} t_1$, then it occurs in the scope of a deployment operator, and t_1 and t_2 are well-defined.
- If $t \equiv A(d)$, then it occurs in the the scope of a deployment operator, and $A(d : D) \stackrel{def}{=} t'$ where t' doesn't contain a parallel or deployment operator.
- If $t \equiv t_1 \parallel t_2$ or $t \equiv \llbracket t_1 \rrbracket_\ell$, then t isn't in the scope of a deployment operator, and t_1 and t_2 are well-defined.

3.3 Execution Mechanisms and Formal Semantics

Our semantics captures the interplay between network layer protocols and the underlying topology as explained in Sect. 2 such that compositionality is maintained. The state of a MANET node is defined by the state of its deployed protocol $\llbracket t \rrbracket_\ell$ and of its data-link layer Q_i , denoted by $\llbracket t \rrbracket_\ell : Q_i$. The latter denotes the state of the queue with capacity K , containing i messages: $Q_i = m_1(u_1) \cdot \dots \cdot m_i(u_i)$ where $i \leq K$. A state of a MANET, called configuration and denoted by \mathcal{T} , results from the aggregate states of its nodes. In a configuration, the data-link layers of different nodes may compete on the shared communication medium to broadcast. To provide the formal semantics for *SRBPT*, we informally examine the behavior of three operators.

Prefix. The configuration $\llbracket (snd(m(u)), r).0 \rrbracket_\ell : Q_0$ specifies that it delivers message $m(u)$ after a delay, exponentially distributed with rate r , to its underlying data-link layer to be transferred to the network. The configuration $\llbracket 0 \rrbracket_\ell : m(u)$ is reached, in which the data-link layer transfers $m(u)$ to the network after an average delay of T_{Mac} , and the configuration $\llbracket 0 \rrbracket_\ell : Q_0$ is reached. The semantic model of $\llbracket (snd(m(u)), r).0 \rrbracket_\ell : Q_0$ has three states with two transitions; message delivery of a protocol to its data-link layer is represented by a transition with the internal action (τ, r) , while sending a message from the data-link layer to the network is represented by a transition with the label $(nsnd(m(d), \ell), r_{Mac})$ where $r_{Mac} = 1/T_{Mac}$.

Choice. In the configuration $\llbracket (snd(m_1(u_1)), r_1).0 + (snd(m_2(u_2)), r_2).0 \rrbracket_\ell : Q_0$, the protocol can deliver message $m_1(u_1)$ or $m_2(u_2)$. Following [2,18], we adopt the *race policy* mechanism for choosing the delayable action to execute: the action sampling the least duration wins the competition and therefore, the choice is resolved probabilistically. Hence the transitions in Fig. 1 are achieved for this configuration. The two transitions labeled by (τ, r_i) denote interaction of the protocol with its data-link layer, executed with the probability $r_i/(r_1 + r_2)$, while the transitions labeled by $(nsnd(m_i(u_i), \ell), r_{Mac})$ denote interaction of the data-link layer with its environment.

A consequence of the adoption of the race policy is that when a delayable and an immediate action compete, the immediate action is executed, since the delayable action

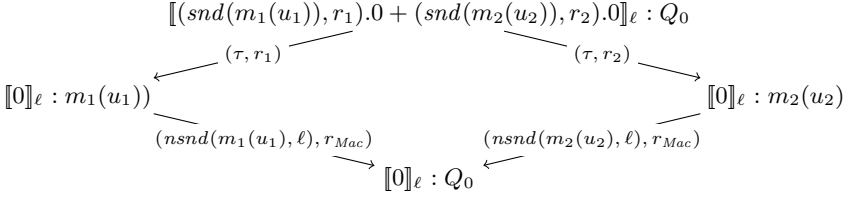


Fig. 1. Transitions of $[[snd(m_1(u_1)), r_1].0 + (snd(m_2(u_2)), r_2).0]_\ell : Q_0$

cannot sample to zero from the associated exponential distribution, $F_X(0) = Pr[X \leq 0] = 0$. In other words, immediate actions take precedence over delayable actions. So stochastically speaking, configurations have the *maximal progress* property [16]. To achieve compositionality as explained in [16], this precedence is not reflected in the semantic rules of Table 1. Instead, we introduce a notion of equality in Sect. 4.1 to take care of it. We adopt non-determinism to choose between passive actions to execute in a state. Therefore the behavior of the choice operator is probabilistic (for delayable actions), prioritized (for immediate and delayable actions) or non-deterministic (for immediate or passive actions) according to its operands.

Parallel. $M_0 \equiv [[snd(m_1(u_1)), r_1].0]_A : Q_0 \parallel [[snd(m_2(u_2)), r_2].0]_B : Q_0$ consists of two nodes with addresses A and B . The transitions are achieved using the memoryless property of exponential distribution as shown in Fig. 2. In the initial configuration, the actions $(snd(m_1(u_1)), r_1)$ and $(snd(m_2(u_2)), r_2)$ compete to execute. If we assume that $snd(m_1(u_1))$ finishes before $snd(m_2(u_1))$, then the remaining time of $snd(m_2(u_1))$ still has a distribution with rate r_2 . Therefore these two actions can be interleaved. Likewise, after delivery of message $m_1(u_1)$ by node A to its data-link layer, this data-link layer and action $snd(m_2(u_1))$ compete to execute. Since the probability of having the same delay for actions $(snd(m_1(u_1)), r_1)$ and $(snd(m_2(u_1)), r_2)$ is zero, there is no transition from M_0 to M_4 .

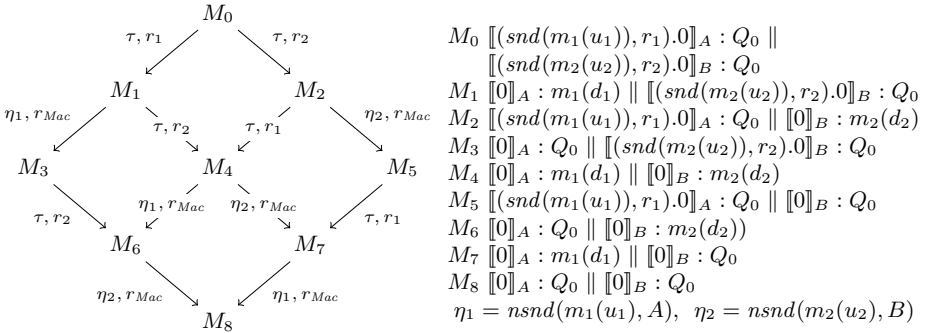


Fig. 2. Transitions of $[[snd(m_1(u_1)), r_1].0]_A : Q_0 \parallel [[snd(m_2(u_2)), r_2).0]_B : Q_0$

Consider the closed configuration $\llbracket t_1 \rrbracket_{\ell_1} : Q_{i_1}^1 \parallel \dots \parallel \llbracket t_n \rrbracket_{\ell_n} : Q_{i_n}^n$ on data-link layer (T_{Mac}, K) , where $i_1, \dots, i_n \leq K$, and P_{UP} is defined. Following the approach of [18], its semantics is given by a labeled multi-transition system with the multi-set transition relation results from the multi-set union of transitions $\mathcal{T} \xrightarrow{(\eta, \lambda)} \mathcal{T}'$ induced by the rules in Table 1, where $\lambda \in \{\top, r\perp\} \cup \mathbb{R}^{>0}$ ($r\perp$ is a shorthand for $r \times \perp$) and η is the unobservable action τ or a network send or receive action $nsnd(m(d), \ell)$ or $nrcv(m(d))$. Let $t \not\xrightarrow{(rcv(m(d)), -)}$ denote there is no t' such that $t \xrightarrow{(rcv(m(d)), \perp)} t'$. Conversely we also use $t \xrightarrow{(m(d)?, -)}$. Let $|\{t^* | t \xrightarrow{(rcv(m(d)), \perp)} t^*\}|$ denote the number of transitions that t can make by performing $(rcv(m(d)), \perp)$, where $\{\} \}$ denotes a multi-set. In Table 1 the symmetric counterparts of *Choice*, *Sync₂* and *Par* are omitted.

Table 1. Operational semantics of MANETs

$$\begin{array}{c}
\frac{}{(\alpha, r).t \xrightarrow{(\alpha, r)} t} : Pre \quad \frac{t_1 \xrightarrow{(\alpha, r)} t'_1}{t_1 + t_2 \xrightarrow{(\alpha, r)} t'_1} : Choice \quad \frac{t[u/d] \xrightarrow{(\alpha, r)} t'}{A(u) \xrightarrow{(\alpha, r)} t'} : Inv, A(d : D) \stackrel{def}{=} t \\
\\
\frac{t[u/d] \xrightarrow{(\alpha, r)} t'}{\sum_{d:D} t \xrightarrow{(\alpha, r)} t'} : Sum \quad \frac{t_1 \xrightarrow{(\alpha, r)} t'_1 \quad \mathcal{ID} \models b = T}{[b]t_1 \diamond t_2 \xrightarrow{(\alpha, r)} t'_1} : Then \quad \frac{t_2 \xrightarrow{(\alpha, r)} t'_2 \quad \mathcal{ID} \models b = F}{[b]t_1 \diamond t_2 \xrightarrow{(\alpha, r)} t'_2} : Else \\
\\
\frac{t \xrightarrow{(snd(m(d)), r)} t' \quad i < K}{\llbracket t \rrbracket_{\ell} : Q_i \xrightarrow{(\tau, r)} \llbracket t' \rrbracket_{\ell} : Q_i \cdot m(d)} : Snd_1 \quad \frac{t \xrightarrow{(snd(m(d)), r)} t' \quad i = K}{\llbracket t \rrbracket_{\ell} : Q_i \xrightarrow{(\tau, r)} \llbracket t' \rrbracket_{\ell} : Q_i} : Snd_2 \\
\\
\frac{t \xrightarrow{(rcv(m(d)), \perp)} t'}{\llbracket t \rrbracket_{\ell} : Q \xrightarrow{(nrcv(m(d)), r(t)\perp)} \llbracket t' \rrbracket_{\ell} : Q} : Rcv_1 \quad \frac{t \xrightarrow{(rcv(m(d)), -)}}{\llbracket t \rrbracket_{\ell'} : Q \xrightarrow{(nrcv(m(d)), r\sim s\perp)} \llbracket t \rrbracket_{\ell'} : Q} : Rcv_2 \\
\\
\frac{t \xrightarrow{(rcv(m(d)), -)}}{\llbracket t \rrbracket_{\ell} \xrightarrow{(nrcv(m(d)), \perp)} \llbracket t \rrbracket_{\ell}} : Rcv_3 \quad \frac{\mathcal{T}_1 \xrightarrow{(nrcv(m(d)), r_1\perp)} \mathcal{T}'_1 \quad \mathcal{T}_2 \xrightarrow{(nrcv(m(d)), r_2\perp)} \mathcal{T}'_2}{\mathcal{T}_1 \parallel \mathcal{T}_2 \xrightarrow{(nrcv(m(d)), r_1 \times r_2\perp)} \mathcal{T}'_1 \parallel \mathcal{T}'_2} : Sync_1 \\
\\
\frac{\mathcal{T}_1 \xrightarrow{(\tau, \lambda)} \mathcal{T}'_1}{\mathcal{T}_1 \parallel \mathcal{T}_2 \xrightarrow{(\tau, \lambda)} \mathcal{T}'_1 \parallel \mathcal{T}_2} : Par \quad \frac{\mathcal{T}_1 \xrightarrow{(nsnd(m(d), \ell), \lambda)} \mathcal{T}'_1 \quad \mathcal{T}_2 \xrightarrow{(nrcv(m(d)), r\perp)} \mathcal{T}'_2}{\mathcal{T}_1 \parallel \mathcal{T}_2 \xrightarrow{(nsnd(m(d), \ell), r \times \lambda)} \mathcal{T}'_1 \parallel \mathcal{T}'_2} : Sync_2 \\
\\
\frac{}{\llbracket t \rrbracket_{\ell} : m(d) \cdot Q \xrightarrow{(nsnd(m(d), \ell), r_{Mac})} \llbracket t \rrbracket_{\ell} : Q} : Bro
\end{array}$$

Pre, *Sum*, *Choice*, *Inv*, *Then* and *Else* are standard rules for basic process algebras. Interactions between protocol t and its data-link layer are specified by *Snd_{1,2}*: when a protocol t transmits a message, it is delivered to the data-link layer, which inserts it at the end of its queue if there is space, else the message is dropped. This action is considered internal from the viewpoint of the data-link layer, and consequently it cannot be synchronized with other actions, as explained by *Par*. After this synchronization the data-link layer transmits the message with average time T_{Mac} . The sojourn time

corresponding to a configuration that ends with a local broadcast (by a node) is an exponentially distributed random variable with rate r_{Mac} . $Brcv$ explains that the data-link layer of a node transmits messages in its queue with rate r_{Mac} , while the network address of the node is appended to this message. Rcv_1 specifies that a process t can receive a message successfully if it has a link to a sender (P_{UP}) and receives the message correctly (P_{rcv}). Therefore the probability of a successful receive action is $r(t) = r_s \times r_t$, where $r_s = P_{rcv} \times P_{UP}$ and $r_t = \frac{1}{|\{t^* | t \xrightarrow{(rcv(m(d)), \perp)} t^*\}|}$ is the normalization factor (since a protocol can non-deterministically execute multiple receive actions, the normalization factor maintains the rate of the sojourn time of a configuration ending with a local broadcast to r_{Mac}). However, if the node does not perform the receive action which was enabled, then the node was either disconnected with probability $1 - P_{UP}$, or it could not receive successfully with probability $P_{UP} \times (1 - P_{rcv})$. Therefore a node with an enabled receive action does not receive with probability $r_{-s} = 1 - P_{UP} \times P_{rcv}$. This behavior is explained in Rcv_2 , by making the node perform the receive action while the state of its process is unchanged. A node with no enabled receive action can be synchronized with probability 1, while the state of its process is unchanged, as explained by Rcv_3 . $Sync_1$ allows to group together nodes that are ready to receive the same message. In this case, the probability of receive actions is a product of all receive coefficients. $Sync_2$ explains what happens when a node broadcasts: the rate of synchronization is the rate of broadcast multiplied by the probability of receivers.

Example. Consider MANET $\llbracket P(A) \rrbracket_A \parallel \llbracket Q(B) \rrbracket_B \parallel \llbracket R \rrbracket_C$ with $P(adr : Loc) \stackrel{def}{=} (snd(elec(adr), r).0, Q(adr : Loc) \stackrel{def}{=} \sum_{lx:Loc} (rcv(elec(lx)), \perp).0 + (snd(elec(adr), r).0$ and $R \stackrel{def}{=} \sum_{lx:Loc} (rcv(elec(x)), \perp).0$, on a data-link layer with $(T_{Mac}, 1)$. By Snd_1 :

$$\llbracket P(A) \rrbracket_A : Q_0 \parallel \llbracket Q(B) \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0 \xrightarrow{(\tau, r)} \llbracket 0 \rrbracket_A : elec(A) \parallel \llbracket Q(B) \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0$$

If node A broadcasts $elec(A)$ and only B receives, then by Rcv_1 , Rcv_2 for nodes B , C respectively and $Sync_2$:

$$\llbracket 0 \rrbracket_A : elec(A) \parallel \llbracket Q(B) \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0 \xrightarrow{(nsnd(elec(A), A), \tau)} \llbracket 0 \rrbracket_A : Q_0 \parallel \llbracket 0 \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0$$

where $\tau = r_{Mac} \times r_s \times r_{-s}$. If B broadcasts and C does not receive, then by Rcv_3 , Rcv_2 for nodes A , C respectively and $Sync_2$:

$$\begin{aligned} \llbracket 0 \rrbracket_A : elec(A) \parallel \llbracket Q(B) \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0 &\xrightarrow{(\tau, r)} \\ \llbracket 0 \rrbracket_A : elec(A) \parallel \llbracket 0 \rrbracket_B : elec(B) \parallel \llbracket R \rrbracket_C : Q_0 &\xrightarrow{(nsnd(elec(B), B)r_{Mac} \times r_{-s})} \\ \llbracket 0 \rrbracket_A : elec(A) \parallel \llbracket 0 \rrbracket_B : Q_0 \parallel \llbracket R \rrbracket_C : Q_0 & \end{aligned}$$

4 From Configuration to CTMC

Our framework aims at the evaluation of MANET protocols by means of CTMCs derived from the semantic model. Immediate actions give rise to non-determinism. To obtain a CTMC, we need to eliminate immediate actions by means of an appropriate congruence relation.

4.1 Weak Markovian Network Bisimilarity

We adapt the notion of weak Markovian bisimilarity from [17] for our framework, which behaves as weak bisimilarity [14] on immediate actions and as Markovian bisimilarity [18] on delayable and passive actions. It is called *weak Markovian network bisimilarity*: delayable and passive actions are treated as in Markovian bisimilarity, but they may be preceded and followed by internal immediate actions.

Let $Q(T_{Mac}, K)$ denote the set of states of the data-link layer, a queue with capacity K and response time T_{Mac} . We write $\xrightarrow{\tau^*}$ for the transitive closure of $\xrightarrow{(\tau, \top)}$ transitions, and $\not\xrightarrow{\tau}$ to denote there is no \mathcal{T}' such that $\mathcal{T} \xrightarrow{(\tau, \top)} \mathcal{T}'$.

The definition of a weak Markovian network bisimulation is obtained in the same manner as [16,17]: a passive/delayable action must be simulated by a matching step, possibly preceded and followed by arbitrarily many immediate internal steps. Stochastically speaking, the cumulated rate of moving by a passive/delayable action to an equivalence class should be equal for each transition and its match. Since its match may be preceded by internal transitions, for an equivalence class $\mathcal{C}_{\mathcal{R}}$ we let $\mathcal{C}_{\mathcal{R}}^{\tau}$ denote the set $\{\mathcal{T}' \mid \exists \mathcal{T} \in \mathcal{C}_{\mathcal{R}} \cdot \mathcal{T}' \xrightarrow{\tau^*} \mathcal{T}\}$. An internal immediate step may be simulated, but can also be mimicked by taking no transition at all, provided the equivalence classes match.

Definition 1. *An equivalence relation \mathcal{R} on configurations is a weak Markovian network bisimulation if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ implies for all equivalence classes $\mathcal{C}_{\mathcal{R}} \in (SRBPT \times Q(T_{Mac}, K))/\mathcal{R}$:*

- if $\mathcal{T}_1 \not\xrightarrow{\tau}$ then $\gamma(\mathcal{T}_1, \eta, \mathcal{C}_{\mathcal{R}}) = \gamma(\mathcal{T}_2', \eta, \mathcal{C}_{\mathcal{R}}^{\tau})$ for some \mathcal{T}_2' with $\mathcal{T}_2 \xrightarrow{\tau^*} \mathcal{T}_2'$, $\mathcal{T}_2' \not\xrightarrow{\tau}$;
- if $\mathcal{T}_2 \not\xrightarrow{\tau}$ then $\gamma(\mathcal{T}_2, \eta, \mathcal{C}_{\mathcal{R}}) = \gamma(\mathcal{T}_1', \eta, \mathcal{C}_{\mathcal{R}}^{\tau})$ for some \mathcal{T}_1' with $\mathcal{T}_1 \xrightarrow{\tau^*} \mathcal{T}_1'$, $\mathcal{T}_1' \not\xrightarrow{\tau}$;
- if $\mathcal{T}_1 \xrightarrow{(\tau, \top)} \mathcal{T}_1'$ then for some $\mathcal{T}_2', \mathcal{T}_2 \xrightarrow{\tau^*} \mathcal{T}_2'$, $\mathcal{T}_1' \mathcal{R} \mathcal{T}_2'$;
- if $\mathcal{T}_2 \xrightarrow{(\tau, \top)} \mathcal{T}_2'$ then for some $\mathcal{T}_1', \mathcal{T}_1 \xrightarrow{\tau^*} \mathcal{T}_1'$, $\mathcal{T}_1' \mathcal{R} \mathcal{T}_2'$.

$\gamma(\mathcal{T}, \eta, \mathcal{C}_{\mathcal{R}}) = \sum \{\lambda \mid \mathcal{T} \xrightarrow{(\eta, \lambda)} \mathcal{T}', \mathcal{T}' \in \mathcal{C}_{\mathcal{R}}\}$, i.e. the summation of all elements in this multiset. Since $r_1 \perp + r_2 \perp = (r_1 + r_2) \perp$ and $r_1 \perp = r_2 \perp$ if and only if $r_1 = r_2$, γ is well-defined. Configurations \mathcal{T}_1 and \mathcal{T}_2 are weak Markovian network bisimilar, denoted $\mathcal{T}_1 \approx_m \mathcal{T}_2$, if $\mathcal{T}_1 \mathcal{R} \mathcal{T}_2$ with \mathcal{R} a weak Markovian network bisimulation.

Theorem 1. *Markovian network bisimilarity is an equivalence relation, and a congruence for configurations.*

See [13] for the proof.

Example. The following equivalences hold:

$$\begin{aligned} \llbracket (\alpha, r).t_1 + (snd(m(d)), \top).t_2 \rrbracket_{\ell} : Q &\approx_m \llbracket (snd(m(d)), \top).t_2 \rrbracket_{\ell} : Q, r \in \mathbb{R}^{>0} \\ \llbracket (snd(m(d)), \top).t \rrbracket_{\ell} : Q &\approx_m \llbracket t \rrbracket_{\ell} : Q \cdot m(d) \end{aligned}$$

The first explains that immediate actions have precedence over delayable actions. The second explains how an immediate action $snd(m(d))$ can be removed while its impact, insertion of $m(d)$ at the end of the queue, is considered.

4.2 Markovian Semantics of MANETs

A Markov model can be derived from a MANET specification, if the non-deterministic choices can be resolved by application of Markovian network bisimilarity: each configuration equivalence class in the MANET is a state of the stochastic process, and the transitions are defined by collapsing transitions carrying active actions between corresponding configuration equivalence classes while adding up the rates; see Theorem. 2.

Definition 2. *The derivative set $ds(\mathcal{T})$ of a MANET model \mathcal{T} is the smallest set of configurations such that:*

- $\mathcal{T} \in ds(\mathcal{T})$; and
- if $\mathcal{T}_i \in ds(\mathcal{T})$, and $\mathcal{T}_i \xrightarrow{(\eta, \lambda)} \mathcal{T}_j$ with $\lambda \in \{\top\} \cup \mathbb{R}^{>0}$ and $\eta \in \{\tau, nsnd(m(d), \ell)\}$, then $\mathcal{T}_j \in ds(\mathcal{T})$.

A Markov process can be derived from a configuration \mathcal{T} if each equivalence class $[\mathcal{T}_i]_{/\approx_m}$ with $\mathcal{T}_i \in ds(\mathcal{T})$ cannot move to another equivalence class by an immediate action:

$$\forall \mathcal{T}_j \in [\mathcal{T}_i]_{/\approx_m} \cdot \mathcal{T}_j \xrightarrow{\tau} \mathcal{T}'_j \Rightarrow \mathcal{T}'_j \in [\mathcal{T}_i]_{/\approx_m}.$$

So immediate actions are removed by weak Markovian network bisimilarity. Such a configuration is called Markovian.

Theorem 2. *Given a finite closed Markovian configuration \mathcal{T} , let the stochastic process $X(t)$ be defined such that $X(t) = [\mathcal{T}_i]_{/\approx_m}$ for some $\mathcal{T}_i \in ds(\mathcal{T})$, indicating that the MANET is in a configuration belonging to $[\mathcal{T}_i]_{/\approx_m}$ at time t . Then $X(t)$ is a Markov process.*

The proof is straightforward (cf. [12,18]). If transition rates are independent of the time at which a transition occurs, the derived CTMC is time-homogeneous, so that it can be used to evaluate the performance of a MANET in terms of different data-link layer service quality, mobility models, and protocol parameter settings.

5 A Leader Election Algorithm for MANETs

In this section we illustrate how the *SRBPT* framework is applicable in analyzing MANET protocols. For this purpose we use the leader election algorithm for MANETs from [27].

5.1 Protocol Specification

Each node has a value associated with it. In the context of MANETs, the leader election algorithms aim at finding the highest-valued node within a connected component during a limited period of time, when the underlying topology is stable. For simplicity the value of a node is the same as its network address. Let $? \in Loc$ denote an unknown address. We assume a total order on network addresses, where $?$ is the minimum. Election is performed in three stages. In the first stage, a spanning tree is grown which (potentially) contains all the nodes within a connected component by broadcasting *elec* messages,

which make nodes join the election and send it in turn. To this aim, each node initially sends a *elec* message after waiting $1/r_{heartbeat}$ in average to receive from a leader in its vicinity. After a node receives *elec*(*xparent*), it sets its parent to *xparent* and then immediately relay the message. In the second stage, values are collected through *ack* messages, which contain the maximum value of a subtree under a node and are passed on to the parent in the spanning tree. Inner nodes of the spanning tree wait for an average time of $1/r_{child_timeout}$ to gather *ack* messages from their potential children and inform their parent. On receiving *ack*, each node updates the maximum value it knows from its subtree.

$$\begin{aligned}
& Node(s_n : Nat, id, lid, max, parent : Loc, elec, ack : Bool) \stackrel{def}{=} \\
& [eq(lid, id)](snd(leader(lid)), r_{heartbeat}).Node(s_n, id, lid, max, parent, elec, ack) \diamond 0 \\
& + [eq(s_n, 0)](snd(elec(id)), r_{heartbeat}).Node(1, id, lid, id, id, T, T) \diamond 0 \\
& + [eq(s_n, 2)](snd(elec(id)), \top).Node(1, id, lid, id, parent, T, T) \diamond 0 \\
& + [s_n < 2 \wedge \neg elec] \sum_{lx:Loc} (rcv(elec(lx)), \perp).Node(2, id, lid, max, lx, elec, ack) \diamond 0 \\
& + [eq(s_n, 1) \wedge ack] \sum_{xmax:Loc} \sum_{xid} (rcv(ack(xmax, xid)), \perp) \\
& \quad .Node(s_n, id, lid, if(xmax > max, xmax, max), parent, elec, ack) \diamond 0 \\
& + [eq(s_n, 1) \wedge ack \wedge \neg eq(parent, id)](snd(ack(max, parent)), r_{child_timeout}) \\
& \quad .Node(s_n, id, lid, max, parent, elec, F) \diamond 0 \\
& + [eq(s_n, 1) \wedge eq(parent, id) \wedge ack](snd(leader(max)), r_{child_timeout}) \\
& \quad .Node(s_n, id, max, max, parent, F, F) \diamond 0 \\
& + [eq(s_n, 3)](snd(leader(lid)), \top).Node(1, id, lid, max, parent, F, F) \diamond 0 \\
& + [s_n < 2 \wedge ((elec \wedge \neg ack) \vee \neg elec)] \sum_{xlid:Loc} (rcv(leader(xlid)), \perp) \\
& \quad .Node(if(\neg elec \vee (\neg ack \wedge xlid > max), 3, 1), id, \\
& \quad \quad if((\neg elec \wedge xlid > lid) \vee (elec \wedge \neg ack \wedge xlid > max), xlid, lid), \\
& \quad \quad if((\neg elec \wedge xlid > lid) \vee (elec \wedge \neg ack \wedge xlid > max), xlid, max), parent, \\
& \quad \quad if((elec \wedge xlid > max), F, elec), ack) \diamond 0 \\
& + [eq(s_n, 1) \wedge \neg eq(lid, ?) \wedge \neg eq(lid, id) \wedge \neg elec] \\
& \quad (snd(elec(id)), r_{hb_timeout}).Node(s_n, id, ?, id, id, T, T) \diamond 0 \\
& + [eq(s_n, 1)](snd(crash), r_{crash_freq}).Node(0, id, ?, ?, F, F) \diamond 0 \\
& + \sum_{xid:Loc} (rcv(probe(xid)), \perp).Node(if(eq(xid, id), 5, s_n), id, lid, max, parent, elec, ack) \\
& + [eq(s_n, 5)](snd(reply(xid)), \top). \\
& \quad Node(if(elec \vee \neg eq(lid, ?), 1, 0), id, lid, max, parent, elec, ack) \diamond 0 \\
& + [eq(s_n, 1) \wedge \neg eq(parent, id) \wedge \neg ack \wedge elec](snd(probe(parent)), r_{probe_freq}) \\
& \quad .Node(4, id, lid, max, parent, elec, ack) \diamond 0 \\
& + [eq(s_n, 4)](\sum_{xparent:Loc} (rcv(reply(xparent)), \perp). \\
& \quad Node(if(eq(xparent, parent), 1, s_n), id, lid, max, parent, elec, ack) \\
& \quad + (snd(leader(max)), r_{reply_timeout}).Node(1, id, max, max, id, F, F) \diamond 0
\end{aligned}$$

Fig. 3. Specification of a leader election algorithm for MANETs

In the third stage, a node declares the maximum value by broadcasting the *leader* message, if it is a root (on expiration of a timer with rate $r_{child_timeout}$ during which *ack*'s are gathered), or it has been disconnected from its parent (on expiration of a timer with rate $r_{reply_timeout}$ to detect its parent). This message is then broadcast periodically

with rate $r_{heartbeat}$ to reestablish leadership of a node, until it is challenged by a greater value. If a node does not hear from its leader for an average time of $1/r_{hb.timeout}$, it initiates a leader election. The spanning tree can change during these stages, since nodes can move into or out of a connected component at will. To keep the tree connected and swiftly respond to changes, a node constantly checks the existence of its parent by sending/receiving *probe/reply* messages.

The leader election algorithm is specified by the protocol name *Node* in Fig. 3. The list of variables maintained by each protocol are: *elec*, *ack* of type *Bool*, where *elec* is *T* when the node is involved in an election, while *ack* is *T* if the node has not sent its *ack* message to its parent; *lid*, *max*, *parent* of type *Loc*, where *lid* denotes the address of the leader (which is updated when the node receives a *leader* message), *max* is the highest value the node is aware of in an election, and *parent* is the address of the node's parent in the spanning tree. The protocol is initially (or after a crash) in a state with $eq(s_n, 0)$, $eq(elec, F)$, $eq(ack, F)$, $eq(parent, ?)$ and $eq(lid, ?)$.

5.2 Protocol Analysis

We focus on evaluating effects of some parameters like r_{Mac} and timer values on protocol performance. We construct configurations by composing several $\llbracket Node(0, \ell, ?, ?, F, F) \rrbracket_\ell : Q_0$, and examine message overhead and the duration from the start of the election until all nodes have found a leader (called election time). It should be examined that configurations are Markovian, and consequently by Theorem. 2 a CTMC can be derived.

Each node immediately sends a message when it is in the state $eq(s_n, 2) \vee eq(s_n, 3) \vee eq(s_n, 5)$. It can be shown that the following equivalences hold (by constructing the Markovian network bisimulation relation $\mathcal{R} = \{(\mathcal{T}_1, \mathcal{T}_2) \mid \forall Q_i \cdot i < K\} \cup \{(\mathcal{T}, \mathcal{T})\}$ for each equivalence relation $\mathcal{T}_1 \approx_m \mathcal{T}_2$):

$$\begin{aligned} \llbracket Node(2, \theta, F, ack) \rrbracket_\ell : Q &\approx_m \llbracket Node(1, \theta, F, ack) \rrbracket_\ell : Q \cdot elec(\ell) \\ \llbracket Node(3, \theta, elec, ack) \rrbracket_\ell : Q &\approx_m \llbracket Node(1, \theta, F, F) \rrbracket_\ell : Q \cdot leader(lid) \\ \llbracket Node(5, \theta', F, ack) \rrbracket_\ell : Q &\approx_m \llbracket Node(0, \theta', F, ack) \rrbracket_\ell : Q \cdot reply(\ell') \\ \llbracket Node(5, \theta, elec, ack) \rrbracket_\ell : Q &\approx_m \llbracket Node(1, \theta, elec, ack) \rrbracket_\ell : Q \cdot reply(\ell'), \text{ where } elec \vee \neg eq(lid, ?) \end{aligned}$$

where θ abbreviates *id*, *lid*, *max*, *parent* and θ' abbreviates *id*, *?*, *max*, *parent*. Therefore, by congruence of \approx_m , in any configuration, nodes in the form of the left-hand side of an equation above can be replaced by the corresponding right-hand side.

We exploit PRISM to derive the overall CTMC resulting from a MANET configuration. To this aim, we cast the resulting CTMC of each node to PRISM such that its parallel composition with other nodes in the MANET results in the overall CTMC. See the [13] for how the encoding is managed. We note that the cast CTMC of a node in PRISM is dependent on all locations in the MANET (no modularity), and it is not straightforward to write the code in PRISM from the scratch. We implemented the *Loc* data sort using the integer type of PRISM (where *?*, *A*, *B*, ... are denoted by 0, 1, 2, ...), and so the cast of the configuration $\llbracket Node(0, \ell, ?, ?, F, F) \rrbracket_\ell : Q_0$ is well-defined.¹

¹ See <http://mehr.sharif.edu/~fghassemi/pcodes.zip>.

Then we can define desired properties in a well-known stochastic temporal logic, Continuous Stochastic Logic [1], which has been extended in PRISM with rewards and queries. By assigning a reward to each send action, we can compute the number of messages sent during an election:

$$R_{|messages|=?}[Flid_A \neq ? \wedge lid_B \neq 0 \wedge lid_C \neq 0 \wedge lid_D \neq 0 \wedge lid_E \neq 0]$$

where the condition $Flid_A \neq ? \wedge lid_B \neq 0 \wedge lid_C \neq 0 \wedge lid_D \neq 0 \wedge lid_E \neq 0$ specifies that all nodes will finally have a leader. We can thus examine the message overhead for different implementation policies. We have also examined another implementation of the protocol; the node tries to participate in the election of a node having the highest value. Thus each node listens to the next election messages, and whenever it receives one with a greater value than its own *parent*, it immediately changes its parent. We use “single” and “multiple” election for the first and second implementation. In Fig. 4, the trend of the message overhead growth is illustrated with respect to the number of nodes for each implementation, which shows the nearly linear relationship between the increase in the number of nodes and the increase in the message overhead. We used the values for $r_{heartbeat} = 0.05$ and $r_{hb_timeout} = 6 \times r_{heartbeat}$ given in [27] and $r_{child_timeout} = 0.02$, $r_{crash_freq} = 0.000028$, $r_{prob_freq} = 0.1$, and $r_{reply_timeout} = 0.1$. P_{UP} is 0.7, and r_{Mac} is 10.

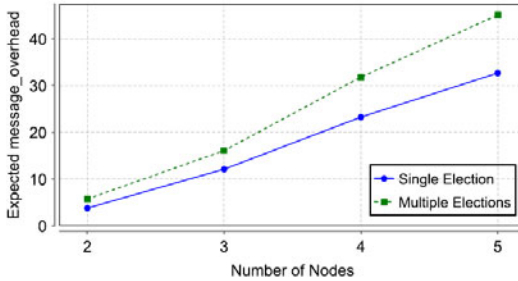


Fig. 4. Message overhead in MANETs of different size

We can compute the distribution function for the duration of an election (for a specific node) in which the highest-valued node is elected as the leader by

$$P_{=?}[lid_A = 0 U_T lid_A = max\{id_A, id_B, id_C, id_D, id_E\}.$$

where the until operator U_T computes the time between when the node has no leader and when its leader has the maximum value. In Fig. 5a, the probability that this election time is less than 40 seconds (for a MANET of five nodes with a data-link layer with capacity 2) is measured in terms of different values of P_{rcv} and $t_{child_timeout} = 1/r_{child_timeout}$. Fig. 5a shows that the optimal choice for child timeout in a network with varying P_{rcv} is 1.0s for the single election implementation. We can again examine the effect of parameters like P_{UP} .

This probability is compared for each implementation when $t_{child_timeout} = 1$ in Fig. 5b: the election time for the multiple election implementation is reduced.

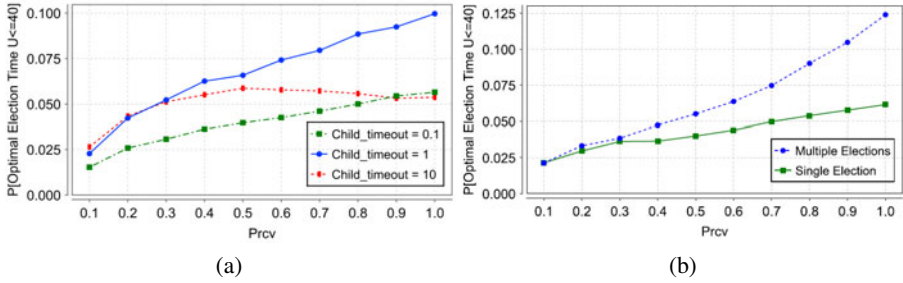


Fig. 5. Effect of P_{rcv} and $t_{child_timeout}$ on election time (left). Probability that the election time is less than 40 sec. for a single and multiple elections (right).

Depending on the quality of service metrics, election time or message overhead, either one of the protocol implementations can be chosen.

6 Conclusion and Future Work

We have extended *RBPT*, an algebraic framework for the specification and verification of MANETs, with stochastic concepts. This is useful for the analysis of protocols in terms of environment (like data-link layer quality of service, mobility of nodes) and protocol parameters.

We plan to extend *SRBPT* following the approach of [9], to arrive at a sound and complete axiomatization. Then we can define a stochastic variant of linear process equations (SLPE) for configurations, which can be exploited in the analysis of MANETs with an arbitrary number of nodes. To this aim, we can also extract the embedded DTMC of an SLPE and then use symbolic confluence reduction [3] or the SCOOP tool [26] to reduce the SLPE. With an increase in the number of nodes, the effect of T_{Mac} on a MANET can be sensed more.

References

1. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.: On the logical characterisation of performability properties. In: Welzl, E., Montanari, U., Rolim, J.D.P. (eds.) *ICALP 2000*. LNCS, vol. 1853, pp. 780–792. Springer, Heidelberg (2000)
2. Bernardo, M., Gorrieri, R.: A tutorial on EMPA: A theory of concurrent processes with non-determinism, priorities, probabilities and time. *Theoretical Computer Science* 202(1-2), 1–54 (1998)
3. Blom, S., van de Pol, J.: State space reduction by proving confluence. In: Brinksma, E., Larsen, K.G. (eds.) *CAV 2002*. LNCS, vol. 2404, pp. 596–609. Springer, Heidelberg (2002)
4. Daws, C., Kwiatkowska, M., Norman, G.: Automatic verification of the IEEE 1394 root contention protocol with KRONOS and PRISM. *Software Tools for Technology Transfer* 5(2), 221–236 (2004)
5. Dufлот, M., Kwiatkowska, M., Norman, G., Parker, D.: A formal analysis of Bluetooth device discovery. In: *Proc. ISOLA 2004*, pp. 268–275 (2004)
6. Ehrich, H., Loeckx, J., Wolf, M.: *Specification of Abstract Data Types*. John Wiley, Chichester (1996)

7. Ene, C., Muntean, T.: Expressiveness of point-to-point versus broadcast communications. In: Ciobanu, G., Păun, G. (eds.) FCT 1999. LNCS, vol. 1684, pp. 258–268. Springer, Heidelberg (1999)
8. Fehnker, A., Fruth, M., McIver, A.K.: Graphical modelling for simulation and formal analysis of wireless network protocols. In: Butler, M., Jones, C., Romanovsky, A., Troubitsyna, E. (eds.) Methods, Models and Tools for Fault Tolerance. LNCS, vol. 5454, pp. 1–24. Springer, Heidelberg (2009)
9. Ghassemi, F., Fokkink, W., Movaghar, A.: Equational reasoning on mobile ad hoc networks. *Fundamenta Informaticae* 103, 1–41 (2010)
10. Ghassemi, F., Fokkink, W., Movaghar, A.: Verification of mobile ad hoc networks: An algebraic approach. *Theoretical Computer Science* 412(28), 3262–3282 (2011)
11. Ghassemi, F., Fokkink, W.J., Movaghar, A.: Restricted broadcast process theory. In: Proc. SEFM 2008, pp. 345–354. IEEE, Los Alamitos (2008)
12. Ghassemi, F., Fokkink, W.J., Movaghar, A.: Towards performance evaluation of mobile ad hoc network protocols. In: Proc. ACSD 2010, pp. 98–105. IEEE, Los Alamitos (2010)
13. Ghassemi, F., Talebi, M., Fokkink, W., Movaghar, A.: Stochastic restricted broadcast process theory. Tech. rep., Sharif University of Technology (2011), <http://mehr.sharif.edu/fghassemi/srbpt-web.pdf>
14. van Glabbeek, R., Weijland, W.: Branching time and abstraction in bisimulation semantics. *Journal of the ACM* 43(3), 555–600 (1996)
15. Groote, J.F., Ponse, A.: Syntax and semantics of μ -CRL. In: Proc. ACP 1994. Workshops in Computing, pp. 26–62. Springer, Heidelberg (1995)
16. Hermanns, H., Herzog, U., Katoen, J.P.: Process algebra for performance evaluation. *Theoretical Computer Science* 274(1-2), 43–87 (2002)
17. Hermanns, H., Rettelbach, M., Weiss, T.: Formal characterisation of immediate actions in spa with nondeterministic branching. *The Computer Journal* 38(7), 530–541 (1995)
18. Hillston, J.: A Compositional Approach to Performance Modelling. Ph.D. thesis, Cambridge University (1996)
19. Khabbazian, M., Kuhn, F., Kowalski, D.R., Lynch, N.A.: Decomposing broadcast algorithms using abstract mac layers. In: Proc. DIALM-PODC, pp. 13–22. ACM, New York (2010)
20. Kloul, L., Valois, F.: Investigating unfairness scenarios in manet using 802.11b. In: Proc. PE-WASUN, pp. 1–8. ACM, New York (2005)
21. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design* 29, 33–78 (2006)
22. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: Hermanns, H., Segala, R. (eds.) PROBMIV 2002, PAMP-PROBMIV 2002, and PAMP 2002. LNCS, vol. 2399, pp. 169–187. Springer, Heidelberg (2002)
23. Lin, T.: Mobile Ad-hoc Network Routing Protocols: Methodologies and Applications. Ph.D. thesis, Virginia Polytechnic Institute and State University (2004)
24. Oliveira, R., Bernardo, L., Pinto, P.: Modelling delay on IEEE 802.11 MAC protocol for unicast and broadcast nonsaturated traffic. In: Proc. WCNC 2007, pp. 463–467. IEEE, Los Alamitos (2007)
25. Razafindralambo, T., Valois, F.: Performance evaluation of backoff algorithms in 802.11 ad-hoc networks. In: Proc. PE-WASUN 2006, pp. 82–89. ACM, New York (2006)
26. Timmer, M.: Scoop: A tool for symbolic optimisations of probabilistic processes. In: QEST 2011 (to appear, 2011)
27. Vasudevan, S., Kurose, J., Towsley, D.: Design and analysis of a leader election algorithm for mobile ad hoc networks. In: Proc. ICNP 2004, pp. 350–360. IEEE, Los Alamitos (2004)
28. Zuniga, M., Krishnamachari, B.: Analyzing the transitional region in low power wireless links. In: Proc. SECON 2004, pp. 517–526. IEEE, Los Alamitos (2004)

Higher Moment Analysis of a Spatial Stochastic Process Algebra

Marcel C. Guenther and Jeremy T. Bradley

Imperial College London, 180 Queen's Gate,
London SW7 2AZ, United Kingdom
{mcg05, jb}@doc.ic.ac.uk

Abstract. We introduce a spatial stochastic process algebra called MASSPA, which provides a formal behavioural description of Markovian Agent Models, a spatial stochastic modelling framework. We provide a translation to a master equation which governs the underlying transition behaviour. This provides a means of simulation and thus comparison of numerical results with simulation that was previously not available. On the theoretical side, we develop a higher moment analysis to allow quantities such as variance to be produced for spatial stochastic models in performance analysis for the first time. We compare the simulation results against resulting ODEs for both mean and standard deviations of model component counts and finish by analysing a distributed wireless sensor network model.

Keywords: Higher Moment Analysis, Spatial Stochastic Process Algebra, Spatial Modelling, MAM, MASSPA.

1 Introduction

Spatial modelling paradigms take into account localised behaviour of individuals or processes in the evaluation of a system. While some domains such as crowd dynamics modelling [1,2] are inherently linked to their environment, other areas which were traditionally analysed without the notion of space gained new insights from spatial dynamics, e.g. the spatial Lotka–Volterra model [3]. As further examples, spatial topology has a modelling impact in wide variety of application domains such as epidemiology [4], wireless sensor networks [5], fire propagation [6] and traffic modelling [7].

Capturing spatial information in the analysis of models comes at a higher computational cost. In discrete spatial modelling paradigms [4,1,5,8], which allow modellers to create lumped CTMCs with a finite number of locations, the state space explosion is even more severe than in non-spatial CTMCs as we need to keep track of the population size for each state in every location. In the past, spatial models could therefore only be analysed using stochastic simulation [9]. Today fluid approximation [10,11] can handle spatial models even if the population and the number of locations become large. Recent work on discrete space, continuous time models, has focused on the approximation of mean

component counts, i.e. the mean population sizes of different states in different locations [4,1,5]. While approximations of the mean population sizes are important metrics, the evaluation of higher order moments is crucial to get a better understanding of the underlying stochastic process. Furthermore it has been shown that higher order moments can be used to get more accurate boundaries for passage time distributions [12]. In this paper we investigate the computation of higher (joint) moments for discrete spatial models. In particular we formalise the Markovian Agent Model (MAM) paradigm [5] by expressing it in a spatial stochastic process algebra and apply higher moment ODE analysis to MAMs for the first time. Moreover, we use stochastic simulation to verify our ODE approximation for mean and standard deviation of component counts in two examples.

The paper is organised as follows. In Sect. 2 we briefly introduce the MAM paradigm. Subsequently in Sect. 3 we summarise Engblom’s approach [13] for the generation of ODEs for higher order stochastic moments of molecule counts in systems expressed by the chemical *Master Equation* [14]. This serves as the basis for the derivation of higher moment ODEs in MAMs (Sect. 4). In Sect. 5 we define a simple but expressive spatial stochastic process algebra for MAMs and show how to translate it to a set of mass action type reactions. In Sect. 6 we describe a wireless sensor network MAM and estimate its mean and standard deviation using ODEs.

2 Markovian Agent Models

In this section we briefly describe the key concepts of Markovian Agent Models (MAM)s. In [5] Gribaudo *et al.* describe a novel agent-based spatio-temporal model called Markovian Agent Model (MAM). The underlying CTMC of the lumped process is approximated using techniques explained in [15]. Space is assumed to be continuous, but for evaluation purposes it is discretised, for instance into a regular 2-dimensional grid. Agents are assumed to be distributed in space according to a spatial Poisson process. A Markovian Agent (MA) in a MAM is a simple sequential component that can have local transitions which occur at a specified exponential rate and possibly emit messages. Additionally each MA can have message induced transitions. When emitting a message, all neighbours of the emitting agent, i.e. agents that are able to receive the message, may execute an induced transition. The function which defines the notion of neighbourhood is the perception function $u(\cdot)$. In MAMs each agent is assumed to act autonomously, i.e. each agent can decide whether to process or ignore incoming messages. Therefore each induced transition in a MA has a probability of accepting an incoming message. Note that in contrast to other process algebras such as PEPA [16], MAMs have no strict form of synchronisation.

We assume that the density of agents of type $c \in C$ in state $i \in S^c$ at time t at location $v \in V$ is given by $\rho_i^c(t, v)$. Vector $\boldsymbol{\rho}^c(t, v)$ contains all $\rho_i^c(t, v)$, $\forall i \in S^c$

and matrix $\rho(t, v)$ contains all $\rho^c(t, v)$, $\forall c \in C$. If the constant total number of agents of type c in location v is $N^c(v)$ then there are

$$N_i^c(t, v) = \rho_i^c(t, v)N^c(v) \quad (1)$$

agents of type c in state i at location v at time t . Hence, we need to have $\sum \rho_i^c(t, v) = 1$. Moreover, each agent of type c in state j produces messages at a Poisson rate of

$$\beta_j^c = \lambda_j^c g_{jj}^c + \sum_{k \in S^c, k \neq j} q_{jk}^c g_{jk}^c \quad (2)$$

where g_{jk}^c is the number of messages that the agent produces when making a transition from state j to k . λ_j^c denotes the rate at which the agent produces messages while sojourning in state j and q_{jk} is the rate at which the agent moves from state j to k . Now let $u(j, c_j, v_j, i, c_i, v_i)$ denote the perception function that scales the intensity with which an agent of type c_i in state i at location v_i receives messages from another agent of type c_j in state j at location v_j . Generally $u(\cdot) \geq 0$. Two agents can communicate iff $u(\cdot) > 0$ for those agents. The rate at which an agent of type c receives messages in state i computed as follows

$$\gamma_{ii}^c(t, v) = \sum_{c' \in C} \sum_{j \in S^{c'}} \sum_{v_j \in V} u(j, c', v_j, i, c, v) \beta_j^{c'} \rho_j^{c'}(t, v_j) N^{c'}(v_j) \quad (3)$$

Note that γ_{ii}^c is a rate of a convoluted Poisson process $X_1^{c'} + X_2^{c'} + \dots$ where each $X_j^{c'}$ represents a rate $\beta_j^{c'}$ modulated by $u(\cdot)$ and scaled by the population size of the sending agent. This convolution is only accurate if we have a spatially independent population distribution. Finally let

$$K^c(t, v) = Q^c + \Gamma^c(t, v)[A^c - I] \quad (4)$$

be the infinitesimal matrix for the CTMC of agents of type c at time t in location v where Q^c is the time invariant infinitesimal matrix describing the rates of internal state transitions for agents of type c , I the identity matrix and A^c the acceptance matrix where each element a_{ij}^c describes the probability of accepting the message and move from state i to j . We say $a_{ii}^c = 1 - \sum_{j \neq i} a_{ij}^c$ is the probability of ignoring the message and assume $\sum_{j \neq i} a_{ij}^c \leq 1$. Hence each row in A^c sums to 1. Γ^c is the diagonal matrix with entries γ_{ii} and since all rows in $A^c - I$ sum to 0 so do the rows in $\Gamma^c(t, v)[A^c - I]$. Thus $K^c(t, v)$ is a valid infinitesimal matrix for the CTMC describing the evolution of $\rho^c(t, v)$ at time t . Vector $\rho^c(t, v)$, which represents the mean values of the underlying stochastic agent densities for states of agent class c at location v , can be approximated using the following equations

$$\begin{cases} \rho^c(0, v) & t = 0 \\ \frac{\delta \rho^c(t, v)}{\delta t} = \rho^c(t, v) K^c(t, v) & t > 0 \end{cases} \quad (5)$$

Note that Eq. 5 uses the mean field assumption $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ (see [15,17,18]). Further generalisations for MAMs which allow multiple types of messages and location dependent transition rates and dynamic MAMs exist but are omitted here. For more details see [19,7].

3 Higher Moment ODEs for the Master Equation

In [13] Engblom derives a general expression for the derivation of ODEs for higher (joint) moments for models that can be expressed by the *Master Equation* [14]. Similar derivations can be found in [20,21]. In this section we summarise Engblom’s work briefly. Later in Sect. 4 we use his technique to obtain ODEs for any higher (joint) moments in MAMs.

In a model of chemical reactions there are D different species and $|R|$ different reactions. $p(x, t)$ describes the probability distribution of the molecule count vector $x \in \mathbb{Z}^{D+} = \{0, 1, 2, \dots\}^D$ at time t . Note that $X(t)$ is a stochastic process. The propensity rate of a reaction $r \in R$ is described by $w_r : \mathbb{Z}^{D+} \rightarrow \mathbb{R}$. The change in x for reaction $r \in R$ is defined by $x_r = x + n_r \xrightarrow{w_r(x_r)} x$ where n_r are the negated stoichiometric coefficients. The resulting Master Equation is

$$\frac{\delta p(x, t)}{\delta t} = \sum_{\substack{r=1 \\ x+n_r^- \geq 0}}^R \underbrace{w_r(x+n_r)p(x+n_r, t)}_{\text{incoming rate}} - \sum_{\substack{r=1 \\ x-n_r^+ \geq 0}}^R \underbrace{w_r(x)p(x, t)}_{\text{outgoing rate}} \quad (6)$$

where $n_r = n_r^+ + n_r^-$ reflect the decrease and increase in the molecule count for reaction $r \in R$ respectively. In contrast to systems biology literature where reaction r usually changes x to $x + n_r^+$ and $x + n_r^-$, Engblom assumes that reaction r changes x to $x - n_r^+$ and $x - n_r^-$, i.e. $n_r^i = -1$ implies that the number of molecule i increases by 1 as a result of reaction r . Also note that integrating the Master Equation will yield the Chapman–Kolmogorov equation for the underlying CTMC. Engblom further shows that

$$\sum_{x \geq 0} T(x) \frac{\delta p(x, t)}{\delta t} = \sum_{r=1}^R \mathbb{E}[(T(X - n_r) - T(X))w_r(X)] \quad (7)$$

where $T : \mathbb{Z}^{D+} \rightarrow \mathbb{R}$ is a suitable test-function in form of a polynomial and $w_r(x) = 0 \forall x \not\geq n_r^+$. Taking $T(x) = x_i$ for example yields the differential equation for the mean of molecule count x_i

$$\dot{\mathbb{E}}[X_i] = \frac{\delta \mathbb{E}[X_i]}{\delta t} = \frac{\delta \mu_i}{\delta t} = - \sum_{r=1}^R n_r^i \mathbb{E}[w_r(X)] \quad (8)$$

Extending Eq. 7 to higher (joint) moments is straightforward.

4 Higher (joint) Moments in MAMs

To date there is no derivation of ODEs for the computation of higher (joint) moments for component counts in MAMs. In [15,19] the authors only approximate the evolution of the mean of the component densities assuming $\mathbb{E}[XY] \approx \mathbb{E}[X]\mathbb{E}[Y]$. In this section we derive an expression for the second moment for MAM component counts using Engblom's technique outlined in Sect. 3.

To map MAMs to the Master Equation we need to define an equivalent set of reactions along with the propensities and the negated stoichiometric coefficients. For simplicity we assume that a MAM has a single location and one type of agent. In [22] we show that this assumption can be generalised to derive ODEs for more complex MAM extensions. Since MAs are autonomous there is no synchronisation between any two agents. In the language of chemical reactions that implies that any reaction expressing an agent transition from state $i \in S$ to state $j \in S$ has the following form

$$\mathbf{N}(t) + n_r \xrightarrow{N_i(t)K_{ij}(\mathbf{N}(t))} \mathbf{N}(t) \quad (9)$$

with negated stoichiometric coefficient $n_r^i = 1$, $n_r^j = -1$ and $n_r^k = 0 \ \forall k \in S \setminus \{i, j\}$ and propensity rate $w_r = w_{ij}(\mathbf{N}(t)) = N_i(t)K_{ij}(\mathbf{N}(t))$. Note that K_{ij} is indeed the ij^{th} element of the K matrix. In contrast to the definition of K in Eq. 4 the parameters of the matrix have changed from (t, v) to $\mathbf{N}(t)$. Formally the parameters are equivalent, since all factors in Eq. 3 apart from the agent distributions are constant with respect to time and location when considering a single location MAM. We can now create such a reaction for every pair of states $(i, j) \in S \times S$, $i \neq j$ with negated stoichiometric coefficients $n^i = 1$ and $n^j = -1$. Note that all reactions of type (i, j) encapsulate both internal agent transitions and transitions induced by incoming messages as the K matrix combines both internal and induced transition rates. If no transition is possible then $K_{ij}(\mathbf{N}(t)) = 0$. Having translated our simplified MAM into a set of reactions we can use Eq. 7 to derive the ODEs for the mean value, the first joint moment and the second order moment of the component counts. We start with the mean. Using Eq. 8 we get

$$\begin{aligned} \dot{\mathbb{E}}[N_i(t)] &= - \sum_r^R n_r^i \mathbb{E}[w_r(\mathbf{N}(t))] \\ &= \sum_{j \in S, j \neq i} \mathbb{E}[N_j(t)K_{ji}(\mathbf{N}(t))] - \sum_{j \in S, j \neq i} \mathbb{E}[N_i(t)K_{ij}(\mathbf{N}(t))] \end{aligned} \quad (10)$$

since

$$\sum_{j \in S, j \neq i} K_{ij}(\mathbf{N}(t)) = -K_{ii}(\mathbf{N}(t)) \quad (11)$$

Eq. 10 becomes

$$\dot{\mathbb{E}}[N_i(t)] = \sum_{j \in S} \mathbb{E}[N_j(t)K_{ji}(\mathbf{N}(t))] \quad (12)$$

applying the mean field approximation to Eq. 12 yields the component count equivalent to Eq. 5 for a MAM with only one location and one agent class

$$\dot{\mathbb{E}}[N_i(t)] \approx \sum_{j \in S} \mathbb{E}[N_j(t)] K_{ji}(\mathbb{E}[\mathbf{N}(t)]) \quad (13)$$

Similarly using Eq. 7 with $T(\mathbf{N}(t)) = N_i(t)N_j(t)$ we get the following ODE for the first joint moment

$$\begin{aligned} \dot{\mathbb{E}}[N_i(t)N_j(t)] &= \sum_r^R \mathbb{E}[\left((N_i(t) - n_r^i)(N_j(t) - n_r^j) - N_i(t)N_j(t)\right)w_r(\mathbf{N}(t))] \\ &= - \sum_r^R n_r^i \mathbb{E}[N_j(t)w_r(\mathbf{N}(t))] - \sum_r^R n_r^j \mathbb{E}[N_i(t)w_r(\mathbf{N}(t))] \\ &\quad + \sum_r^R n_r^i n_r^j \mathbb{E}[w_r(\mathbf{N}(t))] \end{aligned} \quad (14)$$

which becomes

$$\begin{aligned} \dot{\mathbb{E}}[N_i(t)N_j(t)] &= \mathbb{E}[N_i(t) \sum_{k \in S} N_k(t) K_{kj}(\mathbf{N}(t))] + \mathbb{E}[N_j(t) \sum_{k \in S} N_k(t) K_{ki}(\mathbf{N}(t))] \\ &\quad + \sum_r^R n_r^i n_r^j \mathbb{E}[w_r(\mathbf{N}(t))] \end{aligned} \quad (15)$$

where $\sum_r^R n_r^i n_r^j \mathbb{E}[w_r(\mathbf{N}(t))]$

$$= \begin{cases} -\mathbb{E}[N_i(t)K_{ij}(\mathbf{N}(t))] - \mathbb{E}[N_j(t)K_{ji}(\mathbf{N}(t))] & i \neq j \\ -\mathbb{E}[N_i(t)K_{ii}(\mathbf{N}(t))] + \mathbb{E}[\sum_{j \in S, j \neq i} N_j(t)K_{ji}(\mathbf{N}(t))] & i = j \end{cases} \quad (16)$$

The expansion for $i \neq j$ follows from the observation that the only reactions which have non-zero negated stoichiometric coefficient products $n_r^i n_r^j$ are those resulting from transitions from i to j and from j to i . For $i = j$, however, we can consider all reactions involving i , since all of these will yield $(n_r^i)^2 = 1$. Substituting Eq. 16 into Eq. 15 gives the required definition of the first joint moment and the second order moment in MAMs with one location and one agent class. In [22] we show the corresponding ODEs for MAMs with multiple agents, messages and locations.

5 Markovian Agent Spatial Stochastic Process Algebra

In this section we formally introduce a new spatial stochastic process algebra for Markovian Agent Models (MAM)s which we term MASSPA. First we describe a

process algebra that allows us to define Markovian Agents. This process algebra is a blend between π -calculus [23] and a purely sequential version of PEPA [16], with passive actions and action names removed. To describe the spatial dynamics of MAMs we subsequently define the space, the distribution of agents in space and the perception function $u(\cdot)$. The notation for the spatial aspects of the model is similar to the one presented in [8,24]. Having defined MASSPA we give a simple example (Sect. 5.1) and show how MASSPA can be translated into chemical reactions (Sect. 5.2). The grammar of MASSPA is as follows:

$$\begin{array}{l} S ::= \alpha.S \mid S + S \mid ?(m,p).S \mid \alpha!(m,g).S \mid C_S \mid \emptyset \\ P ::= P \underset{u(\cdot)}{\boxtimes} P \end{array}$$

where S denotes a sequential MA, C_S a sequential constant and \emptyset the nil process. The basic MASSPA operators can be interpreted as follows:

Prefix: $\alpha.S'$ describes the possibility of a transition from the current process to process S' . This transition happens at rate α .

Choice: At a given time a process defined as $S + T$ can either behave as S or as T .

Constant: Assign names to patterns of behaviour associated with components, e.g. $N \stackrel{def}{=} S$. In this case N behaves the exact same way as S .

Message Sending: $\alpha!(m,g).S'$ describes a transition from the current process S to process S' where on average $g \in \mathbb{R}^{>0}$ messages of type m are sent as part of this transition. In fact g represents the parameter of a Poisson distribution, which defines the random number of messages generated as part of the transition.

Message Reception: $?(m,p).S'$ describes the possibility of a transition from the current process to process S' that can be induced by an incoming messages of type m . p is the probability of accepting such a message. For each type of message $m \in M$ that a process S listens to, we have to have $0 \leq \sum_{(m,p) \in S} p \leq 1$, i.e. each process can consume at most one type of message at a time.

Parallel: $P \underset{u(\cdot)}{\boxtimes} Q$ means that agent populations (see definition below) P and Q , possibly located in different locations, operate in parallel. The perception function $u(\cdot)$ governs the message exchange between P and Q . This operator is only used to specify our operational semantics. In general we assume all agents in all locations act in parallel under $u(\cdot)$.

We now look at the definition of space, the agent populations and the perception function $u(\cdot)$. We begin with the space. Generally the space can take any discretised form as long as it is finite. Two basic examples are finite 2/3-dimensional regular rectangular and radial grids where each cell/location in the grid has a unique label $l \in L$, e.g. $L = \{A, B, C\}$ or $L = \{(0,0), (0,1), \dots, (x,z)\}$. Within a certain cell we assume a spatial Poisson distribution of agents. In practice we

may simply fix the number of agents in a certain state in a given location l and argue that we can find a corresponding rate for a spatial Poisson process that would on average generate that many agents in l . Assume that we have defined the sequential agents in MASSPA and that C is the set of distinct agent types, i.e. agents that do not share derivative states. Let S_i be the set of all derivative states for a sequential MASSPA agent of type $i \in C$ (cf. Sect. 2), such that $S_i \cap S_j = \emptyset, \forall i, j \in C, i \neq j$. Furthermore let $S = \bigcup_{i \in C} S_i$ be the set of all agent states. We define the set of agent populations for agents of type $i \in C$ as $P_i = \{s@l : s \in S_i, l \in L\}$ and also $P = \bigcup_{i \in C} P_i$. The initial agent population distribution is defined by a mapping $d : P \rightarrow \mathbb{N}^{\geq 0}$. Finally we define $u : Ch \rightarrow \mathbb{R}^{\geq 0}$ where $Ch \subseteq P \times P \times M$ is the set of channels and M the set of all message labels. We say that population $c_1@l_1$ can send messages of type m to $c_2@l_2$ iff $u(\cdot)$ is defined for $(c_1@l_1, c_2@l_2, m) \in Ch$.

The corresponding structured operational semantics are described in Fig. 1. It provides a translation to a labelled transition system which consists of transitions $a \in \mathcal{L}_\alpha \cup \mathcal{L}_M$ where \mathcal{L}_α is the set of exponentially delayed transitions and \mathcal{L}_M , consisting of (α, m, g_m) -sending events and (α, m, p_m) -reception events, is the set of message transitions. Assume $l_1, l_2 \in L$, possibly such that $l_1 = l_2$. $F \xrightarrow{(\alpha, m, g_m)} F$ means F sends g_m messages of type m at rate α and $E@l_1 \xrightarrow[u(\cdot)]{\alpha} F@l_2 \xrightarrow[(x * \alpha * g_m, m, p_m)]{u(\cdot)} E'@l_1 \boxtimes F@l_2$ states that $E@l_1$ goes to $E'@l_1$ at rate $x * \alpha * g_m$ with probability p_m as the result of receiving message(s) of type m .

Prefix

$$\frac{}{E \xrightarrow{\alpha} E'} (E \stackrel{def}{=} \alpha.E') \quad \frac{}{F \xrightarrow{\alpha} F'} (F \stackrel{def}{=} \alpha!(m, g_m).F') \quad \frac{}{F \xrightarrow{(\alpha, m, g_m)} F} (F \stackrel{def}{=} \alpha!(m, g_m).F')$$

Competitive Choice

$$\frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'} \quad \frac{F \xrightarrow{\alpha} F'}{E + F \xrightarrow{\alpha} F'}$$

Parallel

$$\frac{E \xrightarrow{\alpha} E'}{E@l_1 \boxtimes_{u(\cdot)} F@l_2 \xrightarrow{\alpha} E'@l_1 \boxtimes_{u(\cdot)} F@l_2} \quad \frac{F \xrightarrow{\alpha} F'}{E@l_1 \boxtimes_{u(\cdot)} F@l_2 \xrightarrow{\alpha} E@l_1 \boxtimes_{u(\cdot)} F'@l_2}$$

Message Exchange

$$\frac{F \xrightarrow{(\alpha, m, g_m)} F}{E@l_1 \boxtimes_{u(\cdot)} F@l_2 \xrightarrow[(x * \alpha * g_m, m, p_m)]{u(\cdot)} E'@l_1 \boxtimes_{u(\cdot)} F@l_2} (E \stackrel{def}{=} ?(m, p_m).E' + \dots, u(F@l_2, E@l_1, m) = x)$$

Constant

$$\frac{E \xrightarrow{\alpha} E'}{A \xrightarrow{\alpha} E'} (A \stackrel{def}{=} E)$$

Fig. 1. Operational semantics of MASSPA

5.1 A Simple MASSPA Example

In the following we give a simple MASSPA example. In Sect. 6 we look at more complex model. The *simple MAM* (see Fig. 2) has locations $L = \{A, B, C\}$ containing populations of identical agents so that set of all states $S = \{on, off\}$. The agent populations are $P = \{on@A, off@A, on@B, off@B, on@C, off@C\}$.

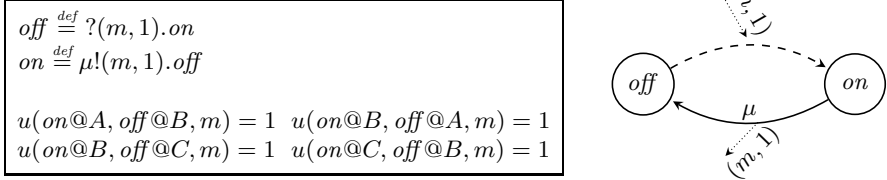


Fig. 2. A simple MAM with 3 locations

5.2 Translating MASSPA into Mass Action Type Reactions

In this section we discuss how any MASSPA model can be translated into a system of chemical reactions (cf. Sect. 3). It is easy to see that the number of different agent populations $|P|$ is equivalent to the number of “molecule” species D . Hence, P is the set of available species. In the simple example above there is only one agent type, i.e. $|C| = 1$. As agents do not synchronise, each reaction has the following form $r : a@l \xrightarrow{w_r} b@l$, $a@l, b@l \in P_i$, $i \in C$. This is a different way of stating the assumption that all reactions for MAMs describe the evolution for a specific type of agent in one location (cf. [22]). Clearly the negated stoichiometric coefficients are 1, -1 for $a@l, b@l$ respectively. To derive the reactions we need to define the w_r terms. By Eq. 9 we have $w_r = w_{a@l b@l} = N_{a@l}(t)K_{a@l b@l}^i(\mathbf{N}(t)) = N_a(t, l)K_{ab}^i(t, l)$, $a, b \in S_i$, $i \in C$, $l \in L$. $N_{a@l}(t)$ is the number of agents of type i in state a at location l . To derive $K_{a@l b@l}^i(\mathbf{N}(t))$ we need to sum the local transition rate from $a@l$ to $b@l$ and the corresponding transition rate induced by incoming messages. The local rate of transitions from $a@l$ to $b@l$ can be obtained directly from the MASSPA agent definition, as this rate is location independent. Let states a, b be derived from process definitions A, B where $A \stackrel{def}{=} \dots + \alpha.B + \dots + \beta!(m, g_m).B$. To obtain the local transition rate we sum up all α and β rates for all prefix operations $\alpha.B$ and sending operations $\beta!(\cdot, \cdot).B$ in the definition of process A . The resulting rate is the ab^{th} element of matrix Q^i (cf. Eq. 4). Now we describe the rate of message induced transitions from $a@l$ to $b@l$. For any term $?(m, p_m).B$ in the definition of A we first need to find all channels $(x@l_1, a@l, m) \in Ch$ for which $u(\cdot)$ is defined. Note that $x \in S_j$ and potentially $a, b \notin S_j$. Let M_a be the set of all messages that can trigger process A to

become process B and Ch_a the channels of type $(x@l_1, a@l, m)$ for which $u(\cdot)$ is defined. The total rate of message induced transitions from population $a@l$ to $b@l$ is

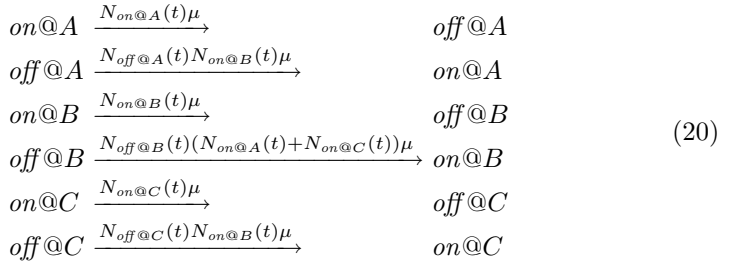
$$\sum_{m \in M_a} p_m(a, b) \sum_{(x@l_1, a@l, m) \in Ch_a} u(x@l_1, a@l, m) * \beta(x, m) * N_{x@l_1}(t) \quad (17)$$

where $p_m(a, b)$ is probability of an agent in state a to accept a message of type m and transit to state b and $\beta(x, m)$ is the total sending rate of messages m of an agent in state x . The former is the sum of all probabilities p_m for message $m \in M$ that are used in the $?(m, p_m).B$ terms in A . The latter is the sum of all rates $\beta * g_m$ from the $\beta!(m, g_m).X'$ terms in the definition of process X that state x refers to. Hence Eq. 17 is the same as the ab^{th} element of $\Gamma^i(t, l)$ in Eq. 4. Therefore MASSPA models do indeed capture the dynamics of static MAMs described in Sect. 2. To give a practical example we now show the reactions for the simple MAM described in Sect. 5.1. First we derive the matrices $K(t, v)$ for each of the three locations.

$$K(t, A) = K(t, C) = \begin{pmatrix} -\mu & \mu \\ N_{on@B}(t)\mu & -N_{on@B}(t)\mu \end{pmatrix} \quad (18)$$

$$K(t, B) = \begin{pmatrix} -\mu & \mu \\ (N_{on@A}(t) + N_{on@C}(t))\mu & -(N_{on@A}(t) + N_{on@C}(t))\mu \end{pmatrix} \quad (19)$$

The top left element in the rate matrix $K(t, X)$ is $K_{on@X \ on@X}$, the top right element $K_{on@X \ off@X}$, the bottom left element $K_{off@X \ on@X}$ and the bottom right element $K_{off@X \ off@X}$. The reactions for the model are



It is important to note that sending agent populations (e.g. $on@B$ in the second reaction) act as catalysts but do not change state as they send a message as there is no synchronisation in MAMs. Hence messages are only ever sent while the sender sojourns in its current state.

Figure 3 shows numerical results for the simple MAM. We found that for small populations the mean approximation deteriorates as time goes on. In this model this is partially due to the fact that simulations for small agent populations tend to a state where no more communication is possible, i.e. $on@A = on@B = on@C = 0$. It is easy to show that this another fixed point for the ODEs, too.

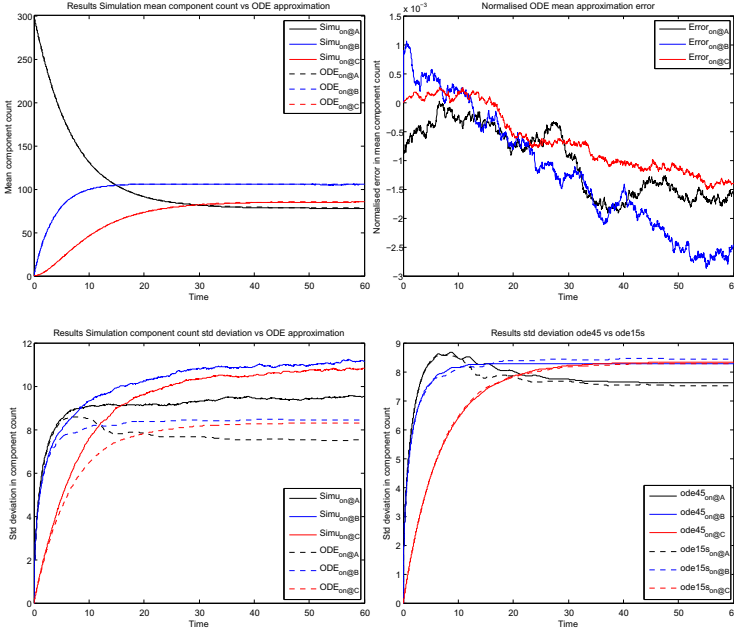


Fig. 3. Numerical results $on@A$, $on@B$, $on@C$ for the simple MAM with initial values 300, 0, 0, 300, 0, 450 for $on@A$, $off@A$, $on@B$, $off@B$, $on@C$, $off@C$ and $\mu = 0.1$. Rates of reactions derived from message induced transitions were divided by the population size of the receiving agent type (see [22]). The error was computed by subtracting the ODE approximation from 10,000 averaged simulation traces and dividing by the population size of the corresponding agent. The first three graphs contain the ODE approximation solved using *ode15s*.

However, the normalised error in the mean became smaller as we increased the population size, which agrees with theoretical mean field results [15,17]. The standard deviation approximation for the simple MAM is quantitatively inaccurate, but qualitatively good as it preserves the relative difference between the standard deviation of $on@A$, $on@B$ and $on@C$ around $t = 60$. More interestingly the comparison between the explicit *ode45* solver and the implicit *ode15s* solver in Matlab shows that the ODEs which determine the second order moments also have multiple fixed points. In this example *ode45* gives quantitatively and qualitatively worse results than *ode15s*.

6 Worked Example: A Simplified Spatial WSN

Our worked example is a simplified version of the Wireless Sensor Network presented in [5]. In this MAM there are wireless sensor nodes (WSN)s which sample

some quantity of their environment, e.g. temperature or humidity, and forward the samples to a sink location via a number of intermediate WSNs. The wireless sensor node is defined in Figs. 4. Each agent samples its environment at rate λ and propagates the measurement to the next agent that is closer to the sink. If it receives a sample from a WSN that is further away from the sink, it first buffers it and then sends it on to the next link at rate μ . Furthermore nodes may go to sleep when they have no buffered messages in order to save energy. WSNs may also fail without chance of recovery.

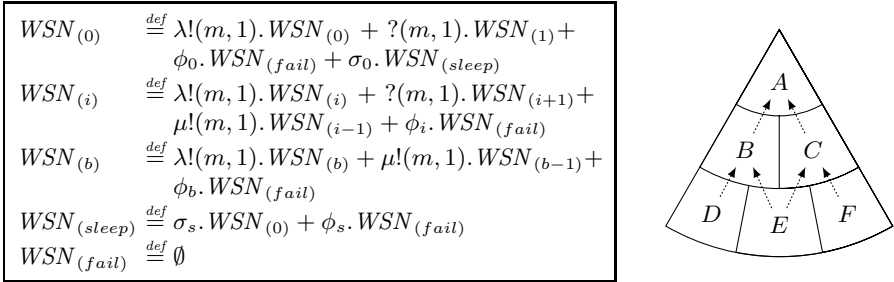


Fig. 4. A simple WSN in MASSPA. b denotes the buffer size. Arrows in the spatial layout on the right define $u(\cdot) = 1$ for locations $L = \{A, B, C, D, E, F\}$. The set of states is $S = \{0, 1, \dots, b, \text{sleep}, \text{fail}\}$.

We now compare the ODE approximations for the mean and standard deviation for the WSN MAM with the exact solution. The ODEs were solved in Matlab using *ode45* and *ode15s*. We experimented with various moment closures for the third order terms in the second order moment ODEs (see [22]). For the ODE traces below we used the $\mathbb{E}[XYZ] \approx \mathbb{E}[XY]\mathbb{E}[Z]$. For stochastic simulation we used Matlab’s *simbiology* tool.

As can be seen in Fig. 5 the mean is approximated well by the ODEs. This holds for all quantities not only for the mean of the component counts in location B shown in Fig. 5. Furthermore we did not observe multiple fixed points for the first order ODEs, which could be due to the fact that the communication will never stop completely as WSNs always sample at rate λ . The more interesting observation made in this model is the behaviour of the ODEs for the second order moments. While the ODEs for WSN components counts in locations A, C, D, E and F were remarkably accurate irrespective of the choice of ODE solver, we found that in location B there was a significant difference in the solution when applying *ode15s* as opposed to *ode45* in Matlab. In Fig. 5d *ode45* is more stable than *ode15s*. The two traces eventually converge around time $t = 2000$ so there is no indication that there are multiple fixed points in the second order ODEs.

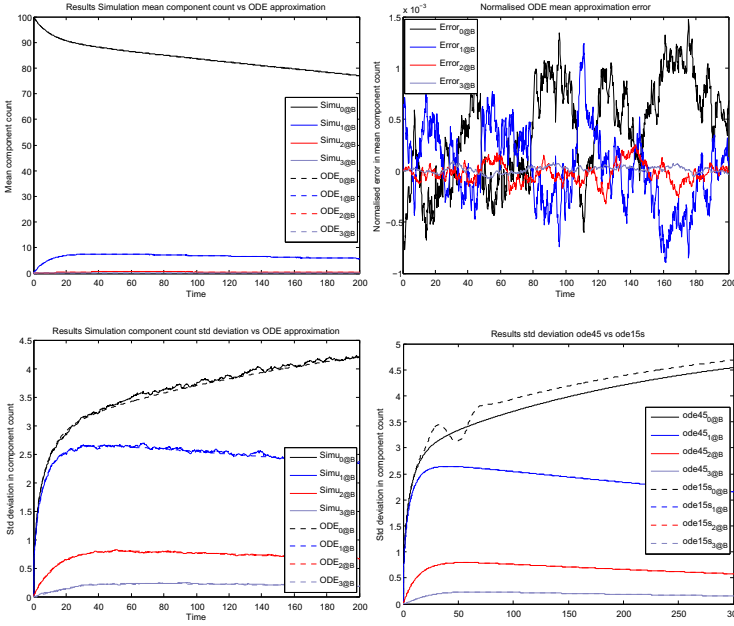


Fig. 5. Numerical results $0@B, 1@B, 2@B, 3@B$ for the WSN MAM with initial values 20, 100, 50, 20, 10, 90 for $0@A, 0@B, 0@C, 0@D, 0@E, 0@F$ and $b = 3, \mu = 0.1, \lambda = 0.03, o_s = 0.0001, o_0 = 0.001$ and zero probability of node failure. Message induced re-actions rates were divided by the population size of the receiving agent type (see [22]). The error was computed by subtracting the ODE approximation from 3,500 averaged simulation traces and dividing by the population size of the corresponding agent. The first three graphs contain the ODE approximations solved using ode45.

7 Conclusion

In this paper we have derived a simple mapping from MAMs to the language of chemical reactions. This mapping is then used to derive second order ODEs for MAMs for the first time. To formalise the description of MAMs further we defined MASSPA, a process algebra for MAMs and showed how MASSPA can be translated into mass action type reactions. Moreover, we have given numerical examples for two models that we defined in MASSPA and shown that their first order ODE approximations are generally good for large populations. As for the second order ODE approximation we got very good results in the WSN model and less accurate results in the simple MAM. We also observed that comparing stochastic simulation traces to the ODE traces for small population sizes gives a good indication as to whether the second order ODEs become more accurate as we increase the overall population size. The $\mathbb{E}[XYZ] \approx \mathbb{E}[XY]\mathbb{E}[Z]$ moment closure assumption proved to be a simple but effective choice. The only closure which gave slightly better results for the simple MAM model was $\mathbb{E}[XYZ] \approx$

$\mathbb{E}[X]\mathbb{E}[YZ]$, but for the WSN model this closure gave rather inaccurate second order approximations. Although the closure behaviour could be entirely model dependent it is not unlikely that $\mathbb{E}[XYZ] \approx \mathbb{E}[XY]\mathbb{E}[Z]$ works well for MAMs as the Z term always represents contributions from the K matrix. Further research is needed to investigate moment closures for MAMs and to find indicators for well behaved spatially motivated moment closures similar to those discussed in [18]. Should good moment closures for MAMs be hard to determine, we might need to look for further evaluation techniques such as those discussed in [25]. Having computed higher moments for ODEs it would be interesting to look at resulting passage time bounds that can be deduced using techniques described in [12].

Acknowledgements. We would like to thank Davide Cerotti and Marco Gribaudo for giving us valuable insights into the MAM paradigm. Moreover, we would like to thank Anton Stefanek and Richard Hayden for their input on moment closures techniques. Finally we would also like to thank the referees for their time and efforts in reviewing the original submission and their helpful comments which have certainly improved the paper.

References

1. Massink, M., Latella, D., Bracciali, A., Hillston, J.: Modelling Crowd Dynamics in Bio-PEPA. In: Proceedings of the 9th Workshop PASTA/Bio-PASTA, pp. 1–11 (2010)
2. Bracciali, A., Hillston, J., Latella, D., Massink, M.: Reconciling Population and Agent Models for Crowd Dynamics. In: 3rd International Workshop on Logics, Agents, and Mobility, LAM 2010 (2010)
3. Dieckmann, U., Law, R.: Relaxation projections and the method of moments. Cambridge studies in adaptive dynamics, ch. 21, pp. 412–455. Cambridge University Press, Cambridge (2000)
4. Stefanek, A., Vigliotti, M., Bradley, J.T.: Spatial extension of stochastic pi calculus. In: 8th Workshop on Process Algebra and Stochastically Timed Activities, pp. 109–117 (2009)
5. Gribaudo, M., Cerotti, D., Bobbio, A.: Analysis of on-off policies in sensor networks using interacting markovian agents. In: 6th IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 300–305 (2008)
6. Cerotti, D., Gribaudo, M., Bobbio, A., Calafate, C.T., Manzoni, P.: A markovian agent model for fire propagation in outdoor environments. In: Aldini, A., Bernardo, M., Bononi, L., Cortellessa, V. (eds.) EPEW 2010. LNCS, vol. 6342, pp. 131–146. Springer, Heidelberg (2010)
7. Cerotti, D., Gribaudo, M., Bobbio, A.: Presenting Dynamic Markovian Agents with a road tunnel application. In: IEEE International Symposium on Modeling Analysis Simulation of Computer and Telecommunication Systems MASCOTS, pp. 1–4. IEEE, Los Alamitos (2009)
8. Galpin, V.: Towards a spatial stochastic process algebra. In: Proceedings of the 7th Workshop on Process Algebra and Stochastically Timed Activities (PASTA), Edinburgh (2008)
9. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)

10. Hillston, J.: Fluid flow approximation of PEPA models. In: Second International Conference on the Quantitative Evaluation of Systems QEST 2005, pp. 33–42 (2005)
11. Hayden, R.A., Bradley, J.T.: A fluid analysis framework for a Markovian process algebra. *Theoretical Computer Science* 411(22-24), 2260–2297 (2010)
12. Hayden, R.A., Stefanek, A., Bradley, J.T.: Fluid computation of passage time distributions in large Markov models. *Theoretical Computer Science* (submitted, 2010)
13. Engblom, S.: Computing the moments of high dimensional solutions of the master equation. *Applied Mathematics and Computation* 180(2), 498–515 (2006)
14. Van Kampen, N.G.: *Stochastic Processes in Physics and Chemistry*. North-Holland personal library, vol. 11. North-Holland, Amsterdam (1992)
15. Bobbio, A., Gribaudo, M., Telek, M.: Mean Field Methods in Performance Analysis. In: Fifth International Conference on Quantitative Evaluation of Systems, QEST, 2008, pp. 215–224 (2008)
16. Hillston, J.: *A Compositional Approach to Performance Modelling*, p. 158. Cambridge University Press, Cambridge (1996)
17. Le Boudec, J.-Y., McDonald, D., Mundinger, J.: A Generic Mean Field Convergence Result for Systems of Interacting Objects. In: Fourth International Conference on the Quantitative Evaluation of Systems QEST 2007, pp. 3–18 (2007)
18. Murrell, D.J., Dieckmann, U., Law, R.: On moment closures for population dynamics in continuous space. *Journal of Theoretical Biology* 229(3), 421–432 (2004)
19. Cerotti, D.: *Interacting Markovian Agents*. PhD thesis, University of Torino (2010)
20. Bortolussi, L.: On the Approximation of Stochastic Concurrent Constraint Programming by Master Equation. *Electronic Notes in Theoretical Computer Science* 220(3), 163–180 (2008)
21. Gillespie, C.S.: Moment-closure approximations for mass-action models. *IET Systems Biology* 3(1), 52–58 (2009)
22. Guenther, M.C., Bradley, J.T.: Higher moment analysis of a spatial stochastic process algebra. Tech. rep., Imperial College London (July 2011), <http://pubs.doc.ic.ac.uk/masspa-higher-moments/>
23. Milner, R.: *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, Cambridge (1999)
24. Galpin, V.: Modelling network performance with a spatial stochastic process algebra. In: Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications (AINA 2009), Bradford, pp. 41–49 (May 2009)
25. Ovaskainen, O., Cornell, S.J.: Space and stochasticity in population dynamics. *Proceedings of the National Academy of Sciences of the United States of America* 103(34), 12781–12786 (2006)

Optimization for Multi-thread Data-Flow Software

Helmut Hlavacs and Michael Nussbaumer

University of Vienna, Research Group Entertainment Computing,
Lenaugasse 2/8, 1080 Vienna, Austria
`{helmut.hlavacs,m.nussbaumer}@univie.ac.at`

Abstract. This work presents an optimization tool that finds the optimal number of threads for multi-thread data-flow software. Threads are assumed to encapsulate parallel executable key functionalities, are connected through finite capacity queues, and require certain hardware resources. We show how a combination of measurement and calculation, based on queueing theory, leads to an algorithm that recursively determines the best combination of threads, i.e. the best configuration of the multi-thread data-flow software on a given host. The algorithm proceeds on the directed graph of a queueing network that models this software. Experiments on different machines verify our optimization approach.

Keywords: Software Optimization, Performance Optimization, Multi-thread Software.

1 Introduction

The trend to many cores inside CPUs enables software engineering towards concurrency. Software is split up in atomic actions that can run in parallel. This may considerably speed up computation, but also causes extra overhead through thread coordination for both, the operation system and also the software engineer. Developing software made of several threads is much more complicated than creating an old-fashioned single thread software. One fundamental problem of the developer might be to find the right strategy for determining the optimal number of threads.

In this work we use the term host for a server hardware platform with a fixed number of cores. Nodes are specified as tasks of the queueing network software and build the data-flow graph. A thread executes node tasks. Each node can have multiple threads that are executing the nodes tasks in parallel. The overall situation is that we want to find the optimal number of threads per node, with the side constraint, that there can only be as many threads as cores available on the host, because each thread runs on a dedicated core.

This paper discusses an approach to use a combination of analytical modeling techniques and measurements to find an optimal configuration of threads on a given host by iteratively increasing the number of threads. The application area

is software following the data-flow paradigm, in our case a commercial tool that computes and persists call data from a telecom network.

Section 2 of this paper presents related work. In Section 3 an example multi-thread data-flow software is introduced, an analytical model is introduced, and theoretical optimization is explained. Section 4 presents our optimization approaches to find the optimal configuration for multi-thread data-flow software. Finally, Section 5 presents experiments conducted on two SUN machines with a real queueing network software, and compares them to a solely analytical approach.

2 Related Work

In the past there have been several approaches for adapting software to various multi-computers, in order to optimize the performance.

FFTW [1] is a free software library that computes the *Discrete Fourier Transform* (DFT) and its various special cases. It uses a planner to adapt its algorithms to the hardware in order to maximize performance. The FFTW planner works by measuring the actual run time of many different plans and by selecting the fastest one.

The project *SPIRAL* [2] aims at automatically generating high performance code for linear *Digital Signal Processing* (DSP) transforms, tuned to a given platform. SPIRAL implements a feedback-driven optimizer that intelligently generates and explores algorithmic and implementation choices to find the best match to the computer's micro architecture.

The *Automatically Tuned Linear Algebra Software* (ATLAS) [3] aims at automatically generating code that provides the best performance for matrix multiplication on a given platform by finding the cache-optimal variant.

In [4] an algorithm is developed for finding the nearly best configuration for a Web system. Up to 500 emulated clients generate traffic for various trading operations like *buy* or *sell*. Optimal configurations are found iteratively, depending on the number of threads and the cache size.

The technology *Grand Central Dispatch*¹ (GCD) by Apple enables to use multicore processors more easily. Firstly released in the Mac OS X 10.6, GCD is implemented by the library *libdispatch*². GCD is a scheduler for tasks organized in a queuing system and acts like a thread manager that queues and schedules tasks for parallel execution on processor cores.

In [5] a queueing network model with finite/single capacity queues and blocking after service discipline is used to model software architectures; more exactly the synchronization constraints and synchronous communication between software components. The information flow (trace) between components is analyzed to identify the kind of communication (fork, join) and reveal the interaction pairs among components that enables to model a queueing network. This way a performance model for a specific software architecture can be derived.

¹ <http://www.apple.com/macosx/technology/#grandcentral>

² <http://libdispatch.macosforge.org>

3 Analytical Model

The system under investigation is a commercial product called Data Flow Engine (DFE) for processing and persisting call data stemming from standard telecom networks. In an industrial cooperation, our task was to find automatic ways for analyzing the performance of the software, and adapt the software to different hardware platforms and workload situations. The software in some sense should utilize the hardware in some optimized way, for example for maximizing the throughput, but also utilize only as much hardware as needed, for example for optimizing the energy consumption if this is supported by the platform.

Data describing call state changes like calling, canceling etc. is represented by tickets or packets being sent from the telecom infrastructure to the software that subsequently receives, extracts, converts, and stores the incoming packets. The software itself follows a data-flow approach, organizing its tasks into a network of processing nodes (Fig. 1). Queues buffer the output for succeeding nodes and incoming tickets pass the graph and visit each node once. Since these nodes technically are lightweight processes (threads), they can be replicated for splitting the load. Nodes model atomic actions, but several instances of a node can exist as threads.

Currently defined nodes are:

- The *Decoder* node takes packets from its queue, extracts the data, and forwards it to the next node’s queue. Hence, the extraction can be done in parallel by several nodes. A *Decoder* can be replicated, each *Decoder* forwards the extracted data to the same queue.
- The *Converter* node takes extracted data from its queue, converts it into a format appropriate for storing and forwards it to a *Serializer* and a *Feeder* node. Thus, the extracted and formatted data is persisted always twice. Also conversion can be done in parallel by several threads of the *Converter*.
- The *Serializer* node takes data from its queue and stores it to disk. *Serializers* may write in parallel, even to different disks.
- The *Feeder* node takes data from its queue and sends it to a database. Since the database is assumed to be capable to provide a connection pool, *Feeders* may send in parallel.

All nodes require certain CPU time, memory, disk space and disk bandwidth, and network bandwidth. Thus, the benefit of replicating nodes is limited by the available hardware. Starting with an initial configuration with one thread per node, the goal is to find the optimal configuration of threads for a certain host.

The main idea is to sequentially add new threads for over-utilized nodes until an optimization goal is reached. For that reason, a queueing network, as base for an analytical model [6–9] for describing the multi-thread software, is shown in Fig. 1 as an open queueing network consisting of 4 queues. Open in the sense that jobs come from an external source, are serviced by an arbitrary number of servers inside the network and eventually leave the network. Further, the

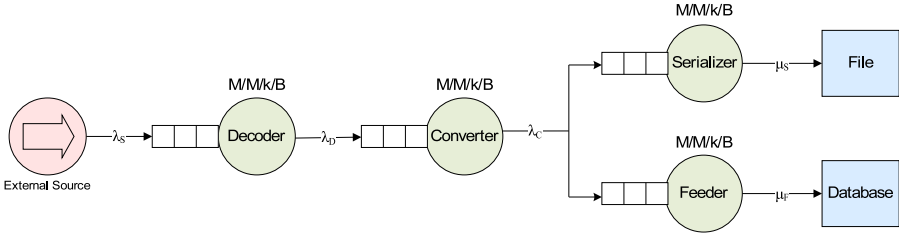


Fig. 1. Multi-thread software modeled as queuing network

suggested queuing network is aperiodic, because no job visits a server twice, the flow goes to one direction. The queuing discipline is always *First-Come, First-Served* (FCFS).

An external source sends events with rate λ_S to the M/M/k/B *Decoder* queue whereas the arrival process (M/././.) is Markovian and follows a Poisson process [10] with independent identically distributed (iid) and exponentially interarrival times $1/\lambda$, which postulates that the next arrival at $t + 1$ is completely independent from the arrival at t . The use of Poisson arrival is motivated by the fact that the workload source consists of possibly millions of independent sources (callers) who create call data independently from each other.

The service time $1/\mu$ of each server k is independent from the arrival process and iid and exponentially distributed (memoryless) with parameter μ and therefore the departure process (./M/./.) is also Markovian. The queue capacity (buffers) is defined by parameter B . An arrival reaching a full queue is blocked. This can be avoided by increasing the queue size B , decreasing the arrival rate λ , or increasing the number of servers k and so increasing the joint service rate μ . This holds also for the *Converter*, *Feeder*, and *Serializer* queues.

After the data is extracted by the *Decoder*, it is forwarded with rate λ_D to the *Converter* queue. The *Converter* then converts the data to the file and database formats and forwards it to the corresponding queues with rate λ_C . The *Feeder* sends the data to the database with rate μ_F whereas the *Serializer* writes data to disk with rate μ_S .

For each node in the graph, there is a race between λ and μ in the sense that under the assumption $\lambda \leq \mu$ for the utilization ρ and given number of servers (i.e., in our case threads) k in that node, the following must hold:

$$\rho = \frac{\lambda}{k\mu} \leq 1 \quad , \quad (1)$$

that leads to a stable node: all jobs in the node's queue can eventually be worked out. Due to different nodes, the bottleneck is the node where $\lambda > \mu$. For finding the best configuration of the queuing network (see Fig. 1), the following performance measures [6–9] are considered for the nodes *Decoder*, *Converter*, *Serializer* and *Feeder*.

The utilization ρ from Equ. (1) of a particular node is the base for most other measures. The probability P_n of n jobs in the node is given by

$$P_n = \frac{(k\rho)^n}{n!} P_0 \quad \text{for } 0 \leq n < k \tag{2}$$

$$P_n = \frac{k^k \rho^n k!}{P} \quad \text{for } k \leq n \leq B \text{ and } B \geq k \tag{3}$$

where k is the number of servers, B the number of buffers (slots in the queue that can be set to a sufficient large number), and P_0 the probability of no jobs in the node, which for $k = 1$ is

$$P_0 = \frac{1 - \rho}{1 - \rho^{B+1}} \quad \text{for } \rho \neq 1 \tag{4}$$

$$P_0 = \frac{1}{B + 1} \quad \text{for } \rho = 1 \tag{5}$$

and for $k > 1$

$$P_0 = \left(1 + \frac{(1 - \rho)^{B-k+1} (k\rho)^k}{k!(1 - \rho)} + \sum_{n=1}^{k-1} \frac{(k\rho)^n}{n!} \right)^{-1} . \tag{6}$$

The expected number of jobs E_s in a node, for $k = 1$ is

$$E_s = \frac{\rho}{1 - \rho} - \frac{(B + 1)\rho^{B+1}}{1 - \rho^{B+1}} \tag{7}$$

and for $k > 1$

$$E_s = \sum_{n=1}^B n p_n \tag{8}$$

where the expected number of jobs in a queue E_q for $k = 1$ is

$$E_q = \frac{\rho}{1 - \rho} - \rho \frac{1 + B\rho^B}{1 - \rho^{B+1}} \tag{9}$$

and for $k > 1$

$$E_q = \sum_{n=k+1}^B (n - k) p_n \tag{10}$$

Since we have queues with finite buffers B , some traffic is blocked. The initial traffic that reaches the queueing network is not equal to the traffic that passes through the queueing network. Reduced to a single node this means that λ depends on a certain blocking probability P_b . This leads to the effective arrival rate λ'

$$\lambda' = \lambda(1 - P_b)$$

where the blocking probability $P_b = P_B$, i.e. the probability of B jobs in a node. The loss rate ϵ is

$$\epsilon = \lambda P_B \quad (11)$$

and the effective utilization ρ' is

$$\rho' = \frac{\lambda'}{k\mu}$$

again with k servers. Next, the mean response time R of a node is

$$R = \frac{E_s}{\lambda'} \quad (12)$$

and the mean waiting time W of a job in the queue is

$$W = \frac{E_q}{\lambda'} \quad (13)$$

Finally, the probability that the buffers of a node are all occupied is denoted by

$$P_k = \frac{\frac{(k\rho)^k}{k!}}{\sum_{j=0}^k \frac{(k\rho)^j}{j!}} \quad (14)$$

Configurations of the queuing network can be evaluated according to these measures. An optimization algorithm, presented in Section 4.1, determines the best configuration.

4 Optimization Approaches

The focus of our work is to find the optimal configuration of a multi-thread data-flow software for a specific host. A configuration is specified as a vector of n tuples with

$$(k_1 - \dots - k_i - \dots - k_n) \quad (15)$$

where k_i denotes the number of threads of node i . With the constraint of

$$\sum_{i=1}^n k_i \leq c \quad (16)$$

where c is the number of available (virtual) cores on a specific host. E.g.: The initial configuration of *Decoder*, *Converter*, *Serializer* and *Feeder* nodes (1-1-1-1) contains one thread per node. Optimization is done recursively by adding threads to over-utilized nodes with two optimization goals:

- Consolidation. With optimization towards consolidation a desired arrival rate is given and the tool determines the minimum number of threads required to ensure that all nodes are below a predefined utilization threshold. With a constant external arrival rate only the number of threads of an over-utilized node with utilization $\geq \text{lim}U$ ($0 < \text{lim}U < 1$) will be incremented for splitting load among available cores as evenly as possible.
- Throughput. With optimization towards throughput we start with the lowest arrival rate of one job per second and the tool increases the arrival rate step-wise until one node exceeds the predefined utilization threshold. Then the number of the corresponding threads is increased as long as the utilization is below the threshold. Again, the arrival rate will be increased and optimization goes on as long as no hardware limit is reached. With $0 < \text{ext}ARInc < 1$ the external arrival rate is increased for each new configuration by $\text{ext}ARInc$ as long as the maximum utilized node does not reach $\text{lim}U$. For the maximum possible number of threads the highest possible throughput is determined.

For calculating the best configuration of nodes, based on a host's resources (for example given by Table 1) the number of CPU cores is the upper bound for the number of nodes that can run in parallel. It is assumed that each node is executed as single thread on a dedicated core and since core sharing is ignored for now, as many nodes as free cores available are possible. Memory, disk space and speed, and network bandwidth are shared by all nodes and are the constraints for optimization. When a configuration exceeds given resources, the algorithm terminates.

Table 1. Resources of a hypothetical host

Resource Type	Quantity
CPU cores (#)	32
Memory (MB)	32000
Disk space (MB)	100000
Disk speed (MB/s)	30
Network (Mbit/s)	100

4.1 Optimization Algorithm

Our first approach in finding the optimal configuration of threads for the proposed queuing network software was to implement the aforementioned performance metrics. We therefore created a Java³ test tool that calculates all the necessary metrics and recursively adds threads to over-utilized nodes.

The optimization process of the calculation module can be described as follows:

³ <http://java.sun.com>

1. The calculation module uses parameters like the external arrival rate, the service rates for each node, the queue sizes for each node and the host's hardware details.
2. The calculation module starts with an initial configuration of one thread per node.
3. The calculation module calculates all the performance metrics mentioned in the previous section.
4. The process is terminated if a utilization goal is reached or hardware restrictions are met.
5. Otherwise the calculation module increases the most over-utilized node (the first node that is utilized more than e.g.: 80%).
6. Goto step 3.

4.2 Real Measurements

Our second approach uses a combination of the previously mentioned calculation module and a measurement approach. Basically, the analytical approach can use hypothetical input data (e.g.: host resources and node service rates) to derive an optimal configuration for the given setup. The goal of our work is to find an optimal solution for a given server software on a specific host. Therefore, another Java module, the measurement module, was developed.

The optimization process of both the calculation and measurement module can be described as follows:

1. The measurement module creates artificial tasks (e.g.: writing data to a file, writing data into a database, data modifications and so forth).
2. These artificial tasks, which should be as close to the tasks of the real queueing network software (e.g.: the DFE) as possible, are assigned to artificial nodes.
3. The measurement module continuously executes the given tasks on a real host and measures the service rate for each node.
4. The measurement module also automatically finds out important hardware specifications of the tested host.
5. The measured service rates, along with the hardware specifications of the tested host are used by the calculation module to recursively find the optimal configuration of the queueing network software for a specific host.

By measuring the service rate on the tested machine, the optimization process now finds the optimal configuration on a given host. In a previously published paper, experiments conducted with the calculation and measurement module are described in more detail in [11]. The focus of this paper, however, is the comparison of the results of our approach with the true optimum configuration for two server machines. These true optimum configurations are derived from experiments conducted with the actual DFE software installed on two server machines (see Section 5).

5 Experiments

To validate our optimization approach a testing environment was set up and experiments were conducted. Therefore, the queueing network software was installed on two hosts (*Goedel* and *Zerberus*).

The first host *Goedel* is a SUN Fire v40z with four dual-core AMD Opteron processors Model 875, each core at 2.2 GHz, has 24 GB of main memory, five 300 GB Ultra320 SCSI HDs, 10/100/1000 Mb/s Ethernet, and runs Linux 2.6.16.60-0.42.7.

The second host *Zerberus* is a Sun SPARC Enterprise T5220, model SED-PCFF1Z with a SPARC V9 architecture (Niagara 2) and a Sun UltraSPARC T2 eight-core processor, each core at 1.2 Ghz and with Chip Multithreading Technology (CMT) for up to 64 simultaneous threads, 32 GB of main memory, two 146 GB Serial Attached SCSI disks, 10/100/1000 Mb/s Ethernet, and runs SunOS 5.10 Generic_127111-11.

Furthermore, an Oracle Database was installed on host *Zerberus*. A ticket generator imitating VoIP devices and sending Diameter tickets to the queueing network software was installed on several test clients.

The procedure of the experiments can be summarized as follows:

- The queueing network software was started with the initial configuration.
- The ticket generator repeatedly sends Diameter tickets to the queueing network software with a given external arrival rate.
- After an experiment cycle, software-internal performance metrics are used to determine the most over-utilized node, which will be increased by one.
- The queueing network software is reconfigured and restarted and a new experiment cycle is started.
- The experiments continue until the optimal configuration is found.

Zerberus. The first experiments were conducted with the actual queueing network software installed on host *Zerberus*. As mentioned before, host *Zerberus* has an Oracle database installed locally and has the possibility to start up to 64 parallel threads.

Again, an over-utilization occurs if the node is utilized more than 80%. Therefore, nodes that show a utilization of over 80% will be increased by one thread.

Table 2 shows the service rate and the utilization of all four nodes with an external arrival rate of 1000 tickets per second. This first experiment makes it obvious that the *Feeder* node is the bottleneck of the system, only managing an average number of 66 tickets per second. Therefore the adaption tool suggests that the *Feeder* node has to be increased by one, creating two *Feeder* threads at the next experiment.

In the next few steps it became obvious that even though the *Feeder* node was recursively extended, the service rate did not increase in the same way. Table 3 shows that the mean service rate of one *Feeder* thread decreases with every newly added *Feeder* thread. Of course, the total service rate of all *Feeder* nodes does not decrease, but adding new *Feeder* threads does not improve the total

Table 2. Initial config. (1-1-1-1) on host *Zerberus* (ext. arrival rate: 1000 tickets/s).

	Service Rate [tickets/s]	Utilization [%]
<i>Decoder</i>	10658	9.38
<i>Converter</i>	12147	8.23
<i>Serializer</i>	2061	48.52
<i>Feeder</i>	66	100.00

service rate of all *Feeder* nodes enough. Even with the maximum number of 61 threads, the *Feeder* node stays the systems bottleneck.

Table 3. Service rates of the *Feeder* node with different configurations on host *Zerberus*

Configuration	Mean Service Rate	
	Individual	Total
(1-1-1-1)	66	66
(1-1-1-2)	44	88
(1-1-1-4)	34	136
(1-1-1-10)	7	70
(1-1-1-20)	5	100
(1-1-1-61)	2	122

Table 4 shows that with the final configuration (one *Decoder*, one *Converter*, one *Serializer* and 61 *Feeder* nodes) all other nodes are of course still under-utilized. This leads to the conclusion that the *Feeder* node should indeed be fixed to one thread per node. The solution to this problem, as mentioned before, could be to allocate a large queue to the *Feeder* node. By doing that, the node can eventually handle queued tickets when the external arrival rate decreases.

Table 4. Final config. (1-1-1-61) on host *Zerberus* (ext. arrival rate: 1000 tickets/s).

	Utilization [%]
<i>Decoder</i>	9.25
<i>Converter</i>	9.67
<i>Serializer</i>	48.95
<i>Feeder</i>	100.00

The final experiment therefore used a *Feeder* node fixed to one thread per node. Table 5 shows that an initial configuration of one thread per each node cannot handle an external arrival rate of 2000 tickets per second without overstepping an utilization of 80%, because the *Serializer* node is already at a

utilization level of 97.04%. Increasing the number of threads per node (without taken the *Feeder* node into account) the final and optimal configuration can handle an external arrival rate of 66900 tickets per second.

Table 5. Initial and final configuration on host *Zerberus*, with a fixed *Feeder* node

Node	Initial Config. 2000 tickets/s		Optimal Config. 66900 tickets/s	
	Util. [%]	Threads	Util. [%]	Threads
<i>Decoder</i>	18.77	1	79.11	9
<i>Converter</i>	16.46	1	70.44	9
<i>Serializer</i>	97.04	1	79.93	45
<i>Feeder</i> (fixed)	100.00	1	100.00	1

Table 5 shows that at an external arrival rate of 66900 tickets per second, all nodes are under a utilization level of 80%. Of course it should be noted, that the *Feeder* node is still highly over-utilized and can only handle about 70 tickets per second. Given the fact that the *Feeder* node and therefore the database is the natural bottleneck, it is necessary to assign a very large queue to the *Feeder* node, to minimize the loss rate.

Goedel. To compare the results and maybe find a different optimal configuration the same experiments were conducted with the queueing network software installed on host *Goedel*. With four dual-core processors, a maximum amount of 8 threads can be started. As mentioned before, host *Goedel* has no local database installed and uses the Oracle database installed on host *Zerberus*.

Table 6 shows the results of the first experiment. At an external arrival rate of 1000 tickets per second, the *Feeder* node is again the systems bottleneck. Table 6 also shows that compared to host *Zerberus*, the *Decoder*, *Converter* and *Serializer* node show a higher service rate during the initial experiment.

Table 6. Initial config. (1-1-1-1) on host *Goedel* (ext. arrival rate: 1000 tickets/s).

	Service Rate [tickets/s]	Utilization [%]
<i>Decoder</i>	32617	3.07
<i>Converter</i>	26058	3.84
<i>Serializer</i>	5202	19.22
<i>Feeder</i>	105	100.00

To start the optimization process, the *Feeder* node again has to be increased. Table 7 shows that on host *Goedel* the individual service rate of one *Feeder* thread

does, to some extent, stay the same, which leads to the fact that the total service rate of all *Feeder* threads is indeed slowly increasing. Table 7 shows that one *Feeder* thread can handle 105 tickets per second, while the final configuration of 5 *Feeder* threads can handle 350 tickets per second.

Table 7. Service rates of the *Feeder* node with different configurations on host *Goedel*

Configuration	Mean Service Rate	
	Individual	Total
(1-1-1-1)	105	105
(1-1-1-2)	79	158
(1-1-1-3)	81	243
(1-1-1-4)	70	280
(1-1-1-5)	70	350

Given the fact that the total service rate of all *Feeder* threads is increasing, it would make sense to stick to this optimization approach. Table 8 therefore shows an initial and optimal configuration on host *Goedel*. With an external arrival rate of 280 tickets per second the *Feeder* node of the initial configuration is over-utilized, but with the optimal configuration of 5 threads, the *Feeder* node is able to stay under the utilization threshold of 80%.

Table 8. Initial and final configuration on host *Goedel*

Node	Initial Config. 280 tickets/s		Optimal Config. 280 tickets/s	
	Util. [%]	Threads	Util. [%]	Threads
<i>Decoder</i>	0.86	1	0.97	1
<i>Converter</i>	1.07	1	1.41	1
<i>Serializer</i>	5.38	1	4.04	1
<i>Feeder</i>	100.00	1	80.00	5

Table 9 shows the results of the experiments, if the software developer decides to fix the *Feeder* node to one thread per node. At an initial configuration of one thread per node and an external arrival rate of 5200 tickets per second, the *Serializer* node would exceed the utilization threshold of 80%. After the optimization process, the system is able to handle up to 15600 tickets per second with the optimal queueing network software configuration (one *Decoder* node, two *Converter* nodes, four *Serializer* nodes and one *Feeder* node), without exceeding the utilization threshold.

Table 9. Initial and final configuration on host *Goedel*, with a fixed *Feeder* node

Node	Initial Config.		Optimal Config.	
	Util. [%]	Threads	Util. [%]	Threads
<i>Decoder</i>	15.94	1	52.10	1
<i>Converter</i>	19.96	1	75.02	2
<i>Serializer</i>	99.96	1	79.89	4
<i>Feeder</i> (fixed)	100.00	1	100.00	1

5.1 Verification of the Analytical Approach

To verify the analytical approach we used the average service rates for each node derived from the experiments done on host *Zerberus* and host *Goedel* (see Table 10) and started the calculation module one more time.

Table 10. Mean service rates of both tested hosts for each node type

Node	Mean Service Rates	
	Zerberus	Goedel
<i>Decoder</i>	9766	30055
<i>Converter</i>	10430	18672
<i>Serializer</i>	1987	6146
<i>Feeder</i>	28	94

Table 11 shows that starting the calculation module with optimization towards throughput, and using the average service rates derived out of the experiments, both, experiments and the calculation module deliver the same optimal configuration. On host *Zerberus* and host *Goedel* a normal optimization would only increase the number of *Feeder* nodes. Due to different hosts and therefore different node service rates, an optimal configuration with a fixed *Feeder* node would lead to an optimal configuration of 9 *Decoder* nodes, 9 *Converter* nodes, 45 *Serializer* nodes and 1 *Feeder* node on host *Zerberus*, and 1 *Decoder* node, 2 *Converter* nodes, 4 *Serializer* nodes and 1 *Feeder* node on host *Goedel*.

These four verifications show the importance of the service rates and if there is no possibility to derive real service rates from an actual software, it is necessary to analyze the used nodes in every detail. As mentioned before, the measurement module is using simulated nodes to derive artificial service rates. Therefore, the simulated tasks have to be as close as they can get to the actual performed tasks of the tested queueing network software.

Table 11. Optimal configuration of threads for both hosts

Optimal Configuration	Zerberus		Goedel	
	Normal	Fixed Feeder	Normal	Fixed Feeder
Experiments	(1-1-1-61)	(9-9-45-1)	(1-1-1-5)	(1-2-4-1)
Analytical Model	(1-1-1-61)	(9-9-45-1)	(1-1-1-5)	(1-2-4-1)

5.2 Removing the Bottleneck

In the above examples we see that the connection to our database is a serious bottleneck that hinders further improvements. Further optimization would therefore try to improve the database connection throughout, e.g., by increasing the network bandwidth, installing new drivers, or installing replicated databases. Consider a hypothetical case where on Zerberus the service rate of the Feeder would be improved by a factor of 5, 10, or even 15.

Table 12. Service rates, number of threads, and utilization in a hypothetical scenario with Feeder being faster by a factor of 5x, 10x, or 15x. The system throughput is increased to 13400, 23000, and 29500 resp.

	5x, EAR: 13400			10x, EAR: 23000			15x, EAR: 29500		
	SR	T	Util. [%]	SR	T	Util. [%]	SR	T	Util. [%]
<i>Decoder</i>	10658	2	62.86	10658	3	71.93	10658	4	69.2
<i>Converter</i>	12147	2	55.16	12147	3	63.12	12147	4	60.71
<i>Serializer</i>	2061	9	72.24	2061	14	79.71	2061	18	79.52
<i>Feeder</i>	330	51	79.62	660	44	79.2	990	38	78.42

Table 12 shows that improving the bottleneck indeed results in a significant improvement of the overall throughput, while keeping all nodes at moderate utilization. Still most threads are invested into the Feeder, which is still the main bottleneck.

6 Conclusion

This work showed how queueing theory can help finding the best configuration of a multi-thread software. By modeling such a software as queueing network consisting of nodes with certain functionalities, optimization towards throughput is possible. As a result the optimal number of threads per node is determined to efficiently use available CPU cores, memory, disk space and speed, and network bandwidth. Experiments evaluated our methodology.

The basic idea behind our three approaches is an online optimization tool that can be placed in front of the queueing network software. After measuring

the respective service times of the nodes, we use an analytical queueing network to find the optimal number of threads for each node.

After that, the data-flow software can go online. The optimization tool should then be able to detect changes in the external arrival rate and – if necessary – recalculate the optimal configuration.

We also conducted several other experiments using a different setup of the queueing network, to see if the proposed approaches can be used for other multi-thread data-flow software. The structure of the queueing network, as well as the used nodes can easily be changed or enhanced, without knowledge of the Java code. Therefore, the mentioned approaches can be used for other multi-thread data-flow software as well.

References

1. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proceedings of the IEEE 93, 216–231 (2005), <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.7045>
2. Püschel, M., Moura, J.M.F., Johnson, J., Padua, D., Veloso, M., Singer, B., Xiong, J., Franchetti, F., Gacic, A., Voronenko, Y., Chen, K., Johnson, R.W., Rizzolo, N.: SPIRAL: Code Generation for DSP Transforms. Proceedings of the IEEE, Special Issue on Program Generation, Optimization, and Adaptation 93(2), 232–275 (2005)
3. Whaley, R.C., Dongarra, J.J.: Automatically tuned linear algebra software. In: Supercomputing 1998: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing (CDROM), pp. 1–27. IEEE Computer Society, Washington, DC (1998)
4. Osogami, T., Kato, S.: Optimizing system configurations quickly by guessing at the performance. SIGMETRICS Perform. Eval. Rev. 35(1), 145–156 (2007)
5. Balsamo, S., Person, V.D.N., Inverardi, P.: A review on queueing network models with finite capacity queues for software architectures performance prediction. Performance Evaluation 51(2-4), 269–288 (2003)
6. Jain, R.K.: The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling. Wiley, Chichester (1991), <http://www.cse.wustl.edu/~jain/books/perfbook.htm>
7. Zukerman, M.: Introduction to Queueing Theory and Stochastic Teletraffic Models. Zukerman (2009), <http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf>
8. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.S.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, 2nd edn. Wiley Blackwell (May 2006), <http://www4.informatik.uni-erlangen.de/QNMC>
9. Kobayashi, H., Mark, B.L.: System Modeling And Analysis - Foundations of System Performance Evaluation, 1st edn., vol. 1. Prentice-Hall, Englewood Cliffs (2008), <http://www.princeton.edu/kobayashi/Book/book.html>
10. Agresti, A.: Categorical Data Analysis, 2nd edn. Wiley-Interscience, Hoboken (2002)
11. Weidlich, R., Nussbaumer, M., Hlavacs, H.: “Optimization towards consolidation or throughput for multi-thread software. In: International Symposium on Parallel Architectures, Algorithms and Programming, pp. 161–168 (2010)

Formal Mapping of WSLA Contracts on Stochastic Models

Rouaa Yassin Kassab and Aad van Moorsel

School of Computing Science, Newcastle University, UK
{rouaa.yassin-kassab, aad.vanmoorsel}@ncl.ac.uk

Abstract. Service level agreement (SLA) specification languages are designed to express monitorable contracts between service providers and consumers. It is of interest to determine if predictive models can be derived for SLAs expressed in such languages, if possible in automated fashion. For this purpose, we study in this paper the mapping of the Web Service Level Agreement (WSLA) into reward metrics defined in the Stochastic Discrete Event Systems (SDES) formalism. We associate a formal semantics with WSLA elements and map these on SDES through a five step mapping process, which includes expressions for the metrics and functions on these metrics, the time at which to predict, and the ultimate service level compliance probability. We illustrate our approach through a stock quote web service example.

1 Introduction

Service Level Agreements (SLA) clarify the relationship between the service provider and their customers regarding the overall quality of the offered service [1,2]. SLA contracts can be written using different languages such as WSLA [3], WS-Agreement [4], and SLAng [5]. Each of these languages has its own syntax and semantics but they have in common declarations of several pieces of information such as, the contractual parties, performance or dependability levels, and the penalties in case of contract breaching [6]. An SLA is typically defined in such a way that it is monitorable. This for instance provides the ability to automate the deployment and configuration of monitoring software based on SLA specification, as pursued in [2].

In this paper we pursue another use of SLAs, namely that as the specification of metrics for a predictive discrete-event stochastic model. Service providers want to be able to predict the level of SLA compliance before agreeing to an SLA with a customer. Thus, the automated conversion from an SLA to a metric in a discrete event stochastic model is of interest. The use of an SLA for predictive modeling is not necessarily easy since SLAs are not written for the purpose of model-based prediction. As an example, SLAs do not define steady-state metrics, but instead functions over periodically monitored variables. Modeling and solving for such metrics is typically more involved than solving for steady-state metrics. In this work, we demonstrate how an existing SLA language can be mapped on (metrics of) a discrete-event stochastic model, in part in automated fashion.

More specifically, we map the Web Service Level Agreement language (WSLA) [3] on SDES, the Stochastic Discrete Event System formalism developed in [7]. We choose

to use WSLA because it is published and publicly available and it is powerful enough to define SLA documents in several domains (like web services). Similarly, SDES is a general purpose stochastic process formalism that includes a variety of reward modeling formalisms, and the mapping to SDES therefore directly translates to that in formalisms that have extensive tool support. We propose a five-step process to map WSLA to SDES. In the first two steps, we determine the semantics of all WSLA elements in terms of SDES. We then give a precise interpretation of the time instances and intervals used in WSLA, and provide a formal interpretation of the functions used in WSLA (such as ‘max’ or ‘span’). The final step in our mapping process then describes the final assessment whether the SLA is met or not. It will turn out that we cannot automate all aspects of the mapping process, since the WSLA document does not provide much information about the system dynamics itself. This implies that certain aspects of the SDES model must be introduced by the modeler, which we will indicate for each aspect of the mapping. Through an example of a stock quote service, we will illustrate the working of our mapping approach.

To the best of our knowledge model-based prediction based on mapping SLAs on a stochastic process has not yet been considered in literature, certainly not in the fashion pursued in this paper. Therefore, there is no literature available that directly relates to what we do in this paper. However, it is important to note the fundamental difference between our approach and using model-based metric definitions as SLAs, such as in [8]. The approaches are opposite: the latter requires engineers or modelers to write and understand sophisticated metrics associated with a stochastic process, while our work starts from an SLA specified by an engineer and provides this engineer with the stochastic metrics that closest fit with the intended meaning of the SLA. We first introduced this idea in our previous work [9], where we illustrated the feasibility of the approach through an ad hoc mapping from WSLA to Möbius rewards [10]. The work in this paper reflects a generic solution independent of the specifics of a tool, providing a formal mapping from WSLA to a general-purpose stochastic process definition.

The rest of the paper is organized as follows. In Sec. 2 we present the relevant background information. In Sec. 3 we outline our approach, while Sec. 4 to 8 provide a detailed mapping from WSLA to SDES. A case study is presented in Sec. 9 before we conclude in Sec. 10.

2 Background

In this paper we map WSLA elements to metrics in the SDES formalism [7], which describes the system dynamics as well as the metrics defined in WSLA. In this section, we review the target formalism SDES and the origin formalism WSLA respectively.

2.1 Stochastic Discrete Event System (SDES)

Stochastic Discrete Event System is a general formalism, in which well-known formalisms such as Stochastic Petri Nets and Queuing Networks can be expressed.

Definition 1. *A stochastic discrete event system is a tuple, $SDES=(SV, A, S, RV)$ [7], where, SV is a set of state variables, A is a set of actions, S is a Sort function $S : SV \rightarrow$*

\mathbb{S} , that gives all possible values of a state variable $sv \in SV$ (where \mathbb{S} is the set of all possible sorts), and RV is a set of reward variables.

An SDES is characterized by its state $\sigma \in \Sigma$, $\Sigma = \prod_{sv \in SV} \mathcal{S}(sv)$, where Σ is the set of all theoretically possible SDES states which not all of them are necessarily reachable. SDES moves between its reachable states through the execution of its actions.

For our purposes, we need to define the reward variable rv further as follows.

Definition 2. An SDES reward variable $rv \in RV$ is a tuple, $rv = (rv_{rate}, rv_{imp}, rv_{int}, rv_{avg})$, where,

- $rv_{rate} : \Sigma \rightarrow \mathbb{R}$: is a rate reward function that specifies the reward obtained while the system is in a specific state.
- $rv_{imp} : A \rightarrow \mathbb{R}$: is an impulse reward function that specifies the reward obtained when a specific action fires.
- $rv_{int} = [lo, hi]$: is an observation interval under consideration specified by the boundaries $lo, hi \in \mathbb{R}^{0+} \cup \{\infty\}$ and $lo \leq hi$. Hence $lo = hi$ implies an instant of time measure and $lo < hi$ an interval of time measure [11].
- $rv_{avg} \in \mathbb{B}$ is a boolean value specifying if the measures should be computed as an average over time ($rv_{avg} = TRUE$) or accumulated ($rv_{avg} = FALSE$).

An SDES model is represented as a stochastic process $SProc = \{\sigma(t), A(t), t \in \mathbb{R}^{0+}\}$, where $\sigma(t) \in \Sigma$ denotes the state at time t , and $A(t) \subset A$ is a set of actions executed at time t . Hence, we can define the reward variable value at time instant t as following: $R(t) = rv_{rate}(\sigma(t)) + \sum_{a \in A(t)} rv_{imp}(a)$. In [7], this is written as:

$$rv = \begin{cases} \lim_{t \rightarrow lo} R(t), & \text{if } lo = hi \text{ and } \neg rv_{avg} \\ \lim_{x \rightarrow lo, y \rightarrow hi} \int_x^y R(t) dt, & \text{if } lo < hi \text{ and } \neg rv_{avg} \end{cases}$$

2.2 Web Service Level Agreement (WSLA)

WSLA is an SLA specification language in XML tailored to web services. Here we review only WSLA elements important to our work. For full details, we refer to [3]. We use courier font to distinguish XML fragments. WSLA consists of three main sections: `Parties`, `ServiceDefinition` and `Obligations`. `Parties` contains information about the contractual parties, `ServiceDefinition` contains the quality attributes defined by a set of metrics and the time interval over which these metrics are collected. Finally, the `Obligations` section contains the Service Level Objectives (SLO) defined through thresholds for the described attributes over a validity period.

Fig. 1 depicts the dependencies between WSLA elements that concern our work. Part (a) shows that an SLO (`ServiceLevelObjective`) is defined by a logical `Expression` that has to be valid during a `Validity` period. This expression has a `Predicate` which compares a quality attribute (`SLAParameter`) to a threshold (`Value`). A logical operator (like `And`, `Or`) can be used to express SLO with nested expressions. WSLA allows the modeler to define how the desired `SLAParameter` is measured and computed

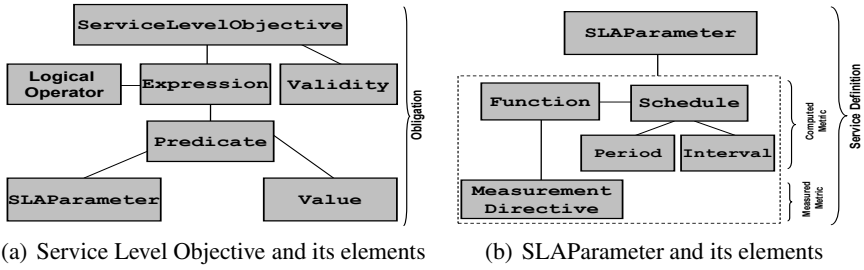


Fig. 1. Main WSLA elements and their dependencies

using Metrics(s) that contain MeasurementDirective, Schedule or Function elements as shown in Fig. 1(b). For example, a measured Metric, using a MeasurementDirective, provides a raw data obtained by intercepting or probing a particular component. If this data is not enough to express an SLAParameter, further manipulations are performed by a computed Metric, using a Function. A Function may create a time series of a metric using a Schedule that defines a Period to specify the time during which the metric is collected (day, month, etc.), and an Interval to specify the instances when a new value is added (minute, hour, etc.). In our paper, we omit Metric usage since it only holds the value of a measurement or a function.

Fig. 2 provides an illustrative WSLA example of a stock quote service adopted from [3] with a change in the validity period as we will discuss in Sec. 9.3. Part (a) of the figure defines an SLO called ContinuousDownTime that specifies the CurrentDownTime to be less than 10 minutes during the last week of the year. Part (b), then, specifies how CurrentDownTime is computed. Here, the StatusRequest measurement in the MeasuredStatus metric checks if the system is up or down (1 or 0). This is used as input to StatusTimeSeries metric that uses a TSConstructor function to

```

<SLA>
  <Parties>...</Parties>
  <ServiceDefinition name="DemoService">
    (c) <Schedule name="availabilityschedule">
      <Period>
        <Start>2001-12-25T14:00:00.000-05:00</Start>
        <End>2001-12-31T14:00:00.000-05:00</End>
      </Period>
      <Interval> <Minutes>1</Minutes> </Interval>
    </Schedule>
    <Operation name="GetQuote"
      xsi:type="WSDLSOAPOperationDescriptionType">
      (b) <SLAParameter name="CurrentDownTime"
        type="long" unit="minutes">
          <Metric>CurrDownTime</Metric>
        </SLAParameter>
        <Metric name="CurrDownTime" type="long" unit="minutes">
          <Function type="Span" resultType="double">
            <Metric>StatusTimeSeries</Metric>
            <Value> <LongScalar>0</LongScalar> </Value>
          </Function>
        </Metric>
        <Metric name="StatusTimeSeries" type="TS" unit="">
          <Function type="TSConstructor" resultType="TS">
            <Schedule>availabilityschedule</Schedule>
            (a) <ServiceLevelObjective name="ContinuousDowntime">
              <Validity>
                <Start>2001-12-25T14:00:00.000-05:00</Start>
                <End>2001-12-31T14:00:00.000-05:00</End>
              </Validity>
              <Expression>
                <Predicate type="Less">
                  <SLAParameter>CurrentDownTime</SLAParameter>
                  <Value>10</Value>
                </Predicate>
              </Expression>
            </ServiceLevelObjective>
          </Function>
        </Metric>
      </Function>
    </Operation>
  </ServiceDefinition>
  <Obligations>
    (c) <Metric>MeasuredStatus</Metric>
    </Function>
    </Metric>
    <Metric name="MeasuredStatus" type="integer" unit="">
      <MeasurementDirective type="StatusRequest"
        resultType="integer">
        <RequestURI>http://y.com/StatusRequest/GetQuote</RequestURI>
      </MeasurementDirective>
    </Metric>
  </Obligations>
</SLA>

```

Fig. 2. WSLA document for a Stock Quote Service (a) Service Level Objective (b) SLAParameter (c) Schedule

define a series of these values. In turn, CurrDownTime metric applies a Span function on that series that gives for a specific position in that time series, the maximum length of an uninterrupted sequence of a value (0 in our example) ending at that position. Finally, this metric value is used as the SLAParameter value. Finally, Part (c) shows the definition of availabilityschedule used by StatusTimeSeries metric to collect MeasuredStatus metric values at each minute for one week. All the previous define the SLO such as that the system will never be down for 10 minutes or more in a row throughout the last week of December.

3 Outline of the Mapping Process

Our approach is given in Fig. 3. It consists of five steps that map WSLA elements on SDES reward variables (solid lines) or on functions of these reward variables (dashed lines). We outline the steps in what follows and then describe them in detail afterwards.

Step 1: Defining basic WSLA semantics. In order to be precise, we need to associate WSLA elements (the dotted boxes in the left of Fig. 3) with mathematical terms.

Step 2: MeasurementDirective(s) Mapping. Completing the interpretation of WSLA's behavioral semantics, it provides a systematic translation of all measured metrics in a WSLA document into SDES reward variables.

Step 3: Schedule Mapping. It provides a systematic translation to obtain the set of observation intervals of the reward variable. This can be a set of either instant or interval of time observation intervals.

Step 4: Function(s) Mapping. Each WSLA Function (such as max and span) is associated a mathematical meaning to further specifying the reward variable in SDES.

Step 5: SLO Result. The outcome of this mapping allows the evaluation of SLO satisfaction probability, i.e., the determination if the service level agreed to is met.

This mapping must be aided in its second step by the modeler (modeler symbol in Fig. 3) because a WSLA document does not provide information about the system dynamics. We will point out when this interference is required later in Sec. 5.

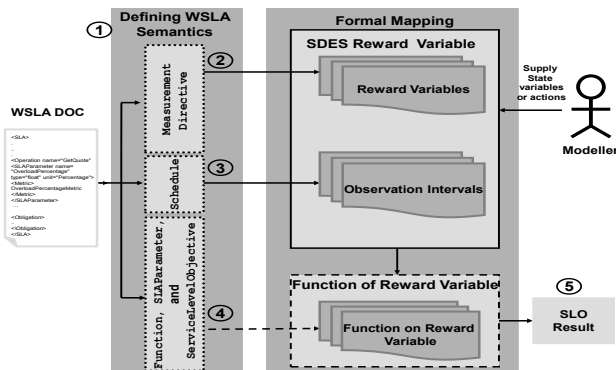


Fig. 3. The Mapping Process from WSLA to SDES

Table 1. Formal elements and associated WSLA location (using XPath 2.0 expressions)

Formal Elements	WSLA Location
slo	/SLA/Obligations/ServiceLevelObjective/@name
$slap$	//ServiceLevelObjective[@name='slo']/Expression/Predicate/SLAPParameter/text()
c	//ServiceLevelObjective[@name='slo']/Expression/Predicate/SLAPParameter[text()='slap']/../@type
v	//ServiceLevelObjective[@name='slo']/Expression/Predicate/SLAPParameter[text()='slap']/../Value/text()
ve	//ServiceLevelObjective[@name='slo']/Validity/End/text()
vs	//ServiceLevelObjective[@name='slo']/Validity/Start/text()
$m \in M$	/SLA/ServiceDefinition/Operation/Metric/MeasurementDirective/@type
$f_m \in F_m$	//Metric[Function/Metric/text()=//Metric/MeasurementDirective[@type='m']]/../@name //Metric/Function[@type='f_m']/../@name/Function/@type
sch	//Operation/Metric[Function/Metric/text()=//Metric/MeasurementDirective[@type='m']]/../@name/Function[@type='TSConstructor']/Schedule/text()
s	/SLA/ServiceDefinition/Schedule[@name='sch']/Period/Start/text()
e	/SLA/ServiceDefinition/Schedule[@name='sch']/Period/End/text()
i	/SLA/ServiceDefinition/Schedule/Interval/node()/text()
o	//Function[@type='Span']/Value/LongScalar/text()

4 Step 1: Formal Definition of Basic WSLA Semantics

In this section, we formalize the structure and the semantics of WSLA's core elements (see Fig. 1). To be able to provide a precise mapping, Table 1 gives the exact locations in the WSLA contract for all elements, using the XPath 2.0 query language[12].¹²

We start by formally representing `ServiceLevelObjective`, since Fig. 1(a) shows it is the key WSLA element, defining its ultimate goal. An SLO in WSLA compares an `SLAPParameter` with some threshold `Value`, for a specific `Validity` period. Hence, we obtain the following definition:

Definition 3. A WSLA `ServiceLevelObjective` can be denoted by a tuple $slo = (slap, c, v, vs, ve)$,³ where:

- $slap \in SLAP$: the desired `SLAPParameter` from the set of all `SLAPParameter(s)` (`SLAP`) defined in a WSLA document. It is defined in detail in Definition 4.
- $c \in C$: where $C = \{=, <, >, \leq, \geq\}$.
- $v \in \mathbb{R}$: the value that the `SLAPParameter` is compared to.
- $vs, ve \in \mathbb{R}^+$, $vs \leq ve$: the start and end of the validity period.

¹ XPath is used to identify elements in an XML document using location path expressions to reach a specific node or set of nodes. A node, in our work, is an element, an attribute or a text node, reached using the `nodename`, `@nodename` and `text()` construct, respectively. A path expression consists of a set of steps separated by a slash `/`, to represent a parent-child relation, or double slash `//` to represent an ancestor-descendant relation (that is, it selects nodes from the node that match the selection regardless where these nodes are). Predicates (inserted into square brackets `[]`) are used to search for a node that matches a particular value. Also two dots `..` is used to select the parent of the current node.

² The XPath expressions have been validated using Altova XMLSpy software [13].

³ We do not need to represent WSLA's `Expression` since it does not matter to the mathematical semantics; Also, for simplicity, we are not considering nested expressions. Therefore, we do not include logical operators in the slo definition.

The next step is to define the `SLAPParameter`, *slap* formally. As depicted in Fig. 1(b), WSLA collects measurements at regular intervals and then applies a set of functions over them to derive the exact *slap*. This allows us to define *slap* as:

Definition 4. A WSLA `SLAPParameter` is a tuple $slap = (M, Sch, F)$, where:

- *M*: Is a non-empty set of $|M|$ elements. Each $m \in M$ specifies a `MeasurementDirective`. To be defined in detail in Sec. 5.
- *sch* $\in Sch$: Is a schedule used by a WSLA function to collect measurements periodically. *sch* is defined as a set of time points:

$$sch = \{t_1, \dots, t_k\}; t_1 = s, t_{j+1} = t_j + i, j = 1 \dots k - 2, t_k = e,$$

where *s*, *e* are the start and end points, *i* is the increment in time, and $k = \lfloor \frac{e-s}{i} \rfloor + 1$ is the number of elements in *sch*.

- *F*: For each $m \in M$, a set of functions $F_m = \{F_{m,1}, \dots, F_{m,|F_m|}\}$ is defined, each refers to a `WSLA Function`. This set can be empty if *slap* represents the value of a single $m \in M$. To be defined further in Sec. 7.

5 Step 2: MeasurementDirective(s) Mapping

In this section, we first present `MeasurementDirective` types in WSLA, then we provide a mapping for each of them. The `MeasurementDirective(s)` are the actual metrics constituting the *slo*. These can be one of seven types, namely `Status`, `StatusRequest`, `Counter`, `Gauge`, `ResponseTime`, `DownTime`, and `InvocationCount`.

Fig. 4 provides the generic structure of a `MeasurementDirective`. The values in italics depend on the measurement type, all other tags remain the same. The measurement type is specified in the `type` attribute, which affects the type of the result specified in the `resultType` attribute. The structure also contains an element that refers to the URI, from where this measurement value can be retrieved. In a model, a URI may indicate the modeler to include a set of actions or state variables that convey a meaning this URI provides. We discuss this when we present each measurement mapping.

```
<MeasurementDirective xsi:type="wsla:Measure_type" resultType="result_type">
    ..additional tags specifying URI name
</MeasurementDirective>
```

Fig. 4. General structure Measurement directives element in WSLA

We provide, in what follows, an unambiguous mapping from each `MeasurementDirective` to a reward variable *rv* in SDES. We will point out the information needs to be input by the modeler to complete this mapping.

5.1 StatusRequest and Status

`StatusRequest` gives 1 if the system is up and 0 otherwise, while `Status` gives a true/false value[3]. This difference is not important when modeling, hence, we treat

them identically. These measurements follow the syntax in Fig. 4 with `resultType` of `"integer"` for `StatusRequest`. The URI is referred to using `<RequestURI>` tag.

We map this measurement as a rate reward variable that returns 1 while the system is in an up state and 0 otherwise. If $\Sigma^* \in \Sigma$ is the set of system state under which this SDES system is considered up, then the reward variable is:

$$rv = \begin{cases} rv_{imp}(a) = 0 & \forall a \in A \\ rv_{rate}(\sigma) = \begin{cases} 1 & \text{if } \sigma \in \Sigma^* \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

URI usage: We propose that it refers to the status of WSDL operation.

Input modeler: The system states Σ^* that correspond to an available system.

Evaluation Interval: WSLA retrieves this measurement at specific time instances. Hence, it is represented as instant of time reward variable, i.e., $rv_{int} = [lo, hi]$, with $lo = hi$ & $rv_{avg} = False$.

5.2 InvocationCount

WSLA defines `InvocationCount` as “the number of usages of an operation” [3]. That means, it corresponds to the throughput of a service operation. Its WSLA syntax follows Fig. 4 with `resultType="integer"` and a `<CounterURI>` tag for specifying the URI.

The natural manner to describe this measurement in SDES is to associate an impulse reward of value 1 each time an action $a^* \in A$ that represent the WSDL operation is fired:

$$rv = \begin{cases} rv_{rate}(\sigma) = 0 & \forall \sigma \in \Sigma \\ rv_{imp}(a) = \begin{cases} 1 & \text{if } a = a^* \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

URI Usage: Refers to include an action that represents the WSDL operation.

Input modeler: Action a^* .

Evaluation Interval: We interpret WSLA’s use of this measurement as the increment in the number of the service invocations from one reading to the next. The reward variable should therefore be evaluated as interval of time reward variable to keep track of the increment in invocation count, i.e., $rv_{int} = [lo, hi]$, with $lo < hi$, & $rv_{avg} = False$.

5.3 Gauge

Gauge is defined in WSLA as “a non-negative integer that may increase or decrease, and is used to measure the current value of some entity” [3]. It has a `resultType` of `"double"` and the URI is referred to using `<MeasurementURI>` tag.

In essence, `Gauge` corresponds to the current value of a state variable, and in SDES terms we can allow rate as well as impulse rewards to add to it. The reward definition is then unrestricted, and the modeler can assign any rewards to the gauge. We therefore also provide a special gauge, corresponding to a single state variable representing the gauge value (which is the common case). Depending on the model at hand, this simplifies the job of the modeler. Assume the state variable that is intended to hold the number of particular tasks in the system is $sv \in SV$, with $sv(\sigma)$ being the value of sv in a system state σ , then the reward variable is defined as:

$$rv = \begin{cases} rv_{imp}(a) = 0 & \forall a \in A \\ rv_{rate}(\sigma) = \begin{cases} sv(\sigma) & \forall \sigma \in \Sigma \\ 0 & otherwise \end{cases} \end{cases}$$

URI usage: Hints to the addition of a state variable that performs a special function. In the example below, *noReqInQueue* indicates the modeler to include a state variable that stores the arriving requests in the model.

```
<MeasurementDirective xsi:type="wsla:Gauge" resultType="double">
  <MeasurementURI>http://support1.com/noReqInQueue</MeasurementURI>
</MeasurementDirective>
```

Input modeler: Chooses or introduces a state variable sv_i .

Evaluation Interval: The reward variable is an instant of time variable to give the current value of a system component, i.e., $rv_{int} = [lo, hi]$, with $lo = hi$ & $rv_{avg} = False$.

5.4 Counter

Counter according to WSLA “describes the relevant information to retrieve a counter from the instrumentation of a service or managed resource” [3] and it is used to count specific events of a service. It has a `resultType` of “*integer*” and the URI is referred to using `<MeasurementURI>` tag. We map Counter as an impulse reward variable of an action $a_i \in A$ in the model:

$$rv = \begin{cases} rv_{rate}(\sigma) = 0 & \forall \sigma \in \Sigma \\ rv_{imp}(a) = \begin{cases} 1 & \text{if } a = a_i \\ 0 & otherwise \end{cases} \end{cases}$$

URI usage: Hints to specify an action that performs a special function. In the example below, *ipPacketsIn* hints to add an action that indicates an IP packet arrival.

```
<MeasurementDirective xsi:type="wsla:Counter" resultType="double">
  <MeasurementURI>http://support1.com/ipPacketsIn</MeasurementURI>
</MeasurementDirective>
```

Input modeler: Action a_i .

Evaluation Interval: The reward variable is interval of time reward variable, i.e., $rv_{int} = [lo, hi]$, with $lo < hi$ & $rv_{avg} = False$.

5.5 ResponseTime

It denotes the time between sending a request and receiving its response. The syntax has a `resultType` = “*double*” and the URI specified inside a `<MeasurementURI>` tag.

To express the response time in term of rewards, we use an additional state variable $sv \in SV$ that signals the receipt of the response. sv is initially set to 0 and can jump to 1 once only, indicating the response has been received. Then, $RT(t)$, the probability that the response time is less than t , is equal to the probability that the state variable is 1 at time t . Hence, the response time of an operation is determined by checking at each

time instance if the state variable equals 1. That is: $RT(t) = P(\text{response time} \leq t) = P(sv(\sigma) = 1 \text{ at time } t)$. This is represented using a rate based reward function as:

$$rv = \begin{cases} rv_{imp}(a) = 0 & \forall a \in A \\ rv_{rate}(\sigma) = \begin{cases} 1 & \text{if } \sigma \in \Sigma^* \\ 0 & \text{otherwise} \end{cases} \end{cases}$$

Where Σ^* represents all system states σ where $sv(\sigma) = 1$. If the model is closed, we can use the state variable connected to the sending action as an indicator to whether a response is returned back.

URI Usage: We propose it refers to the WSDL operation.

Input modeler: Introduces a state variable sv in the model to be set to 1 when the response is received, and determine an appropriate initial state for the model.

Evaluation Interval: The reward variable is an instant of time reward variable to check system response at this instant, i.e., $rv_{int} = [lo, hi]$, with $lo = hi$ & $rv_{avg} = False$.

5.6 DownTime

DownTime gives a direct reading of the total time for which the system is in a down status [3]. Hence, the mapping is identical to **Status**, but measured as an interval of time rather than an instant of time. This measurements has a **resultType** of "double".

URI Usage: **DownTime** does not specify any URI.

Evaluation Interval: The reward variable is interval of time reward variable to check system down period. This means: $rv_{int} = [lo, hi]$, with $lo < hi$ and $rv_{avg} = False$.

6 Step 3: Schedule Mapping

After mapping each measurement $m \in M$ to a specific reward variable $rv \in RV$ and determining if it is an instant or interval of time variable, we should find out what these instants/intervals are. For that, it is of particular interest to map WSLA monitoring times defined in the **Schedule** element to time in SDES. We defined WSLA schedule sch in Sec. 4 as a set of time points:

$$sch = \{t_1, \dots, t_k\}; t_1 = s, t_{j+1} = t_j + i, j = 1, \dots, k - 2, t_k = e$$

We want to map these time points on time in SDES. Since SDES has a construct rv_{int} that defines a single observation interval for a reward variable $rv \in RV$, we need to define a set $\{rv_{int}\}$ that contains multiple observation intervals for each reward variable:

$$\{rv_{int}\} = \{rv_{int_j} = [lo_j, hi_j]\}, \quad lo_j, hi_j \in \mathbb{R}^+, j = 1, \dots, k$$

Because the reward variable $rv \in RV$ could be either instant or interval, the boundaries lo_j, hi_j of each rv_{int_j} will vary accordingly.

In case $m \in M$ is mapped as an instant of time reward variable, then $lo_j = hi_j$ in each observation interval rv_{int_j} . Hence, $t_j \in sch, j = 1, \dots, k$ is mapped as a set of instant of time observation intervals. We write this as a set: $\{rv_{int}\} = \{\{t_j, t_j\}\}, j = 1, \dots, k$.

However, if $m \in M$ is mapped as interval of time reward variable, then $lo_j < hi_j$ in each observation interval rv_{int_j} . In this case, $t_j \in sch$, $j = 1, \dots, k$, is mapped as a set of interval of time observation intervals, each interval is between two sequential time points in sch . We write this as the set: $\{rv_{int}\} = \{[t_j, t_{j+1}]\}$, $j = 1, \dots, k - 1$.

7 Step 4: Function(s) Mapping

In WSLA, a `MeasurementDirective` is used as the basis for performing other WSLA computations to produce the top most metric that represents the required `SLAParameter`. These computations are done through `Function(s)`. WSLA defines a set of 17 `Function` types in its standard specification. Each one corresponds to either series constructors (like `TSConstructor`), series manipulation functions (like `Span`), arithmetic functions (like `Plus`), or statistical functions (like `Mean`). These `Function(s)` can be applied on a `MeasurementDirective` or on other `Function(s)` to obtain the exact `SLAParameter`.

WSLA rarely depends on a single measurement $m \in M$ to represent an `SLAParameter`, *slap*. Rather, it applies a set of functions $F_m = \{F_{m,1}, \dots, F_{m,|F_m|}\}$ on m , where the function $F_{m,i}$, $i = 1, \dots, |F_m|$ is the i -th function to be applied on m . The functions in F_m set could be any of the aforementioned function types. However, from our observation of existing WSLA contracts [3,14,15], we have obtained a common order in which function types are applied described in the following steps:

1. Firstly, WSLA applies a time series function (`TSConstructor`)⁴ to create a time series that collects measures of m by the use of a schedule sch . Hence, $F_{m,1}$ is the `TSConstructor` function and its output is a series of measurements:

$$\{m(t_1), \dots, m(t_k)\}, \{t_1, \dots, t_k\} \in sch,$$

where $m(t_j)$ is the measurement m at time t_j , $j = 1, \dots, k$.

2. Then, a function $F_{m,2}$ is applied on this series so that a single output is produced⁵.
3. Finally, additional functions can be applied so that the exact *slap* is obtained.

We need to map WSLA functions on SDES. The mapping is provided for each of the steps described earlier as follows:

1. Since the measurement is mapped as an *rv* and the schedule is mapped as $\{rv_{int}\}$, then the time series constructor function in SDES evaluates the reward variable $rv \in RV$ for each evaluation interval in $\{rv_{int}\}$. This can be expressed as a set: $\{rv(t_1), \dots, rv(t_k)\}$, where $rv(t_j)$ is the reward variable with the evaluation interval $rv_{int_j} = [t_j, t_j]$ in case of instant of time reward variable, and $rv_{int_j} = [t_j, t_{j+1}]$ in case of interval of time reward variable. In SDES, each $rv(t_j)$ can be thought of as a random variable $X_{t_j} : \Sigma \rightarrow \mathbb{R}$. Accordingly, we can write the previous set as a set of random variables as follows:

$$\{X_{t_1}, \dots, X_{t_k}\}$$

⁴ We refer to the formal semantic of WSLA functions, measurements, and schedules by emphasizing them to differentiate them from the XML tag.

⁵ Except for series constructors functions that return a series instead of a single value.

2. The function $F_{m,2}$ is applied on the above set of random variables. Due to space limitations, we are not providing an exact mapping of each WSLA function, rather, we generally describe how these functions are applied on the set. Any function over a set of random variables results in a new random variable whose probability distribution is determined by the probability distribution of each random variable [16].

$$X_{F_{m,2}} = F_{m,2}(X_{t_1}, \dots, X_{t_k})$$

3. The rest of functions $F_{m,3}, \dots, F_{m,|F_m|}$ in the set F_m will be applied in sequence. This also results in a new random variable each time a new function is applied.

$$X_{F_{m,i}} = F_{m,i}(X_{F_{m,i-1}}), i = 3, \dots, |F_m|$$

The random variable $X_{F_{m,|F_m|}}$ that results from applying the last function $F_{m,|F_m|} \in F_m$, represents the value of *slap*.

It should be noted that some WSLA functions applied on a set of measurements specify an additional operand called *Value*. For example, the *Value* element in the *Span* function specifies the value the function counts the occurrence of inside the set. Hence, we can write these WSLA functions as: $F_{m,2}(m(t_1), \dots, m(t_k), o)$, $o \in \mathbb{R}$, where o is the value the function is looking for. The WSLA location for o value (for *Span* in particular) is specified in Table 1. Accordingly, we can write these functions after mapping to SDES as: $F_{m,2}(X_{t_1}, \dots, X_{t_k}, o)$, $o \in \mathbb{R}$.

8 Step 5: SLO Result

In the previous section, we describe how the mapping of all functions F_m results in a single random variable $X_{F_{m,|F_m|}}$ that represents *SLAPparameter*, *slap*. Hence, the random variable X_{slap} can be defined as:

$$X_{slap} \triangleq X_{F_{m,|F_m|}}$$

Depending on this random variable, an *slo* is evaluated. This is done by performing a comparison of type c of X_{slap} against a value v (as defined in Sec. 4). For example if $c = \leq$ then we can write the probability that an *slo* is met as: $P(slo) = P(X_{slap} \leq v)$

Note that v_s and v_e of the validity period specified for an *slo* have often the same value as s and e for the schedule *sch* defined within the *slap*. Thus, v_s and v_e are implicit in the definition of *sch* and hence all values of *slap* are already between v_s and v_e .

9 Case Study

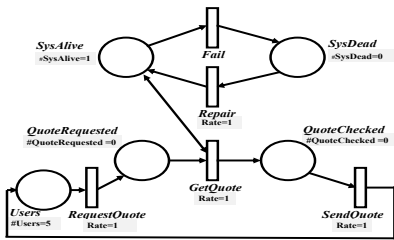
In this section, we present an example of a system modeled using a specific SDES model, namely Stochastic Activity Networks (SAN) [17]. We first describe the system, then give the mapping of the associated WSLA contract, and provide some discussion.

9.1 System Description

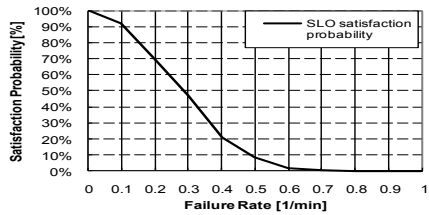
The example described here is a stock quote service that fits with the WSLA contract in Fig. 2. When a user requests a quote, the system puts the request in a queue and then checks its value. Next, it creates a response that waits in another queue before being sent back to the user. Checking the stock value may fail, in which case the requests wait in the queue until the system is up again. As described by the WSLA contract of Fig. 2, the provider offers the service in the last week of the year with high availability: down time should be less than ten minutes.

The service provider wants to predict the probability with which the system meets its SLO. For that, we are mapping WSLA on an SDES model (SAN in our example) of the described service. As we described earlier, the modeler is responsible for creating parts of the model but since web services are described using WSDL, hints from WSDL and WSLA files could be useful for the modeler. For example, in our WSLA contract, the *SLAParameter* is defined for the *GetQuote* WSDL operation and a *StatusRequest* is used as a *MeasurementDirective*. This implies, according to Sec. 5.1, that the model should have an action (SAN activity) that represents this operation, and the modeler should specify how the system goes down (how the *getQuote* operation fails). An automated mapping from WSDL to a Petri Net model is possible according to [18], but we did not exploit this yet in our approach.

The SAN model of the system is illustrated in Fig. 5(a). The system up/down states are described using a single token in *SysAlive* and *SysDead* places respectively. The modeler can choose alternatives or more detailed failure behaviours, such as system resources (CPUs) that run out. Our model alternates between the up/down states through activities *Repair* and *Fail*. All activities in the model have an exponential distribution, their rates (except for failure rate) are shown in the figure along with the proposed system’s initial state (which is also chosen by the modeler).



(a) SAN model of the Quote service



(b) SLO satisfaction probability

Fig. 5. Stock Quote service

9.2 Mapping WSLA to SDES

We now illustrate the application of the five steps mapping process defined in Sec. 3.

Step 1: Defining WSLA Semantics (refer to Sec. 4) *slo* elements along with their correspondences in the WSLA contract (Using Table 1), are given in Definition 3:

$$ContinuousDownTime = (slap, c, v, vs, ve) = (CurrentDownTime, <, 10, 0, 10080),$$

Also, from Definition 4 we see that *CurrentDownTime* is represented by (M, Sch, F) . From the contract we obtain one measurement $m \in M$ and a set of two functions F_m applied on it, and a schedule $sch \in Sch$ as follows:

- $m = StatusRequest$
- $F_m = \{F_{S_{StatusRequest}}\} = \{TSCConstructor, Span\}$
- $sch = availabilitySchedule = \{0 \dots 10080\}$

We note that the WSLA start and end time point are specified in *DateTime*, while the increment is specified using elements like *Hour*, *minutes* and *seconds*. This allows us to use the increment unit as the unified unit of time in our model. We also set the start time to zero, and the end time as the subtraction of the start and end dates. In our example, the increment time unit is minutes, hence the start time point is 0 and the end time 10080 is obtained by expressing 7 days in minutes. The parameters v_s and v_e in Definition 3 can be obtained in similar manners, resulting in $v_s = 0$ and $v_e = 10080$.

Step 2: MeasurementDirective Mapping (refer to Sec. 5.1) *StatusRequest* is mapped as instant of time rate reward variable, where the modeler needs to define Σ^* , the system up states. In our model, this equates to the place *SysAlive* containing 1 token: $\Sigma^* = \{\sigma : SysAlive(\sigma) = 1\}$.

Step 3: Schedule Mapping (refer to Sec. 6.) Since *StatusRequest* is an instant of time reward variable, the model needs to provide instant of time results at the following points in time: $\{rv_{int}\} = \{[0, 0], [1, 1], \dots, [10080, 10080]\}$. (obeying SDES notational conventions)

Step 4: Function(s) Mapping (refer to Sec. 7) We identified the two functions present in the WSLA contract in Step 1 above. The time series function $F_{S_{StatusRequest},1} = TSCConstructor$ is mapped to a set of random variables $\{X_{t_1}, \dots, X_{t_k}\}$, that represents the reward variable at each time instant. In our example, the random variables' state space is $\{0, 1\}$, since the system can be either up (1) or down (0). Thus, for $j = 1, \dots, k$:

$$X_{t_j} = \begin{cases} 1 & \text{if system is up at time } t_j \\ 0 & \text{if system is down at time } t_j \end{cases}$$

The *Span* function $F_{S_{StatusRequest},2} = Span$, counts the number of consecutive random variables with an identical value, which is 0 in our example (refer to the last paragraph in Sec. 7). For $j = 1, \dots, k$, the j -th element of the *Span* function then is:

$$[Span(X_{t_1} \dots X_{t_k}, 0)]_j = \begin{cases} u & \text{if } X_{t_j} = 0 \wedge \dots \wedge X_{t_{j-u+1}} = 0 \wedge X_{t_{j-u}} = 1, \text{ with } 0 < u < j \\ 0 & \text{if } X_{t_j} = 1 \\ j & \text{otherwise (that is, } X_{t_1} = 0, \dots, X_{t_j} = 0) \end{cases}$$

The SLO is satisfied if all *Span* values for $j = 1, \dots, k$ are smaller than the agreed value $v=10$ (see Step 1). So, X_{Span} can be defined as the maximum over all elements $[Span(X_{t_1} \dots X_{t_k}, 0)]_j$.

Step 5: SLO result (refer to Sec. 8.) The service level agreement is evaluated using the random variable X_{Span} , that is: $P(slo) = P(X_{slap} < 10) = P(X_{Span} < 10)$.

We ran experiments using the terminating discrete event simulation in the Möbius modeling tool [10]. We set the repair rate to 1 and varied the failure rate from 0 to

1 to discover how this will affect SLO satisfaction. We run the simulation for at least 100 replicas and count how many of the replicas have a span function value less than 10 (following the definition of $X_{S_{pan}}$ in Step 4 above). We found out, as depicted in Fig. 5(b), that for failure rates more than 0.1, a significant drop in SLO satisfaction is predicted.

9.3 Discussion

The above case study demonstrates that an effective mapping from WSLA to SDES is possible and software tools such as Möbius [10] can be used to obtain the desired reward metrics using discrete-event simulation. However, these tools are not designed to compute functions over random variables, such as the WSLA span function. To determine the probability an SLO will be met, we therefore needed to save results for each replica (corresponding to one realization of the time series X_{t_j} for $j = 1, \dots, k$), and compute the span function afterwards. Note also that although our mapping solution aimed at allowing numerical solutions, numerical solution of the WSLA metrics (instead of deriving them using discrete-event simulation) can be expected to be prohibitively difficult in many cases. This is especially the case if the joint distribution of $\{X_{t_1}, \dots, X_{t_k}\}$ needs to be derived (where the random variables X_{t_j} are not independent), such as is the case for the WSLA span function. In any case, in the next phase of our research, we plan to develop a tool support to augment existing modeling tools with support for all WSLA functions. It should be noted also that in our approach, the reward variable has to be solved for every instant when an observation is made which makes the model expensive to solve. In our example, the Möbius tool took one hour to complete a simulation run of 100 replicas using a single 2.19 GHz processor. This run solve the reward variable for 10080 instants specified in a week validity period. Running time will be longer if we solve for each minute in a one month period as specified in the original example [3]. This raises the problem of scalability that will be investigated more in the future.

10 Conclusion

In this paper, we investigated if WSLA can be used for model-based prediction of SLA compliance, or, to be more precise, if WSLA can serve as a specification of reward metrics in a stochastic model. The process of translating from WSLA to reward variables in Stochastic Discrete Event Systems developed in this paper includes defining the semantics of WSLA elements in terms of stochastic processes, a mapping on SDES reward variables of the WSLA measurements, their evaluation intervals, and associated functions. The outcome of this mapping is a prediction of the probability of complying to the service level objective. The presented translation process establishes a basis to implement a mapping from WSLA to SEDS reward metrics that is as automated as much as possible. The paper shows that there is potential in deriving predictive models from WSLA specifications, and we are therefore pursuing further efforts to complete our tool support for predictive modeling based on WSLA, thus further simplifying the definition of predictive models for web service engineers.

References

1. Molina-Jiménez, C., Pruyne, J., van Moorsel, A.: The role of agreements in IT management software. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) *Architecting Dependable Systems III*. LNCS, vol. 3549, pp. 36–58. Springer, Heidelberg (2005)
2. Sahai, A., Machiraju, V., Sayal, M., van Moorsel, A., Casati, F.: Automated SLA monitoring for web services. In: Feridun, M., Kropf, P.G., Babin, G. (eds.) *DSOM 2002*. LNCS, vol. 2506, pp. 28–41. Springer, Heidelberg (2002)
3. Ludwig, H., Keller, A., Dan, A., King, R., Franck, R.: *Web Service Level Agreement (WSLA) Language Specification*. IBM (January 2003)
4. Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Nakata, T., Pruyne, J., Rofrano, J., Tuecke, S., Xu, M.: *Web Services Agreement Specification (WS-Agreement)*. Open Grid Forum, version 2005/09 edition
5. Davide Lamanna, D., Skene, J., Emmerich, W.: SLAng: A language for defining service level agreements. In: *The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 100–106 (2003)
6. Paschke, A., Schnappinger-Gerull, E.: A categorization scheme for SLA metrics. In: *Service Oriented Electronic Commerce*, pp. 25–40 (2006)
7. Zimmermann, A.: *Stochastic Discrete Event Systems: Modeling, Evaluation, Applications*. Springer-Verlag New York, Inc., Secaucus (2007)
8. Dingle, N.J., Knottenbelt, W.J., Wang, L.: Service level agreement specification, compliance prediction and monitoring with performance trees. In: *22nd Annual European Simulation and Modelling Conference (ESM 2008)*, pp. 137–14 (September 2008)
9. Kassab, R.Y., van Moorsel, A.: Mapping WSLA on reward constructs in Möbius. In: *24th UK Performance Engineering Workshop*, pp. 137–147 (2008)
10. Sanders, W.H.: *Möbius User Manual, Version 2.3.1*. University of Illinois, US (May 2010)
11. Sanders, W.H., Meyer, J.F.: A unified approach for specifying measures of performance, dependability, and performability. *Dependable Computing and Fault-Tolerant Systems: Dependable Computing for Critical Applications 4*, 215–237 (1991)
12. Berglund, A., Boag, S., Chamberlin, D., Fernández, M.F., Kay, M., Robie, J., Simon, J.: *XML path language (XPath) 2.0*. Technical report, World Wide Web Consortium (January 2007)
13. Altova XMLSpy v2011r3 enterprise edition (2011), <http://www.altova.com>
14. Keller, A., Ludwig, H.: The WSLA framework: Specifying and monitoring service level agreements for web services. *J. Netw. Syst. Manage.* 11(1), 57–81 (2003)
15. Ludwig, H., Keller, A., Dan, A., King, R., Franck, R.: A service level agreement language for dynamic electronic services 3, 43–59 (January 2003)
16. Grinstead, C.M., Snell, J.L.: *Introduction to probability*. American Mathematical Society, Providence (1997)
17. Sanders, W.H., Meyer, J.F.: Stochastic activity networks: Formal definitions and concepts. In: *European Educational Forum*, pp. 315–343 (2000)
18. Thomas, J.P., Thomas, M., Ghinea, G.: Modeling of web services flow. In: *CEC*, pp. 391–398 (2003)

Comparison of the Mean-Field Approach and Simulation in a Peer-to-Peer Botnet Case Study

Anna Kolesnichenko¹, Anne Remke¹,
Pieter-Tjerk de Boer¹, and Boudewijn R. Haverkort^{1,2}

¹ Centre for Telematics & Information Technology, University of Twente,
Enschede, The Netherlands

² Embedded Systems Institute, Eindhoven, The Netherlands

{a.v.kolesnichenko,a.k.i.remke,p.t.deboer,b.r.h.m.haverkort}@utwente.nl

Abstract. Peer-to-peer botnets, as exemplified by the Storm Worm, and the spreading phase of Stuxnet, are a relatively new threat to security on the internet: infected computers automatically search for other computers to be infected, thus spreading the infection rapidly. In a recent paper, such botnets have been modeled using Stochastic Activity Networks, allowing the use of discrete-event simulation to judge strategies for combating their spread. In the present paper, we develop a mean-field model for analyzing botnet behavior and compare it with simulations obtained from the Moebius tool. We show that the mean-field approach provides accurate and orders-of-magnitude faster computation, thus providing very useful insight in spread characteristics and the effectiveness of countermeasures.

Keywords: mean-field approximation, simulation, differential equations, peer-to-peer botnet spread.

1 Introduction

A peer-to-peer botnet can be seen as a very large population (possibly all computers in the Internet) of interacting components (peers), where infected nodes infect more and more other computers. Due to the large numbers of (potentially) active components, the analysis of the spreading speed of such large-scale systems is time consuming and computationally expensive. Recently, a stochastic model for the growth of peer-to-peer botnets was introduced in [16]. The model is a Stochastic Activity Network (SAN) [17] with an unbounded number of tokens per place, hence, no solutions can be obtained analytically. The authors of [16] simulate the model with the Moebius toolset [7] to gain insight into the effectiveness of defense strategies. Although such simulation is possible, it is very time-consuming and does result in statistical uncertainties.

Recently, much work has been done on the analysis of large populations of interacting objects. Markovian Agents have been used to predict the propagation of earth quake waves [5] or the behavior of sensor networks [11]. The dissemination of gossip information [1] and disease spread between islands [2]

was analyzed using mean-field approximation. Hybrid approaches, combining mean-field and simulation, have been proposed for general systems of interacting objects [14], but also to predict predator and prey behavior [13]. Ordinary differential equations (ODEs) have been used to analyze the behavior of intracellular signaling pathways [4] and together with PEPA for epidemiological models [6]. Note that the relationship between the different techniques is currently not well investigated; what all have in common is that they provide an approach to deal efficiently with very large numbers of similar interacting objects.

Out of the many available approaches, in this paper we limit ourselves to mean-field analysis and the direct derivation of ODEs from the SAN. The Markovian agent approach was deemed less suitable here, because it explicitly takes into account locality (i.e., the amount of interaction between entities depends on their location), which is not present in the botnet model studied in [16]. We also have not explicitly tried the approach of deriving ODEs from PEPA, as done in [3]; however, due to similarities among the methods, we expect the resulting ODEs to be the same as the ones we obtained.

While in this paper we are not directly interested in obtaining new insights on botnet behavior, our goal is to show how a quicker analysis method can be used to obtain different measures of interest that cannot be obtained using simulation. We use a model based on the one developed in [16] in order to compare simulation and mean-field approximation. The comparison shows that the mean-field method is much faster than simulation, therefore it allows to address more complicated and resource consuming questions, such as the dependency of botnet spread on several parameters. We explore the speed-up, and show that it can be used it to obtain more insight into the botnet behavior, by taking into account costs for running antimalware software and costs that occur due to computers being infected. Furthermore, we discuss the differences between the mean-Field method and simulation and the resulting suitability in different settings.

For completeness, we point out that the spreading phase of the botnet is quite similar to worm propagation, of which several studies using differential equations have been published. For example, [8] briefly introduces a simple model of worm behavior and compares results with the real measured data, while the main focus of the paper is on the discussion of the different types of worms. The authors of [9] use Interactive Markov Chains to calculate simple bounds of the infected population size and compare results with simulation. A density-dependent Markov jump process model and hybrid deterministic/stochastic model for Random Constant Scanning worm are presented in [15]. A continuous-time approximation of process-algebra models is used in [3] for analysis of the worms spread.

The paper is further organized as follows. In Section 2 we develop a model describing the behavior of an individual computer in a botnet. In Section 3 the mean-field approximation method is applied to the botnet spread model. The mean-field results and the simulation results obtained from Moebius are compared in Section 4. In Section 5 the full potential of the mean-field method in consequence of the attained speedup is explored. In Section 6 the applicability

of other large-scale analysis methods to the botnet spread model is discussed. The conclusions and future work are discussed in Section 7.

2 Model of the Botnet Behavior

A Stochastic Activity Network (SAN) model was introduced for Peer-To-Peer botnets in [16]. It models how the infection spreads through an infinite population of computers. The model closely reflects the states a computer goes through after the initial infection has taken place. The original SAN model consists of: (i) one place for each phase of infection a system can be in, that can each hold an unbounded number of tokens, representing the number of computers per phase, and (ii) transitions, which move tokens from place to place, as the infection spreads. The SAN model represents the entire population of infected computers, hence, the number of computers in each state (phase) can be directly derived from the model. However, as the population of computers can be very large or even infinite, it is only possible to derive measures of interest from the SAN model using simulation. This is very time consuming and computationally expensive. Using mean-field analysis, it is possible to compute performance measures of a large population of identical components in a very efficient way. For this, a model is needed which reflects the individual behavior of a single computer. We develop a continuous time Markov chain (CTMC) model for the behavior of an individual computer based on the SAN model from [16], as follows.

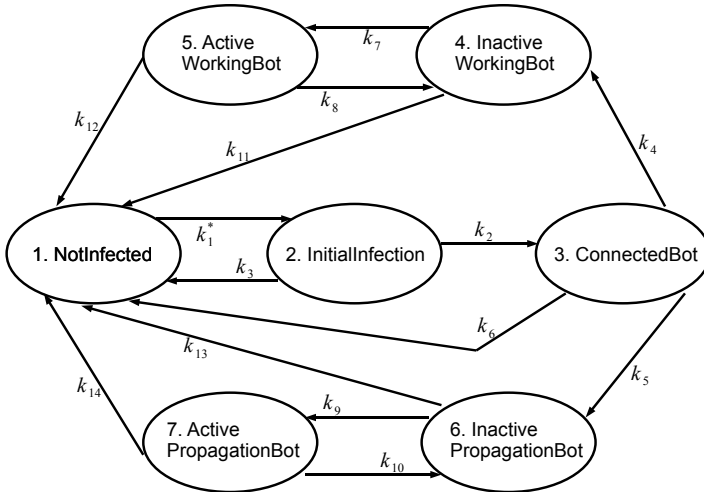


Fig. 1. CTMC for an individual computer

Each place in the SAN model is also represented in the CTMC model as an individual state, however, an additional state is added, that represents a computer which is not infected yet. The CTMC model is depicted in Figure 1 and the corresponding transition rates can be found in Table 1.

Table 1. Transition rates for the CTMC of a single computer

k_1	RateOfAttack·ProbInstallInitialInfection
k_1^*	Rate depends on k_1 and the environment
k_2	RateConnectBotToPeers·ProbConnectToPeers
k_3	RateConnectBotToPeers·(1-ProbConnectToPeers)
k_4	RateSecondaryInjection·ProbSecondaryInjectionSuccess·(1-ProbPropagationBot)
k_5	RateSecondaryInjection·ProbSecondaryInjectionSuccess·ProbPropagationBot
k_6	RateSecondaryInjection·(1-ProbSecondaryInjectionSuccess)
k_7	RateWorkingBotWakens
k_8	RateWorkingBotSleeps
k_9	RatePropagationBotWakens
k_{10}	RatePropagationBotSleeps
k_{11}	RateInactiveWorkingBotRemoved
k_{12}	RateActiveWorkingBotRemoved
k_{13}	RateInactivePropagationBotRemoved
k_{14}	RateActivePropagationBotRemoved

A computer which is not infected yet enters the *InitialInfection* state with rate k_1^* and it becomes initially infected. Then, it connects to the other bots in the botnet, downloads the next part of the malware and possibly moves to state *ConnectedBot* with rate k_2 . If the computer for any reason is not able to download the malware it returns to the state *NotInfected* with rate k_3 .

After downloading the malware, the computer joins the botnet as either *InactiveWorkingBot* or as *InactivePropagationBot* with rates k_4 and k_5 , respectively. If downloading the malware is not possible, for example, because the connection has failed, the computer moves back to the *NotInfected* state with rate k_6 . Once the bot becomes either an *InactiveWorkingBot* or an *InactivePropagationBot* it never switches between *Working* or *Propagation* bot. In order not to be detected, the bot is inactive most of the time and only becomes active for a very short period of time. Transitions from *InactivePropagationBot* to *ActivePropagationBot* and back occur with rates k_7 and k_8 , respectively. The transition rates for moving from *InactiveWorkingBot* to *ActiveWorkingBot* and back are denoted k_9 and k_{10} , respectively.

The computer can recover from its infection, e.g., if an antimalware software discovers the virus, or if the computer is physically disconnected from the network. It then leaves the *InactivePropagationBot* or the *ActivePropagationBot* state and moves to the *NotInfected* state with rates k_{11} , k_{12} , respectively. The same holds for the *working bots*; the transition rates from *InactiveWorkingBot* and *ActiveWorkingBot* are k_{13} , k_{14} , respectively.

The transition rates in the SAN model are marking dependent, e.g., the rate of moving from state s_1 to state s_2 linearly depends on the number of computers in state s_1 . The transition rates for the CTMC model of the single computer, as proposed here, are constant, with the only exception of k_1^* , which depends on the number of active propagation bots in the environment; seen from this perspective this CTMC is a non-homogeneous CTMC.

The CTMC model consists of seven states (S_1, \dots, S_7), where each state from the state space $S = \{NotInfected, \dots, ActiveWorkingBot\}$, $|S| = 7$, represents a certain phase of the infection of a single computer. The rate matrix R of the CTMC is written as:

$$R = \begin{pmatrix} 0 & k_1^* & 0 & 0 & 0 & 0 & 0 \\ k_3 & 0 & k_2 & 0 & 0 & 0 & 0 \\ k_6 & 0 & 0 & k_4 & 0 & k_5 & 0 \\ k_{11} & 0 & 0 & 0 & k_7 & 0 & 0 \\ k_{12} & 0 & 0 & k_8 & 0 & 0 & 0 \\ k_{13} & 0 & 0 & 0 & 0 & 0 & k_9 \\ k_{14} & 0 & 0 & 0 & 0 & 0 & k_{10} \end{pmatrix} \tag{1}$$

The $|S| \times |S|$ generator matrix \mathbf{Q} is given as follows: $\mathbf{Q}_{s_1s_2}$ is equal to the transition rate $R_{s_1s_2}$ to move from the state s_1 to the state s_2 and \mathbf{Q}_{ss} is equal to the negative sum of all the rates in row s . The only exception is the rate k_1^* to move from the *NotInfected* state (S_1) to the *InitialInfection* state (S_2). As discussed above, this rate depends on k_1 and on the number of computers in the *ActivePropagationBot* state, but it should not depend on the total number of computers in the environment. We provide an expression for k_1^* in the next section.

In the following, we use the mean-field method to model and study the population of computers in a network that can possibly be infected, assuming that all computers behave individually according to the CTMC model described above. We leave the discussion of actual parameter values to Section 4.

3 Mean-Field Approximation

The mean-field method allows to compute the exact limiting behavior of an infinite population of identical components, and suggests an approximation when the number of components is sufficiently large. The global idea of the method is to describe the behavior of the infinitely large population (overall behavior) via the average behavior of the individual components, which are indistinguishable. Both the overall behavior of the population and the individual behavior of a single component are modeled as a Markov chain, where the transition rates in the overall Markov model depend on the number of components in each state, but not on the total number of components in the system.

Previously we discussed the CTMC (see Fig. 1) and the corresponding state space S which describes the behavior of a single computer in the botnet. The overall behavior of the population of N computers is then given by a CTMC with a state space of size $|S|^N$. This can be lumped due to the identicality of the computers' behavior, and then described by the number of computers in each state $s \in S$ at time t , i.e., by $\underline{M}(t) = (M_1(t), M_2(t) \dots M_{|S|}(t))$, where $M_s(t)$ is the number of computers in state s .

Recall that the generator matrix \mathbf{Q} , as discussed in the previous section, has one transition rate that depends on the environment: k_1^* . We can now express

this k_1^* in terms of $M(t)$ from the overall CTMC, as follows. The total rate of infections produced by all bots that are in the active propagation state is $k_1 \cdot M_7(t)$. These infections are spread out randomly over all not-yet infected computers, of which there are $M_1(t)$ ¹. Hence, the infection rate k_1^* perceived by each individual computer is given by the ratio:

$$k_1^*(\underline{M}(t)) = \frac{k_1 \cdot M_7(t)}{M_1(t)}. \tag{2}$$

The above equation completes the definition of \mathbf{Q} and, hence, allows to build a complete mean-field model. Given a population of N computers, we denote the fraction of objects in an arbitrary state s at time t as $m_s(t) = M_s(t)/N$, where $0 \leq \underline{m}(t) \leq 1$ is the normalized occupancy vector $\underline{m}(t) = (m_1(t), m_2(t), \dots, m_{|S|}(t))$, which does not depend on N . The generator matrix $\mathbf{Q}(\underline{m}(t))$ for the normalized vector $\underline{m}(t)$ is the same as $\mathbf{Q}(\underline{M}(t))$ for the unnormalized vector, since its components depend only the ratios of the vector elements.

We apply the mean-field method for the overall system using Theorem 1 from [10], which states that the normalized occupancy vector $\underline{m}(t)$ of the overall behavior tends to be deterministic in distribution and satisfies the following differential equations when N tends to infinity:

$$\frac{d\underline{m}(t)}{dt} = \underline{m}(t)\mathbf{Q}(\underline{m}(t)). \tag{3}$$

The system of equations describing the population behavior in the botnet then equals:

$$\begin{cases} \dot{m}_1(t) = k_3m_2(t) + k_6m_3(t) + k_{11}m_4(t) \\ \quad + k_{12}m_5(t) + k_{13}m_6(t) + (k_{14} - k_1)m_7(t), \\ \dot{m}_2(t) = -(k_2 + k_3)m_2(t) + k_1m_7(t), \\ \dot{m}_3(t) = k_2m_2(t) - (k_4 + k_5 + k_6)m_3(t), \\ \dot{m}_4(t) = k_4m_3(t) - (k_7 + k_{11})m_4(t) + k_8m_5(t), \\ \dot{m}_5(t) = k_7m_4(t) - (k_8 + k_{12})m_5(t), \\ \dot{m}_6(t) = k_5m_3(t) - (k_9 + k_{13})m_6(t) + k_{10}m_7(t), \\ \dot{m}_7(t) = k_9m_6(t) - (k_{10} + k_{14})m_7(t). \end{cases} \tag{4}$$

Note that in the above, k_1^* is used as in (2), where the entries of the unnormalized vector $\underline{M}(t)$ have been replaced with the corresponding entries of the normalized occupancy vector $\underline{m}(t)$. These equations can be solved analytically, however the closed forms are impractically large. We use the Wolfram Mathematica tool to obtain the analytical solution. The system of ODEs (4) is applicable when N tends to infinity. To obtain the approximation for the case when N is finite but sufficiently large, we use the Corollary 2 from [10], which states, that:

When N is sufficiently large, the normalized state vector of the lumped process, $\underline{m}(t)$, is a random vector whose mean can be approximated by the following differential equation

¹ Note that the above modeling decision was made to match the existing SAN model and may not completely reflect realistic botnet spreading.

Table 2. The setups for the different experiments. Bold font indicates difference w.r.t. baseline experiment.

Parameter	Experiments						
	Baseline	1	2	3	4	5	6
ProbInstallInitialInfection	0.1	0.06	0.04	0.1	0.1	0.1	0.1
ProbConnectToPeers	1	1	1	1	1	1	1
ProbSecondaryInjectionSuccess	1	1	1	1	1	1	1
ProbPropagationBot	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RateOfAttack	10.0	10.0	10.0	10.0	10.0	10.0	10.0
RateConnectBotToPeers	12.0	12.0	12.0	12.0	12.0	12.0	12.0
RateSecondaryInjection	14.0	14.0	14.0	14.0	14.0	14.0	14.0
RateWorkingBotWakens	0.001	0.001	0.001	0.001	0.001	0.001	0.001
RateWorkingBotSleeps	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RatePropagationBotWakens	0.001	0.001	0.001	0.001	0.001	0.001	0.001
RatePropagationBotSleeps	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RateInactiveWorkingBotRemoved	0.0001	0.0001	0.0001	0.001	0.001	0.001	0.001
RateActiveWorkingBotRemoved	0.01	0.01	0.01	0.01	0.01	0.01	0.01
RateInactivePropagationBotRemoved	0.0001	0.0001	0.0001	0.001	0.001	0.001	0.001
RateActivePropagationBotRemoved	0.01	0.01	0.01	0.07	0.04	0.02	0.015

$$\frac{dE(m_i(t))}{dt} \approx \sum_{j \in S} E(m_j(t)) Q_{ji}(\underline{m}(t)), i \in S. \quad (5)$$

Considering this, the expected occupancy vector $E(\underline{M}(t))$ is given as follows: $E(\underline{M}(t)) \approx N \cdot \underline{m}(t)$, where $\underline{m}(t)$ is the solution of (4). In our experiments we set $N = 10^7$.

4 Results

In this section we discuss the mean-field results in detail and compare them to the simulation results we obtained from the model given in [16]. We carried out a similar series of experiments as in [16]; the chosen parameters for all these experiments are given in Table 2.

The simulation of the model was done using the Moebius tool [7]. Each experiment covered one week of simulated time. Each experiment was replicated 1000 times; the mean values and 95% confidence intervals of the measures of interest are shown. The initial conditions for each experiment are as follows: 200 computers are located in the place *ActivePropagationBots* in the SAN, and all the other places are empty. Note that the simulation results shown here differ from those in [16]. Together with the authors of [16] we found a small mistake in the simulator settings they used: because the rates in the SAN model are marking dependent, a flag has to be set in the Moebius tool to ensure that the rates are updated frequently. Not setting this flag can result in inaccurate numbers of propagation bots, as illustrated below in Figure 2.

We use Mathematica [18] to obtain solutions for the set of differential equations (4) coupled with the transition rates from Table 2. To obtain the same

initial conditions for the mean-field model as for the SAN model we need to take the additional state in the CTMC into account. Given an overall population of $N = 10^7$, the fraction of computers in the state *NotInfected* is initialized as $m_1(0) = (N - 200)/N$, the fraction of computers in the state *ActivePropagation-Bot* is initialized as $m_7(0) = 200/N$, and the fractions of computers in all other states are initialized as zero.

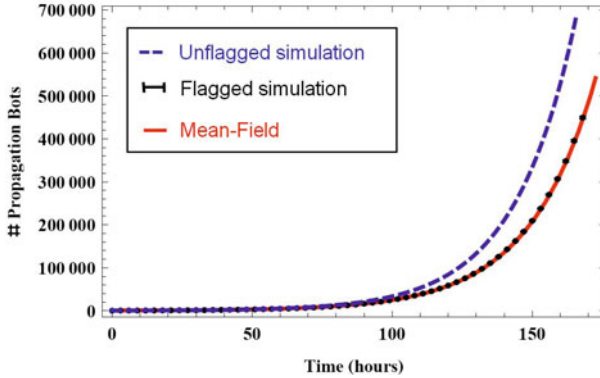


Fig. 2. Number of propagation bots over time in the Baseline experiment obtained from Moebius simulations with and without the rates-updating flag set, as well as obtained from mean-field approximations

Figure 2 shows the number of the propagation bots in a botnet. The number of propagation bots (both active and inactive, states S_6 and S_7) has been taken as measure of interest since they actively infect "healthy" computers. The lower solid line depicts the mean-field results of the Baseline experiment together with the 95% confidence intervals of the corrected Moebius simulation. As can be seen, the mean-field results are very accurate in this case, since they lie mostly within the confidence intervals, even though the confidence intervals are very narrow. The upper solid line represents the mean value of the original Moebius simulation from [16]. Comparing the original Moebius results with the new results from the correct simulator setting, reveals that the number of propagation bots (both *active* and *inactive*) differs from the results stated in [16]; during the first fifty hours the unflagged simulation provides slightly lower results (about 20%), however on this scale the difference is hardly noticeable. Starting from fifty hours, the unflagged simulation over-estimates; after a week the difference is about 42% (754755 vs. 440073). Note that the simulation takes longer than 5 days of runtime, versus 1 second for the mean-field method. We come back to this at the end of the section.

The goal of this paper is not to study the growth of botnets under different conditions, but to compare the results obtained from mean-field approximation with those obtained from simulations. Hence, we compare results for a representative selection of experiments in order to discuss the advantages and disadvantages of both approaches.

To investigate how a reduced infection spread would influence the growth of botnets, Experiments 1 and 2 were done in [16]. The "user factor" (*ProbInstalInfection*) is reduced to 60% and 40%, respectively, as compared to the Baseline experiment to represent a lower probability of, e.g., opening infected files. The results are, together with those from the Baseline experiment, presented in Figure 3. A logarithmic scale has been chosen for the number of propagation bots, in order to better visualize the exponential growth. For both experiments, the results obtained with the mean-field model are very accurate and lie well within the confidence intervals most of the time.

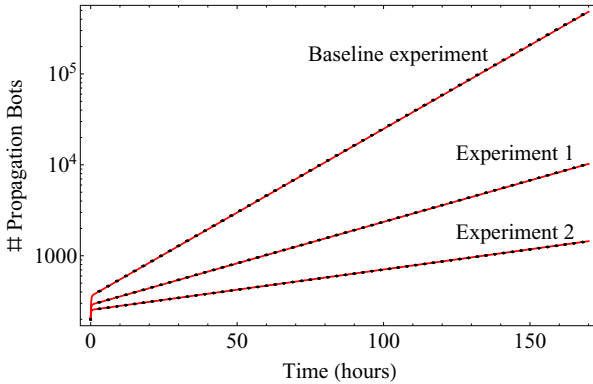


Fig. 3. Number of propagation bots over time in the Baseline experiment and in Experiment 1 and 2 obtained from Moebius simulation and from mean-field analysis

To investigate how efficient anti-malware software can control or even stop the botnet spread, experiments with increased removal rates were done in [16]. To make a comparison of the approaches, we conducted a series of experiments, where the removal rate of active propagation bots varies between 0.01 and 0.1. The mean-field approximation provides an explicit result, which in most of the cases lies well within the 95% confidence intervals (see Figure 4).

At first sight the high accuracy of the analytical results might be surprising, since the underlying assumption of mean-field approximation is that the number of interacting components is large. However, apparently in Experiment 3 (cf. Figure 4) the initial set of active propagation bots hardly gets a chance to infect more computers before being disinfected themselves. In terms of the local CTMC, it means that the transition from the state *NotInfected* to the state *InitialInfection* is taken by (almost) none of the computers. This transition happens to be the only one whose rate depends on the environment; if we remove it from the local CTMC, we are left with a CTMC with constant rates. With all rates in the CTMC constant, the ODEs (4) are easily seen to be the Kolmogorov differential equations, whose solution is the probability distribution over the states of the CTMC as a function of time. Also, removing this transition in the SAN simulation model reduces it to a set of many *independent* CTMCs. Taking the number of markings per state as a function of time, and dividing by the total,

obviously results in an unbiased estimate of the probability distribution of the CTMC in the course of time. Thus, clearly the two approaches should match, as they in fact do and this explains why the mean-field results are still accurate, even though in this case the overall number of components is small.

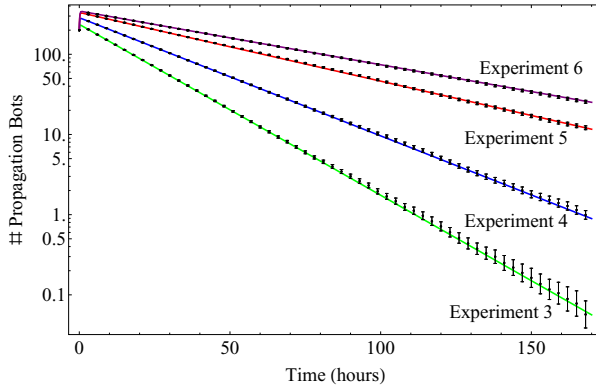


Fig. 4. Number of propagation bots over time in Experiments 5, 6, 7, and 8, obtained from Moebius simulation and from mean-field analysis

It is interesting that the confidence intervals in Experiment 6 are much narrower than the ones in Experiment 3. As the average number of propagation bots decreases over time, the confidence intervals seem to get wider on the logarithmic scale (see Figure 4). In fact, however, the absolute width of the intervals gets smaller, but less quickly than the estimate itself. The reason for this is that the actual number of propagation bots always is a non-negative integer; therefore, when the estimated *average* decreases much below 1, it must be the average of many 0's and a few 1's (or even fewer higher integers). Such an estimate inherently has a large coefficient of variation; in fact, this is the main problem of *rare-event simulation*, cf. [12].

Another thing to remark about these experiments, is that when the number of propagation bots reaches 0, and there are also no bots in the states *InitialInfection* and *ConnectedBot* anymore, no new infections can occur. The number of propagation bots will then remain 0. Thus, when the graph indicates that after a week the *average* number of propagation bots is about 0.01, this means that in most (about 99%) of the simulation runs the botnet is extinct and will stay so, while only a few runs still have some botnet activity.

In Table 3, the computer run times for the simulation and for the mean-field computation are compared. The results were obtained on a Core-i7 processor with 3 GB RAM and 4 cores and hyper threading. One sees that the simulation can take a very long time, namely up to several days, while the mean-field approximation is always done within one second. The difference between the simulation time for the different experiments is due to the marking dependency of the rates. For example, in the Baseline experiment the number of *ActivePropagationBots* is large, hence, the rate of infection becomes very large and more

Table 3. Time spent on simulation and mean-field approximation

Experiment	Simulation	mean-field
Baseline	5 d 3 h 25 min	1 sec
Exp. 1	9 h 51 min	1 sec
Exp. 2	5 h 37 min	1 sec
Exp. 3	31 min	1 sec
Exp. 4	40 min	1 sec
Exp. 5	45 min	1 sec
Exp. 6	36 min	1 sec

time is needed to simulate the resulting large number of events. The time spent on the simulation of the experiments with lower numbers of computers involved is reasonably smaller; however the mean-field approximation is still much faster in all cases. In any case, the simulation times should only be taken as indications, since the simulations were not run completely independently, but with 2, 3 or 4 of them simultaneously on a 4-core computer, so the jobs may have interfered with each other.

5 Exploiting the Speed-Up

In the previous section we have shown how fast and efficient the mean-field method is (cf. Table 3), and that it gives correct results. This allows us to use the mean field method in this section to address problems which are not feasible using simulation: (i) we study the dependence of the botnet spread on two parameters, while the results in the previous section are only functions of time for a given set of parameter values, (ii) and we study the behavior of the botnet in the presence of cost constraints. The purpose of this section is to show the difference between the simulation and the mean-field capabilities, and, at the same time, to show the advantages of the fast analysis.

The authors of [16] used time-consuming simulation to show in a couple of examples that there is no considerable difference in increasing the detection of active or inactive bots (namely increasing the removal rates k_{11} , k_{13} or k_{12} , k_{14}). The mean-field method allows to make the analysis faster and to obtain more information. We calculate the number of propagation bots as a function of k_{13} and k_{14} (see Figure 5). As one can see, there is no considerable difference in a relative increase of one or the other parameter. It is known that inactive computers are much harder to detect (increasing k_{13} is more difficult), therefore the above results might be helpful for the antivirus software developers to find the better strategy for botnet removal.

Next, we introduce a cost concept to analyze the economical side of an infection. In the following, two types of costs are considered: (i) the cost of a computer being infected, that occurs for example due to the loss of information or productivity, and (ii) the cost of more frequent checking with antivirus software. On one hand the number of infected computers, and hence their cost grows if computers are not frequently checked. On the other hand, if computers are checked

too often the botnet is not growing, but running the antivirus software becomes very expensive. We analyze this trade-off in more detail in the following. We calculate the cumulative cost as follows:

$$C(t_0, t_1, RR, D_1, D_2) = \int_{t_0}^{t_1} (D_1 \cdot \text{InfBots}(t, RR) + D_2 \cdot RR \cdot \text{AllComp}) dt \tag{6}$$

where RR is the change in removal rates k_{11}, \dots, k_{14} with respect to the rates in the Baseline experiment, i.e. $k_{11} = RR \cdot k_{11, \text{baseline}}$ and similarly for k_{12}, k_{13}, k_{14} ; D_1 is the cost of infection; $\text{InfBots}(t, RR)$ is the number of infected computers for a given RR , at time t , including active and inactive working and propagation bots; D_2 is the cost of one computer being checked, which probably is much lower than the cost of infection (D_1); AllComp is the number of the computers in the system. We calculate the cumulative cost of the system performance for three days. For RR from the interval $[0.001; 5]$ we calculate the cost as a function of time for given D_1 and D_2 . Results are depicted in Figure 6 and, one can see, that the cost grows exponentially with time and quite linearly with decreasing RR if the computers are not checked frequently (for the RR between 0 and 1). However, if antimalware software is used too often (RR above 2), the cost grows linearly with RR .

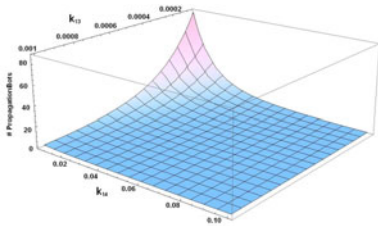


Fig. 5. Number of propagation bots for $(k_{13}, k_{14}) \in [8 \cdot 10^{-5}; 10^{-3}] \times [8 \cdot 10^{-3}; 10^{-1}]$ at time $T = 3 \text{days}$, all other parameters are the same as for Baseline experiment (see Table 2)

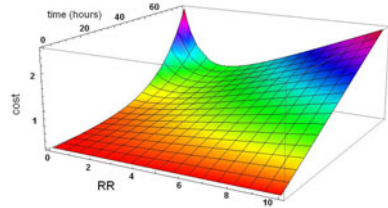


Fig. 6. Cost of the system performance for $D_1 = 0.01, D_2 = 4 \cdot 10^{-5}$

We see that the mean-field method can be easily used for finding the removal rates which minimize the cost at a given moment of time. It can help network managers with careful decision-making, based on the situation at hand. Even though not all parameters might be known in reality, such analysis can help to obtain a better understanding of the characteristics of botnet spread.

In this section we studied different aspects of botnet spread and gained a deeper understanding of the trade-off which occurs when costs are induced. The set of problems discussed in this section demonstrates the efficient application of the mean-field method. One can think of other questions to address, however our aim was to show the potential of the method by addressing problems which can not be solved using simulation.

6 Variations of the Method

As an alternative to the method described in Section 3, we also applied a discrete-time approximation to the model. The uniformisation of the CTMC was done and the corresponding DTMC for the single node behavior was obtained. We used the mean-field convergence theorem from [1] to obtain the approximation. As was expected, the results for discrete time approximation are identical to the results for the continuous time approximation, and therefore omitted here.

In [4], a method is proposed to systematically derive a set of ordinary differential equations governing the concentrations of reactants in (bio)chemical systems. This approach can also be applied to the botnet model, by interpreting each of the 7 states as a chemical "reactant", and each transition between the states as a "chemical reaction". The concentrations then represent the fraction of all computers that are in that particular state; the method allows sufficient freedom in the dependence of the reaction rates on the concentrations to fit the botnet model. Applying the systematic method from [4] to this model results in a set of ODEs that is identical to the equations (4) which we derived using the mean-field approximation.

It turns out that there is a good explanation for the match between this ODE method and the mean-field approximation. The main premise in mean-field analysis is that there is a large number N of identical entities (computers in the botnet case), of which some number are in each state. The quantity of interest then is the *fraction* of the entities that is in each state (the vector $\underline{m}(t)$). Now, a *concentration* in the (bio)chemical context of [4] is also a fraction, namely the fraction of all molecules that are of a certain type (assuming that the number of solvent molecules far exceeds the others, so that the total number of molecules is practically constant). With this interpretation, the two methods are identical. The main difference between the two methods is that in mean-field analysis, the total number of entities is called N and is explicitly taken in the limit to infinity. In contrast, this limit is not made explicit in the (bio)chemistry ODE method; this may be justified by the typically extremely large number of molecules in chemical systems (cf. Avogadro's number).

A similar argument holds for ODEs derived from a PEPA model, as done in [3]. The PEPA model describes a large CTMC, which in fact models many identical computers, each of which can be in one of a small number of states. For deriving ODEs, only the total number per state is taken into account. These total numbers are then treated as continuous rather than discrete variables, which is equivalent to setting the total number to infinity. Although we did not explicitly do so, it is clear that this PEPA-based approach to our botnet model would again result in the same set of ODEs.

7 Conclusion

In this paper, we have compared different approaches for evaluating a Markov model for peer-to-peer botnet spreading.

We have shown that the mean-field approach is much faster than simulation, taking about 1 second instead of minutes to days of computation time. The results from the mean-field analysis match those from the simulation very well, being mostly inside the 95% confidence interval.

Due to the speed-up of the mean-field method we have been able to address various questions which cannot practically be answered with simulation, such as questions involving cost trade-offs; this is useful in typical engineering applications. We also considered several other approaches that can be used for the analysis of large-scale systems, such as automatically deriving ODEs and deriving ODEs from process algebra models, and found them to be equivalent to the mean-field approach.

In general, the mean-field method is only a first-order approximation to the real Markov chain model, which becomes better as the number of entities involved increases. However, in the present model we did not observe any significant difference between the mean-field results and the simulation results. In contrast to the mean-field approximation, the precision of the simulation results suffers when the mean number of bots being estimated, becomes close to zero. This is because the standard deviation does not go to zero as fast as the mean value. Note that this is, in fact, the problem that *rare-event simulation* addresses.

The present research shows the usefulness of the mean-field approach, as it is able to provide very accurate results very quickly. However, even in cases where mean-field results are less accurate for small population numbers, it can be useful as a quick check of the simulation. In fact, the simulator setting problem discussed in Section 4 was found due to the mismatch with the mean-field results.

Future work will involve further exploration of the conditions under which mean-field results are correct. As noted above, even when the number of entities involved was small, our mean-field results remained correct, and we could explain why this was the case. Presumably, more general conditions for such correctness can be found.

Acknowledgments. This work is supported by the Netherlands Organization for Scientific Research (NWO), the Centre for Telematics and Information Technology (CTIT), and the Centre for Dependable ICT Systems (CeDICT). Furthermore, we thank the authors of [16] for providing the source code of their model and discussions on the settings of the simulator.

References

1. Bakhshi, R., Cloth, L., Fokkink, W., Haverkort, B.: Mean-Field Analysis for the Evaluation of Gossip Protocols. In: 6th Int. Conference on Quantitative Evaluation of Systems (QEST 2009), pp. 247–256. IEEE CS Press, Los Alamitos (2009)
2. Bakhshi, R., Endrullis, J., Endrullis, S., Fokkink, W., Haverkort, B.: Automating the mean-field method for large dynamic gossip networks. In: 7th Int. Conference on Quantitative Evaluation of Systems (QEST 2010). IEEE CS Press, Los Alamitos (2010)

3. Bradley, J., Gilmore, S., Hillston, J.: Analysing distributed internet worm attacks using continuous state-space approximation of process algebra models. *Journal of Computer and System Sciences* 74(6), 1013–1032 (2008)
4. Calder, M., Gilmore, S., Hillston, J.: Automatically deriving ODEs from process algebra models of signalling pathways. In: *Proceedings of Computational Methods in Systems Biology (CMSB 2005)*, pp. 204–215 (2005)
5. Cerotti, D., Gribaudo, M., Bobbio, A.: Disaster propagation in heterogeneous media via markovian agents. In: Setola, R., Geretshuber, S. (eds.) *CRITIS 2008. LNCS*, vol. 5508, pp. 328–335. Springer, Heidelberg (2009)
6. Ciocchetta, F., Hillston, J.: Bio-PEPA for epidemiological models. *Electronic Notes in Theoretical Computer Science* 261, 43–69 (2010)
7. Deavours, D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J., Sanders, W., Webster, P.: The Mobius framework and its implementation. *IEEE Transactions on Software Engineering* 28(10), 956–969 (2002)
8. Feamster, N., Gao, L., Rexford, J.: How to lease the internet in your spare time. *SIGCOMM Comput. Commun. Rev.* 37, 61–64 (2007), <http://doi.acm.org/10.1145/1198255.1198265>
9. Garetto, M., Gong, W., Towsley, D.: Modeling malware spreading dynamics. In: *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications, INFOCOM 2003, IEEE Societies, March-3 April 2003*, vol. 3, pp. 1869–1879 (2003)
10. Gribaudo, M.: Analysis of large populations of interacting objects with mean field and markovian agents. In: Bradley, J.T. (ed.) *EPEW 2009. LNCS*, vol. 5652, pp. 218–219. Springer, Heidelberg (2009)
11. Gribaudo, M., Cerotti, D., Bobbio, A.: Analysis of on-off policies in sensor networks using interacting markovian agents. In: *Sixth Annual IEEE International Conference on Pervasive Computing and Communications, PerCom 2008*, pp. 300–305. IEEE, Los Alamitos (2008)
12. Heidelberger, P.: Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation* 5, 43–85 (1995)
13. Henzinger, T.A., Mateescu, M., Mikeev, L., Wolf, V.: Hybrid Numerical Solution of the Chemical Master Equation. In: *Proceedings of Computational Methods in Systems Biology, CMSB 2010* (2010); preprint arXiv:1005.0747
14. Le Boudec, J.-Y., McDonald, D., Mundinger, J.: A generic mean field convergence result for systems of interacting objects. In: *4th Int. Conference on Quantitative Evaluation of SysTems (QEST 2007)*, pp. 3–18. IEEE CS Press, Los Alamitos (2007)
15. Rohloff, K., Basar, T.: Stochastic behavior of random constant scanning worms. In: *Proceedings. 14th International Conference on Computer Communications and Networks, ICCCN 2005*, pp. 339–344 (October 2005)
16. van Ruitenbeek, E., Sanders, W.H.: Modeling peer-to-peer botnets. In: *5th Int. Conference on Quantitative Evaluation of SysTems (QEST 2008)*, pp. 307–316. IEEE CS Press, Los Alamitos (2008)
17. Sanders, W., Meyer, J.: Stochastic Activity Networks: Formal Definitions and Concepts? *Lectures on Formal Methods and Performance Analysis*, 315–343 (2001)
18. Wolfram Research, Inc.: Mathematica tutorial (2010), <http://reference.wolfram.com/mathematica/tutorial/IntroductionToManipulate.html>

WMTools - Assessing Parallel Application Memory Utilisation at Scale

Oliver Perks, Simon D. Hammond, Simon J. Pennycook, and Stephen A. Jarvis

Performance Computing and Visualisation
Department of Computer Science
University of Warwick, UK
{ofjp,sdh,sjp,saj}@dcs.warwick.ac.uk

Abstract. The divergence between processor and memory performance has been a well discussed aspect of computer architecture literature for some years. The recent use of multi-core processor designs has, however, brought new problems to the design of memory architectures - as more cores are added to each successive generation of processor, equivalent improvement in memory capacity and memory sub-systems must be made if the compute components of the processor are to remain sufficiently supplied with data. These issues combined with the traditional problem of designing cache-efficient code help to ensure that memory remains an on-going challenge for application and machine designers.

In this paper we present a comprehensive discussion of WMTools - a trace-based toolkit designed to support the analysis of memory allocation for parallel applications. This paper features an extended discussion of the WMTrace tracing tool presented in previous work including a revised discussion on trace-compression and several refinements to the tracing methodology to reduce overheads and improve tool scalability.

The second half of this paper features a case study in which we apply WMTools to five parallel scientific applications and benchmarks, demonstrating its effectiveness at recording high-water mark memory consumption as well as memory use per-function over time. An in-depth analysis is provided for an unstructured mesh benchmark which reveals significant memory allocation imbalance across its participating processes. This study demonstrates the use of WMTools in elucidating memory allocation issues in high-performance scientific codes.

Keywords: Memory, Multi-core, Tracing, Analysis.

1 Introduction

In the forty-five years since Gordon Moore predicted the rate at which processor transistor counts would increase, the performance of individual processors found in large supercomputing machines has grown by over four orders of magnitude. The designers of supercomputers have used this property, coupled with increased machine scale, to deliver exceptional improvements in compute performance with each successive generation of machine. The performance offered

by such machines, augmented with developments in parallel algorithms, has created significant opportunities in a variety of scientific domains. In many cases, a limiting factor in the rate of improvement has been the availability of sufficient Random Access Memory (RAM) to house the required application datasets.

Historically, the divergence between processor compute performance and memory access time, the so called “memory wall” [13], has been regarded as one of the greatest concerns in computer architecture. However, the use of multi-core processors is presenting several additional memory-related challenges. The most immediate of these to be felt by users of large parallel scientific codes has been the decrease in available memory per-core - a result of the rapid growth in processor core density without matched improvement in memory capacity. More subtle, but nonetheless as important, has been increasing memory latency, a product of increased contention for memory access which results from having additional cores access memory through a single memory sub-system. As the High Performance Computing (HPC) industry looks to the future, these problems look set to become more problematic - additional cores in the form of graphics processing units (GPUs) will introduce memory placement to the list of the concerns as well as placing a severe limits on memory capacity per processing element.

Traditionally, strong scaling - the practice of using additional processors to solve a fixed problem size - has been a technique to address any lack of available memory. In general, this is typically successful if the problem is implemented to decompose of the available processing cores. However, where data is static or decomposes poorly this approach can yield disappointing or even negative results. The authors of [7,8] illustrate how rank-to-rank communication buffers within middleware, specifically the Message Passing Interface (MPI), can consume increasing volumes of memory at scale. Although solution to these problems have been engineered [6], the solution has yet to be generally adopted.

Memory, therefore, continues to pose a challenge in the design and optimisation of parallel algorithms which must scale. Understanding how an application requests, utilises and frees memory during execution and how these requests relate to the design of the underlying algorithm, remains a key activity associated with application design. If memory requirements across an application workflow can be sufficiently understood, opportunities may be created to reduce the memory configuration of a machine during procurement, reducing initial capital expenditure or permitting execution over fewer processor cores.

In this paper we introduce WMTools, a toolkit designed to support the analysis of memory utilisation for parallel applications. This framework consists of the WMTrace memory tracer, a dynamic shared library which requires no source code modification, and a parallel analysis tool for interpreting trace data post-execution. The tools are intended to provide sufficient information to developers to enable the diagnosis of memory allocation over time and by application function. By comparing traces from multiple scales of application run developers are aided in diagnosing which memory allocations scale in size and which are likely to prevent scaling. Such analysis is a vital precursor to the optimisation of application memory utilisation - a key area of focus for the design of parallel applications.

The specific contributions of this work are:

- Presentation of WMTools, a new lightweight toolkit based on application tracing. The distinguishing features of this tool are the ability to record memory allocation traces combined with temporal and stack information without the introduction of significant overhead. This data is sufficient to identify peak memory usage and analyse memory usage over time, on a per function basis, for a set of parallel tasks;
- Profiling and tracing of five established scientific applications and benchmark codes using WMTools. The five codes are selected to cover a variety of programming languages and scientific domains reflecting the generality of our approach and its ability to scale to applications of varying size. In particular we are able to utilise WMTools in assessing memory allocations for these codes with increasing problem size and core count demonstrating how application developers may choose to use our toolkit in identifying potential scaling bottlenecks;
- Detailed application analysis for the unstructured-mesh engineering benchmark, phdMesh. We demonstrate the identification of maximum and minimum high-water marks for the benchmark, indicating a load imbalance; memory use over time, showing high memory use during initialisation but crucially not during the main body of execution; and finally a breakdown of memory use by function at minimal and maximal high-water marks as the number of processor cores is scaled.

The remainder of this paper is structured as follows: in Section 2 we catalogue a short list of previously reported memory analysis tools and summarise techniques which are relevant to this work; Section 3 contains the presentation of our tracing tool, WMTrace; a case study applying our tool to analysing the memory allocation behaviour of several industry standard applications and benchmarks is described in Section 4; in Section 5 we present a more in-depth investigation into a single code, via information obtained with WMTrace; finally, we conclude the paper with a summary of our findings in Section 6 and identify areas for further work.

2 Related Work

Memory has the potential to be a significant bottleneck for HPC centres - in many cases a lack of memory capacity can act to place a limit on scientific delivery and in-sufficient attention to memory use can be a contributor to poor code performance. Due to this, a collection of tools already exist to assist developers in analysing application memory usage and management within their code.

The main distinction between memory tracing tools is the level of analysis performed and the granularity of the data collected. Broadly, memory analysis tools form two classes - lightweight and heavyweight, depending on their overheads and analysis depth.

The tools which provide the most detailed level of data collection have an inherent overhead in either additional memory consumption or runtime – and

often both. The large volumes of data generated may also require extensive post-processing to derive any meaningful interpretation from the results. This class of tools often collects data at the hardware counter level, and may require code instrumentation. The alternative category of tools attempt to avoid this overhead and are thus limited in the data they can collect. As lightweight tools, they are often loaded dynamically at run time and therefore do not require code instrumentation.

The closest tool to WMTools is the memP library from Lawrence Livermore National Laboratory (LLNL) [1]. memP is a lightweight library designed for collecting basic memory consumption information. The primary aim is to collect high-water mark usage data from all of the processes in a parallel job, which it achieves by re-implementing the memory management functions to allow tracing. All of the data is collected and stored within in-memory data structures, eliminating the need for a post-processing stage. Maintaining these in-memory data structures introduces some performance overhead and additional memory consumption. The resulting data set is minimal, providing maximal heap and stack usage across the processes together with aggregated statistics over this data (*e.g.* standard deviation and coefficient of variation).

Where WMTools differs from memP is that it does not store the complete data-set in memory, but instead streams it to file, thus minimising overheads and having as little influence on the execution of the program as possible. As statistics are not generated upon job completion, WMTools utilises a post-processing phase to extract memory usage statistics and has the facility to provide further in-depth analysis which is difficult to compute efficiently during application execution.

Another tool, mprof, provides a lightweight framework for memory analysis with greater granularity than memP. mprof was written when memory constraints prevented the tool from consuming much system resource, with the authors claiming to require only 50KB of additional memory and incurring a maximum 10x slowdown [14]. This tool also favours in-memory data structures as, at the time of writing, streaming trace information to disk was too expensive. One of the main limitations of mprof is it provides a stack traversal to a fixed depth (currently five). In codes which make use of external libraries, a stack depth of five is often insufficient to reach user code. WMTools removes any arbitrary limit on stack depth, allowing for greater accuracy in analysis.

The volume of data generated by stack tracing tools has been a topic of detailed research, with a number of methods to reduce or compress stack data being proposed [3,4]. Many of the techniques developed employ *a-priori* knowledge about the data structures and repetition of the data within the trace. The data management approach used in WMTools is two-stage, firstly, a custom stack-dictionary is maintained to eliminate repetitive writing of stack information and secondly, trace information is streamed to file using the popular Z-lib library. In our experimentation this approach balances reducing trace size without significant impact on runtime.

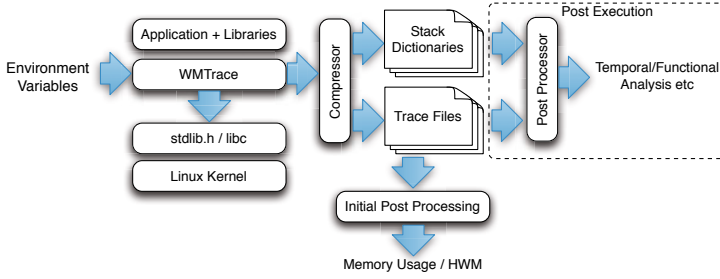


Fig. 1. WMTrace and Analysis Workflow

Many modern heavyweight analysis tools are built upon dynamic binary instrumentation (DBI) frameworks, such as Valgrind [11] and Pin [9]. These DBI frameworks facilitate the use of shadow memory, where either meta data regarding access frequency or even value representation is stored for every byte of application allocated memory. This is exploited by many different shadow memory tools to detect accesses to un-addressable memory, perform type checking and identify data races amongst others. The overheads of such tools are widely acknowledged to be large; the authors of Memcheck document a mean slowdown of 22.2x [10]. Tools like Memcheck introduce overheads by monitoring memory allocations and accesses, and while this is partially the case with WMTTools we face the additional overheads of data generation and storage. Differences in the functionality of the two tools mean that WMTTools does not incur the same mid-execution processing costs as Memcheck, reducing the overall runtime overheads.

3 The WMTrace Library and Analysis Framework

Memory allocation analysis using WMTTools is conducted via a two-stage process (shown in Figure 1). In the first stage, a parallel application is executed with the WMTrace tracing library, linked at runtime by the operating system linker. Immediately prior to execution, calls to the standard POSIX memory handling functions (`malloc`, `calloc`, `realloc` and `free`) are dynamically linked to those exported by WMTrace. Each function is implemented within WMTrace by recording the memory allocation/free event in the tool’s internal buffer and passing the call through to the operating system libraries. The interposition of functions using this approach results in the tracing tool being application and implementation language agnostic and therefore applicable to any application using conventional POSIX memory management. Recorded events are periodically written to a compressed per-process trace file for the second stage of the framework – post-execution analysis.

In this post-execution stage (described in Section 3.2), events in the trace file are read serially and each allocation event attributed to the top level function from the call stack. The returned pointer address from the allocation is stored with each event (at runtime) so that subsequent deallocations can be correctly mapped, allowing the correct calculation of freed memory.

Each aspect of WMTrace’s behaviour is configurable at runtime through the specification of environmental variables. This permits flexible levels of runtime tracing, including the ability to disable stack recording or alter trace compression behaviour.

3.1 Stage 1: Memory Event Tracing

As illustrated in Figure 1, memory events are traced in WMTrace through the use of dynamic function interposition, in which POSIX memory handling functions are intercepted and recorded prior to being passed to the operating system for execution. WMTrace utilises an internal, compile-time configurable buffer to temporarily store event traces in memory during execution. Each MPI process maintains a unique buffer and operates on a dedicated trace file ensuring that the overhead associated with recording trace information is as low as possible.

Memory events recorded by WMTrace are stored as a series of ‘frames’ which permit rapid movement through the trace file during post-execution analysis. Currently four types of events are stored, which map to the four main POSIX memory handling functions: `malloc`, which allocates a single memory block; `calloc`, which allocates a contiguous block of memory for a set of items; `realloc`, which frees a previously defined block of memory and allocates a second block; and `free`, which frees previously defined memory blocks and returns the memory to the unallocated system pool. Each event is stored in a fixed length frame containing the event specific data along with a time stamp, for temporal analysis. For events where memory is allocated the call site stack is recorded to facilitate a functional breakdown during the analysis phase.

The potential volume of data generated by stack tracing tools maybe very large if the application being traced makes frequent calls to memory handling functions, has a deep call hierarchy or runs for a considerable amount of time. The WMTrace library reduces stack trace output through a two-stage compression process designed to balance trace file size with runtime overhead. In the first stage, a custom stack dictionary is used to relate a specific stack to an identifier. Each new stack is added to the dictionary and assigned an identifier so that, if it is encountered at a later point of execution, the identifier can be used. Since scientific codes typically iterate, the probability of encountering stacks later in execution is high, enabling considerable compression to occur through custom stack handling. Data is then written to an internal tracing buffer, when full a second compression stage takes place using Z-lib [5]. Our empirical tests indicate that these two stages balance the size of the trace file with increase in runtime overhead and enable us to provide lossless compression for improved accuracy during the analysis stage.

3.2 Stage 2: Post-Execution Trace Analysis

Following the tracing of a parallel application, the second stage of our framework, WMAAnalysis, conducts analysis over the compressed trace files – one for each MPI process. In this stage, the events of each trace file are read serially and a map of allocations to size of requested memory is created on a per-function basis.

Post-processing takes place in two phases; the first is an initial analysis to establish the application high-water mark memory usage and identify threads of interest and the second is a targeted in-depth analysis which generates data over time or by application function.

The initial post-processing is performed in parallel at the end of job execution, and provides aggregated statistics on the memory usage of the application, including high-water mark for each process, maximal, minimal and mean high-water mark, and standard deviation between high-water mark values. No temporal or functional analysis is performed during this phase to decrease the time to generate initial results.

During the second post-processing phase temporal and functional analysis is performed using the stack data associated with each event. A record of the allocation, and the address attributed to it, is then added to the function's list. When a `free` event is encountered, the address associated with the event can then be searched for in memory and the appropriate block of memory removed from the function's list. Therefore, at any point during the analysis we have a complete record of the memory requested by each function in the application. We are also able to utilise memory requests performed by each function to estimate the memory requirements of each library utilised by the application – a factor often overlooked by application developers who treat libraries as black-box entities. The time-stamp associated with every event enables further temporal analysis.

This in-depth post-processing can have significant runtime requirements depending on the granularity of the analysis, motivating execution as a later post-processing stage. The structure of the trace files allows this in-depth analysis to be performed at a later date or at an alternative location as no dependencies to the machine or code are maintained.

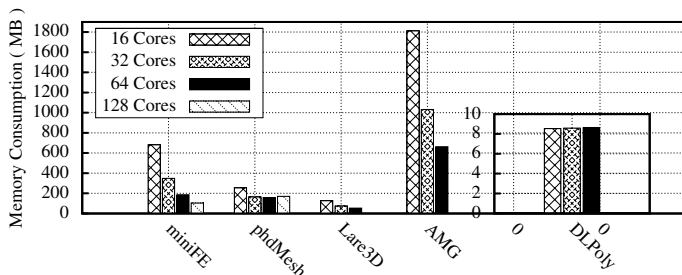
The motivation for separating the analysis from the job execution is to limit the runtime impact of the tracing tool. It also allows the analysis to be performed multiple times, at different granularity levels, on the same trace output, allowing the user to gradually refine their investigation. This separation also reduces the volume of data stored in memory during job execution, thus limiting disruption to cache and helping to reduce context switching.

4 Case Study

In the following section we illustrate the usage of WMTTools with five different scientific applications and benchmarks, identified in Table 1. Either independently or as part of a larger workflow/application, the routines represented by

Table 1. Applications and Benchmarks used for Tracing

	Language	Description
miniFE	C++	Unstructured finite element solver
phdMesh	C++	Unstructured mesh contact search
DLPoly	Fortran 90	Molecular dynamics simulator
Lare3D	Fortran 90	Non-linear molecular hydrodynamics
AMG	C	Parallel algebraic multigrid solver

**Fig. 2.** Peak Memory Consumption on Minerva

these codes use significant proportions of the compute time at supercomputing sites ranging from universities to national laboratories such as Daresbury and EPCC in the UK or Lawrence Livermore and Sandia National Laboratories in the United States. These codes are also interesting from a technical perspective because they represent the three principle implementation languages – C, C++ and Fortran 90 – which are used to write modern parallel scientific applications. The ability to successfully trace each of these languages is critical if our tool is to be generally applicable to HPC codes. For each of these codes an appropriately sized problem has been created and executed with the WMTrace library to record memory behaviour over a variety of core configurations. To illustrate how the memory consumed alters as scale is increased, the problem has been strong scaled over 16, 32, 64 and 128-core executions.

The following experiments were performed on the recently installed Minerva supercomputer, located at the Centre for Scientific Computing (CSC) at the University of Warwick. This machine comprises of 258 dual-socket, hex-core Intel Westmere-EP X5650, nodes connected via Infiniband. Each node provides 24GB of system memory (2GB per core). The runs presented utilised the GNU 4.3.4 compiler toolkit with OpenMPI 1.4.3, using the `-O3` optimisation flag and debugging symbols.

This case study illustrates the ability of WMTools to calculate memory high-water marks and analyse memory consumption over time. We demonstrate how this is achieved with minimal overheads, whilst still producing an accurate result, when compared with the alternative tool memP.

4.1 High Water Mark

Figure 2 presents the peak high-water mark, the maximum high-water mark of all processes within a job, of the selected codes. Note that the scaling of memory

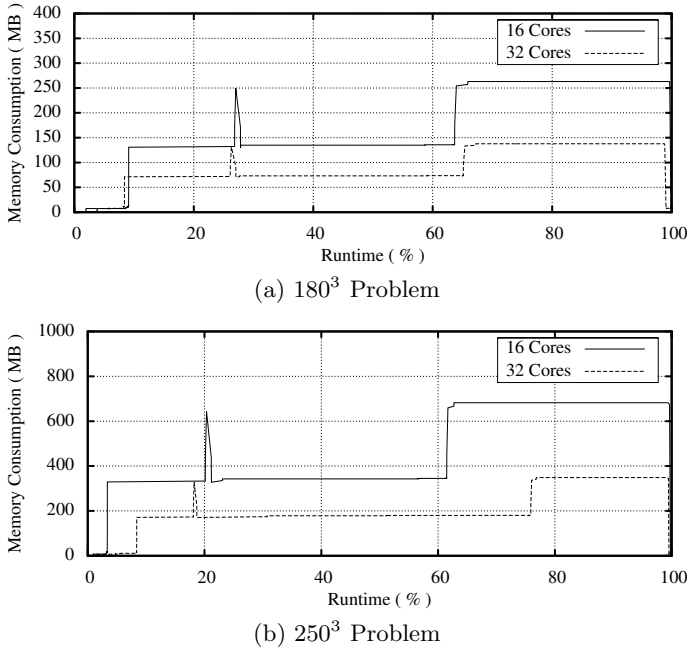


Fig. 3. Temporal memory trace for miniFE at different problem sizes and scale

allocations is not consistent for all codes. Whilst the memory utilised by miniFE scales well with increasing core count, the memory usage of phdMesh actually increases when moving from 64 to 128 cores. The usage of memory in DLPoly remains fairly static (around 8.5MB) despite an increase in the number of cores.

4.2 Memory Usage over Time

In Figures 3a and 3b we present the memory utilisation of the miniFE benchmark over the course of runs on 16 and 32 cores for a 180³ and 250³ problem size respectively. The application runtime has been made relative to permit direct comparison of the shape of memory utilisation between the runs (hence the x-axis represents percentage of runtime). miniFE’s behaviour is characterised by a startup phase in which the application creates and fills a mesh before assembling the Finite Element data (prior to solving). On our graphs this is represented by the initial spike - in which matrices are created. This is followed by a period of population in which no additional memory is allocated, and then an increase in memory as the data is copied into matrices for solving using the CG-method.

4.3 Tool Overhead and Comparison

To measure the overheads of WMTrace we time the execution of the code both with and without the profiler loaded. We also time code execution with the memP profiler loaded, to give a comparison of tool performance.

Table 2. Runtime slowdown comparison: WMTrace and memP

	Cores	Runtime (Secs)	WMTrace			memP
			Slowdown (x)	Post-Processing (Sec)	Trace Size (MB)	Slowdown (x)
miniFE	16	22.67	1.01	1.21	32	1.13
	32	17.56	1.09	1.67	53	1.00
	64	6.00	1.34	1.03	134	2.90
	128	5.07	1.40	1.40	257	3.00
phdMesh	16	35.06	9.17	19.26	3072	80.43
	32	29.85	8.90	19.64	3994	30.79
	64	18.23	10.36	14.09	5018	20.24
	128	13.17	11.50	6.83	6042	11.25
DLPoly	16	57.76	1.81	1.97	331	1.06
	32	79.10	1.43	2.72	421	1.00
	64	107.57	1.26	2.76	894	1.07
Lare3D	16	2069.37	1.00	3.66	51	1.00
	32	1066.85	1.00	2.07	97	1.01
	64	579.14	1.00	1.61	168	1.05
AMG	16	442.37	1.08	4.59	482	-
	32	164.79	1.11	4.85	494	13.09
	64	76.58	1.06	4.99	525	8.28

Table 2 illustrates the overheads introduced by WMTrace. It is clear to see that the tool performs much better on some codes than others; the mean slowdown ranges from 1.0x for Lare3D to 11.5x for phdMesh. We also shows our runtime overheads are in line with those of memP, if not slightly lower. Our initial post-processing overheads are also presented to illustrate the minimal runtime impact of parallel post-processing at the end of job execution. The overly high runtime overheads introduced into phdMesh are attributed to the number of memory functions intercepted. This is in part due to the nature of the code, as it performs multiple allocations, specifically C++ object initiations, in each iteration, and then destroys them at the end of that iteration. A trend to note in these results is that as the number of processors is scaled the slowdown factor worsens. This is due to the compression and I/O times for the trace files. In Section 4.4 we discuss output size in more detail but it is clear that the total volume of data generated scales with the core count, whilst the application runtime generally decreases. The results presented represent a significant improvement over our previous published results [12], both in terms of runtime and tool slowdown factor. We attribute these results in part to the transition from Woodcrest (X5160) to Westmere (X5650) and to engineering improvements to our tool set.

For validation purposes we evaluated the high-water mark results of WMTools against those of memP, and found a consistent over prediction by around 7MB, we attribute this to our capture of allocations ignored by memP. We believe that our pessimistic result is preferable to the optimistic memP result, as it guarantees execution when close to a memory limit.

As an initial comparison to a heavyweight tool we compare WMTools to two memory tools from the Valgrind suite: Massif, a heap profiler, and Memcheck, a memory error detector. Massif is designed to provide a similar level of analysis to both WMTrace and memP but incurs the overhead of the Valgrind framework whilst Memcheck provides different functionality, focusing on leak detection and

invalid accesses, but illustrates the cost of the shadow memory method of memory tracking. We compare the overheads of the three tools for the execution miniFE during a serial execution. We experienced a 5x slowdown with Massif and a 20x slowdown with Memcheck, for WMTrace we did not experience any observable slowdown. The performance overheads of the Valgrind tools are in-line with those discussed on the Valgrind website [2], between 5x and 100x, and Nethercote’s study of Memcheck [10], 22.2x.

The memory consumption of the WMTrace library is marginal with the primary consumer of memory being the output buffer. The size of this buffer can be varied to trade off memory consumption and volume of I/O operations. For our experiments a buffer of 5MB was used as a suitable trade-off. The custom stack dictionary is stored in memory, to allow for fast comparison, the memory required for this structure varies with the number of unique call stacks, but was benchmarked between 453KB and 1.2MB per process during the experiments presented.

4.4 Compression

An important feature of WMTrace is the ability to compress trace data on the fly. Through the use of Z-Lib, WMTrace is able to obtain around 24x compression on each output trace file. As the number of cores is scaled these trace files do not tend to shrink in size for all codes, but as there are more processes generating trace files the total volume of data scales roughly with the core count. With a mean compressed trace file of 20.5MB, and a maximum of 213MB, per core compression this represents on average a 2.5x improvement over our previous work, but data storage remains an important factor. Compression achieved through the custom stack dictionary is vital due to the depth of the call stacks. During these experiments we witnessed an mean stack depth of 10.3 with a maximum depth of 35 occurring in a 128 core run of phdMesh.

5 Analysis

In Section 4 we identified a problem with the memory scaling of phdMesh. Using information collected via WMTrace we conduct an in-depth analysis into why the memory requirements increase as the core count is scaled. Figure 2 illustrates that phdMesh requires more memory to run on 128 cores than it does for 32 or 64 cores. Firstly we study the variance between the high-water marks of all processes within a job, to identify if this memory increase is endemic to all processes or if there is variation. An increase in high-water mark for all process may indicate the storage of per process information (*e.g.* communication buffers), whereas an increase within a minority of processes may be indicative of a memory imbalance as a result of data set decomposition.

Table 3 shows how the high-water mark values vary for each process in a run of phdMesh as we scale the number of cores. Increasing the core count reduces both the minimum and mean high-water marks, whilst the maximal high-water mark

Table 3. Per process memory HWM comparison for phdMesh

Cores	Min (MB)	Max (MB)	Mean (MB)	Std. Dev. (MB)
16	239.50	255.26	251.54	4.33
32	149.34	164.96	161.75	3.56
64	122.40	158.46	154.87	5.14
128	92.18	170.21	139.49	18.14

increases for 128 cores. The standard deviation between the high-water mark values identifies large discrepancies between memory consumption on different processes, particularly in the case of 128 cores. This is indicative of problems in the data decomposition.

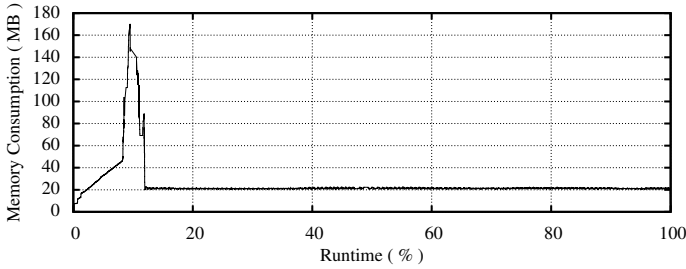
By studying the temporal high-water mark analysis of phdMesh on 128 cores (see Figure 4) we can analyse the differences in memory consumption between the maximal and minimal high-water mark processes. Figure 4a shows the maximal high-water mark thread, with the memory consumption of 170MB; Figure 4b the minimal high water mark process, at 92MB. It is clear that both processes have a very similar temporal memory trace, despite the difference in peak memory consumption.

We see a start up phase with significantly increased memory consumption, until around 15% of their execution, than the sustained consumption after this point. Despite the large variation in high-water mark values (an 85% increase from minimal) the sustained memory consumption is very similar, at around 20MB. At the end of the start up phase in phdMesh a re-balance is performed – to ensure a consistent decomposition – which coincides with the decrease in memory from our temporal analysis. This is suggestive of the application preloading data which can then be discarded for the actual computation phase. It is highly likely that this operation could be arranged in a more efficient configuration which would massively reduce the application’s high-water mark, and the initial variation between processes.

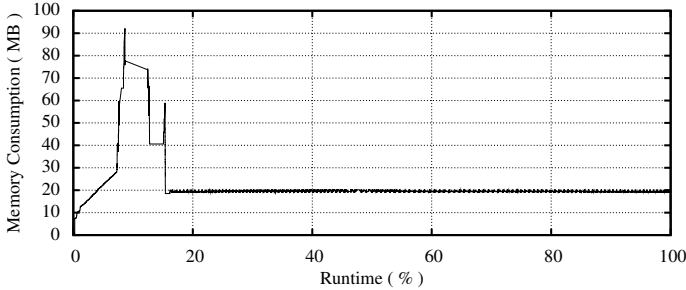
To aid in the start of redesign, we analyse the functional breakdown of the high-water mark for both the maximal and minimal processes for each run of phdMesh (see Figure 5). From this breakdown, we see that – despite the variations in memory consumption – the proportions of each function remain similar between the maximal and minimal high-water mark threads and between runs of different size. This indicates that the memory consumption is distributed across all of the primary functions, rather than just being limited to a single function involved in the start up. Although this suggests it would be difficult to uncouple the data causing the high-water mark from the algorithm, it does suggest that the initial problem set distribution is the root cause of the high-water mark.

6 Conclusions

The diverging gap between compute-processor and memory-chip performance has been a well documented feature of computer architecture literature for some time. As processor designers have utilised multi-core chip design to improve compute performance still further, a series of additional concerns in architecture



(a) phdMesh maximal HWM on 128 cores



(b) phdMesh minimal HWM on 128 cores

Fig. 4. Temporal memory trace for phdMesh illustrating memory variation

design have arisen. First, that the slow rate of improvement in memory capacity has resulted in a reduction in the memory available per-core, and, second, that the increase in core density has resulted in higher levels of contention for memory channels. When combined with the increasing scale of contemporary supercomputers, which is placing pressure on the implementation of middleware, the efficient utilisation of memory at runtime is rapidly becoming a concern for the design of applications which must scale.

In this paper we present WMTTools, a parallel application memory utilisation analysis framework, comprised of a memory allocation tracing tool, WMTrace, and analysis tool – WMAAnalysis. This tool enables users to trace calls to POSIX memory handling functions in distributed MPI codes without modification to application source or recompilation being necessary. The second half of this paper describes a case study in which we apply WMTTools to tracing the memory allocation behaviour of five applications and scientific benchmarks. The results of this study demonstrates the use of WMTTools in:

- Analysing the allocation and freeing of memory during application execution. The ability to track memory use over time represents a clear advantage over existing tools which report only aggregated statistics such as high-water mark and, more importantly in the context of diminishing memory per-core, the opportunity to investigate isolated points during execution where memory usage spikes. In our study we utilised this technique to show the memory usage of the miniFE benchmark as the problem size and core count is varied;

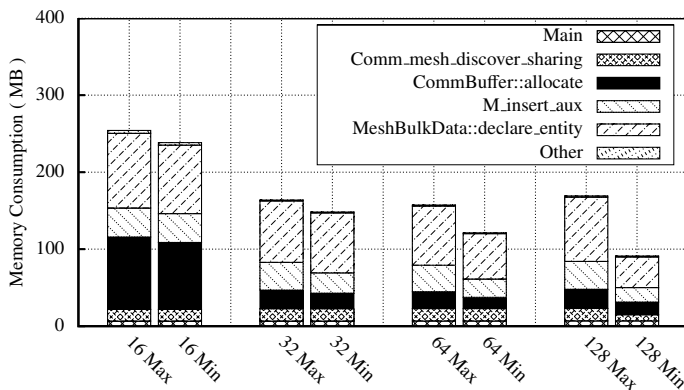


Fig. 5. Functional breakdown of phdMesh for minimal and maximal HWM

- Comparing the high-water mark memory usage between codes and processor core-counts. The direct comparison between different applications which are present in a workflow is vital during procurement when memory capacities per-core must be specified. Machine designers may choose to trade memory capacity and runtime for reduced cost - high-water marks are vital if this is to be achieved accurately;
- Conducting in-depth analysis of memory consumption per-function. By recording the call stack leading to each allocation request, WMTrace offers the ability to relate memory requests to each function and to do so over time. This is a pre-requisite activity associated with the optimisation of memory use as the functions which contribute most to the high-water mark can be addressed in turn. In our study we demonstrated how such an activity may be performed on phdMesh - an unstructured mesh Engineering benchmark. This code makes large requests at the initialisation stage before an efficient decomposition can be found. Further investigations of the memory utilisation of each function across the parallel execution were able to demonstrate potential load imbalance leading to an increased high water mark on some nodes.

As we continue to develop WMTools and apply it to larger and more sophisticated production applications we expect to extend the framework described to conduct further types of analysis - in particular, the matching of allocations and memory-free requests will enable memory leaks to be diagnosed. We are also actively investigating the potential use of WMTrace and memory shadowing to investigate the relationship between memory allocation requests and use by either the function or its child-calls since memory optimisation opportunities may exist at later points in execution.

WMTools is a memory tracing framework which supports the tracing of POSIX memory allocation requests. The tool is able to dynamically attach to existing application binaries and perform tracing with low levels of overhead - in many cases overheads are comparable or lower than equivalent tools. The use

of a post-execution analysis step which utilises traces recorded during execution allows for considerably deeper levels of analysis to be conducted. In this paper we have demonstrated how memory allocation over time and by-function can be generated to support the study and, potentially, optimisation of memory use - an activity we expect to become commonplace in the future development of parallel applications at scale.

References

1. memP (2011), <http://sourceforge.net/projects/memp/>
2. Valgrind (2011), <http://valgrind.org/info/>
3. Budanur, S., Mueller, F., Gamblin, T.: Memory Trace Compression and Replay for SPMD Systems using Extended PRSDs. SIGMETRICS Perform. Eval. Rev. 38, 30–36 (2011)
4. Burtscher, M.: VPC3: A Fast and Effective Trace-Compression Algorithm. SIGMETRICS Perform. Eval. Rev. 32, 167–176 (2004)
5. Deutsch, P., Gailly, J.-L.: ZLIB Compressed Data Format Specification (version 3.3). Request for Comments RFC:1950, Internet Engineering Task Force (IETF) (May 1996)
6. Koop, M., Jones, T., Panda, D.: Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach. In: Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGRID 2007), pp. 495–504 (May 2007)
7. Koop, M.J., Sur, S., Gao, Q., Panda, D.K.: High Performance MPI Design using Unreliable Datagram for Ultra-scale InfiniBand Clusters. In: Proceedings of the 21st IEEE/ACM International Conference on Supercomputing, ICS 2007, pp. 180–189. ACM, New York (2007)
8. Liu, J., et al.: Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In: Proceedings of the 2003 ACM/IEEE International Conference on Supercomputing, SC 2003, p. 58. ACM, New York (2003)
9. Luk, C.-K., et al.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In: Programming Language Design and Implementation, pp. 190–200. ACM Press, New York (2005)
10. Nethercote, N., Seward, J.: How to Shadow Every Byte of Memory used by a Program. In: Proceedings of the 3rd International Conference on Virtual Execution Environments, VEE 2007, pp. 65–74. ACM, New York (2007)
11. Nethercote, N., Seward, J.: Valgrind: a Framework for Heavyweight Dynamic Binary Instrumentation. In: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2007, pp. 89–100. ACM, New York (2007)
12. Perks, O., Hammond, S.D., Pennycook, S.J., Jarvis, S.A.: WMTrace - A Lightweight Memory Allocation Tracker and Analysis Framework. In: Proceedings of the UK Performance Engineering Workshop (UKPEW 2011) (2011)
13. Wulf, W.A., McKee, S.A.: Hitting the Memory Wall: Implications of the Obvious. SIGARCH Comput. Archit. News 23, 20–24 (1995)
14. Zorn, B., Hilfinger, P.: A Memory Allocation Profiler for C and Lisp Programs. In: Proceedings of the Summer 1988 USENIX Conference, pp. 223–237 (1988)

On Stochastic Fault-Injection for IP-Packet Loss Emulation

Philipp Reinecke and Katinka Wolter

Freie Universität Berlin
Institut für Informatik
Takustraße 9
14195 Berlin, Germany
{philipp.reinecke,katinka.wolter}@fu-berlin.de

Abstract. Injection of IP packet loss is a versatile method for emulating real-world network conditions in performance studies. In order to reproduce realistic packet-loss patterns, stochastic fault-models are used. While fault-models can be derived from measurements, inappropriate implementation may introduce artifacts in the experiment process that might invalidate results. In this paper we study the effects of different fault-models in different experiment setups. We illustrate that care should be taken to select an appropriate fault-injection method for the scenario under study.

Keywords: Stochastic Fault-Injection, Gilbert-Elliot model, Packet Loss, Testbeds.

1 Introduction

Testbed-driven performance and dependability evaluation of distributed systems requires the emulation of a realistic networking environment. Common disturbances such as packet loss may have an adverse effect on both dependability and performance of the network. As illustrated in e.g. [1, 2], packet loss may affect dependability of higher networking and system layers even with the reliable TCP protocol, which guarantees reliable data transmission. Consequently, methods for reproducing disturbances are required.

Injection of IP packet loss is an indispensable tool when evaluating fault-tolerant network protocols. However, fault injection at the IP level also provides a convenient way for assessing performance and reliability of distributed systems in which the network stack is considered just an off-the-shelf component. As the injected faults force the network stack to apply the same fault-handling procedures that are executed under real-world operating conditions (such as e.g. packet retransmissions in TCP), the same operational patterns of the network stack that are present in a real network can be expected to emerge in the testbed.

IP packet loss patterns have been the focus of intense study in the past decades (e.g. [3–5]). While in the simplest case packet loss may be described by a Bernoulli model, packet loss is often comprised of bursts of elevated loss

probability, which can be modelled more closely with Gilbert-Elliot (GE) models [3–5]. A Gilbert-Elliot model describes the loss process as a Markov Chain with different loss probabilities for each state.

Our focus in this paper is on the application of observed fault-models in fault-injection driven performance and dependability studies of distributed systems. Considering three examples of typical scenarios differing in their communication patterns, we investigate the impact of the choice of fault-model on the obtained results. The remainder of the paper is structured as follows: In Section 2 we describe common methods to measure and inject packet loss and introduce the three fault-injectors we compare. In Section 3 we evaluate the impact of faults injected with each of the three FI methods on different types of arrival streams.

2 Measurement, Modelling, and Injection of IP Packet Loss

In order to inject realistic loss patterns, an understanding of loss patterns in real-world networks is required, which can be obtained by measurements. There are a number of methods for obtaining packet-loss traces. Measurements may be performed using monitoring approaches [6–8], router measurements [9], and probes [3, 4]. Since monitoring and router measurements require a dedicated infrastructure, probes are often the preferred approach to gather large, representative data sets. Probe-based approaches typically consist of one host sending numbered probe messages to another host. The receiver either records their arrival or replies with an acknowledgment to the sender. Insight into the characteristics of the loss process is then obtained by an analysis of the recorded traces of sent and received packets. While implementation details may vary (e.g. one could use the Ping utility or the Zing utility [3] for different arrival processes), measurement studies using this methodology share the common property that the arrival process is independent of the loss process. That is, the frequency of probe packets is not influenced by packet loss.

2.1 Packet-Loss Models

The result of a measurement study is typically a trace of the observed loss process. While such traces may be used for fault-injection by simply replaying them [10], this method suffers from a number of shortcomings [11]. In particular, sharing of traces is difficult due to their size and privacy issues, and traces can be neither generalised nor parameterised. Consequently, one typically derives models that describe important characteristics of the traces and uses these models in further evaluation steps. In the literature there exist two common models for the IP packet loss process [12]: Bernoulli loss models and Markovian loss models.

Bernoulli Loss Models. Bernoulli loss models describe the packet loss process as a sequence of Bernoulli trials with identical loss probability l . For each packet, a Bernoulli trial is performed, and the packet is dropped according to the outcome of the trial.

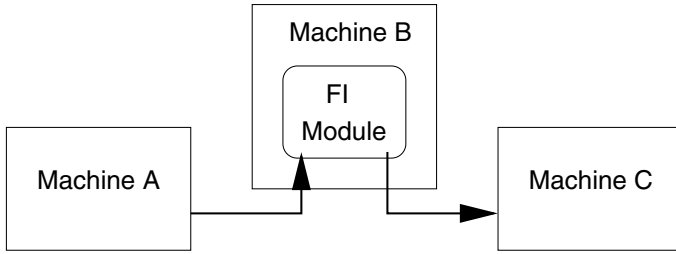


Fig. 1. General approach to packet-level fault-injection

Markovian Loss Models. Since traces often show burstiness in the loss process, that is, interchanging periods of different loss rates [3, 4], Markovian models have been proposed as an alternative to the simple Bernoulli model. With Markovian models, the loss process is described by a Markov chain of length n whose states s_1, \dots, s_n correspond to Bernoulli trials with different loss probabilities l_1, \dots, l_n . Packet traces are modelled by parameterising the model such that it reflects the distribution of the length of times for which loss levels have been stationary. In the simplest case, one can model lossy and loss-free periods [3] individually by a two-state Markov chain, i.e. assume loss probabilities $l_1 = 0$ and $l_2 = 1$ for the two states. However, larger models may capture the loss process better [4]. By convention, Markovian loss models are often called (extended) Gilbert or Gilbert-Elliot models. A good overview of such models is given in [5].

2.2 Stochastic Fault-Injection

The most straightforward way of injecting packet loss according to a loss process described by a loss model is depicted in Figure 1: One computer is set up to act as a router between two or more hosts or networks. On this machine, the TCP/IP stack is augmented by a fault-injection (FI) module. For each arriving outside packet the FI module determines whether to forward the packet to its destination or to discard it. With a Bernoulli model, for each packet one trial is performed and the packet is dropped according to the outcome of this trial. With a Markovian model, the dropping probability for the Bernoulli trial is selected according to the current state of the model when the packet arrives.

2.3 The Problem with Markovian Models in Fault-Injection

Note that in the above we have omitted describing when state-changes occur in the fault-injector's Gilbert-Elliot model. In fact, so far we have intentionally refrained from mentioning the time domain for Markovian models at all.

In the literature (e.g. [5]), Gilbert models are defined as embedded Discrete-Time Markov Chains (DTMCs). The left part of Figure 2 shows an example of a two-state discrete-time Markovian loss model with loss probabilities l_1 and l_2 and state transition probabilities p_{12} and p_{21} . In a DTMC loss model, state changes

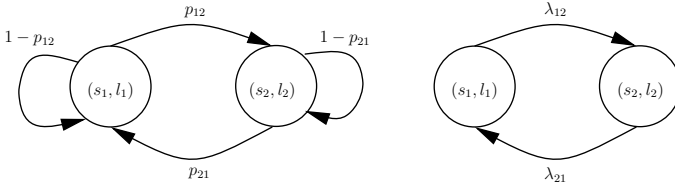


Fig. 2. Gilbert model as a DTMC (left) and CTMC (right)

occur at discrete time intervals. Certainly, this view is appropriate for modelling the loss process as described by a packet trace: Each packet constitutes a time instant at which a state-change may occur. Implementation of fault-injection using a discrete-time model is straightforward as well: With each arriving packet the next state is selected randomly from the possible successor states (including the same state) of the current state.

An alternative definition of the Gilbert-Elliot model considers the model a Continuous-Time Markov Chain (CTMC). In this view, state sojourn times are described by exponentially distributed random variables. The right-hand side of Figure 2 illustrates a two-state continuous-time Markovian loss model, again with loss rates l_1, l_2 . Transition rates are λ_{12} and λ_{21} . Packet-loss injection according to a continuous-time Gilbert model is implemented by playing the CTMC, as follows: Upon entering a state, the state sojourn time is drawn from an exponential distribution with rate equal to the sum of outgoing rates. When this time has passed, the next state is chosen based on the embedded Markov chain.

Let us now consider the relation of these different formulations of the loss model. With respect to *describing* packet-loss traces, the CTMC model is obviously equivalent to the embedded DTMC model, since both can capture interchanging periods of different loss levels. With respect to *generating* packet loss, however, we note a difference: In the embedded DTMC model, state-changes in the model may only occur upon arrival of packets, i.e., the packet-loss process is dependent on the arrival process. This is in contrast to the CTMC model, where state-changes occur independently, and requires a closer look at the arrival process. As described above, packet-loss measurements are typically obtained using an arrival process whose packet interarrival times are independent of packet transmissions. While there exist network protocols where this is the case, most protocols adapt their sending rate based on the success of previous packet transmissions. In particular, the Transmission Control Protocol (TCP) employs various mechanisms to reduce the sending rate if packet loss occurs. These reductions can be quite drastic, especially during the connection-setup phase, where the retransmission interval, starting at 3s, is doubled for each timeout (i.e. packet loss) [13]. If such a reactive protocol is studied using fault-injection with a DTMC Gilbert model, the protocol’s sending rate is affected by the loss process, while at the same time the loss process depends on the sending rate. This observation raises the question whether the obtained results are sound.

2.4 Scenarios for Fault Injection

In the following sections we study the impact of the loss model in three scenarios where fault-injection is used to assess the performance of a distributed system where two hosts communicate over a lossy link. Our scenarios differ in the characteristics of the arrival process of the network traffic.

Independent Arrival Stream. First, we consider a scenario where one host sends messages at a constant rate, thereby generating an arrival stream that is independent of the loss process. Examples for this type of traffic pattern may be heartbeats, periodic log messages, or clock frequency synchronisation as used in PTP [14]. Note that, while such a protocol might employ acknowledgments and retransmissions of lost packets to ensure data transmission, retransmissions are assumed to take place at the time a normal packet transmission would have happened. That is, in this scenario retransmissions due to packet loss do not affect the arrival process observed by the network.

Long TCP Arrival Stream. Second, we study the performance of a large data upload over a single long-lived TCP connection. Since TCP adapts its sending rate dynamically based on the available bandwidth, using, among others, packet loss events as indicators of network overload [13], in this scenario the arrival process is not independent of the loss process.

Short TCP Arrival Stream. In our third scenario we consider the performance of HTTP when using TCP connections over a lossy link. This scenario differs from the previous one with respect to the length of the connections and the amount of data to be transmitted. Here, we assume short connections, such as may be observed when downloading small web pages or in client-server communication in Web Services scenarios.

2.5 Modules for Fault Injection

In order to inject packet loss according to the three models described in Section 2.1, an appropriate fault-injector implementation must be selected.

NetEm (Bernoulli Packet Loss). The Linux operating system supplies the kernel module NetEm for injecting various disturbances at the IP level. In particular, this module supports the Bernoulli loss model, which can be parameterised by providing the desired loss rate.

NetEm CLG (Discrete-Time Markovian). The NetEm-CLG (Correlated Loss Generator) module [15] extends the default NetEm implementation by a discrete-time extended Gilbert-Elliot model. The module supports up to four states with fixed transition structure. In the following we use it to implement a two-state Gilbert model.

Netem CG (Continuous-Time Markovian). In our group we recently developed the NetEm-CG (Continuous Gilbert) module [16] as an extension to the default NetEm module. The module support continuous-time Gilbert models with an arbitrary number of states and an arbitrary structure. As with the NetEm-CLG module, we configure NetEm-CG such that it provides a two-state Gilbert model.

3 Experiments

In our experiments we want to study the impact of different fault models when used to investigate the three example scenarios. We use three Linux machines to set up the experiment environment shown in Figure 1. Machines A and C run Linux kernel 2.6.26, while machine B runs 2.6.37. We routed packets from A to C over B, while packets from C to A were transmitted directly, in order to ensure that the chosen packet-loss rate is achieved.

3.1 Scenarios

We generate the arrival streams required by our scenarios as follows: For the independent arrival stream, we use the Ping utility to send packets with a constant interarrival time of 200ms. Since Ping does not retransmit lost packets, this reflects the scenario where the packet arrival process is independent of the loss process. Ping sends a stream of ICMP packets. For each packet, the recipient replies with another ICMP packet. Upon receipt, the original sender prints out the sequence number of each received packet. We run the Ping test for 30 min and record the sequence numbers.

We emulate a long TCP connection using the TCP_STREAM test of Netperf 2.4.5 [17]. In our experiments, Netperf generates a unidirectional TCP stream from machine A to machine C and measures overall throughput as well as averages taken over time-windows of approximately 10s length. The test duration was originally set to 5 min. Due to our observation that for higher loss rates the test failed to transmit a sufficient amount of data within 5 min to get a measurement of the throughput, we also ran experiments where we set the test duration by specifying a limit on the amount of data to be transmitted instead (100 MB).

Performance of short HTTP connections was studied using the Apache JMeter tool [18] and the DHTTDP/1.02a web server [19]. We set up the web server on machine C and configured JMeter to repeatedly download the server's default homepage without any embedded content. The default homepage is a single static HTML file of 5258 bytes length; consequently, we expect the processing overhead of the web server to be negligible. We configured JMeter such that the KeepAlive feature of HTTP was not used. Combined with the small file size this results in very short TCP connections, consisting almost entirely of connection setup and connection teardown. We originally ran the test for 10 min, but increased the time to 30 min for higher loss rates with the discrete-time loss model, in order to obtain at least 500 samples for each loss rate.

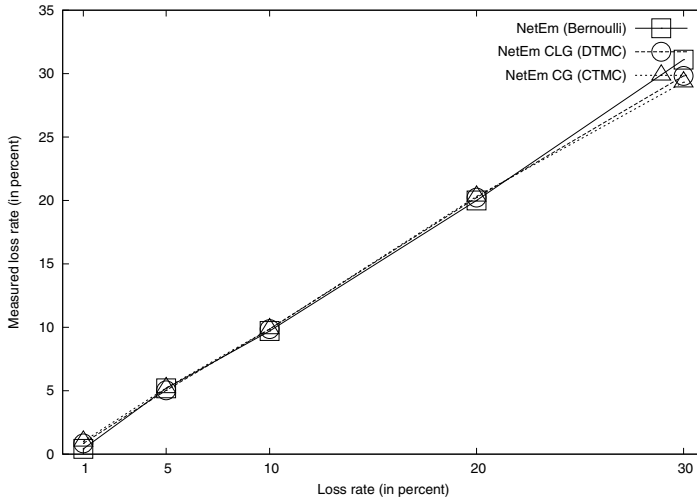


Fig. 3. Packet loss rates in the Ping test (independent arrival stream)

The loss-injector modules are loaded in machine B. We parameterise the models such that they produce packet-loss rates of $l = 0.01, 0.05, 0.1, 0.2$, and 0.3 .) For the Bernoulli model, parameterisation consists in setting the loss rate to the desired value. For the NetEm-CLG module we specify the desired loss rate l of the model. The model is then parameterised such that $p_{12} = l$, $p_{21} = 1 - p_{12}$, $l_1 = 0$ and $l_2 = 1$. With the NetEm-CG module we use the same loss rates l_1, l_2 , set $\lambda_{12} = 1$, and choose λ_{21} such that the loss rate l is achieved. Sojourn times were scaled to give a mean sojourn time for the loss-free state of 100 ms.

4 Results

Figures 3 through 5 show the average loss rate, average throughput, and mean HTTP download times obtained using Ping, Netperf and JMeter, respectively. Considering first the loss rate (Figure 3), we observe that all fault injectors generated the desired loss rates. This demonstrates that, with respect to the loss rate, the fault model implementations are equivalent when an independent arrival process is used. Turning towards throughput measurements (Figure 4), however, we note that with the Bernoulli model and with the DTMC model performance drops much faster than with the CTMC model. Finally, consider the mean HTTP download times shown in Figure 5. Here, a similar tendency can be observed: Packet loss generated using the CTMC model has a much lower impact on download times than Bernoulli and DTMC Gilbert-model loss.

The last scenario is especially useful for explaining the difference in the results, as the short duration of individual HTTP transmissions implies that each connection consists almost entirely of the TCP three-way handshake in the connection setup phase. In this phase, TCP detects packet loss by the RTO timeout,

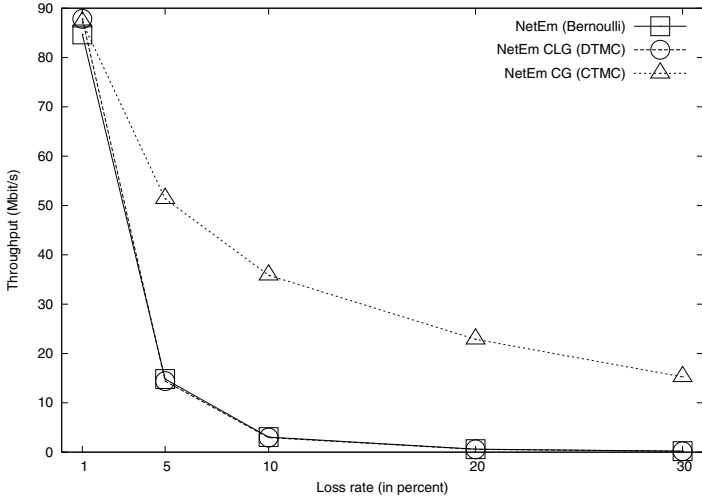


Fig. 4. Average throughput in a long-lived TCP connection

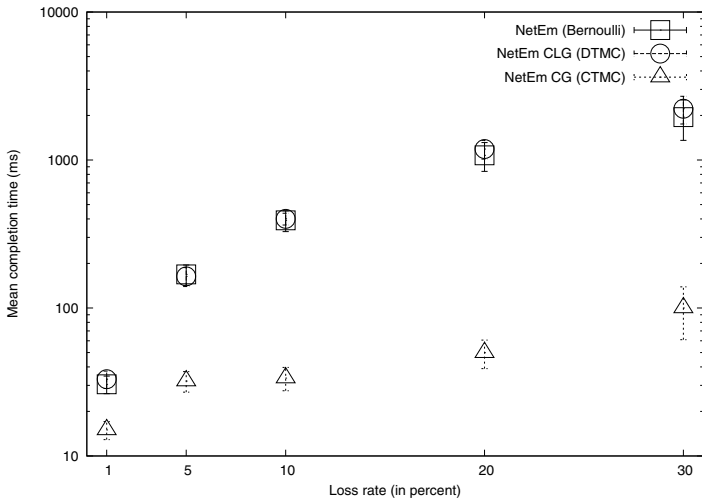


Fig. 5. Mean HTTP download times (short TCP connections, log scale) with 95% confidence intervals

which starts at 3 s. That is, if the initial SYN packet is lost, the first retransmission happens after 3 s. For each timeout event, the RTO is doubled, i.e. the next retransmission occurs at 6 s, and so on [13]. Since the delay in our network is negligible, it is reasonable to assume that samples larger than or equal to 6 s have been affected by at least two packet losses in the connection-setup phase. From Figure 6 we see that the probability of such samples grows with

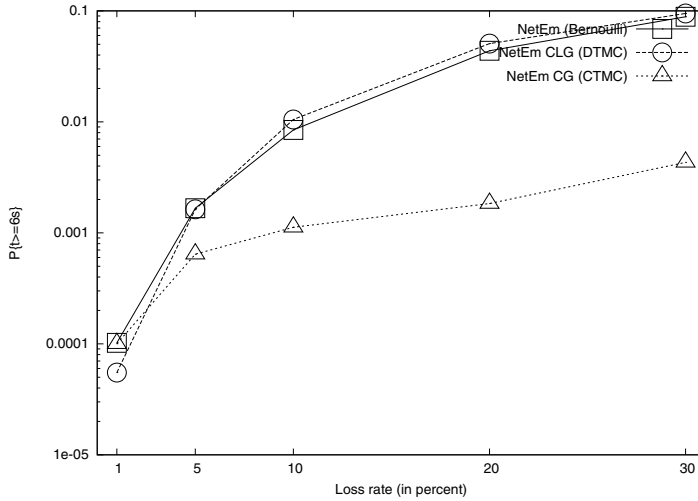


Fig. 6. Probability of HTTP connection times larger than 6 s

the loss rate for all models. More important, however, is that the discrete-time Gilbert model results in a much higher probability of two-packet losses during TCP connection setup than the continuous-time model. This can be attributed directly to the discrete-time loss process: The model can only leave the loss state upon packet arrivals. That is, if the model is in the loss state after dropping one packet, it will still be in the loss state when the next packet arrives, irrespective of the interarrival time between both packets. In contrast, the continuous-time model changes its state independently of packet arrivals and may thus be in another state when the next packet (or retransmission) arrives.

5 Concluding Discussion

In this paper we have investigated the impact of the interaction between the arrival stream and the fault model used in fault-injection studies. We observed that with dependent arrival streams as generated by TCP the results obtained using discrete-time and continuous-time Markovian loss models differ significantly, with discrete-time models resulting in much lower performance. When used in a performance study, the two models may lead to different conclusions. With a discrete-time model, where changes occur only at packet arrivals, very short retransmission timeouts may improve performance significantly, as they increase the rate of state transitions, and thus allow the model to reach the loss-free state earlier. With a continuous-time model, shorter timeouts might still improve performance marginally (due to the fact that the loss-free period is detected earlier), but will not have as big an impact, since the loss model changes state independently of packet arrivals. On the other hand, the increase in packet transmissions will increase the load and the effective packet loss rate.

We cannot conclude this paper with a clear recommendation which model one should use to reflect reality in a fault-injection experiment. Such a recommendation needs to be based on knowledge of whether the packet-loss process is independent of the arrival process. Currently available measurement studies of packet loss only provide traces which can be described equally well by both models, and give no indication as to dependence or independence of the loss process. Investigating this question is still part of future work. However, as it appears rather unlikely that the loss process is governed by the arrival process, we do provide a tentative recommendation to use CTMC models.

References

1. Reinecke, P., van Moorsel, A.P.A., Wolter, K.: The Fast and the Fair: A Fault-Injection-Driven Comparison of Restart Oracles for Reliable Web Services. In: QEST 2006: Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems, pp. 375–384. IEEE Computer Society, Washington, DC (2006)
2. Reinecke, P., Wolter, K.: Phase-Type Approximations for Message Transmission Times in Web Services Reliable Messaging. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 191–207. Springer, Heidelberg (2008)
3. Zhang, Y., Paxson, V., Shenker, S.: The Stationarity of Internet Path Properties: Routing, Loss, and Throughput. ACIRI Technical Report (2000)
4. Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the Constancy of Internet Path Properties. In: IMW 2001: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pp. 197–211. ACM, New York (2001)
5. Hohlfeld, O., Geib, R., Haßlinger, G.: Packet Loss in Real-Time Services: Markovian Models Generating QoE Impairments. In: Proc. of the 16th International Workshop on Quality of Service (IWQoS), pp. 239–248 (June 2008)
6. Chung, S.H., Won, Y.J., Agrawal, D., Hong, S.C., Ju, H.T., Park, K.: Detection and Analysis of Packet Loss on Underutilized Enterprise Network Links. In: E2EMON 2005: Proceedings of the Workshop on End-to-End Monitoring Techniques and Services 2005, pp. 164–176. IEEE Computer Society, Washington, DC (2005)
7. Hacker, T., Smith, P.: Building a Network Simulation Model of the TeraGrid Network. In: TeraGrid 2008 Conference, Las Vegas, NV, June 9-13 (2008)
8. Hacker, T., Noble, B., Athey, B.: The Effects of Systemic Packet Loss on Aggregate TCP Flows. In: Proceedings of the ACM/IEEE 2002 Conference on Supercomputing, pp. 1–15 (2002)
9. Barford, P., Sommers, J.: A Comparison of Probe-based and Router-based Methods for Measuring Packet Loss. Technical report, University of Wisconsin-Madison (September 2003)
10. Keller, A., Baumann, R., Fiedler, U.: TCN Trace Control for Netem (2009), <http://tcn.hypert.net/> (last seen June 6, 2011)
11. Reinecke, P., Wolter, K.: Towards a multi-level fault-injection test-bed for service-oriented architectures: Requirements for parameterisation. In: SRDS Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems, Naples, Italy, AMBER (2008)
12. Reinecke, P., Wolter, K., Malek, M.: A Survey on Fault-Models for QoS Studies of Service-Oriented Systems. Technical Report B-2010-02, Freie Universität Berlin (February 2010)

13. Krishnamurthy, B., Rexford, J.: Web Protocols and Practice. Addison Wesley, Reading (2001)
14. IEEE: Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4579760
15. Salsano, S., Ludovici, F., Ordine, A.: Definition of a general and intuitive loss model for packet networks and its implementation in the Netem module in the Linux kernel. Technical report, University of Rome "Tor Vergata"
16. Dräger, M.: Entwurf und Implementierung eines Moduls zur stochastischen Fehlerinjektion für IP-Netzwerke gemäß eines erweiterten Gilbert-Modells. Bachelor thesis, Freie Universität Berlin (2011) (in German)
17. Jones, R.: Netperf 2.4.5 (June 2009), <http://www.netperf.org/netperf/> (last seen June 6, 2011)
18. The Apache Software Foundation: Apache JMeter, <http://jakarta.apache.org/jmeter> (last seen June 6, 2011)
19. Klouwen, W.: Dhttpd/1.02a, <http://sourceforge.net/projects/dhttpd/> (last seen June 6, 2011)

Analysis of Gossip-Based Information Propagation in Wireless Mesh Networks

Abolhassan Shamsaie¹, Wan Fokkink², and Jafar Habibi¹

¹ Department of Computer Engineering, Sharif University of Technology, Iran

² Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
shamsaie@mehr.sharif.edu, w.j.fokkink@vu.nl, jhabibi@sharif.edu

Abstract. Analytical models exist for evaluating gossip-based information propagation. Up to now these models were developed only for fully connected networks. We provide analytical models for information propagation of a push-pull gossiping protocol in a wireless mesh network. The underlying topology is abstracted away by assuming that the wireless nodes are uniformly deployed. We compare our models with simulation results for different topologies.

Keywords: Gossiping protocols, Analytical model, Information propagation, wireless mesh network.

1 Introduction

In traditional distributed computing, one or more nodes in a network, called servers, organize all other nodes in the network, centrally. The emergence of new distributed networks like peer to peer, ad hoc and sensor networks has introduced new challenges in organizing networks to efficiently route or disseminate information in a distributed manner. The topology of these networks is mostly partially connected and can change over time. Nodes can join and leave the network without informing other nodes. Each node has just a partial view of other nodes, which may continuously change. Mostly, partial views contain some information about neighbors or a history of information from a fraction of other nodes. In peer to peer networks, a node's neighbors are a random subset of other nodes, while in wireless ad hoc or sensor networks they are located in the radio range of that node. This makes it very hard to obtain some sort of global view in such networks. In contrast, information propagation in such networks has been demonstrated to be feasible. To this aim, nodes continuously exchange information with their neighbors, to acquire some recent information about e.g. their proximity or the network state. Nowadays, a lot of research concerns this challenge, considering scalability, fault tolerance and robustness, graceful degradation, and adaptability.

Epidemic-style or gossip-based protocols [5] are a fundamental solution for information propagation in dynamic networks. These protocols are simple, probabilistic in nature, and based on local operations. They are in general periodic

and execute in a fully distributed manner. They have been exploited primarily to provide the required underlying infrastructure. Their performance has been evaluated by empirical evaluation, simulation and analytical models, see e.g. [10,2,7]. Empirical evaluation is helpful to study protocols or their implementation in a real deployment. However, it is costly and mostly performed in a small-scale experiment, which may not reflect the large-scale behavior of a protocol. In contrast, simulation and analytical models may help to extract the large-scale, long-time behavior of a protocol, as well as its correctness and performance. However, these approaches do not consider all aspects of real deployment scenarios.

Related works. Analytical models tend to abstract away from the behavior of a protocol with regard to some performance measures, and can provide a good view to tune parameters of a protocol to improve performance. Models for the aggregation of information by gossip-based protocols (e.g. [3,4,9,8]) draw their inspiration from the mathematical theory of epidemics, or use Markov chains or random walks. In [1] mean-field analysis is used for modeling gossip-based protocols in networks with a very large number of identical nodes. These models all assume a fully connected network, which is a reasonable assumption for peer to peer networks. It is sometimes hidden behind a peer sampling service which allows nodes to select a node uniformly at random. In [2] a model has been developed for information propagation by a simple push-pull gossip-based protocol, called the shuffle protocol [6]. The local behavior of protocol, i.e. the pairwise node interaction which is considered as an atomic operation, is modeled by a probabilistic state transition system. Through differential equation, two properties of the shuffle protocol are expressed as functions of protocol parameters:

replication ratio The fraction of nodes having a copy of a given item at a certain moment in time

coverage ratio the fraction of nodes that have seen a copy of a given item before a certain moment in time

The model from [2] is also only valid for fully connected networks.

In a wireless network, a node can only communicate directly to nodes in its radio range. A path between two nodes may pass through other nodes, so for propagating a piece of information, cooperation and coordination among nodes are needed. Different topologies for wireless networks have been introduced in the literature, such as grid, partial or fully connected mesh, star and tree to name a few. Among these, we consider a partial mesh network in which nodes are deployed uniformly in a disk-like area and connected to each other according to their proximity. There is assumed to be a path between all pairs of nodes. Disk-like grid and fully connected mesh networks are also covered by our model, as they are cases of uniformly deployed wireless mesh networks (WMNs). In this paper, we adapt the model from [2] to model the propagation of information for the shuffle protocol on uniformly deployed WMNs. We use a probabilistic state transition system and convergence or steady state characteristics of the shuffle protocol to model propagation of an item in a WMN. Next to a symbolic

analysis, we compare our model against simulations with an implementation of the shuffle protocol, for different topologies and parameter settings. These simulation results demonstrate that our model provides a good prediction of the behavior of the shuffle protocol. To the best of our knowledge, this is the first attempt to model information propagation of a gossip-based protocol in WMNs.

The paper is organized as follows. Sect. 2 gives a brief description of the shuffle protocol from [6], and of the analytical model of its local behavior from [2]. In Sect. 3, analytical models are developed for WMNs, which are evaluated in Sect. 4. Finally Sect. 5 contains conclusions and future work.

2 Shuffle Protocol

Gossip-based protocols are pairwise and probabilistic in nature. In a network of size N , these protocols consist of N local, pairwise and periodical gossiping operations between neighboring nodes which lead to some global characteristic for the network, like robustness or convergence. The locality of operations in these protocols makes them robust to network churn, while their probabilistic nature can provide convergence. Gossip-based protocols can be push-based, pull-based or a combination of them.

Our paper is based on a gossip-based protocol named the shuffle protocol, introduced in [6]. It is a push-pull protocol in which each node has a cache of items of size c and periodically exchanges a random subset of size s from these items with a random neighbor.

2.1 Description

Nodes periodically initiate a shuffle. Although nodes are not synchronized, they all have an inner timer which periodically times out with approximately the same frequency and changes the state of a node from passive to active. We call such a period a round hereafter, and assume that it takes one time unit. When a node switches to the active state, it initiates a shuffle and then switches back to the passive state. In the passive state, a node is waiting for shuffles which are initiated by others. Below a brief description of a shuffle in this protocol is given. For more details we refer the reader to [6].

1. Node A selects one of its neighbors B uniformly at random, and sends a copy of a random subset of size s from items in its local cache to B .
2. Node B receives s items from node A , in response sends a copy of a random subset of size s from items in its local cache to A , and updates its cache (step 4).
3. Node A receives s items from node B , and updates its local cache (step 4).
4. Received items for which a copy is already present in the local cache, are redundant. To update the cache, each node replaces non-redundant sent items with non-redundant received items.

This protocol ensures that no item will disappear completely from the network.

When a new item is introduced into the network at one of the nodes, the number of nodes which hold (or have seen) a copy of this item will increase over time. If the number of distinct items in the network does not exceed the capacity of local caches ($c \geq n$), all nodes will eventually hold a copy of this item, and the replication ratio and coverage ratio will both converge to 1. But if $c < n$, the replication ratio does not converge to 1.

2.2 Probabilistic State Transition System

To model the propagation of a given item in the network, in [2] a shuffle between two nodes is modeled by a probabilistic state transition system, depicted in Fig. 1. A node is in state 1 if it has the given item, and in state 0 otherwise. All probabilities are of the form $P(a_2b_2|a_1b_1)$ with $a_1, a_2, b_1, b_2 \in \{0, 1\}$. Given an information exchange between two nodes in state a_1 and b_1 respectively, it indicates the probability that after the exchange these nodes are in state a_2 and b_2 respectively. [2] determined the transition probabilities for one shuffle, i.e., for one information exchange between two nodes. They are presented below; in the formulae, n refers to the number of items, and c and s to the cache and exchange buffer size respectively:

$$\begin{aligned}
 P(01|01) &= P(10|10) = \frac{c-s}{c} & P(01|11) &= P(10|11) = \frac{s}{c} \cdot \frac{c-s}{c} \cdot \frac{n-c}{n-s} \\
 P(10|01) &= P(01|10) = \frac{s}{c} \cdot \frac{n-c}{n-s} & P(00|00) &= 1 \\
 P(11|01) &= P(11|10) = \frac{s}{c} \cdot \frac{c-s}{n-s} & P(11|11) &= 1 - 2 \cdot \frac{s}{c} \cdot \frac{c-s}{c} \cdot \frac{n-c}{n-s}
 \end{aligned}$$

These probabilities model local behavior of the shuffle protocol. In [2] these probabilities were used to model the global behavior (replication and coverage ratio) in fully connected mesh networks using differential equations. In this paper we carry over this work to WMNs.

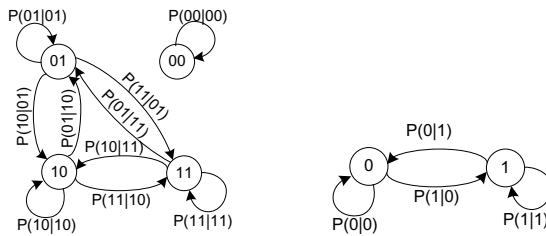


Fig. 1. State transition systems of a shuffle and of a node

Based on the state transition system of a shuffle, shown at the left-hand side of Fig. 1, a state transition system for each node can be derived, as shown at the right-hand side of Fig. 1. For instance the transition probability $P(0|1)$ or $P(1|0)$ denotes the probability that a node loses or gains the given item in a shuffle, respectively. Thus $P(0|1) = P(01|10) \cdot P_0 + P(01|11) \cdot P_1$ and $P(1|0) = (P(10|01) + P(11|01)) \cdot P_1$. In the shuffle protocol, after convergence of the replication ratio

for the given item, the average number of nodes which go from state 0 to state 1 is equal to the average number of nodes which go from state 1 to state 0 during a round. This property expresses a steady state for all nodes, giving the following equation:

$$P_1 \cdot P(0|1) = P_0 \cdot P(1|0)$$

In the above equations, P_1 (P_0) is the probability that a node has (does not have) the given item. Replacing $P(0|1)$ and $P(1|0)$ by the formulas above (and P_0 by $1 - P_1$) yields $P_1 = \frac{c}{n}$. That is, the replication ratio should converge to $\frac{c}{n}$, which is in line with a uniformly random distribution of all items over the network.

3 WMN Model of the Shuffle Protocol

We want to model propagation of information by the shuffle protocol in a uniformly deployed WMN. At time 0, a fresh item is introduced at the central node of the network. The shuffle protocol propagates this item in the network. Our model aims to estimate the average behavior of all possible propagation scenarios.

When the item is inserted into the network, its replication ratio and coverage ratio are both $\frac{1}{N}$, where N is the number of nodes. We assume $c < n$. As mentioned in Sect. 2.2, the replication and coverage ratio should converge to $\frac{c}{n}$ and 1 respectively. The nodes which have a copy of the given item are called source nodes. We assume a disk-like area with radius R ; the initial source node is located at the center of this disk and the other nodes are uniformly deployed throughout this area. Thus the node density (d) is $\frac{N}{\pi \cdot R^2}$. Nodes can communicate to nodes which are located in their radio range; we assume all nodes have radio range $r \leq 2 \cdot R$. The neighbors of a node are uniformly distributed in the intersection of the disk-like area and the radio range of the node. So the average number of neighbors (\overline{H}) for each node is less than $\pi \cdot r^2 \cdot d - 1$. Substituting $\frac{N}{\pi \cdot R^2}$ for d in this formula yields $\overline{H} \leq (\frac{r}{R})^2 \cdot N - 1$. When $r \ll R$, \overline{H} is approximately $(\frac{r}{R})^2 \cdot N - 1$. The average length of a hop (\overline{L}) is approximately the radius of a circle around the node which contains half of the $\pi \cdot r^2 \cdot d - 1$ nodes. The node itself is inside too, so there are $\frac{\pi \cdot r^2 \cdot d - 1}{2} + 1$ possible nodes inside this circle, which gives the equation $\pi \cdot \overline{L}^2 \cdot d = \frac{\pi \cdot r^2 \cdot d - 1}{2} + 1$. Substituting $\frac{N}{\pi \cdot R^2}$ for d in this formula yields:

$$\overline{L} = \sqrt{\frac{r^2}{2} + \frac{R^2}{2 \cdot N}} \tag{1}$$

At the left side of Fig. 2, a sample of a disk-like uniformly deployed WMN with parameters $r = 2$, $R = 10\sqrt{10}$ and $N = 3150$ is shown. The distribution of neighbors, the number of nodes having a certain number of neighbors, is given at the right. The vertical dotted line in this figure shows the average number of neighbors, which is 11.6102. The aforementioned formula for the average number

of neighbors yields $\overline{H} = 11.6000$, which is close to the real scenario depicted in Fig. 2. The average length of hops in this scenario is 1.4679, which is close to $\overline{L} = 1.4693$ calculated from the related formula.

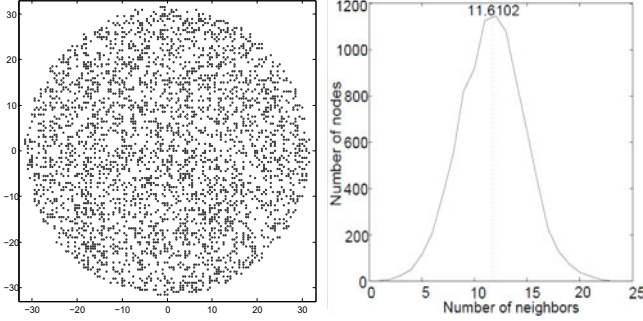


Fig. 2. A sample uniformly deployed WMN (left) and its distribution of neighbors (right)

We aim to model the propagation of the given item by the shuffle protocol according to the replication and coverage ratios. We exploit the convergence characteristic of the shuffle protocol to mathematically analyze how the replication ratio $\alpha(t+1)$ (given $\alpha(t)$) and coverage ratio $\beta(t+1)$ (given $\alpha(t)$ and $\beta(t)$) change over time. Our problem has six parameters: network parameters r , R and N , and protocol parameters c , s and n . In a snapshot of an execution of the shuffle protocol at some time t before convergence, we can observe a circular area around the central node in which the relative replication ratio has converged to $\frac{c}{n}$, while in the remaining area it is less than $\frac{c}{n}$. We implemented the shuffle protocol, and ran it on a uniformly deployed WMN with parameters $r = 2$, $R = 10\sqrt{10}$, $N = 3150$, $c = 100$, $s = 50$ and $n = 500$. Fig. 3 shows some snapshots of an execution of this implementation. In each snapshot, the circle indicates the convergence circle: the largest circular area in which the relative replication ratio has converged to $\frac{c}{n}$. We denote the replication and coverage ratio inside a convergence circle by α' and β' respectively. We also denote the number of interior nodes and source nodes in such a circle by N' and S' respectively; so $\alpha' = \frac{S'}{N'}$. Let $r_c(t)$ and $\alpha(t)$ denote the convergence circle radius and the replication ratio in a snapshot at time t . By definition the number of source nodes at time t is $N \cdot \alpha(t)$. We have $N' = \pi \cdot r_c(t)^2 \cdot d$. Substituting $\frac{N}{\pi \cdot R^2}$ for d yields $r_c(t) = R \cdot \sqrt{\frac{N'}{N}}$. According to the definition of a convergence circle, $\alpha'(t)$ has converged to $\frac{c}{n}$, so $N' = \frac{n}{c} \cdot S'$, which implies $N' \leq \frac{n}{c} \cdot N \cdot \alpha(t)$. Hence $r_c(t) \leq R \cdot \sqrt{\frac{n}{c} \cdot \alpha(t)}$. We define $r_{c_{max}}(t)$ as the maximum possible $r_c(t)$ in a snapshot at time t , so $r_{c_{max}}(t) = R \cdot \sqrt{\frac{n}{c} \cdot \alpha(t)}$. Thus:

$$r_c(t) \leq r_{c_{max}}(t) \quad (2)$$

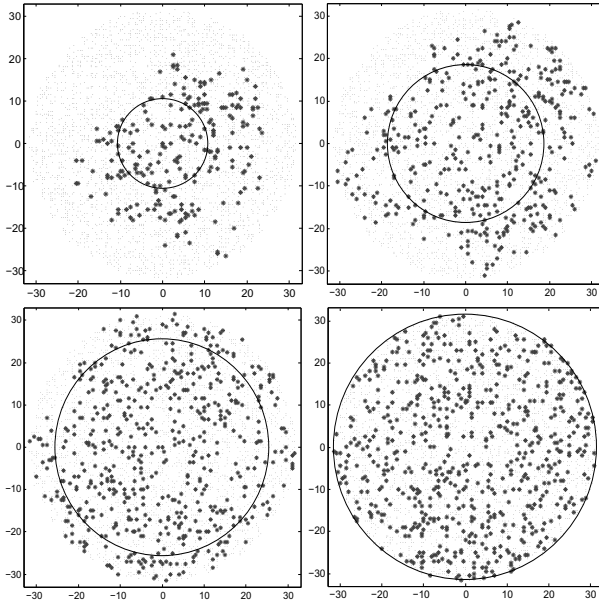


Fig. 3. Snapshots of the shuffle protocol, depicting the source nodes and the convergence circle at times $t = 100, 150, 170$ and 190 respectively

Given a snapshot at time t , the propagation circle is the smallest circle centered by the central node which contains all source nodes and has a relative replication ratio of at most $\frac{c}{n}$ (or the entire disk of radius R if $\alpha(t) \geq \frac{c}{n}$). Let $r_p(t)$ denote the radius of the propagation circle at time t . Clearly,

$$r_{c_{max}}(t) \leq r_p(t) \tag{3}$$

To illustrate the applicability of $r_p(t)$, consider a snapshot of an execution of the shuffle protocol in a WMN at time t . Only nodes inside or just outside the propagation circle of this snapshot had a chance to perform a shuffle with source nodes during the previous round. Possibly some nodes further than $r_p(t)$ had the given item during round t but dropped it in the same round.

3.1 Lower Bound on $\overline{r_p}(t + 1)$

In this section, we obtain a lower bound on the average of all possible values of $r_p(t + 1)$, denoted by $\overline{r_p}(t + 1)$, for a given snapshot at time t , in which the replication ratio is $\alpha(t)$. This provides a lower bound on the average number of nodes which will have a chance to perform a shuffle with source nodes during the next round. This lower bound will in turn help us to find a lower bound of the average of all possible replication and coverage ratios in the next round.

To this aim, we assume a situation in which $r_p(t)$ takes the lowest possible value for the given replication ratio $\alpha(t)$. Thus, according to inequality 3, we

assume $r_p(t) = r_{c_{max}}(t)$, which implies $r_c(t) = r_{c_{max}}(t)$. We calculate the average of all possible values of $r_p(t + 1)$, which provides a lower bound for $\bar{r}_p(t + 1)$.

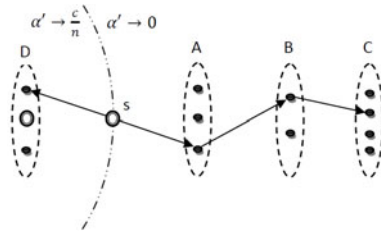


Fig. 4. A scenario in which the given item travels multiple hops away from the central node in one round

Fig. 4 illustrates the mentioned scenario at the start of round $t + 1$. The arc is a portion of the maximum possible convergence circle area with radius $r_{c_{max}}(t)$. In this scenario, at the start of the next round, all source nodes (empty bullets) are inside or on the circle. Some of them (like s) may exchange the given item to nodes outside this circle during the next round. In Fig. 4, D and A, B, C are groups of neighbors of s which are nearer to and further from the central node than s , respectively. In the next round, s may shuffle and send the given item to a node in group A , then that node may send it to a node in group B , and so on. Clearly, in such a scenario, $r_p(t + 1)$ is greater than $r_{c_{max}}(t)$. On the other hand, in round $t + 1$, s may shuffle and send the given item to a node in group D and drop it. In this case, $r_p(t + 1)$ might decrease, but only if all other source nodes that are located near the the convergence circle in round t act in this same fashion in round $t + 1$. Consequently, the probability that $r_p(t + 1)$ decreases has a probability close to 0. Thus, to find a lower bound on the average of all possible values of $r_p(t + 1)$, for each $i \geq 0$ we calculate the probability that the given item travels i hops further than $r_{c_{max}}(t)$ in round $t + 1$, and we ignore the case $i < 0$.

The neighbors of an arbitrary node s are divided into two groups: D contains the neighbors closer to the central node than s , and A contains the neighbors further away. (On average, the number of neighbors of s with exactly the same distance to the central node is almost zero, so these can be ignored.) The fraction of neighbors of s which are closer to or further from the central node can be approximated by the areas of D and A . The area of D can be calculated as follows, where ℓ is the distance of s to the central node.¹

$$area(D) = \begin{cases} \pi \cdot \ell^2 & 0 < \ell \leq \frac{r}{2} \\ r^2 \cdot \cos^{-1}\left(\frac{r}{2 \cdot d}\right) + \ell^2 \cdot \cos^{-1}\left(\frac{2 \cdot \ell^2 - r^2}{2 \cdot \ell^2}\right) & \frac{r}{2} \leq \ell \leq R \\ -\frac{r}{2} \cdot \sqrt{4 \cdot \ell^2 - r^2} & \end{cases}$$

¹ <http://mathworld.wolfram.com/Circle-CircleIntersection.html>

On average approximately half of the neighbors of a node s are in D . Clearly, on average each node shuffles twice in each round, once as an initiator and on average once as a selected node. Therefore we assume that in a round each node shuffles once with a nearer node in D and once with a further node in A .

We assume $r_c(t) = r_{c_{max}}(t)$, so the relative replication ratio inside the arc is close to $\frac{c}{n}$, and outside it is close to 0. For each $i \geq 0$, we calculate the probability $P(\Delta h = i)$ that the given item in a source node like s travels i hops away from the central node in the next round. To find the transition probabilities, we use P_s and P_d (introduced in [2]), which denote the probability that a node selects respectively drops the given item in a shuffle. We define one more probability P_n , which expresses the chance of the given item to go one hop nearer to the central node and be removed from a source node s , when s shuffles with a nearer node u . The given item will be removed from s and go to u if (1) s selects the given item and sends it to u with probability P_s , (2) u does not send the item to s because either it does not have the item with probability $\frac{n-c}{n}$, or it has the item with probability $\frac{c}{n}$ but does not select it with probability $(1 - P_s)$, and (3) s drops the item with probability P_d . So:

$$P_n = P_s \cdot \left(\frac{c}{n} \cdot (1 - P_s) + \frac{n - c}{n} \right) \cdot P_d$$

Replacing P_s and P_d by $\frac{s}{c}$ and $\frac{n-c}{n-s}$, as computed in [2], we get:

$$P_n = \frac{s}{c} \cdot \frac{n - c}{n}$$

We write $P_{\neg n}$ and $P_{\neg s}$ for $(1 - P_n)$ and $(1 - P_s)$ respectively. By assumption, in a round, each node on average shuffles once with a nearer node and once with a further node. The probability that s remains as a source node and does not send the given item further is:

$$P(\Delta h = 0) = \frac{1}{2} \cdot P_{\neg n} \cdot P_{\neg s} + \frac{1}{2} \cdot P_{\neg s} \cdot P_{\neg n}$$

In the above equations, $\frac{1}{2}$ in each term denotes the probability of each of two possible scenarios: the shuffle with a nearer node or with a further node executes first. The other two parts of each term denote that the given item stays at s and does not move further. Replacing P_s and P_n by $\frac{s}{c}$ and $\frac{n-c}{n-s}$ yields:

$$P(\Delta h = 0) = \frac{c - s}{c} \cdot \left(\frac{c - s}{c} + \frac{s}{n} \right)$$

When the given item passes through a path of i hops from s to a node e in one round, the following three steps should take place:

1. In s , one of the following two scenarios should happen. Either the first shuffle is with a nearer node and the given item is not removed from s , and then in the second shuffle it moves to a further node. Or the first shuffle is with a further node and the given item moves to it.

2. For nodes in the middle of the path from s to e , the shuffle with a nearer node should be executed first to take the given item, and then the shuffle with a further node should be executed to send it further.
3. When the given item reaches e , one of the following two scenarios should happen. Either the given item reaches e in the second shuffle. Or the given item reaches e in the first shuffle with a nearer node, and does not move further in the second shuffle.

According to the above steps, $P(\Delta h = i)$ for $i \geq 1$ is:

$$P(\Delta h = i) = \left(\frac{1}{2} \cdot P_{-n} \cdot P_s + \frac{1}{2} \cdot P_s\right) \cdot \left(\frac{1}{2} \cdot P_s\right)^{i-1} \cdot \left(\frac{1}{2} + \frac{1}{2} \cdot P_{-s}\right)$$

Replacing P_s and P_n in this equation yields:

$$P(\Delta h = i) = \frac{2 \cdot c - s}{2 \cdot c} \cdot \left(\frac{2 \cdot c - s}{c} + \frac{s}{n}\right) \cdot \left(\frac{1}{2} \cdot \frac{s}{c}\right)^i$$

Let ℓ denote a random variable that expresses the distance of an arbitrary node to the central node. According to uniform deployment, for any value ℓ' between 0 and R , the cumulative density function² $F(\ell')$, which expresses the probability that $\ell \leq \ell'$, equals $\frac{\pi \cdot \ell'^2}{\pi \cdot R^2} = \frac{\ell'^2}{R^2}$. The expected value $\bar{\ell}$ of ℓ is $\int_0^R \ell' \cdot \frac{dF(\ell')}{d\ell'} d\ell' = \int_0^R \ell' \cdot \frac{2 \cdot \ell'}{R^2} d\ell' = \frac{2 \cdot R}{3}$. On average the given item goes at most distance $R - \bar{\ell}$ beyond a source node s . On average this distance is covered by $\lceil \frac{R - \bar{\ell}}{L} \rceil = \lceil \frac{R}{3 \cdot L} \rceil$ hops. (Recall that \bar{L} denotes the average length of each hop, see equation 1.) Thus, we approximate the average of Δh as follows:

$$\overline{\Delta h} = \frac{\sum_{0 \leq i \leq \lceil \frac{R}{3 \cdot L} \rceil} i \cdot P(\Delta h = i)}{\sum_{0 \leq i \leq \lceil \frac{R}{3 \cdot L} \rceil} P(\Delta h = i)}$$

Concluding we arrive at the following lower bound for $\bar{r}_p(t + 1)$:

$$\min(r_{c_{max}}(t) + \overline{\Delta h} \cdot \bar{L}, R) \leq \bar{r}_p(t + 1) \tag{4}$$

This lower bound will help us to find an estimation of the replication and coverage ratio in the next round.

3.2 Modeling Replication and Coverage Ratio with Differential Equations

We now construct two differential equations that approximate the replication and coverage ratio of the given item from a round-based perspective. They express the long-term behavior of the system as a function of the six parameters. We need to know the probabilities of a node in a given state (0 or 1) to interact

² <http://mathworld.wolfram.com/DistributionFunction.html>

with another node in a given state. In WMNs, before convergence, the density of source nodes decreases from the central node to the edge of the disk, so the distribution of source nodes over the disk is not uniform.

Given a snapshot at time t with replication ratio $\alpha(t)$, we found a lower bound of $\overline{r_p}(t + 1)$. We model the protocol behavior considering a greedy scenario in which $r_p(t + 1)$ is equal to this lower bound. With this greedy scenario we use the fact that when the propagation radius does not increase much, the distribution of source nodes inside becomes uniform relatively fast. We call the circle with a radius one hop (of average length) more than the propagation radius the shuffle circle, and denote its radius by $r_{sh}(t + 1)$. All nodes in the shuffle circle have approximately the same chance to shuffle with a source node in round $t + 1$. According to inequality 4 and the definition of the shuffle circle:

$$r_{sh}(t + 1) = \min(r_{c_{max}}(t) + (\overline{\Delta h} + 1) \cdot \overline{L}, R) \tag{5}$$

At time t there are $\pi \cdot r_{sh}(t + 1)^2 \cdot d$ nodes in the shuffle circle. Approximately $N \cdot \alpha(t)$ of them are source nodes, so the relative replication ratio of the shuffle circle is $\alpha' = \frac{N}{\pi \cdot r_{sh}(t+1)^2 \cdot d} \cdot \alpha(t)$. The shuffles that can influence the number of source nodes in the next round are restricted to the shuffle circle. So the variation of α per round, $\frac{d\alpha}{dt}$, can be derived from α' and the probability that the given item replicates or disappears in each shuffle in the shuffle circle:

$$\frac{d\alpha}{dt} = [(P(11|10) + P(11|01)) \cdot (1 - \alpha') \cdot \alpha' - (P(10|11) + P(01|11)) \cdot \alpha' \cdot \alpha'] \cdot \frac{\pi \cdot r_{sh}(t+1)^2 \cdot d}{N}$$

The first part, between brackets, indicates the probability that the given item replicates or disappears during a shuffle within the shuffle circle during the next round. The first term expresses the probability that a source node shuffles with a non-source node in the shuffle circle and replication increases by one, while the second term expresses the probability that two source nodes shuffle in the shuffle circle and replication decreases by one. The second part normalizes the result as a differential equation for all networks. Simplifying the right-hand side of the equation yields

$$\frac{d\alpha}{dt} = 2 \cdot \frac{s}{c} \cdot \frac{c - s}{n - s} \cdot \alpha(t) \cdot \left(1 - \frac{n}{c} \cdot \sigma(t) \cdot \alpha(t)\right)$$

where $\sigma(t)$ denotes $\frac{1}{\min\left(\sqrt{\frac{n}{c} \cdot \alpha(t) + (\overline{\Delta h} + 1) \cdot \sqrt{\frac{r^2}{2 \cdot R^2} + \frac{1}{2 \cdot N}}}, 1\right)}$, called the stepwise coef-

ficient hereafter. By imposing $\frac{d\alpha}{dt} = 0$, we find the stationary solution $\frac{c}{n}$, meaning that eventually the replication ratio of the given item stabilizes as $\frac{c}{n}$. Compared to [2] for fully connected networks, we have the extra stepwise coefficient $\sigma(t)$, which reflects the impact of uniformly deployed WMN topologies on the speed of information propagation. In our model, if the network is fully connected then $r = 2 \cdot R$, and so $\sigma(t) = 1$. Hence our model coincides with [2] for fully connected networks.

To model the coverage ratio $\beta(t)$ over time, the same approach is followed. At time t there are about $\beta(t) \cdot N$ covered nodes in the shuffle circle, and

$\beta' = \frac{N}{\pi \cdot r_{sh}(t+1)^2 \cdot d} \cdot \beta(t)$ is the fraction of nodes in the shuffle circle that are covered nodes. A non-covered node will be covered in the next round if it receives the given item from a source node and does not lose it in the remaining time of the next round. Namely, each node only checks its cache for new items at the end of each round, so new items which come and leave during a round are not covered. Thus the variation of β per time slot (round) $\frac{d\beta}{dt}$ can be derived from α' and β' and the probability that the given item replicates to a non-covered node and does not disappear until the end of this round:

$$\frac{d\beta}{dt} = \left[\frac{1}{2} \cdot P(1|0) \cdot P(1|1) \cdot (1 - \beta') + \frac{1}{2} \cdot P(1|0) \cdot (1 - \beta') \right] \cdot \frac{\pi \cdot r_{sh}(t+1)^2 \cdot d}{N}$$

The first part, between brackets, indicates the probability of increase in the number of covered nodes during the next round inside the shuffle circle, and the second part normalizes the result as a differential equation for all networks. There are two terms between brackets, showing two scenarios: the first and second term refer to the scenarios in which a node is covered in its first or second shuffle, respectively. The factor $\frac{1}{2}$ needs to be included because when a node is covered this happens with 50% chance in its first shuffle and with 50% chance in its second shuffle in a round. In the first term, the $P(1|1)$ ensures that the node that receives the given item in its first shuffle, will not lose it in its next shuffle in the same round. We have

$$\begin{aligned} P(1|1) &= (P(10|10) + P(11|10)) \cdot (1 - \alpha') + (P(10|11) + P(11|11)) \cdot \alpha' \\ P(1|0) &= (P(10|01) + P(11|01)) \cdot \alpha' \end{aligned}$$

Substituting $P(1|0)$ and $P(1|1)$ in the differential equation of the coverage ratio and simplifying the right-hand side of the equation yields:

$$\frac{d\beta}{dt} = \frac{1}{2} \cdot \frac{s}{c} \cdot (1 - \sigma(t) \cdot \beta(t)) \cdot \alpha(t) \cdot \left[1 + \frac{c-s}{n-s} \cdot \frac{n}{c} + \left(1 - \frac{c-s}{n-s} \cdot \left(\frac{s}{c} \cdot \frac{n-c}{c} + \frac{n}{c} \right) \right) \cdot \sigma(t) \cdot \alpha(t) \right]$$

To evaluate our model, we will perform a stepwise calculation of α and β for a given parameter setting, taking into account that $\alpha(0) = \beta(0) = \frac{1}{N}$.

4 Evaluation of the Model

The model from Sect. 3.2 gives the replication and coverage ratio of the shuffle protocol in a uniformly deployed WMN, for a parameter setting c, s, n, r, R, N . As mentioned before, we also implemented the shuffle protocol. To evaluate our model, we compare its outcomes with simulations of our implementation of the shuffle protocol, for a disk-like area with radius $R = 10\sqrt{10}$ and $N = 3150$.

In the first experiment, we considered a radio range of $r = 1$ for all nodes, and deployed them in a grid structure. We simulated the shuffle protocol for different settings of n, c, s , and compared the results of these simulations to our model. Figs 5 and 6 compare how the replication and coverage ratios of the simulations and of our model grow over time, for $n = 2100, c = 300$, and different values for s . In each of these figures, the curves in the graph at the left represent the

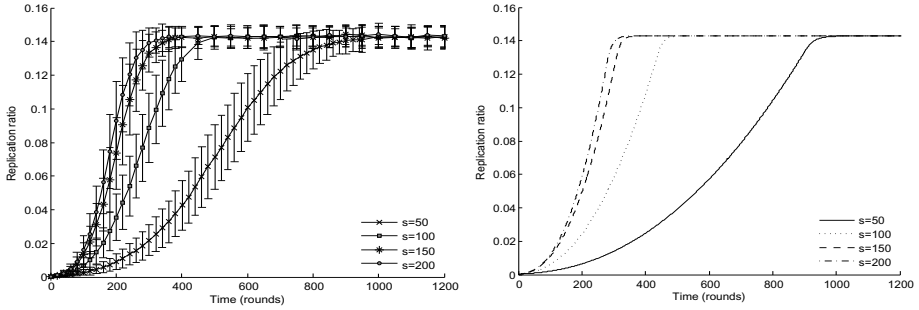


Fig. 5. Replication ratio in simulations and model for $r = 1$, $n = 2100$, $c = 300$ and different values of s

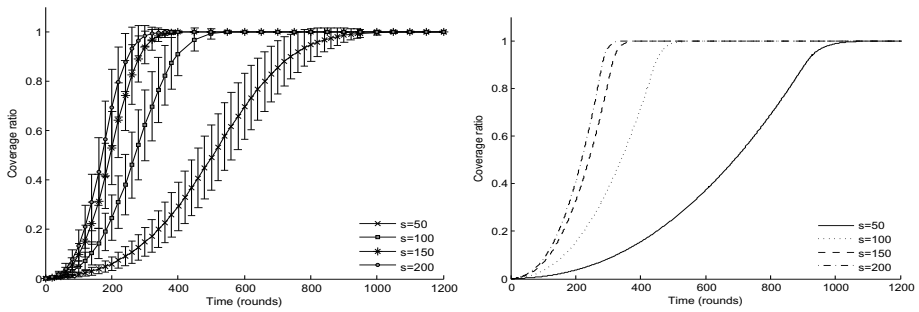


Fig. 6. Coverage ratio in simulations and model for $r = 1$, $n = 2100$, $c = 300$ and different values of s

average and standard deviation over 100 simulation runs, and the curves in the graph at the right are calculated with our model. We separated the results of the model and the simulations to avoid having a tangled view of the curves. These figures demonstrate that our model gives a good estimation of the growth of the replication and coverage ratios over time. Experiments with other values of n, c, s gave similar good results.

In the second experiment, we compared simulation results for two types of topology and the model. We considered a radio range of $r = 2$ for all nodes, and deployed them in a grid as well as in a uniform at random network. Simulation results in both deployments, for different settings of n, c, s , were compared to our model. Figs 7 and 8 present the outcomes for $n = 2100, c = 300$ and $n = 500, c = 100$ respectively and different values for s . For clarity of presentation we leave out the standard deviations in these figures. The curves in each figure contain the average over 100 simulation runs and the outcomes of our model in both deployments. In these figures, the graph at the left represents the comparisons of the replication ratios and the graph at the right represents the comparison of the coverage ratios. The figures demonstrates that our model gives a good estimation of the behavior of the shuffle protocol. Experiments with other values of n, c, s gave similar good results.

In all figures, we expect that our model gives an underestimate of the behavior of the shuffle protocol, but some rounds before the convergence we face to a more speed in our model. This is the result of using the $\overline{\Delta h}$ in our model instead of Δh for simplification which overestimate it when the given item reaches near the border of area. Thus some rounds before convergence $r_{sh}(t + 1)$ increases faster until it reaches to R and consequently the model estimates (not underestimate) the behavior of the shuffle protocol some rounds before the convergence.

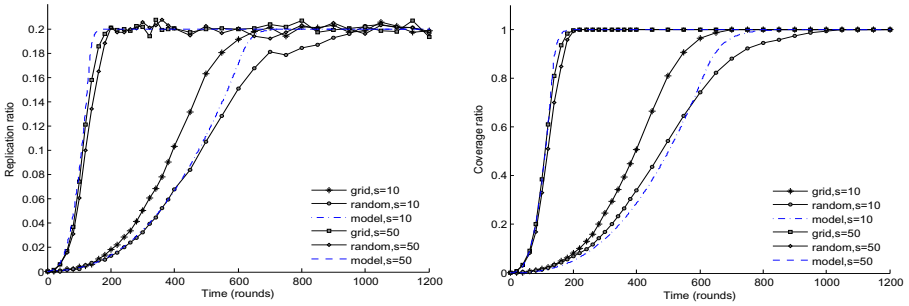


Fig. 7. Replication (left) and coverage (right) ratio in simulations and model for $r = 2$, $n = 500$, $c = 100$ and different values of s in two uniformly deployed WMNs (grid, random)

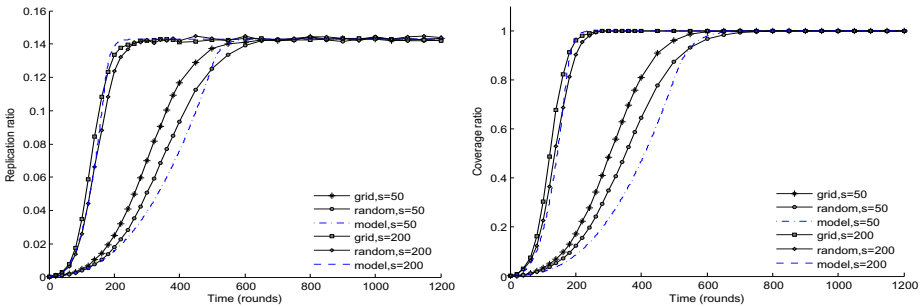


Fig. 8. Replication (left) and coverage (right) ratio in simulations and model for $r = 2$, $n = 2100$, $c = 300$ and different values of s in two uniformly deployed WMNs (grid, random)

5 Conclusion and Future Work

The analytical models for information propagation of a push-pull gossiping protocol are the main contribution of our paper. We have demonstrated that information propagation of a push-pull gossip-based protocol can be analytically modeled in WMNs. To this aim, we considered a disk-like and uniform deployed area in which all nodes have the same radio power. We assumed that links and

nodes do not crash and the network topology is connected. Our models support different uniformly deployed topologies like grids and fully or partially connected meshes.

We have developed differential equations for computing how the replication and coverage ratio of a given item in the network develops over time, for the push-pull gossip-based shuffle protocol; the given item is introduced at the central node at time 0. Simulations with an implementation of the shuffle protocol showed that our analytical models provide an accurate prediction. Our models can help to find optimal parameter settings for the shuffle protocol.

For the future we would like to extend our models for different deployments and areas with different shapes or density. We also intend to extend our models to take into account that nodes may crash or links may be down. We are also curious to explore the usefulness of our approach in modeling other gossip-based protocols in wireless networks.

Acknowledgments. The first author would like to thank Rena Bakhshi for useful discussions.

References

1. Bakhshi, R., Cloth, L., Fokkink, W., Haverkort, B.R.: Mean-field analysis for the evaluation of gossip protocols. In: QEST, pp. 247–256. IEEE Computer Society, Los Alamitos (2009)
2. Bakhshi, R., Gavidia, D., Fokkink, W., van Steen, M.: An analytical model of information dissemination for a gossip-based protocol. *Computer Networks* 53(13), 2288–2303 (2009)
3. Boyd, S.P., Ghosh, A., Prabhakar, B., Shah, D.: Randomized gossip algorithms. *IEEE Transactions on Information Theory* 52(6), 2508–2530 (2006)
4. Dimakis, A.G., Sarwate, A.D., Wainwright, M.J.: Geographic gossip: efficient aggregation for sensor networks. In: IPSN, pp. 69–76. ACM, New York (2006)
5. Eugster, P.T., Guerraoui, R., Kermarrec, A.M., Massoulié, L.: Epidemic information dissemination in distributed systems. *IEEE Computer* 37(5), 60–67 (2004)
6. Gavidia, D., Voulgaris, S., van Steen, M.: Gossip-based distributed news service for wireless mesh networks. In: WONS, pp. 59–67. IEEE, Los Alamitos (2006)
7. Kermarrec, A.M., Massoulié, L., Ganesh, A.J.: Probabilistic reliable dissemination in large-scale systems. *IEEE Trans. Parallel Distrib. Syst.* 14(3), 248–258 (2003)
8. Khelil, A., Becker, C., Tian, J., Rothermel, K.: An epidemic model for information diffusion in manets. In: MSWiM, pp. 54–60. ACM, New York (2002)
9. Luk, V.W.H., Wong, A.K.S., Ouyang, W.R., Lea, C.T.A.: Gossip-based delay-sensitive n-to-n information dissemination protocol. In: GLOBECOM, pp. 2499–2503. IEEE, Los Alamitos (2008)
10. Vogels, W., van Renesse, R., Birman, K.P.: The power of epidemics: robust communication for large-scale distributed systems. *Computer Communication Review* 33(1), 131–135 (2003)

Multi-class Network with Phase Type Service Time and Group Deletion Signal

Thu-Ha Dao-Thi¹, Jean-Michel Fourneau¹, and Minh-Anh Tran²

¹ PRiSM, Université de Versailles St-Quentin, CNRS,
UniverSud Paris, Versailles, France

² Université de Paris Est Créteil, France

{thu-ha.dao-thi,jmf}@prism.uvsq.fr, minh-anh.tran@univ-paris12.fr

Abstract. We consider networks with multiple classes of customers which receive service with a Phase type distribution. The service discipline is Last In First Out. We consider negative signal and a new type of signal: the group deletion signal. Negative signals eliminate a customer in service (if there are any) and group deletion signal delete all consecutive customers in the same class and same phase at the back-end of the buffer. We prove that the network has product form solution.

Keywords: Queue, Network, phase type, quasi-reversible, LIFO, product form.

1 Introduction

Traditional queueing networks model systems are used to represent contention among customers for a set of resources. Customers moves from server to server, waiting for service. But the customers do not interact among themselves or modify the queue or the server. G-network models overcome some of the limitations of conventional queueing network models adding signals and interactions between signals and customers. Despite this deep modification of the model, G-networks still preserve the product form property of some Markovian queueing networks. In his seminal paper [10], Gelenbe introduced negative customer, the first type of signal. A negative customer is never queued. A negative customer deletes a positive customer at its arrival at a backlogged queue. Positive customers are usual customers which are queued and receive service or are deleted by negative customers. Under typical assumptions for Markovian queueing networks (Poisson arrival for both types of customers, exponential service time for positive customers, Markovian routing of customers, open topology, independence) Gelenbe proved that such a network has a product form solution for its steady-state behavior. The results are more complex than Jackson's networks. The G-networks flow equations exhibit some uncommon properties: they are neither linear as in closed queueing networks nor contracting as in Jackson queueing networks. Therefore the existence of a solution had to be proved [11] by new techniques, a numerical algorithm was also developed [8].

G-networks had been extended in many directions. First many signals were introduced and shown to lead to product form solution: triggers which redirect other customers among the queues, catastrophes which flush all the customers out of a queue [12,13] and resets [14]. Multiple class versions of these models have also been derived [6,9,15].

Most of the signals studied so far have a globally negative effect on the queues. Indeed, the balance of customers in the queues involved in a signal is negative (triggers, deletion, catastrophes) or negative in expectation (resets). Recently more complex interactions were introduced: change the class of the customer in service [4], change the phase of the customer in service for Phase type service distribution, synchronised arrivals in a set of queues [5]. For a review, one can see these books [3,16], and many references therein.

G-networks had also motivated new important results in the theory of queues. As negative customers lead to customer deletions, the original description of quasi-reversibility does not hold anymore and new versions have been proposed. At the time being, the description proposed by Chao and his co-authors in [3] looks sufficient to study queues with customers and signals. At the same time a completely different approach, based on Stochastic Process Algebra, was proposed by Harrison [17,18]. The main results (CAT and RCAT theorems and their extensions [1,17,18,19]) give some sufficient conditions for product form stationary distributions. Thus Harrison's technique clearly has a different range of applications as it allows to represent component based models which are much more general than networks of queues. An interesting open question is to mix both results to obtain a quasi-reversibility characterisation directly from a SPA specification using a master-slave description of the system (like in RCAT) rather than arrivals, departure and internal transitions as proposed in [3].

Here we introduce a new type of signal which deletes several customers of the same type (same class and same phase) at the back end of a LIFO queue. Batch deletion were studied by Gelenbe in [12] for single class model. To the best of our knowledge the multiclass problem was not considered until now, except the catastrophe in a PS queue studied in [7] which is easier to model. Moreover, the group-deletion signal is not like the previously studied batch which was studied as we seek to delete all customers of the same type, which means that it will depend on the type of customers, while the effect of a batch or a catastrophe does not depend on class of customers. We also assume that the service time distributions are Phase type.

The following of the paper is as follows. Section 2 is devoted to the definition of quasi-reversibility as it has been generalised by Miyazawa and his co-authors to take into account signals. In section 3, we show that the queues are quasi-reversible. Finally in section 4, using quasi-reversibility we prove that the steady-state has a product form solution.

2 Preliminaries

In this section, we will introduce the network of quasi-reversible queues which is introduced by Chao, Miyazawa and Pinedo in [3]. All the results presented in this

section comes from [3] and have been already used in the context of G-networks in [4] to prove that networks with Phase type services and another type of signal also have a product form steady-state solution. We summarize the results of [3] for the readers.

2.1 Definition of Quasi-reversibility of Chao, Miyazawa and Pinedo

Let us introduce the definition of quasi-reversibility of a queue with signals and instantaneous movement.

Consider a queue where the queue-content evolves as a continuous time Markov chain on state space \mathcal{S} . For a pair of states (\mathbf{x}, \mathbf{y}) , we decompose the transition rate function $q(\mathbf{x}, \mathbf{y})$ of queue into three types of rates: $q_u^A(\mathbf{x}, \mathbf{y}), u \in T$; $q_u^D(\mathbf{x}, \mathbf{y}), u \in T$; $q^I(\mathbf{x}, \mathbf{y})$, where T is the set of the classes of arrivals and departures, which is countable. The transition rate of the queue can be written as:

$$q(\mathbf{x}, \mathbf{y}) = \sum_{u \in T} q_u^A(\mathbf{x}, \mathbf{y}) + \sum_{u \in T} q_u^D(\mathbf{x}, \mathbf{y}) + q^I(\mathbf{x}, \mathbf{y}), \quad \mathbf{x}, \mathbf{y} \in \mathcal{S}.$$

The transition rate functions q_u^A, q_u^D and q^I generate the point processes corresponding to class u arrivals, class u departures and the internal transitions, respectively. ‘‘A’’, ‘‘D’’ and ‘‘I’’ stand for ‘‘arrival’’, ‘‘departure’’ and ‘‘internal’’.

Suppose that q admits a stationary distribution π . Furthermore, assume that when a class u arrives and changes the state of the queue from \mathbf{x} to \mathbf{y} , it instantaneously triggers a class v departure with probability $f_{u,v}(\mathbf{x}, \mathbf{y})$, where:

$$\sum_v f_{u,v}(\mathbf{x}, \mathbf{y}) \leq 1, \quad u \in T, \quad \mathbf{x}, \mathbf{y} \in \mathcal{S}.$$

With probability $1 - \sum_v f_{u,v}(\mathbf{x}, \mathbf{y})$ the class u arrival does not trigger any departure. The function $f_{u,v}(\mathbf{x}, \mathbf{y})$ is the *triggering probability*.

The quasi-reversibility of instantaneous movement is defined as follows.

Definition 2.1. *If there exist two sets of non-negative numbers $\{\alpha_u, u \in T\}$ and $\{\beta_u, u \in T\}$ such that: for all $\mathbf{x} \in \mathcal{S}, u \in T$,*

$$\sum_{\mathbf{y} \in \mathcal{S}} q_u^A(\mathbf{x}, \mathbf{y}) = \alpha_u, \tag{1}$$

$$\sum_{\mathbf{y} \in \mathcal{S}} \pi(\mathbf{y}) \left[q_u^D(\mathbf{y}, \mathbf{x}) + \sum_{v \in T} q_v^A(\mathbf{y}, \mathbf{x}) f_{v,u}(\mathbf{y}, \mathbf{x}) \right] = \beta_u \pi(\mathbf{x}), \tag{2}$$

then the queue with signal is said to be quasi-reversible with respect to $\{q_u^A, f_{u,v}, u, v \in T\}, \{q_u^D, u \in T\}$ and $\{q^I\}$.

The non-negative numbers α_u and β_u are called the arrival rate and departure rate of class u customers.

Example 2.1. Let us give an example of a G-queue $M/M/1/\infty$ with negative signal to introduce the definition. Customers arrive with rate λ , service rate is μ . Negative signals arrive with rate λ^- . A negative signal will eliminate a customer if there is any in the queue.

We use the indices c for customers and $-$ for negative signals. The transition rates of the queue are given by:

$$\begin{aligned} q_c^A(n, n + 1) &= \lambda, \quad n \geq 0, \\ q_c^A(n, n - 1) &= \lambda^-, \quad n \geq 1, \\ q_c^A(0, 0) &= \lambda^-, \\ q_c^D(n, n - 1) &= \mu, \quad n \geq 1. \end{aligned}$$

We add the rate $q_c^A(0, 0)$ for equation (1) to hold for both c and $-$, where $\alpha_c = \lambda$ and $\alpha_- = \lambda^-$. Note that $q_c^A(0, 0)$ is a dummy transition. Therefore it is possible to add such a transition rate.

The stationary distribution π is given by

$$\pi(n) = \pi(0) \left(\frac{\lambda}{\mu + \lambda^-} \right)^n.$$

Equation (2) is satisfied for c with

$$\beta_c = \frac{\lambda\mu}{\mu + \lambda^-}.$$

We now consider the negative signals. We add the triggering probability $f_{-,-}(n+1, n) = 1$ for $n \geq 0$ in the queue. Hence, one obtains equation (2) for $-$ with

$$\beta_- = \frac{\lambda\lambda^-}{\mu + \lambda^-}.$$

Chao et al. proved that this definition of queue without instantaneous movements is equivalent to the quasi-reversible definition of Kelly in [20]. This implies that the arrival processes and the departure (triggered and non-triggered) of class u customers are Poisson.

We use the definition of Chao, Miyazawa and Pinedo as it is more convenient for G-networks with instantaneous movements.

2.2 Network of Quasi-reversible Queues with Signals and Instantaneous Movement

Consider a network of N queues. Each queue is a quasi-reversible queue with signals as described above. The set of arrival and departure classes is T (we may have a set T_i for each queue i , however, for the sake of simplicity, we take $T = \cup_i T_i$).

Let \mathbf{x}_i be the state of queue i with state space \mathcal{S}_i . The Poisson source has index 0 and for the sake of simplification, we assume that the source has only one state which is denoted as 0.

For queue i , we introduce functions p_{iu}^A , q_{iu}^D , q_i^I and $f_{iu,v}$ on the state space \mathcal{S}_i :

- $p_{iu}^A(\mathbf{x}_i, \mathbf{y}_i)$ = the probability that a class u arrival at queue i changes the state from \mathbf{x}_i to \mathbf{y}_i , where it is assumed that $\sum_{\mathbf{y} \in \mathcal{S}_i} p_{iu}^A(\mathbf{x}_i, \mathbf{y}_i) = 1$, $\mathbf{x}_i \in \mathcal{S}_i$;
- $q_{iu}^D(\mathbf{x}_i, \mathbf{y}_i)$ = the rate at which class u departures change the state of queue i from \mathbf{x}_i to \mathbf{y}_i ;
- $q_i^I(\mathbf{x}_i, \mathbf{y}_i)$ = the rate at which internal transitions change the state of queue i from \mathbf{x}_i to \mathbf{y}_i ;
- $f_{iu,v}(\mathbf{x}_i, \mathbf{y}_i)$ = the triggering probability that when a class u arrival occurs at queue i and the state changes from \mathbf{x}_i to \mathbf{y}_i , it simultaneously induces a class v departure, where $\sum_{v \in T} f_{iu,v}(\mathbf{x}_i, \mathbf{y}_i) \leq 1$, $i \leq N$, $u \in T$, $\mathbf{x}_i, \mathbf{y}_i \in \mathcal{S}_i$.

For source 0, we set $p_{0u}^A(0, 0) = 1$, $p_{0u}^A(0, 0) = \beta_{0u}$, $q_0^I(0, 0) = 0$ and $f_{0u,v} \equiv 0$. Here, β_{0u}^A is the arrival rate to the network from the outside (the source).

In Chao's model, a queue is defined by three rates q_u^A , q_u^D and q^I . In that case, the arrival effect function may be defined as:

$$p_u^A(\mathbf{x}, \mathbf{y}) = \frac{q_u^A(\mathbf{x}, \mathbf{y})}{\sum_z q_u^A(\mathbf{x}, \mathbf{z})},$$

and q_u^D and q^I are the departure and internal transition functions.

The dynamics of the network are described as follows. Customers of class u arrive to the network from outside (the source) according to a Poisson process with rate β_{0u} , and are routed to queue i as a class v arrival with probability $r_{0u,iv}$. A class u departure from queue i , either trigger or non-trigger, enters queues j as a class v arrival with probability $r_{iu,jv}$. It is assumed that:

$$\sum_{j=0}^N \sum_v r_{iu,jv} = 1, \quad i = 0, 1, \dots, N, \quad u \in T.$$

Furthermore, whenever there is a class u arrival at queue i , either from the outside or from other queues, it makes the state of the queue change from \mathbf{x}_i to \mathbf{y}_i with probability $p_{iu}^A(\mathbf{x}_i, \mathbf{y}_i)$, it also triggers a class u departure with probability $f_{iu,v}(\mathbf{x}_i, \mathbf{y}_i)$, and it triggers no departure from queue i with probability $1 - \sum_{v \in T} f_{iu,v}(\mathbf{x}_i, \mathbf{y}_i)$, $i = 0, 1, \dots, N$.

The transition rate function of the network is denoted by $q(\mathbf{x}, \mathbf{y})$, $\mathbf{x}, \mathbf{y} \in \mathcal{S} = \mathcal{S}_1 \times \dots \times \mathcal{S}_N$ (note that we accept the case where $q(\mathbf{x}, \mathbf{x}) \neq 0$).

Consider for each queue i the following auxiliary process:

$$q_i^{(\alpha_i)}(\mathbf{x}_i, \mathbf{y}_i) = \sum_{u \in T} \left(\alpha_{iu} p_{iu}^A(\mathbf{x}_i, \mathbf{y}_i) + q_{iu}^D(\mathbf{x}_i, \mathbf{y}_i) \right) + q_i^D(\mathbf{x}_i, \mathbf{y}_i),$$

where $(\alpha_i) = (\alpha_{iu}, u \in T)$ are considered as dummy parameters and their values are determined by the traffic equations.

Suppose that $q_i^{(\alpha_i)}$ has a stationary distribution $\pi_i^{(\alpha_i)}$. Note that this is always true for the source 0, as for all α_0 , $\pi_0^{(\alpha_0)(0)} = 1$. We now require that $q_i^{(\alpha_i)}$ be quasi-reversible.

We always have

$$\sum_{\mathbf{y}_i \in \mathcal{S}_i} \alpha_{iu} p_{iu}^A(\mathbf{x}_i, \mathbf{y}_i) = \alpha_{iu}, \quad i = 1, \dots, N, u \in T.$$

Hence, the quasi-reversibility of $q_i^{(\alpha_i)}$ for $i = 1, \dots, N$ is equivalent to the existence of a set of non-negative numbers $\beta_{iu}, u \in T$ such that:

$$\begin{aligned} \sum_{\mathbf{y}_i} \pi_i^{(\alpha_i)}(\mathbf{y}_i) \left[q_{iu}^D(\mathbf{y}_i, \mathbf{x}_i) + \sum_{v \in T} \alpha_{iv} p_{iv}^A(\mathbf{y}_i, \mathbf{x}_i) f_{iv,u}(\mathbf{y}_i, \mathbf{x}_i) \right] \\ = \beta_{iu} \pi_i^{(\alpha_i)}(\mathbf{x}_i), \end{aligned} \tag{3}$$

for all $\mathbf{x}_i \in \mathcal{S}_i, i = 1, \dots, N$ and $u \in T$.

Queue i in isolation is said to be quasi-reversible with α_i if (3) is satisfied.

Since α_{iu} and β_{iu} are the arrival and the departure rates of class u customers at queue i , we have the following traffic equations:

$$\alpha_{iu} = \sum_{j=0}^N \sum_v \beta_{jv} r_{jv,iu}, \quad i = 0, 1, \dots, N. \tag{4}$$

We need the following condition to ensure that the network process is regular:

$$\sum_{i=1}^N \sum_{\mathbf{x}_i \in \mathcal{S}_i} \pi_i^{(\alpha_i)} \sum_{\mathbf{y}_i \in \mathcal{S}_i} q_i^{(\alpha_i)}(\mathbf{x}_i, \mathbf{y}_i) < \infty.$$

We have the theorem:

Theorem 2.1. *If each queue i with signals, $i = 1, \dots, N$, is quasi-reversible with α_i which are the solution to the traffic equations (4), then the queueing network with signal has the product form stationary distribution*

$$\pi(\mathbf{x}) = \prod_{i=1}^N \pi_i^{(\alpha_i)}(\mathbf{x}_i),$$

where $\pi_i^{(\alpha_i)}$ is the stationary distribution of $q_i^{(\alpha_i)}, i = 1, \dots, N$.

3 A LIFO Multi-class PH Queue with Signal Deletion Customers of Same Sub-class

The goal is to model a generalized network of multiple classes of (positive) customers where the service times of each class are assumed to be Phase type and 2 types of signal: negative signal and group deletion signal.

In [2], Bonald and Tran modelled the Phase type service by considering a change of class inside the queue after service. More precisely, the phase is modelled by an absorbing DTMC with $k + 1$ states where 0 is the only absorbing

state. The transition probability matrix of this DTMC is denoted as H . It can be viewed as follow: each “phase” ph demands an exponential service time. After “phase” ph , a customer can change to some “phase” ph' with some probability (given by matrix H) and still stay in the queue. The arriving rate of each “phase” will respect the ratio of the initial probability of the Phase-type.

Example 3.1. Consider a queue of 2 classes of customers:

- Class 1 which arrives with rate λ_1 and asks for exponential service of rate μ_1 ;
- Class 2 which arrives with rate λ_2 and asks a PH service with two transient states (and an absorbing state): ph, ph' and initial distribution ν . State ph and ph' demands exponential services time μ_2 and μ'_2 .

Hence, the matrix H is given by

$$H = \left(\begin{array}{c|cc|c} 0 & 0 & 0 & H[1,0] \\ \hline 0 & 0 & H[ph, ph'] & H[ph, 0] \\ \hline 0 & H[ph', ph] & 0 & H[ph', 0] \\ \hline 0 & 0 & 0 & 0 \end{array} \right),$$

where

- $H[1, 0] = 1$, which means that if a customer reach phase 1, then after service completion, it will reach the absorbing state.
- $H[ph, ph']$ and $H[ph', ph]$ are the probabilities in which after service completion of phase ph and ph' , respectively, customer of class 2 can ask for another “phase” ph' , ph , respectively.
- $H[ph, 0]$ and $H[ph', 0]$ are the probabilities in which after of phase ph and ph' , respectively, customer of class 2 will reach the absorbing state.

Then, inside the queue, there are changes of class after service (in this case, ph to ph' and ph' to ph). The arrival rate for phase 1 is λ_1 , while it is $\lambda_2\nu(ph)$ and $\lambda_2\nu(ph')$ for phase 2 (ph) and 3 (ph').

We will use this presentation to model our network. A multi-class network of queues with phase type service times can be modelled as a multi-class network of queues with exponential service times where the customers can change class inside the queue after completion of an exponential service.

3.1 Description

Consider the LIFO multi-class queue with the set of customers given by \mathcal{C} and a special class index 0 ($0 \notin \mathcal{C}$) which denotes the “absorbing state”. There are two types of signals: negative signals and group-deletion signals. The set of negative signals is given by $\mathcal{S}^- = \{s_c^-\}_{c \in \mathcal{C}}$ and the set of group-deletion signals is given by $\mathcal{S} = \{s_c\}_{s \in \mathcal{C}}$.

Remark 3.1. We can consider only one class of negative signal and one class of group-deletion signal, with different probability of success for each class of customer. However, we consider different classes to have more flexible routing while connecting the networks.

Customers of class c arrive according to a Poisson process of rate $\lambda^{(c)}$, they require exponential service times with mean $1/\mu^{(c)}$, for $c \in \mathcal{C}$. Denote by λ the sum of all λ^c : $\lambda = \sum_{c \in \mathcal{C}} \lambda^{(c)}$. A customer of class c after completion of service will change to class k (it demands another service) with probability $H[c, k]$ or reach the state 0 (it quits the queue at the completion of its service) with probability $H[c, 0]$. The following condition is satisfied:

$$H[c, 0] + \sum_{k \in \mathcal{C}} H[c, k] = 1.$$

Negative signals of class s_c^- arrive according to a Poisson process of rate $\lambda^-(c)$. Arriving signal of type s_c^- will eliminate a customer c in service (customer at the back-end of the buffer).

Group deletion signals of class s_c arrive according to a Poisson process of rate $\lambda^s(c)$. If customer in service is of class c , then the arriving signal of class s_c will cancel all consecutive customers of class c at the back-end of the buffer (it will eliminate all customers of class c until it finds a customer of another class).

If the queue length is n , then the state of queue is

$$\mathbf{x} = (x(1), x(2), \dots, x(n)),$$

where $x(l)$ is the class of customer in position l .

Remark 3.2. We consider also triggering effect in the queue to have instantaneous movements when considering network model. However, details of the triggering effect will be given later when we study the quasi-reversibility.

3.2 Stationary Distribution

We first give the system of equation which plays an important role in calculating the stationary distribution.

Definition 3.1. *The PH Group-deletion Equations associated to the considered LIFO queue are the equations of the variable $\boldsymbol{\rho} = \{\rho(c)\}_{c \in \mathcal{C}} \in \mathbb{R}_+^{\mathcal{C}}$, defined by*

$$\lambda(c) + \sum_{k \in \mathcal{C}} \rho(k)\mu(k)H[k, c] = \frac{\rho(c)\lambda^s(c)}{1 - \rho(c)} + \rho(c)\mu(c) + \rho(c)\lambda^-(c), \quad \text{if } \lambda^s(c) > 0, \quad (5)$$

$$\lambda(c) + \sum_{k \in \mathcal{C}} \rho(k)\mu(k)H[k, c] = \rho(c)(\mu(c) + \lambda^-(c)), \quad \text{if } \lambda^s(c) = 0. \quad (6)$$

Let \mathcal{C}^+ be a subset of \mathcal{C} : $\mathcal{C}^+ = \{c \in \mathcal{C} \mid \lambda^s(c) > 0\}$.

Remark 3.3. If ρ is a solution of (5,6), then taking the sum over all c in \mathcal{C} , and using the fact that $H[c, 0] + \sum_{k \in \mathcal{C}} H[c, k] = 1$, one has

$$\begin{aligned} \lambda &= \sum_{c \in \mathcal{C}} (\rho(c)\mu(c) + \rho(c)\lambda^-(c)) + \sum_{c \in \mathcal{C}^+} \frac{\rho(c)\lambda^s(c)}{1 - \rho(c)} - \sum_{c, k \in \mathcal{C}} \rho(k)\mu(k)H[k, c] \\ &= \sum_{c \in \mathcal{C}} \rho(c)\mu(c)H[c, 0] + \sum_{c \in \mathcal{C}} \rho(c)\lambda^-(c) + \sum_{c \in \mathcal{C}^+} \frac{\rho(c)\lambda^s(c)}{1 - \rho(c)}. \end{aligned}$$

We now give lemmas which prove the existence of the solution of *PH Group-deletion Equations*. We first prove this property in the extreme cases: $\mathcal{C}^+ = \mathcal{C}$ or $\mathcal{C}^+ = \emptyset$. Then we treat the general case.

Lemma 3.1. *If $\mathcal{C}^+ = \emptyset$ (which means that there is no deletion-group signal in the queue), then there exists a solution $(\rho(c))_{c \in \mathcal{C}}$ of the system (5,6).*

Proof. The system can be rewritten as follows:

$$\rho = \eta + \rho M,$$

where $\eta = (\eta(c))_{c \in \mathcal{C}}$ is a vector and $M = M[c, k]_{c, k \in \mathcal{C}}$ is a matrix the entries of which are given by:

$$\eta(c) = \frac{\lambda(c)}{\mu(c) + \lambda^-(c)}, \quad M[c, k] = \frac{\mu(k)H[k, c]}{\mu(c) + \lambda^-(c)}.$$

One has that $rank(Id - M) = |\mathcal{C}|$ as at least one variable $H[c, 0] > 0$. Hence, there exist a unique solution given by

$$\rho = \eta(Id - M)^{-1}.$$

This completes the proof.

Lemma 3.2. *If $\mathcal{C}^+ = \mathcal{C}$ (which means that $H[c, k] = 0$ for all $c, k \in \mathcal{C}$), then there exists a solution $(\rho(c))_{c \in \mathcal{C}}$ of the system (5,6) which satisfies*

$$0 \leq \rho(c) < 1.$$

Proof. $\rho(c)$ is a root of a polynomial of degree 2:

$$\begin{aligned} P^c(\rho(c)) &= \rho(c)^2(\mu(c) + \lambda^-(c) - \mu(c)H[c, c]) - \rho(c)(\lambda^s(c) + \mu(c) + \lambda^-(c)) \\ &\quad + \sum_{k \neq c} \rho(k)\mu(k)H[k, c] - \mu(c)H[c, c] + (\lambda[c] + \sum_{k \neq c} \rho(k)\mu(k)H[k, c]) \end{aligned}$$

Polynomial $P^c(X) = 0$ has a solution in $[0,1]$ as $P^c(0) \geq 0$ and $P^c(1) < 0$. As the multiplication of the two solutions of $P^c(X)$ is positive ($\lambda(c)/(\mu(c) + \lambda^-(c))$), we have that $P^c(X) = 0$ has 2 positive roots. It implies that $P^c(X)$ has a unique solution in $[0,1]$ (which is depending on $\rho(k)_{k \neq c}$).

Consider the function $\Phi(\rho) : [0, 1]^{\mathcal{C}^+} \rightarrow [0, 1]^{\mathcal{C}^+}$ where $\Phi(\rho)(c)$ is the unique solution in $[0,1)$ of $P^c(X)$. As $[0, 1]^{\mathcal{C}^+}$ is a non-empty compact, then applying Brouwer's fixed point theorem, one has that there exists a fixed point solution in $[0, 1]^{\mathcal{C}^+}$ of the equation: $\Phi(\rho) = \rho$, or $\Phi(\rho)(c) = \rho(c)$. Clearly the solution will be in the set $[0, 1]^{\mathcal{C}^+}$.

This completes the proof.

We now have the result in the general case.

Lemma 3.3. *There exists a solution $(\rho(c))_{c \in \mathcal{C}}$ of the system (5,6) which satisfies*

$$0 \leq \rho(c) < 1, \quad \text{for all } c \in \mathcal{C}^+.$$

Proof. Consider $c \in \mathcal{C} \setminus \mathcal{C}^+$. Similarly to Lemma 3.1, one has that there exists a solution $\rho_1 = (\rho(c))_{c \in \mathcal{C} \setminus \mathcal{C}^+}$ which depends on $\rho_2 = (\rho(c))_{c \in \mathcal{C}^+}$ determined by:

$$\rho_1 = \eta_1(Id - M_1)^{-1},$$

where

$$\eta_1(c) = \frac{\lambda(c) + \sum_{k \in \mathcal{C}^+} \mu(k)H[k, c]}{\mu(c) + \lambda^-(c)}, \quad M_1[c, c'] = \frac{\mu(c')H[c', c]}{\mu(c) + \lambda^-(c)}, \quad \text{for } c, c' \notin \mathcal{C}^+.$$

This implies that for $c \notin \mathcal{C}$, $\rho(c)$ is a linear combination of $(\rho(c))_{c \in \mathcal{C}^+}$, which is denoted by $\Phi_1(c)$.

For $c \in \mathcal{C}^+$, consider the polynomial

$$\begin{aligned} P^c(X) &= X^2(\mu(c) + \lambda^-(c)) - X(\lambda^s(c) + \mu(c) + \lambda^-(c) + \sum_{k \in \mathcal{C}^+} \rho(k)\mu(k)H[k, c]) \\ &\quad + \sum_{k \notin \mathcal{C}^+} \Phi_1(k)\mu(k)H[k, c] + (\lambda(c) + \sum_{k \in \mathcal{C}^+} \rho(k)\mu(k)H[k, c]) \\ &\quad + \sum_{k \notin \mathcal{C}^+} \Phi_1(k)\mu(k)H[k, c]. \end{aligned}$$

Similarly to Lemma 3.2, one has that there is a unique solution in $[0, 1)$ of $P^c(X)$.

Consider the function $\Phi_2(\rho_2) : [0, 1]^{\mathcal{C}^+} \rightarrow [0, 1]^{\mathcal{C}^+}$ where $\Phi(\rho)(c)$ is the unique solution in $[0,1)$ of $P^c(X)$. Similarly to Lemma 3.2, we have a solution to the fixed point equation: $\Phi_2(\rho_2)(c) = \rho_2(c)$ which satisfies $\rho_2(c) \in [0, 1)$.

This completes the proof.

Lemma 3.4. *Let ρ be a solution to the system of equations (5,6). The considered LIFO queue has an invariant measure p defined by*

$$p(x(1), x(2), \dots, x(n)) = \prod_{l \leq n} \rho(x(l)). \tag{7}$$

Proof. To prove that p is an invariant measure, one has to check that for all $\mathbf{x} = x(1), x(2), \dots, x(n)$, one has

$$\sum_{\mathbf{y}} p(\mathbf{x})Q(\mathbf{x}, \mathbf{y}) = \sum_{\mathbf{y}} p(\mathbf{y})Q(\mathbf{y}, \mathbf{x}),$$

where Q is the infinitesimal generator of the queue.

The left-hand side is given by

$$L = \lambda, \quad \text{if } n = 0,$$

or

$$L = p(\mathbf{x})(\lambda + \mu(x(n)) + \lambda^-(x(n)) + \lambda^s(x(n))), \quad \text{if } n > 0.$$

Using the equation $\sum_{k>0} \rho(c)^k = \rho(c)/(1 - \rho(c))$ when $\rho(c) < 1$, the right-hand side is given by

$$R = \sum_{c \in \mathcal{C}} \rho(c)\mu(c)H[c, 0] + \sum_{c \in \mathcal{C}} \rho(c)\lambda^-(c) + \sum_{c \in \mathcal{C}^+} \frac{\rho(c)\lambda^s(c)}{1 - \rho(c)}, \quad \text{if } n = 0$$

or

$$R = \frac{p(\mathbf{x})}{\rho(x(n))} \left\{ \lambda(x(n)) + \rho(x(n)) \sum_{k \in \mathcal{C}} \rho(k)\lambda^-(k) + \rho(x(n)) \sum_{k \in \mathcal{C}} \rho(k)\mu(k)H[k, 0] \right. \\ \left. + \sum_{k \in \mathcal{C}^+, k \neq x(n)} \frac{\rho(k)\lambda^s(k)}{1 - \rho(k)} \rho(x(n)) + \sum_{k \in \mathcal{C}} \rho(k)\mu(k)H[k, x(n)] \right\}, \quad \text{if } n > 0.$$

Let us comment about the expression of R . In the sum, the first term corresponds to an arrival; the second term corresponds to an elimination caused by a negative signal; the third term corresponds to a completion of service of phase k to reach absorbing state 0 and leave the queue; the fourth term corresponds to an elimination caused by a group deletion signal of class s_k , for $k \neq x(n)$; and the last term corresponds to a service completion of phase k which provokes another service of phase $x(n)$.

When $n = 0$, we have $L = R$ as in remark 3.3. Using this equation, for $n > 0$, one has

$$R \frac{\rho(x(n))}{p(\mathbf{x})} = \lambda(x(n)) + \sum_{k \in \mathcal{C}} \rho(k)\mu(k)H[k, x(n)] \\ + \rho(x(n)) \left(\lambda - \mathbb{1}_{x(n) \in \mathcal{C}^+} \frac{\rho(x(n)\lambda^s(x(n)))}{1 - \rho(x(n))} \right)$$

The equations (5,6) imply that

$$R \frac{\rho(x(n))}{p(\mathbf{x})} = \rho(x(n))(\mu(x(n)) + \lambda^-(x(n)) + \lambda) + \mathbb{1}_{x(n) \in \mathcal{C}^+} \rho(x(n)\lambda^s(x(n))),$$

which yields that $L = R$ when $n > 0$.

This completes the proof.

We now have the main result for one queue.

Theorem 3.1. *Consider the LIFO multi-class PH queue with negative signal and group-deletion signal where the variables are $(\lambda(c), \mu(c), \lambda^-(c), \lambda^s(c))$. Let ρ be a solution of the PH Group-deletion Equations (6,5) which satisfies $\rho(c) < 1$ for $c \in \mathcal{C}^+$.*

If we have

$$\sum_{c \in \mathcal{C}} \rho(c) < 1$$

then the queue is stable and the stationary distribution is given by: for $\mathbf{x} = \{x(1), \dots, x(n)\}$

$$\pi(\mathbf{x}) = K \prod_1^n \rho(x(l)), \tag{8}$$

where K is the normalization constant given by $K = 1 - \sum_{c \in \mathcal{C}} \rho(c)$.

Proof. If $\sum_{c \in \mathcal{C}} \rho(c) < 1$, then one has that $\sum_{\mathbf{x}} \pi(\mathbf{x}) = 1$. Using Lemma 3.4, one has that π is the stationary distribution. This completes the proof.

3.3 Quasi-reversibility

In this section, we will study the quasi-reversibility as defined by Chao et. al.

Consider the departure process of customer of class c , one has

$$\frac{\sum_{\mathbf{y}} q_c^D(\mathbf{y}, \mathbf{x}) \pi(\mathbf{y})}{\pi(\mathbf{x})} = \rho(c) \mu(c) H[c, 0].$$

Consider the negative signal of class s_c^- , one has

$$\frac{\sum_{\mathbf{y}} q_{s_c^-}^A(\mathbf{y}, \mathbf{x}) \pi(\mathbf{y})}{\pi(\mathbf{x})} = \rho(c) \lambda^-(c).$$

The problem is more complicated while considering the group-deletion signal of class s_c , one has

$$\frac{\sum_{\mathbf{y}} q_{s_c}^A(\mathbf{y}, \mathbf{x}) \pi(\mathbf{y})}{\pi(\mathbf{x})} = \begin{cases} \sum_{k \geq 1} \rho(c)^k \lambda^s(c) = \frac{\lambda^s(c) \rho(c)}{1 - \rho(c)}, & \text{if } x(n) \neq c; \\ 0, & \text{if } x(n) = c. \end{cases}$$

Hence, we have the quasi-reversibility for customers which reach the absorbing state after finishing service. We can add triggering probability for negative signal when there are a successful elimination

$$f_{s_c^-, s_c^-}(x(1) \cdots x(n)c, x(1) \cdots x(n)) = 1,$$

then we also have the quasi-reversibility for negative signal.

To have the quasi-reversibility for group-deleting signal, one needs to modify the network as follows:

- Triggering probability for group-deleting signal when there are a successful group-deletion: $f_{s_c, s_c}(x(1) \cdots x(n)c^k, x(1) \cdots x(n)) = 1$,
- When the state of the queue is $x(1) \cdots x(n)$, then there is a process of group-deletion signal of class $s_{x(n)}$ is active to departure with rate $\lambda^s(c)\rho(c)/(1 - \rho(c))$.

We now have all ingredients to study the network.

4 Network of LIFO Multi-class PH Queues with Negative Signal and Group-Deletion Signal

4.1 Description

Consider the network of N LIFO multi-class Ph queues with negative signal and group-deletion signal.

At queue i , the set of classes of customers is given by \mathcal{C}^i . The arrival rate and service rate of customer of type c are $\lambda^i(c)$ and $\mu^i(c)$, respectively. The matrix which determines the service rate is given by $H^i[c, k]$ and $H^i[c, 0]$ for $c, k \in \mathcal{C}^i$. The arrival rate of negative signal of type s_c^- is $\lambda^{i,-}(c)$. The arrival rate of group-deletion signal of type s_c is $\lambda^{i,s}(c)$.

At queue i , after service completion at phase c , a customer asks another service of phase k with probability $H^i[c, k]$; or reaches absorbing state 0 at queue i with probability $H^i[c, 0]$, and then

- leaves to queue j as a customer of class c' with probability $P^{i,j}(c, c')$, as a negative signal of class $s^- c'$ with probability $P^{i,j}(c, s_c^-)$, as a group-deletion signal of class $s_{c'}$ with probability $P^{i,j}(c, s_{c'})$,
- or quits the network with probability $d^i(c)$.

The following condition is satisfied:

$$\sum_j \sum_{c' \in \mathcal{C}^j} (P^{i,j}(c, c')P^{i,j}(c, s_c^-)P^{i,j}(c, s_{c'})) + d^i(c) = 1.$$

A negative signal of class s_c^- arrives to queue i finding customer of class c in service will eliminate this customer, and then

- moves to queue j as a customer of class c' with probability $P^{i,j}(s_c^-, c')$, as a negative signal of class $s^- c'$ with probability $P^{i,j}(s_c^-, s_{c'}^-)$, as a group-deletion signal of class $s_{c'}$ with probability $P^{i,j}(s_c^-, s_{c'})$,
- or quits the network with probability $d^i(s_c^-)$.

The following condition is satisfied:

$$\sum_j \sum_{c' \in \mathcal{C}^j} (P^{i,j}(s_c^-, c')P^{i,j}(s_c^-, s_{c'}^-)P^{i,j}(s_c^-, s_{c'})) + d^i(s_c^-) = 1.$$

A group-deletion signal of class s_c arrive to queue i finding customer of class c in service will cancel all customers of class c at the back-end of the buffer.

Remark 4.1. We know that we can generalize this mechanism of the group-deletion signal with probability of successful deletion in the network as we will have the quasi-reversibility. However, for the sake of simplicity, we will not introduce the generalized version of group-deletion signal.

4.2 Stationary Distribution

Definition 4.1. *The Traffic Equations of the network are given by*

$$\begin{aligned} \Lambda^i(c) &= \lambda^i(c) + \sum_{k \in \mathcal{C}^i} \rho^i(k) \mu^i(k) H^i[k, c] \\ &\quad + \sum_{j, k \in \mathcal{C}^j} \rho^j(k) \{ \mu^j(k) H[k, 0] P^{j,i}(k, c) + \Lambda^{j,-}(k) P^{j,i}(s_k^-, c) \}, \end{aligned} \quad (9)$$

$$\Lambda^{i,-}(c) = \lambda^{i,-}(c) + \sum_{j, k \in \mathcal{C}^j} \rho^j(k) \{ \mu^j(k) H[k, 0] P^{j,i}(k, s_c^-) + \Lambda^{j,-}(k) P^{j,i}(s_k^-, s_c^-) \}, \quad (10)$$

$$\Lambda^{i,s}(c) = \lambda^{i,s}(c) + \sum_{j, k \in \mathcal{C}^j} \rho^j(k) \{ \mu^j(k) H[k, 0] P^{j,i}(k, s_c) + \Lambda^{j,-}(k) P^{j,i}(s_k^-, s_c) \}, \quad (11)$$

where ρ^i is the solution to the PH group-deletion Equations associated to the variables $(\Lambda^i(c), \Lambda^{i,-}(c), \Lambda^{i,s}(c), \mu^i(c))$.

Remark 4.2. As mentioned in section 3.3, we can consider a modified network to have the quasi-reversible property for group deletion signal. However, we do not give the detail in this paper. We only consider the “simple” network as presented.

We now have the main result to the paper.

Theorem 4.1. *Consider the network of N LIFO multi-class PH queues with negative signals and group-deletion signals. If there exists a solution to the traffic equations (9,10,11) which satisfies: for all i*

$$\sum_{c \in \mathcal{C}^i} \rho^i(c) < 1,$$

then the network is stable and the stationary distribution has a product form given by

$$\pi(\mathbf{x}^1, \dots, \mathbf{x}^N) = \prod_{i=1}^N \pi^i(\mathbf{x}^i) = K \prod_{i=1}^N \rho^i(x^i(1) \cdots x^i(n_i)),$$

where $\mathbf{x}^i = (x^i(1) \cdots x^i(n_i))$ and K is the normalisation constant.

The theorem can be deduced by using the result in Theorem 2.1.

5 Concluding Remarks

We hope that this new result will improve the applicability of G-network to model systems with complex destruction mechanism of customers. Extension to other queueing discipline is not that trivial because of the combinatorial problem for description of set of customers to be deleted when the queueing discipline is a general symmetric discipline as defined by Kelly.

References

1. Balsamo, S., Harrison, P.G., Marin, A.: A unifying approach to product-forms in networks with finite capacity constraints. In: Misra, V., Barford, P., Squillante, M.S. (eds.) SIGMETRICS, pp. 25–36. ACM, New York (2010)
2. Bonald, T., Tran, M.A.: On Kelly networks with shuffling. *Queueing Syst. Theory Appl.* 59(1), 53–61 (2008)
3. Chao, X., Miyazawa, M., Pinedo, M.: *Queueing networks: Customers, signals, and product form solutions*. Wiley, Chichester (1999)
4. Dao-Thi, T.-H., Fourneau, J.-M., Tran, M.-A.: Multiple class symmetric g-networks with phase type service times. *Comput. J.* 54(2), 274–284 (2011)
5. Dao-Thi, T.-H., Fourneau, J.-M., Tran, M.-A.: G-networks with synchronised arrivals. *Perform. Eval.* 68(4), 309–319 (2011)
6. Fourneau, J.-M., Gelenbe, E., Suros, R.: G-networks with multiple classes of negative and positive customers. *Theoret. Comput. Sci.* 155(1), 141–156 (1996)
7. Fourneau, J.-M., Kloul, L., Quessette, F.: Multiple class G-Networks with jumps back to zero. In: MASCOTS 1995: Proceedings of the 3rd International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 28–32. IEEE Computer Society, Washington, DC (1995)
8. Fourneau, J.-M.: Computing the steady-state distribution of networks with positive and negative customers. In: 13th IMACS World Congress on Computation and Applied Mathematics, Dublin (1991)
9. Fourneau, J.-M.: Closed G-networks with resets: product form solution. In: Fourth International Conference on the Quantitative Evaluation of Systems (QEST 2007), Edinburgh, UK, pp. 287–296. IEEE Computer Society, Los Alamitos (2007)
10. Gelenbe, E.: Product-form queueing networks with negative and positive customers. *J. Appl. Probab.* 28(3), 656–663 (1991)
11. Gelenbe, E.: Stability of G-Networks. *Probability in the Engineering and Informational Sciences* 6, 271–276 (1992)
12. Gelenbe, E.: G-Networks with signals and batch removal. *Probability in the Engineering and Informational Sciences* 7, 335–342 (1993)
13. Gelenbe, E.: G-networks with instantaneous customer movement. *Journal of Applied Probability* 30(3), 742–748 (1993)
14. Gelenbe, E., Fourneau, J.-M.: G-networks with resets. *Perform. Eval.* 49(1-4), 179–191 (2002)
15. Gelenbe, E., Labed, A.: G-networks with multiple classes of signals and positive customers. *European Journal of Operational Research* 108(2), 293–305 (1998)
16. Gelenbe, E., Mitrani, I.: *Analysis and Synthesis of Computer Systems*. Imperial College Press (World Scientific), London (2010)
17. Harrison, P.G.: G-networks with propagating resets via RCAT. *SIGMETRICS Performance Evaluation Review* 31(2), 3–5 (2003)
18. Harrison, P.G.: Compositional reversed Markov processes, with applications to G-networks. *Perform. Eval.* 57(3), 379–408 (2004)
19. Harrison, P.G.: Product-forms and functional rates. *Performance Evaluation* 66(11), 660–663 (2009)
20. Kelly, F.: *Reversibility and stochastic networks*. Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons Ltd., Chichester (1979)

Critical Level Policies in Lost Sales Inventory Systems with Different Demand Classes

Aleksander Wieczorek^{1,4}, Ana Bušić¹, and Emmanuel Hyon^{2,3}

¹ INRIA and ENS, Paris, France

² Université Paris Ouest Nanterre, Nanterre, France

³ LIP6, UPMC, Paris, France

⁴ Institute of Computing Science, Poznan University of Technology, Poznan, Poland

aleksanderwieczorek@gmail.com, ana.busic@inria.fr,

emmanuel.hyon@u-paris10.fr

Abstract. We consider a single-item lost sales inventory model with different classes of customers. Each customer class may have different lost sale penalty costs. We assume that the demands follow a Poisson process and we consider a single replenishment hypoexponential server. We give a Markov decision process associated with this optimal control problem and prove some structural properties of its dynamic programming operator. This allows us to show that the optimal policy is a critical level policy. We then discuss some possible extensions to other replenishment distributions and give some numerical results for the hyperexponential server case.

Keywords: Markov decision process, critical level policies, inventory systems.

1 Introduction

We consider an inventory system with single item type and a finite stock capacity. Upon a demand, we can decide to either deliver an item or reject the demand. We assume different demand classes with different rejection penalty costs. The objective is to find a sequence of optimal decisions that minimizes the costs. This problem can be easily modeled as a Markov Decision Process (MDP). We refer the reader to Puterman [12] for general theory on MDPs and to Porteus [11] for inventory systems and MDPs.

Finding a sequence of optimal decisions (an optimal policy) is in general a very difficult problem. When the state space and the set of possible decisions (actions) are finite, one can solve the problem numerically. However, this can be done effectively only for very small instances. The aim is therefore to prove some structural properties of an optimal policy, which then simplifies the computation by reducing significantly the search space of all admissible policies.

We are interested here in the class of critical level policies. A critical level policy can be seen as a set of thresholds (a threshold by class) which determine the optimal action to perform (either rejection or acceptance of a demand). Such

policies and their applications have been largely studied in the literature. We will focus here only on the results closely related to our problem; for a survey with applications to communication networks see Altman [1] or Koole [7], and for applications to web services see Mazzucco et al. [10].

In the case of an exponential replenishment server, Ha [4] has shown the optimality of a critical level policy. This result was further extended by de Véricourt, Karaesmen and Dallery [14], by authorizing the backordering of unsatisfied demands, where critical level policy are also shown to be optimal. Both previous results assume exponential service times. Dekker et al. [2], and Kranenburg and van Houtum [8] consider general service times and they exhibit exact and heuristic methods to compute the real value of the optimal levels, under the assumption of a critical level policy. Nevertheless, there is no warranty that the critical level policies are optimal under the general service distributions.

In the case of Erlang distributed service times, the optimality of a critical level policy was proven by Ha [5]. However, his model relies on the assumption that the service operator has a constant rate in any state of the model.

We consider here the case of hypoexponential service (see Svoronos [13] for examples of complex services in inventory problems). We prove the optimality of critical level policies. Note that this represents the simplest case of a phase-type service where the service rate depends on the service phase. Also, we consider a two-dimensional state space that is more suitable to possible extensions to other distributions. This new state-space representation allows us to prove that the switching curve is monotone in k (service phase). We expect that our results represent a first step towards general phase-type service. Our numerical experiments suggest that the result may still be preserved in the hyperexponential case.

This paper is organized as follows. In Section 2 we give the formal description of the MDP problem considered in the paper. Section 3 is devoted to the hypoexponential service case. We prove first in Section 3.3 that the dynamic programming operator preserves convexity, which then has two different applications: i) that the optimal policy is of threshold type and ii) that the thresholds are ordered for different classes. The preservation of submodularity, proved in Section 3.5, implies the monotonicity of the switching curve for a fixed demand class. In Section 4 we discuss the possible extensions of the model. Section 5 contains conclusions and some future directions. Details of proofs can be found in the extended version of this article [15].

2 Problem Formulation

2.1 Model Description

We consider a single-item inventory model. We denote by S the maximum stock level. This single-item is demanded by J different types of customers (or demand classes). Demands arrive according to a Poisson process of parameter λ and the probability that an arriving demand is of class j is equal to p_j . Let λ_j be the arrival rate of class j . Then $\lambda_j = p_j\lambda$ and $\sum_{j=1}^J \lambda_j = \lambda$.

When an item is not delivered, the item is lost and a penalty cost is incurred. We denote by c_j the penalty cost of demand class j and we assume without loss of generality that the costs are ordered such that $c_1 \leq c_2 \leq \dots \leq c_J$. When a demand arrives, one has to decide whether to satisfy or to refuse it. Since the stock level is finite, rejecting a demand causes a penalty cost but it also saves an item that can be used for future demands (of possibly higher class).

We assume a single replenishment server and the replenishment times can follow different distributions. In this paper we will consider hypoexponential and hyperexponential replenishments.

We detail now how this inventory problem can be expressed as an admission control problem in a queuing system with different classes of customers. We represent the items in replenishment for the inventory system by a queue of capacity S . An arriving demand of class j is an arriving customer of class j . Delivering an item is the equivalent to admitting the customer into the queue. The penalty cost due to a lost demand is equivalent to the cost of rejection. From now on, we will use this queuing model formalism.

2.2 Optimal Control Problem

The problem of finding the best decision (rejection or acceptance), to perform when customers arrive is an optimal control problem that can be modeled as a (time-homogeneous) MDP. We detail now the elements of this MDP.

The system behaves in continuous time, but we can restrict our attention to the transition epochs. This can be done without loss of generality since we only consider costs that depend on the state of the system. At these transition instants an event occurs and induces a change in the system. When the event is an arrival, then we have to choose the best action to trigger. Although the system behaves continuously, we consider here a discrete time MDP. Indeed, the use of the uniformization standard method makes the problem discrete (see Lippman [9] or Puterman [12]).

The state space of our MDP is $\mathcal{X} = \{1, \dots, S\} \times \{1, \dots, N\} \cup \{(0, 1)\}$. For any $(x, k) \in \mathcal{X}$, the parameter x denotes the number of customers in the system, while k describes the phase of the customer being served (hypoexponential and hyperexponential have different phases). Actually, in state $(0, 1)$ no customer is served and the indication of the phase is useless but we keep it for readability. Note that the state space does not depend on the classes of customers.

We denote by $\mathcal{A} = \{0, 1\}$ the action set where action 1 is an acceptance and action 0 is a rejection. A *policy* π is a sequence of decision rules that maps the information history (past states and actions) to the action set \mathcal{A} . Since in our case the action set and the state space are finite, and since the costs are thus trivially bounded, we restrict our attention to Markov deterministic policies. Indeed, standard results (see Puterman [12]) insure that an optimal policy belongs to the set of Markov deterministic policies. A Markov deterministic policy is of the form $(a(\cdot), a(\cdot), \dots)$ where $a(\cdot)$ is a single deterministic decision rule that maps

the current state to a decision (hence, in our case $a(\cdot)$ is a function from \mathcal{X} to \mathcal{A}). Under a policy π the evolution of the system generates a random sequence of states and actions y_n and a_n .

We denote by $C(y, a)$ with $y \in \mathcal{X}$ and $a \in \mathcal{A}$ the instantaneous cost in state y when action a is triggered. Here $C(y, 0) = c_j$ according to the class of the customer we reject, and $C(y, 1) = 0$ in case of acceptance.

Since we are interested in the average cost criteria, we define the objective, known as the *minimal long-run average costs*, by

$$\bar{v}^* = \min_{\pi} \lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{E}_y^{\pi} \left(\sum_{\ell=0}^{n-1} C(y_{\ell}, a_{\ell}) \right),$$

where the initial state $y_0 = y$. Our aim is to characterize the optimal policy π^* that reaches \bar{v}^* . This means to determine the optimal decision rule denoted by a^* , with $\pi^* = (a^*, a^*, \dots)$.

Total cost criteria. We adopt the framework of Koole [7] and we will work with the total cost criteria to characterize the optimal average policy. We define the *minimal (expected) n-stage total cost* by

$$V_n(y) = \min_{\pi(n)} \mathbb{E}_y^{\pi(n)} \left(\sum_{\ell=0}^{n-1} C(y_{\ell}, a_{\ell}) \right), \forall y \in \mathcal{X}, \tag{1}$$

where $\pi(n)$ is a policy over n steps. We assume that the initial state $y_0 = y$ and that, by convention, $V_0(y) = 0, \forall y \in \mathcal{X}$. We call V_n the n -stage total cost value function.

From Koole [7] and Puterman [12, Chapter 8], the minimal n -stage total cost value function V_n does not converge when n tends to infinity. Instead, the difference $V_{n+1}(y) - V_n(y)$ converges to the minimal long run average cost. On the other hand, still from Koole [7], the n -stage policy which minimizes V_n tends to the optimal average policy π^* when n tends to infinity.

Finally, since the instantaneous costs are positive, from standard results on MDPs (see Puterman [12]), V_n is well defined for all n and satisfies the dynamic programming equation (Bellman equation):

$$V_{n+1} = TV_n,$$

where T is the dynamic programming operator. More precisely, T operates on the total cost value functions and is defined by:

$$(Tf)(y) = \min_a (\hat{T}f)(y, a) = \min_a \left(C(y, a) + \sum_{y' \in \mathcal{X}} \mathbb{P}(y'|y, a) f(y') \right), \tag{2}$$

with $\mathbb{P}(y'|y, a)$ the probability to move to y' from state $y \in \mathcal{X}$ with action $a \in \mathcal{A}$, and f a function that belongs to the total cost value function set. The optimal decision rule can then be deduced from Eq. (2):

$$a(y) = \min \left(\arg \min (\hat{T}f)(y, a) \right). \tag{3}$$

2.3 Critical Level Policies

In this work, we do not want to numerically compute the optimal policy but rather to characterize some “good” properties that the optimal policy should verify. Indeed, very often the optimal policy admits a simple form. The aim of our work is to prove that the optimal policy for our model is a critical level policy.

Definition 1 (Critical level policy). *A policy is called a critical level policy if for any fixed k and any customer class j it exists a level in x : $t_{k,j}$, such that in state (x, k) :*

- for all $0 \leq x < t_{k,j}$ it is optimal to accept any customer of class j ,
- for all $x \geq t_{k,j}$ it is optimal to reject any customer of class j .

Structured policy framework. In order to show that the optimal policy exhibits some special properties we use a standard framework presented in the monograph of Koole [7]. Most of the results presented now follow from this monograph in which detailed proofs can be found. This framework is that of property preservation by the Dynamic Programming operator. It consists of three steps: First, identify two related sets. The one of the value functions that exhibit a special form (say \mathcal{V}) and the one one of the decision rules \mathcal{D} that exhibit a special form. The relation between these two sets being such that when the value function belongs to \mathcal{V} , then the optimal decision belongs to \mathcal{D} . Next, show that the properties of $v \in \mathcal{V}$ are conserved by the operator T . Finally, check that these properties are kept when passing to the limit.

We present now some special forms for value functions. Let f be a function from $\mathcal{X} \mapsto \mathbb{R}_+$.

Definition 2 (Monotonicity). *We say that a function f is increasing in x (denoted by $\text{Incr}(x)$) if, for all $y = (x, k)$, we have $f(x+1, k) \geq f(x, k)$. Similarly, we say that a function f is decreasing in k (denoted by $\text{Decr}(k)$) if, for all $y = (x, k)$, we have $f(x, k+1) \leq f(x, k)$.*

Definition 3 (Convexity). *We say that a function f is convex in x (denoted by $\text{Convex}(x)$) if, for all $y = (x, k)$, we have:*

$$2f(x+1, k) \leq f(x, k) + f(x+2, k). \quad (4)$$

Definition 4 (Submodularity). *We say that a function f is submodular in x and k (denoted by $\text{Sub}(x, k)$) if, for all $y = (x, k)$, we have:*

$$f(x+1, k+1) + f(x) \leq f(x+1, k) + f(x, k+1). \quad (5)$$

We give now the relationship between the properties of the value functions and the optimal decisions.

Proposition 1 (Th 8.1 [7]). *Let $a(y)$, with $y = (x, k)$, be the optimal decision rule defined by Eq. (3).*

- i) If $f \in \text{Convex}(x)$, then $a(y)$ is decreasing in x .
- ii) If $f \in \text{Sub}(x, k)$, then $a(y)$ is increasing in k .

Assume a critical level policy and consider a decision for a fixed demand class j .

Definition 5 (Switching curve). For every k , we define a level $t(k)$ such that when we are in state (x, k) decision 1 is taken if and only if $x \leq t(k)$ and 0 otherwise. The mapping $k \mapsto t(k)$ is called a switching curve.

Let us notice that a switching curve separates the state space \mathcal{X} in two regions \mathcal{X}_0 , in which decision 0 is taken, and \mathcal{X}_1 , in which decision 1 is taken.

Definition 6 (Monotone switching curve). We say that a decision rule is of the monotone switching curve type if the mapping $k \mapsto t(k)$ is monotone.

3 Hypoexponential Service

In order to show that the optimal level policy is of critical level type we will prove increasingness (Section 3.2) and convexity (Section 3.3) properties, while submodularity (proved in Section 3.5) implies monotonicity of the switching curve. Furthermore, although the problem was given under its most general formulation in Section 2.2, for the sake of readability and in order to simplify the proofs, in the rest of the paper we will decompose the dynamic programming operator according to the event-based approach presented in Glasserman and Yao [3], and Koole [7]. This means a decomposition of the dynamic programming operator into operators related to a dedicated event and an observation of the system after decision epoch.

3.1 Model Description

Let us detail the event-based decomposition. The dynamic programming equation becomes:

$$V_n(x, k) = T_{unif} \left(\sum_{i=1}^J p_i T_{CA(i)}(V_{n-1}), T_D(V_{n-1}) \right), \tag{6}$$

where $V_0(x, k) \equiv 0$, and T_{unif} , $T_{CA(i)}$ and T_D are different event operators.

If the service (replenishment) time is hypoexponential, then the random value R that describes the service time is defined by $R = \sum_{k=1}^N E_k$, where E_k is an exponential random value of parameter μ_k , $1 \leq k \leq N$. Let $\alpha = \max_k \mu_k$. We will use the uniformization rate equal to $\lambda + \alpha$. The uniformization operator, T_{unif} , is defined by:

$$T_{unif}(f(x, k), g(x, k)) = \frac{\lambda}{\lambda + \alpha} f(x, k) + \frac{\alpha}{\lambda + \alpha} g(x, k).$$

Let $\mu'_k = \mu_k/\alpha$, the departure operator, $T_D f$, is defined by:

$$T_D f(x, k) = \mu'_k \begin{cases} f(x, k+1) & \text{if } (k < N) \text{ and } (x > 0), \\ f((x-1)^+, 1) & \text{if } (k = N) \text{ or } (x = 0 \text{ and } k = 0) \end{cases} \\ + (1 - \mu'_k) f(x, k).$$

The controlled arrival operator of a customer of class i , $T_{CA(i)}$, is defined by:

$$T_{CA(i)} f(x, k) = \begin{cases} \min\{f(x+1, k), f(x, k) + c_i\} & \text{if } x < S, \\ f(x, k) + c_i & \text{if } x = S. \end{cases}$$

3.2 Increasing Property

We first notice that the state space admits a linear order (denoted \preceq) which is a lexicographic order defined to be increasing with respect to the first dimension and decreasing with respect to the second dimension:

Definition 7 (Linear order \preceq). For every (x, i) and (y, j) in \mathcal{X} ,

$$(x, i) \preceq (y, j) \iff (x < y) \text{ or } (x = y, i \geq j).$$

Let us define the increasing property of functions with respect to \preceq .

Definition 8 (Increasing property). For any function $f : \mathcal{X} \mapsto \mathbb{R}$ we say that $f \in \text{Incr}(\preceq)$ if $(x, i) \preceq (y, j) \Rightarrow f(x, i) \leq f(y, j)$.

We now show that operators $T_{CA(i)}$ and T_D preserve this increasing property.

Lemma 1. Let $f : \mathcal{X} \mapsto \mathbb{R}$ such that $f \in \text{Incr}(\preceq)$, then $T_{CA(i)} f \in \text{Incr}(\preceq)$.

Proof. Assume that $f \in \text{Incr}(\preceq)$. In order to prove that $T_{CA(i)} f \in \text{Incr}(\preceq)$, it sufficient to prove that:

$$T_{CA(i)} f(x, k) \leq T_{CA(i)} f(x, k-1), \forall x > 0, \forall k > 1, \quad (7)$$

$$T_{CA(i)} f(x, 1) \leq T_{CA(i)} f(x+1, N), \forall x, \forall k. \quad (8)$$

Let us consider first (7). For $0 < x < S$ and $k > 1$ we get:

$$T_{CA(i)} f(x, k) = \min\{f(x, k) + c_i, f(x+1, k)\} \\ \leq \min\{f(x, k-1) + c_i, f(x+1, k-1)\} = T_{CA(i)} f(x, k-1),$$

For $x = S$ and $k > 1$ we have $T_{CA(i)} f(S, k) = f(S, k) + c_i \leq f(S, k-1) + c_i = T_{CA(i)} f(S, k-1)$.

We now prove (8). For $x < S-1$ we get:

$$T_{CA(i)} f(x, 1) = \min\{f(x, 1) + c_i, f(x+1, 1)\} \\ \leq \min\{f(x+1, k) + c_i, f(x+2, k)\} = T_{CA(i)} f(x+1, k).$$

For $x = S-1$, $T_{CA(i)} f(S-1, 1) = \min\{f(S-1, 1) + c_i, f(S, 1)\} \leq f(S, k) + c_i = T_{CA(i)} f(S, k)$. \square

Lemma 2. *Let $f : \mathcal{X} \mapsto \mathbb{R}$ such that $f \in \text{Incr}(\preceq)$, then $T_D f \in \text{Incr}(\preceq)$.*

Proof. Assume that f is $\text{Incr}(\preceq)$. In order to prove that $T_D f \in \text{Incr}(\preceq)$, it sufficient to prove that:

$$T_D f(x, k) \leq T_D f(x, k - 1), \forall x > 0, \forall k > 1, \tag{9}$$

$$T_D f(x, 1) \leq T_D f(x + 1, N), \forall x, \forall k. \tag{10}$$

Let us consider first (9). For $x > 0$ and $k = N$,

$$\begin{aligned} T_D f(x, N) &= \mu'_N f(x - 1, 1) + (1 - \mu'_N) f(x, N) \\ &\leq \mu'_{N-1} f(x, N) + (1 - \mu'_{N-1}) f(x, N - 1) = T_D f(x, N - 1). \end{aligned}$$

Indeed, using Definitions 8, one has that both $f(x - 1, 1)$ and $f(x, N)$ are smaller or equal than $f(x, N)$ and $f(x, N - 1)$ and the same holds for any of their convex combinations. Similarly, for $x > 0$ and $1 < k < N$ we have:

$$\begin{aligned} T_D f(x, k) &= \mu'_i f(x, k + 1) + (1 - \mu'_i) f(x, k) \\ &\leq \mu'_{k-1} f(x, k) + (1 - \mu'_{k-1}) f(x, k - 1) = T_D f(x, k - 1). \end{aligned}$$

Let us now consider (10). For $x > 0$:

$$\begin{aligned} T_D f(x, 1) &= \mu'_1 f(x, 2) + (1 - \mu'_1) f(x, 1) \\ &\leq \mu'_N f(x, 1) + (1 - \mu'_N) f(x + 1, N) = T_D f(x + 1, N) \end{aligned}$$

For $x = 0$: $T_D f(0, 1) = f(0, 1) = T_D f(1, N)$. □

Remark 1. From Definition 7, it follows that $\text{Incr}(\preceq)$ implies $\text{Incr}(x)$ and $\text{Decr}(k)$.

3.3 Convexity

In this part we prove that the convexity with respect to the first dimension is preserved by operators $T_{CA(i)}$ and T_D . However, due to the state space form, this requires to introduce an additional property (called augmented convexity) at state $(0, 1)$.

Definition 9 (Augmented convexity). *We say that a function $f : \mathcal{X} \mapsto \mathbb{R}$ is augmented convex in x (denoted by $A\text{Convex}(x)$) if for all $k \in \{1, \dots, N\}$ we have $f(0, 1) + f(2, k) \geq 2f(1, k)$.*

Lemma 3. *Let f be a function such that $f \in \text{Incr}(\preceq) \cap \text{Convex}(x) \cap A\text{Convex}(x)$, then $T_{CA(i)} f \in \text{Incr}(\preceq) \cap \text{Convex}(x) \cap A\text{Convex}(x)$.*

Proof. Assume that f is $\text{Incr}(\preceq) \cap \text{Convex}(x) \cap A\text{Convex}(x)$.

(i) Preservation of $\text{Convex}(x)$ shall be proved for $x > 0$. For $x = S - 2$,

$$\begin{aligned} T_{CA(i)} f(x, k) + T_{CA(i)} f(S, k) &= \min \left\{ \begin{aligned} &f(x + 1, k) + f(S, k) + c_i \\ &f(x, k) + c_i + f(S, k) + c_i \geq 2f(x + 1, k) + 2c_i \\ &\geq 2 \min \{ f(S, k), f(x + 1, k) + c_i \} = 2T_{CA(i)} f(x + 1, k). \end{aligned} \right. \end{aligned}$$

For $0 < x < S - 2$, the sum $T_{CA(i)}f(x, k) + T_{CA(i)}f(x + 2, k)$ is equal to

$$\min \begin{cases} f(x + 1, k) + f(x + 3, k) \geq 2f(x + 2, k) \\ f(x + 1, k) + f(x + 2, k) + c_i \\ f(x, k) + c_i + f(x + 3, k) + f(x + 2, k) - f(x + 2, k) \\ \geq 2f(x + 1, k) + f(x + 3, k) - f(x + 2, k) + c_i \\ \geq f(x + 1, k) + f(x + 2, k) + c_i \\ f(x, k) + c_i + f(x + 2, k) + c_i \geq 2f(x + 1, k) + 2c_i \end{cases} \\ \geq 2 \min\{f(x + 2, k), f(x + 1, k) + c_i\} = 2T_{CA(i)}f(x + 1, k).$$

Since for every expression on the left hand side of the inequality there exists a smaller or equal expression on the right hand side thereof, the minimum on the right hand side is smaller than the one on the left hand side.

(ii) For $x = 0$ augmented convexity is preserved:

$$T_{CA(i)}f(0, 1) + T_{CA(i)}f(2, k) = \min \begin{cases} f(1, 1) + f(3, k) \geq f(1, k) + f(3, k) \geq 2f(2, k) \\ f(1, 1) + f(2, k) + c_i \\ f(0, 1) + c_i + f(3, k) + f(2, k) - f(2, k) \\ \geq 2f(1, k) + f(3, k) - f(2, k) + c_i \\ \geq f(1, k) + f(2, k) + c_i \\ f(0, 1) + c_i + f(2, k) + c_i \geq 2f(1, k) + 2c_i \end{cases} \\ \geq 2 \min\{f(2, k), f(1, k) + c_i\} = 2T_{CA(i)}f(1, k),$$

Preservation of $\text{Incr}(\preceq)$ follows from Lemma 1. □

Lemma 4. *Let f be a function such that $f \in \text{Incr}(\preceq) \cap \text{Convex}(x) \cap \text{AConvex}(x)$, then $T_Df \in \text{Incr}(\preceq) \cap \text{Convex}(x) \cap \text{AConvex}(x)$.*

Proof. Assume that f is in $\text{Incr}(\preceq) \cap \text{Convex}(x) \cap \text{AConvex}(x)$.

(i) Preservation of $\text{Convex}(x)$ shall be proved for $x > 0$. For $x > 0$ and $k < N$:

$$T_Df(x, k) + T_Df(x + 2, k) \\ = \mu'_k f(x, k + 1) + (1 - \mu'_k) f(x, k) + \mu'_k f(x + 2, k + 1) + (1 - \mu'_k) f(x + 2, k) \\ \geq 2\mu'_k f(x + 1, k + 1) + 2(1 - \mu'_k) f(x + 1, k) = 2T_Df(x + 1, k)$$

For $x > 0$ and $k = N$:

$$T_Df(x, N) + T_Df(x + 2, N) \\ = \mu'_N f(x - 1, 1) + (1 - \mu'_N) f(x, N) + \mu'_N f(x + 1, 1) + (1 - \mu'_N) f(x + 2, N) \\ \geq 2\mu'_N f(x, 1) + 2(1 - \mu'_N) f(x + 1, N) = 2T_Df(x + 1, N)$$

(ii) For $x = 0$ augmented convexity is preserved. For $x = 0$ and $k < N$:

$$T_Df(0, 1) + T_Df(2, k) = \mu'_1 f(0, 1) + (1 - \mu'_1) f(0, 1) + \mu'_k f(2, k + 1) + (1 - \mu'_k) f(2, k) \\ = \mu'_k f(0, 1) + (1 - \mu'_k) f(0, 1) + \mu'_k f(2, k + 1) + (1 - \mu'_k) f(2, k) \\ \geq 2\mu'_k f(1, k + 1) + 2(1 - \mu'_k) f(1, k) = 2T_Df(1, k)$$

For $x = 0$ and $k = N$:

$$\begin{aligned} T_D f(0, 1) + T_D f(2, N) &= \mu'_1 f(0, 1) + (1 - \mu'_1) f(0, 1) + \mu'_N f(1, 1) + (1 - \mu'_N) f(2, N) \\ &= \mu'_N f(0, 1) + \mu'_N f(1, 1) + (1 - \mu'_N) f(0, 1) + (1 - \mu'_N) f(2, N) \\ &\geq 2\mu'_N f(0, 1) + 2(1 - \mu'_N) f(1, N) = 2T_D f(1, N) \end{aligned}$$

Preservation of $\text{Incr}(\preceq)$ follows from Lemma 2. □

Since the operator T_{unif} is a convex sum of operators $T_{CA(i)}$ and T_D , the following proposition follows by induction from Lemmas 3 and 4 (see details in [15]).

Proposition 2. *Let V_n be a n -steps total cost value function that satisfy Equation 6. Then, for all $n \geq 0$, V_n is in $\text{Incr}(\preceq) \cap \text{AConvex}(x) \cap \text{Convex}(x)$.*

3.4 Critical Level Policy

The next step is to show that the optimal policy, as defined in Section 2.2, is a critical level policy (see Definition 1).

Theorem 1. *The optimal policy is a critical level policy.*

Proof. First, any value function V_n satisfying Eq. (6) belongs to $\text{Convex}(x)$ this follows from Proposition 2. From Proposition 1 it follows that, for any n , the action minimizing V_n when $V_n \in \text{Convex}(x)$ defines a critical level policy. Finally, from the discussion in 2.2 it follows that the policy minimizing V_n converges (with a simple convergence) to the optimal policy when n tends to infinity. Since convexity is kept with simple convergence, then the optimal policy is of critical level policy type. □

Example 1. Consider a model with the following parameters as an example: $J = 3$ (number of customer classes), $N = 5$ (number of phases), $S = 10$ (stock size), $\lambda = 3$, $\mu = [2, 6, 9, 4, 7]$ (service rates in different phases), $p = [0.3, 0.4, 0.3]$ (probabilities of J customer classes), $c = [30, 40, 50]$ (rejection costs).

Numerical computation of the optimal policy using value iteration method results in a critical level policy as presented in Figure 1. It can be seen that for every customer class j and for every phase k there exists a unique threshold in x : $t_{k,j}$. These thresholds are represented on the figure as the transition between acceptance for different classes (blue circle – all classes are accepted, green triangle – classes 2 and 3 are accepted, pink square – only class 3 is accepted, red asterisk – rejection of any class). This example and Figure 1 also serve as an illustration for Theorems 2 and 3.

Now we show that when the rejection costs are ordered for consecutive customer classes, one can deduce order on the levels in x .

Theorem 2. *For any critical level policy (see Definition 1), if the rejection costs are nondecreasing ($c_1 \leq \dots \leq c_J$), then the levels $t_{k,j}$ are nondecreasing with respect to customer class j , i.e. $t_{k,j} \leq t_{k,j+1}$.*

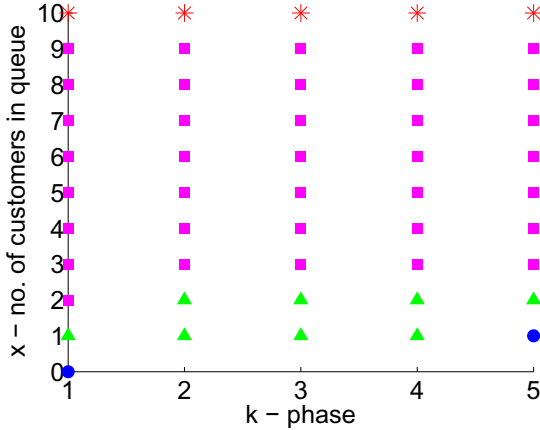


Fig. 1. Acceptance points for different customer classes. Blue circle – all classes are accepted, green triangle – classes 2 and 3 are accepted, pink square — only class 3 is accepted, red asterisk – rejection of any class.

Proof. First recall that the operator $T_{CA(j)}$ is defined as: $T_{CA(j)}f(x, k) = \min \{f(x+1, k), f(x, k) + c_j\}$ for $x < S$ (for $x = S$ the only allowed action is rejection).

Observe now that by definition $t_{k,j}$ is a number such that: in state (x, k) for all $0 \leq x < t_{k,j}$ it is optimal to accept any customer of class j and to reject it for all $x \geq t_{k,j}$. This, by definition of $T_{CA(j)}$, means that:

$$\begin{aligned} V_n(x + 1, k) - V_n(x, k) &< c_j \quad \forall x < t_{k,j} \\ V_n(x + 1, k) - V_n(x, k) &> c_j \quad \forall x \geq t_{k,j} \end{aligned} \tag{11}$$

Owing to the assumption $c_j \leq c_{j+1}$, it yields $V_n(x + 1, k) - V_n(x, k) < c_j \leq c_{j+1}$ which, by definition of $t_{k,j+1}$ holds for all $x < t_{k,j+1}$. But, by Eq. (11), this also holds for all $x < t_{k,j}$. This means that the set of x such that $x < t_{k,j}$ is a subset of the set of x such that $x < t_{k,j+1}$, which, by definition of $t_{k,j}$ implies that $t_{k,j} \leq t_{k,j+1}$. □

3.5 Submodularity

In this part we prove the preservation by operators $T_{CA(i)}$ and T_D of submodularity with respect to the first and second dimension ($Sub(x, k)$). Similarly to convexity, we need an additional property (boundary-submodularity) and subsequently its preservation along with submodularity shall be proved too.

Definition 10 (Boundary-submodularity). We say that a function f is in $BSub(x, k)$ if $\forall 0 < x < S$ we have $f(x, 1) + f(x, N) \leq f(x - 1, 1) + f(x + 1, N)$.

Lemma 5. Let f be a function such that $f \in Sub(x, k) \cap BSub(x, k)$. Then $T_{CA(i)}f \in Sub(x, k) \cap BSub(x, k)$.

The proof is given in [15].

Lemma 6. *Let f be a function such that $f \in \text{Sub}(x, k) \cap \text{BSub}(x, k)$. Then $T_D f \in \text{Sub}(x, k) \cap \text{BSub}(x, k)$.*

The proof is given in [15].

The combination of Lemmas 5 and 6 implies that the n -step value total cost value function is submodular (the proof follows easily from (6) and the definition of T_{unif}).

Proposition 3. *Let V_n be a n -step total cost value function which satisfies Eq. (6). Then for all $(n \geq 0)$ V_n is in $\text{Sub}(x, k) \cap \text{BSub}(x, k)$.*

3.6 Monotone Switching Curve

Henceforth we can now refine the optimal policy using Definition 5.

Theorem 3. *The optimal policy defines a monotone switching curve.*

Proof. For all $n \geq 0$, a value function V_n that checks Eq. (6) belongs to $\text{Sub}(x, k)$ (this follows from Proposition 3). Thus, by Theorem 1, we know that action is decreasing in x and that there exists a critical level policy. Since the action set consists of two elements and since the switching curve is the boundary between inverse images of these two elements with respect to the Markov deterministic policy, then Theorem 1 yields that for a fixed customer class the boundary (switching curve) is a function $k \mapsto t_{k,j}$. Furthermore, from Prop. 1 it follows that the action minimizing V_n for $V_n \in \text{Sub}(x, k)$ is increasing in k . This, combined with the previous observation, implies that the switching curve is increasing.

Observe now, that (from the discussion in 2.2) the policy minimizing V_n converges to the optimal policy when n tends to infinity. Since submodularity is kept by simple convergence, thus the optimal policy is a monotone switching curve.

At last, it has to be noted that submodularity is not well defined for the state $(0, 1)$ (indeed when $x = 0$ only one phase is possible). Therefore the same reasoning based on Proposition 1 cannot be applied to the zero state. However, Theorem 1 ensures that an unique level $t_{1,j}$ for $k = 1$ exists and submodularity, by application of Proposition 1, implies the fact that the switching curve is increasing for $x > 0$. Finally, the only event not covered by the previous discussion (*i.e.* the existence of the switching curve at $(0, 1)$) does not impair the monotonicity of the switching curve, since it is the smallest possible state. \square

4 Model Extensions

We discuss the possible extensions of the model in two different directions: hyperexponential service times and including different instantaneous costs.

4.1 Hyperexponential Service

We now assume that the service times follow a hyperexponential distribution (an exponential distribution of rate μ_k with a probability γ_k). Let V_n be the n -step value function, the dynamic programming equation is similar: $V_n(x, k) = T_{unif} \left(\sum_{i=1}^J p_i T_{CA(i)}(V_{n-1}), T_D(V_{n-1}) \right)$, with $V_0(x, k) \equiv 0$. However, since the dynamics of the system changes compared to the hypoexponential case, the definitions of controlled arrivals $T_{CA(i)}$ and departure operators T_D differ.

Let us recall that α is the maximum over all μ_k and that $\mu'_k = \frac{\mu_k}{\alpha}$. We assume, without loss of generality, that $\mu_1 \leq \mu_2 \leq \dots \leq \mu_N$. The departure operator T_D is now defined by:

$$T_D f(x, k) = \mu'_k \begin{cases} \sum_{i=1}^N a_i f(x-1, i) & \text{if } x > 1 \\ f(0, 1) & \text{if } x \leq 1 \end{cases} + (1 - \mu'_k) f(x, k),$$

and the controlled arrivals $T_{CA(i)}$ by:

$$T_{CA(i)} f(x, k) = \begin{cases} f(x, k) + c_i & \text{if } x = S, \\ \min\{f(x+1, k), f(x, k) + c_i\} & \text{if } 0 < x < S, \\ \min\{f(0, 1) + c_i, \sum_{j=1}^N a_j f(1, j)\} & \text{if } x = 0. \end{cases}$$

Example 2. Consider a model with the following parameters: $J = 3$ (number of customer classes), $N = 5$ (number of phases), $S = 10$ (stock size), $\lambda = 3$, $\mu = [2, 4, 6, 7, 9]$ (service rates for different phases), $p = [0.3, 0.4, 0.3]$ (probabilities of J customer classes), $c = [0.5, 1, 3]$ (rejection costs). The optimal policy is given in Figure 2. For this example, and many others that we have tested, the optimal policy is still of critical level type. This suggests that even in the hyperexponential service case, the optimal policies are critical level policies. However, since the arrival operator is now more complex at the zero state, the proof technique used for hypoexponential case cannot be used directly, and were not able to prove any results yet about critical level optimality.

4.2 Adding Holding Costs

We now discuss the possibility of adding the holding costs to Eq. (6). As mentioned in De Véricourt et al. [14], this breaks the queueing model and the inventory system similarities.

First note that the admission control in a queueing system model with holding costs has a wide range of applications (see Kleywegt et al. [6]). In the context of inventory systems, it models the make-to-stock problem (see Ha [4]).

We obtain the similar results as before when we add linear and increasing holding costs in admission control model.

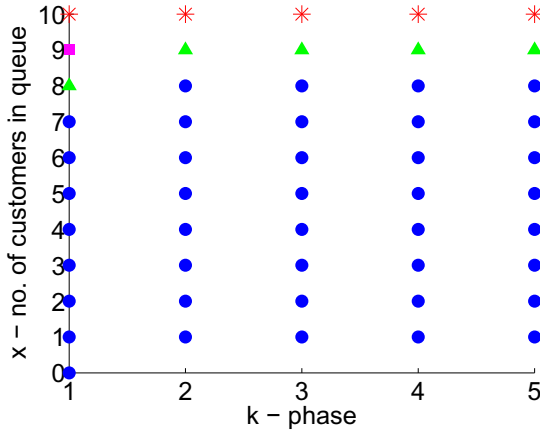


Fig. 2. Acceptance points for different customer classes. Blue circle – all classes are accepted, green triangle – classes 2 and 3 are accepted, pink square – only class 3 is accepted, red asterisk – rejection of any class.

Corollary 1 (Critical level policy with holding costs). Let T_{cost} be the operator defined by

$$T_{cost}f(x, k) = \frac{c_h x}{\lambda + \alpha} + f(x, k),$$

with c_h the per capita holding cost. The value function is given by $V_n(x, k) = T_{cost}(T_p(V_{n-1}))$ where T_p is the operator given by Eq. (6).

The optimal policy is of critical level policy type. The levels are increasing with respect to the customer class and the optimal policy defines a monotone switching curve.

Proof. We only give a sketch of proof by noticing that the increasing, convexity, and submodularity properties are kept by the operator. Then the results follow using similar arguments as in the proofs of Theorems 1, 2, and 3. \square

5 Conclusions

The main contribution of this paper is the proof that in the case of a hypoexponential service case we still have an optimal policy of critical level type. This extends the previous results from the literature to the simplest case of a phase-type service where the service rate depends on the service phase. We expect that our result is a first step towards more general results for phase-type service. Our numerical experiments suggest that the result is still preserved for hyperexponential case. However, the particularity of the zero state (i.e. when the queue is empty) becomes more challenging because of the possibility of choice in the arrival operator when the queue is empty. Technically, on the boundary around zero state convexity and submodularity must be replaced with some more elaborate properties. Defining the boundary properties for general phase type service

and finding the conditions under which they are preserved by the dynamical programming operator represents our first direction for future extensions of this work. The second direction concerns the extensions of the result for different types of immediate costs. Finally, it would be interesting to consider other types of arrival processes or adding the possibility of decision for service.

Acknowledgements. This work was supported by INRIA ARC OCOQS grant.

References

1. Altman, E.: Applications of Markov Decision Processes in communication networks. In: Feinberg, E.A., Shwartz, A. (eds.) *Handbook of Markov Decision Processes*, ch.16. Kluwer, Dordrecht (2002)
2. Dekker, R., Hill, R., Kleijn, M., Teunter, R.: On the (S-1,S) lost sales inventory model with priority demand classes. *Naval Research Logistics* 49, 593–610 (2002)
3. Glasserman, P., Yao, D.: *Monotone Structure in Discrete-Event Systems*. Wiley, Chichester (1994)
4. Ha, A.: Inventory rationing in a make-to-stock production system with several demand classes and lost sales. *Management Science* 43(8), 1093–1103 (1997)
5. Ha, A.: Stock rationing in an $M/E_k/1$ make-to-stock queue. *Management Science* 46(1), 77–87 (2000)
6. Kleywegt, A.J., Papastavrou, J.D.: The dynamic and stochastic knapsack problem. *Operations Research* 46, 17–35 (1998)
7. Koole, G.: Monotonicity in Markov reward and decision chains: Theory and applications. *Foundation and Trends in Stochastic Systems* 1(1) (2006)
8. Kranenburg, A., van Houtum, G.: Cost optimization in the (S-1, S) lost sales inventory model with multiple demand classes. *Oper. Res. Lett.* 35(4), 493–502 (2007)
9. Lippman, S.: Applying a new device in the optimization of exponential queueing systems. *Operations Research* 23, 687–710 (1975)
10. Mazzucco, M., Mitrani, I., Palmer, J., Fisher, M., McKee, P.: Web service hosting and revenue maximization. In: *Fifth European Conference on Web Services, ECOWS 2007*, pp. 45–54 (2007)
11. Porteus, E.L.: *Foundations of Stochastic Inventory Theory*. Stanford University Press, Stanford (2002)
12. Puterman, M.: *Markov Decision Processes Discrete Stochastic Dynamic Programming*. Wiley, Chichester (2005)
13. Svoronos, A., Zipkin, P.: Evaluation of one-for-one replenishment policies for multiechelon inventory systems. *Management Science* 37(1) (1991)
14. de Véricourt, F., Karaesmen, F., Dallery, Y.: Optimal stock allocation for a capacitated supply system. *Management Science* 48, 1486–1501 (2002)
15. Wiecek, A., Bušić, A., Hyon, E.: Critical level policies in lost sales inventory systems with different demand classes. Tech. rep., INRIA (2011)

Model-Based Evaluation and Improvement of PTP Syntonisation Accuracy in Packet-Switched Backhaul Networks for Mobile Applications

Katinka Wolter¹, Philipp Reinecke¹, and Alfons Mittermaier²

¹ Freie Universität Berlin

Institut für Informatik

Takustraße 9

14195 Berlin, Germany

{philipp.reinecke,katinka.wolter}@fu-berlin.de

² highstreet technologies GmbH

Berlin, Germany

alfons.mittermaier@highstreet-technologies.com

Abstract. Base stations in mobile networks have very strict frequency synchronisation (also referred to as syntonisation) requirements. As backhaul networks are migrated to asynchronous packet-switching technology, timing over packet using the IEEE 1588 Precision Time Protocol (PTP) replaces current synchronisation methods that rely on the synchronous bit clock of the network. With PTP, base-station clocks derive their frequency from the inter-packet delays of Sync messages sent by a high-quality time source at regular time intervals. Packet-delay variation (PDV) thus has a major impact on the achievable synchronisation quality, and the amount of PDV of a backhaul network determines whether the network can support frequency synchronisation of base stations. We present a simulation and an analytical approach to assessing the suitability of backhaul networks for clock synchronisation using PTP and derive two methods for reducing PDV.

Keywords: Precision Time Protocol, Frequency Synchronisation, Mobile Backhaul Networks.

1 Introduction

In order to enable handover between neighbouring base stations in mobile backhaul networks for GSM, WCDMA and LTE, the frequency on the air interface of the base stations must not deviate by more than 50 ppb¹ from the nominal frequency [1]. In practice, this requires that the frequency of the local oscillator of base stations is accurate to about 15 ppb. As local oscillators cannot guarantee this accuracy in free-running operation, frequency synchronisation (also referred

¹ Parts per billion, defined as 10^{-9} .

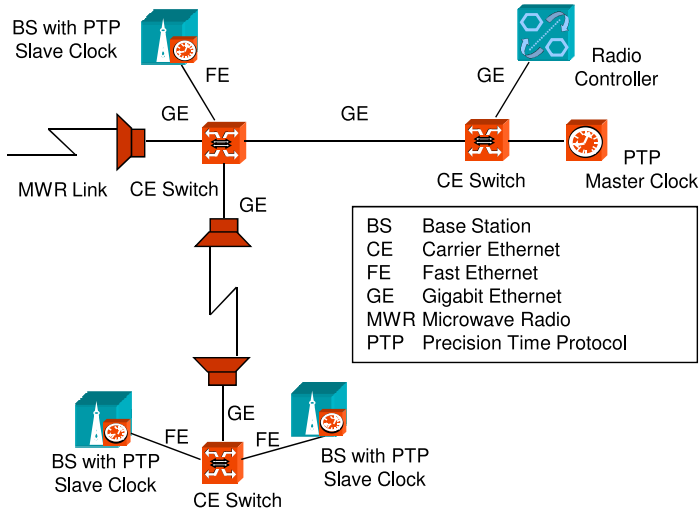


Fig. 1. Network using synchronisation of base stations with ToP using PTP. The master clock (in the upper right corner) sends PTP Sync messages to the slaves. Note that in this application domain, network traffic is usually shown as flowing from right to left.

to as syntonisation) of base-station clocks (called slave clocks) to a high-quality primary reference clock (master clock) is mandatory.²

Whereas current synchronous TDM-based networks can convey the frequency of the master clock to the slaves via the bit clock of the network itself, this becomes impossible with the migration of backhaul networks to asynchronous packet-switching networks (e.g. Carrier Ethernet, CE). In such networks, timing-over-packet (ToP) using the precision time protocol (PTP) according to IEEE standard 1588-2008 [2] is the preferred syntonisation method.

The accuracy attainable by PTP is limited by variation in packet transmission delays, which in turn depends on a variety of factors, including the networking components, network topology and background traffic. Whether a network design will be able to sustain syntonisation using PTP is an important question in engineering backhaul networks. Still, the dominant approach in evaluating backhaul networks with respect to PTP consists in black-box measurements on real hardware (e.g. [3,4,5,6,7,8]). While guidelines in ITU-T G.8261 (04/2008) [9] provide a structure to these measurements, time and cost constraints severely limit the coverage of such an approach. Simulation and analytical methods may help to overcome this restriction. However, as of yet such approaches have not been widely applied to the problem.

In this paper we propose both simulation and analysis as approaches to evaluating PTP syntonisation accuracy in packet-switched networks, illustrating their usefulness by deriving methods for improving accuracy.

² In contrast, time and phase synchronisation are less important.

The remainder of the paper is structured as follows: In the next section we provide a detailed discussion of PTP and introduce Packet Delay Variation (PDV) as a metric for the suitability of a network for PTP. In Section 3 we describe our approach to develop highly-detailed and efficient simulation models for PTP evaluation. Based on these models, we identify a major cause for high PDV and propose a countermeasure. Section 4 presents our analytical approach for typical backhaul network topologies. Our method allows us to derive a closed-form solution for reducing PDV. We conclude the paper with an outlook on further work in Section 6.

2 Technical Background

The operation of PTP can be illustrated using the scenario shown in Figure 1: A PTP master clock syntonises PTP slave clocks in a single-ended way. The network has one PTP master clock on the right-hand side and several PTP slave clocks. At constant time intervals, the PTP master clock sends Sync messages to the slaves. The slaves use the interarrival time of consecutive Sync messages to synchronise their local clock frequency to that of the master clock. Network connections in this scenario can be either optical fiber Gigabit Ethernet (GE) links connecting carrier-ethernet (CE) switches, or microwave radio (MWR) links with radio antennas on either end. The last link to the mobile base station (BS) usually is a fast ethernet (FE) connection. Some PTP slaves are only a few links away from the master clock, but others can be at a distance of up to 20 links.

The Packet Transfer Delay (PTD) of the i th Sync message is defined as the difference between the time the message was received and the time the message was sent:

$$PTD_i = t_i^{received} - t_i^{sent}.$$

In the following, let PTD denote the packet transfer delay distribution, defined on $\mathbf{R}^+ \cup \{0\}$. Ignoring packet loss, the PTD of a network is bounded to an interval $[PTD_{min}, PTD_{max}]$ of the best and worst case transmission delay. In practice, PTD_{min} is the time required by a Sync message when there is no background traffic. PTD_{min} thus depends solely on the networking hardware, and $PTD_{min} > 0$. If there is background traffic, Sync messages may be delayed by processing of background packets. Then, $PTD_{max} > PTD_{min}$.

Syntonisation accuracy depends only on the variation in PTD samples, but not on the constant offset PTD_{min} . The variation is described by the Packet Delay Variation (PDV), defined as the shifted PTD distribution (section 6.2.4.1 in [10])³:

$$PDV := PTD - PTD_{min}.$$

In the following we consider two ways of characterising PDV . A simple measure is given by the peak-to-peak PDV (p2pPDV),

³ Note that there exist different terminologies and definitions for the variation in transmission delays (sometimes also referred to as jitter), e.g. the instantaneous PDV, as defined by [11].

$$p2pPDV := PTD_{max} - PTD_{min} = PDV_{max}.$$

In practice, slave clocks typically filter out samples from slow PTP packets, as slow packets are likely to increase PDV. Slow packets are easily identified by computing the difference between timestamps set by the sender and receiver. The slave clock then uses only the fastest packets, with typical thresholds set at the 1% quantile of the PDV distribution or below. The quantiles of the PDV distribution thus provide a practical measure for PDV. Note that this measure is equal to the $p2pPDV$ of the PDV distribution truncated at the respective quantile.

The upper PDV bound tolerable by a PTP slave clock depends on the PTP implementation and the accuracy of the local oscillators. These properties vary between vendors and are often considered proprietary information. However, a 1% quantile PDV between master and slave of $216\ \mu s$ can be considered a reasonable target. Although we cannot embark on a detailed discussion of the implementation of a slave clock, we note that slave clocks average PTD values over a certain time period and that the upper bound of $216\ \mu s$ corresponds to a maximum deviation of 15 ppb with integration window size 4 hours.

3 Accurate and Efficient Simulation of Backhaul Networks

The need for simulation models is dictated by the limited coverage of measurement studies on practical networking equipment. Even though there exist guidelines for conducting measurement studies [9], cost and time constraints render exhaustive tests infeasible. Due to this restriction, network operators cannot evaluate and compare different backhaul network design choices with respect to their suitability for PTP. Discrete-event simulation allows evaluation in a much more efficient way than black-box testing. However, the high accuracy requirements of PTP demand very precise models, because the delay variation experienced by the fastest packets when passing through a node must be quantifiable with microsecond precision under a wide range of load and other conditions. As these requirements are far beyond those of typical applications, and product documentation does not provide the necessary detail, sufficiently accurate models for networking equipment are not available in state-of-the-art network simulators. In particular, current models do not include many of the internal structures that have an effect on the PDV.

In this section we present our simulation approach to PDV evaluation. We start with an overview of the characteristics of packet delay variation, as known from measurement studies.

3.1 PDV Characteristics

The characteristics of the packet delay variation encountered in a network depend on the networking hardware, the path length, and the background traffic load. As evidenced by measurement studies [8,3,4,5,12,6,7,13], these parameters

may result in a wide variety of phenomena. In order to be useful, our simulation models must be able to reproduce these effects. In the following we focus on the PDV of a single switch, as this will give us the basic building block of models for networks with longer paths between master and slave clock. Furthermore, we assume that the switch is configured such that PTP packets have highest priority, because this configuration is common practice when engineering backhaul networks for syntonisation using PTP.

With these prerequisites we find two patterns in PDV measurements. Abstracting from device-specific constant offsets or scaling factors, these patterns can be identified in most measurement studies. We base our discussion on the data shown in Figure 22 of [8]. The upper part of the figure shows PTD samples obtained with a step-wise change in background load every two hours, with the following load levels: 10%, 40%, 70%, 100%, 70%, 40%, 10%, 100%, 10%, 70%, 0%, corresponding to an average load of about 47% (Figure 19 in [8]).

The data of [8] clearly demonstrates that a background load above 0% results in increased p2pPDV, even though PTP packets have highest priority. This effect is caused by the non-preemptive operation mode of switches: If a PTP packet arrives while the switch is transmitting a background packet, processing of the PTP packet is delayed until the background packet has been sent. The delay experienced by the PTP packet is bounded from above by the maximum time required for transmitting a background packet.⁴ With a constant data rate, this time depends only on the size of the largest possible background packet. The PDV is then a mixture of the point mass at zero (no background packet was being processed) and the distribution of background packet transmission times. In practice, these interactions result in a characteristic, load-dependent shape for the density of the PDV distribution. The effect of load is visible in Figure 12 of [3], and in Figure 8 of [13] where the density of the PTD distribution is shown: For low load levels, the density of the PDV distribution is close to the point mass at zero (recall that the PDV distribution is the shifted PTD distribution, i.e. any constant offset in the PTD distribution can be ignored). The higher the load, the more likely it becomes that PTP packets must wait for a background packet, and therefore the distribution of background packet transmission times becomes more dominant. Note that the shape of the PDV density at 50% in [3] is very similar to those in Figure 5 of [5] (50% load) and Figure 22 of [8] (about 47% load). There is a peak at low values, and a drawn-out block reminiscent of the density of the uniform distribution. The first requirement on simulation models is thus to be able to reproduce this characteristic, load-dependent shape of the PDV density.

Considering again the PTD data shown in Figure 22 of [8], we observe that at a background load of 100% both PTD_{min} and PTD_{max} increase drastically, giving the plot the appearance of being ‘shifted up’. The same ‘delay step’ at 100% load can be observed in Figures 5 and 7 of [12], Figures 11, 14, and 15 of [6], and in [7]. This delay step is due to the internal layout of typical switches. As discussed in the next section, packets leaving the switch must pass a transmit

⁴ Cf. [14], where this is referred to as the ‘jumbo packet phenomenon’.

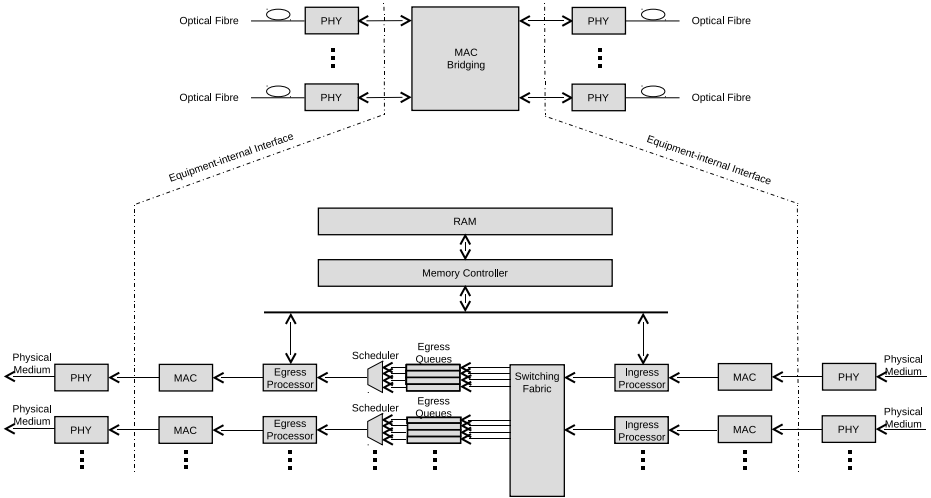


Fig. 2. Functional model of a single-stage Carrier Ethernet switch

FIFO buffer before being transmitted. This buffer, however, does not support priorities. If the transmit FIFO buffer runs into overload, it throttles the egress scheduler in order to reduce the load. The additional delay affects all packets, including the high-priority PTP packets. Our second requirement on simulation models is thus that they are able to represent the delay step at overload.

Note that the increased delay in overload situations does not constitute a problem in itself, since a constant delay offset can be ignored by the slave clocks. However, overload situations occur only intermittently. Then, the resulting delay steps increase overall PDV, and thus affect PTP syntonisation accuracy.

3.2 Simulation Models for PDV Evaluation

A structural analysis is a prerequisite for building an accurate simulation models of a switch. As we will illustrate in the validation step, this requires detailed insight into the internal structure of the switch. The level of detail required is usually not reflected in manufacturer’s data sheets.

Consider the functional structure of a typical single-stage Carrier Ethernet switch with a MAC bridging device as its central component, as shown in Figure 2. Virtually all MAC bridging devices implement a transmit FIFO buffer between the egress scheduler and the transmit interface. Since this transmit FIFO buffer is behind the priority queueing and scheduling block (which applies strict priority queueing), it completely ignores any packet priority. It can be modelled as a rate-matching buffer with the low and the high thresholds being vendor-discretionary and sometimes configurable. The PDV attributable to equipment-internal packet processing and forwarding including packet storage and physical interfaces can be modelled by an arbitrary delay element. MAC bridging device manufacturers usually guarantee the PDV caused by device-internal packet

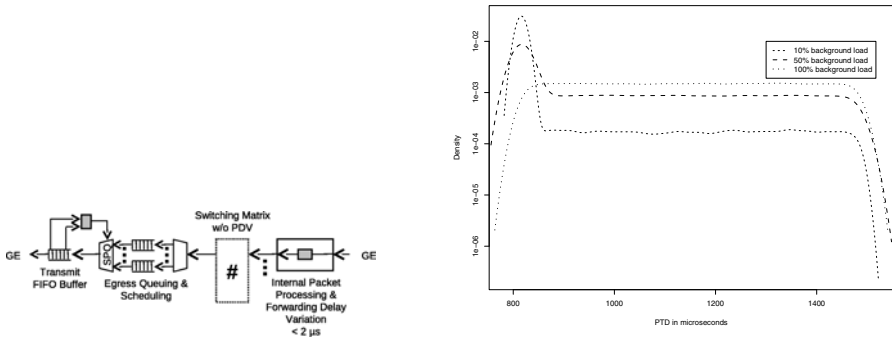


Fig. 3. Delay model of a CE switch (left) and PTD densities for increasing background load (simulation of one switch, right).

processing and forwarding to be less than $1 \mu s$. With the p2pPDV of a Gigabit Ethernet interface being less than $0.2 \mu s$ [15], and leaving some margin, the PDV of the arbitrary delay element may be assumed to be less than $2 \mu s$. Based on this analysis, we model the switch as shown in Figure 3.

Note that neither the transmit FIFO buffer nor the arbitrary delay element are documented in typical manufacturer data sheets. A simulation model of the switch that only took into account the documented features of the switch would fail to represent their effects, as we will illustrate in the next section.

In order to evaluate the PDV of a backhaul network, we connect several copies of the detailed switch model to form a chain between the master and slave clocks. We then generate background traffic on the intermediate switches and transmit PTP packets through this chain, measuring the delay. We use the state-of-the-art discrete-event simulator ns-2 [16] for the simulation. Due to the high data rates in state-of-the-art backhaul networks, discrete-event simulation of high loads and long network paths entails very large numbers of events, which in turn increase simulation times. This makes it difficult to evaluate backhaul networks where the PTP packets must traverse long paths.

If background packets can enter and leave the path at any intermediate switch, the problem of long simulation times can be addressed by a hybrid approach to simulation. First, we measure PDV on a highly-detailed model of a single switch. We then approximate the PDV distribution by fitting a phase-type (PH) distribution (cf. eg. [17,18]) to the data. Approximation with PH distributions has the advantage that PH distributions provide both good fitting quality and efficient methods for random-variate generation [19,20]. By replacing the detailed switch model with a simple delay element that delays packets according to samples drawn from the approximating PH distribution we can greatly decrease the number of events, as we do not have to simulate individual background packets.

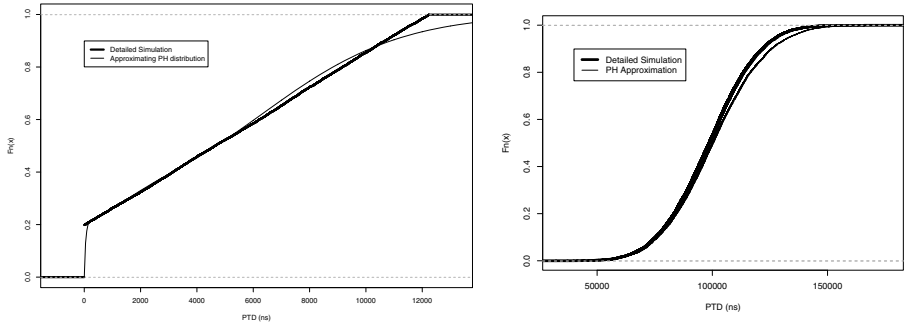


Fig. 4. Cumulative Density Function (CDF) of PTD at 80% background load with detailed simulation and PH approximation. Results for 1 switch are shown on the left, results for 20 switches are shown on the right.

3.3 Validation

As discussed in Section 3.1, the PDV characteristics of networking equipment vary depending on a number of factors. Therefore, validation of simulation models against a specific set of measurements would only show that the models are capable of reproducing this exact set of data. On the other hand, validating the models against the two patterns we identified in Section 3.1 will serve to show that these models can be used to simulate the typical behaviour of networking equipment, without being restricted to specific hardware.

The first step in the validation is thus to show that our models are capable of reproducing the load-dependent shape of the PDV probability density function, which is a mixture of the point mass at zero and the background packet transmission time distribution. We simulated one switch with increasing levels of background load (packet size 64 bytes) and a 1 GB connection and measured the packet transfer delay of PTP packets. Figure 3 shows the resulting PTD densities⁵ for this scenario. Observe that, as background load increases, the spread of the distribution increases as well, thus closely mirroring the measurements reported in [3,13]. Furthermore, note that the density for 50% background load shows the same shape as those in Figure 5 of [5] and Figure 22 of [8].

The second characteristic we identified in Section 3.1 is the delay step occurring in overload situations. In Figure 6 we show the 1% quantile of the PDV for growing background load obtained with a standard model, as constructed from the data sheet, and our detailed model. The standard model does not include the transmit FIFO buffer. Note that the 1% quantile of the PDV stays close to zero for all load levels, i.e. the standard model fails to reproduce the delay step at 100% observed in e.g. [8]. In contrast, our detailed model shows this behaviour very clearly: As the load reaches 100%, the 1% quantile of the PDV increases by about 10000 μs .

⁵ Recall that the PDV distribution is defined as the shifted PTD distribution.

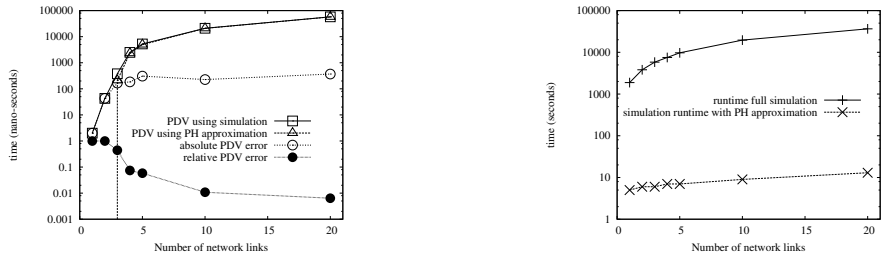


Fig. 5. Error in the 1% quantile of the PTD distribution using PH approximation (left). Simulation times with detailed simulation and PH approximation (right).

We thus conclude that our detailed simulation models capture the behaviour of typical networking hardware well. In order to illustrate the validity of the PH approximations we simulated scenarios with 1 link and with 20 links, using both the PH approximation and the detailed models. The Cumulative Density Functions (CDFs) of the resulting PDV distributions are shown in Figures 4 and 4. Note that PDV CDFs from the simulations using the approximations fit the lower quantiles of the PDV CDFs from the detailed simulations well, but tend to diverge on the higher quantiles. This is due to the fact that PH distributions have infinite support, in contrast to the limited support of the PDV distribution. However, as PTP slave clocks only use data from the lower quantiles of the PDV distribution, this inaccuracy is negligible in PTP simulations. We investigate the error of the 1% quantile of the PDV in more detail in Figure 5. Note that both the relative and the absolute error stay well below $1 \mu\text{s}$, even for a large number of links.

The advantage of using PH approximations is illustrated in Figure 5, where we show the simulation times for increasing numbers of links. Observe that the time required for the detailed simulation rises quickly, while the time for the approximated simulations stays in the order of minutes.

3.4 Delay-Step Elimination

Our simulation model provides sufficient insight to allow us to propose a solution to the problem of excessive PDV caused by the delay step in overload situations. Recall that the delay step is caused by the transmit FIFO buffer throttling the scheduler upon overload. By implementing a leaky-bucket egress shaper [21] before the transmit FIFO buffer, we can reduce the input data rate of the buffer such that no overload occurs in the FIFO, and no throttling becomes necessary. The third graph in Figure 6 confirms the effectiveness of this approach, as the delay step vanishes when the egress shaper is active. Note that typical switches already contain a leaky-bucket egress shaper, however, this function is usually turned off, as the egress shaper reduces the available data rate. In backhaul networks, however, the resulting improvement in syntonisation accuracy outweighs the small reduction in the data rate.

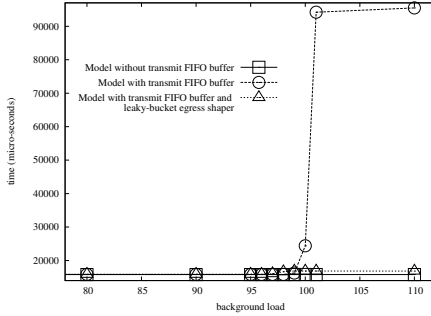


Fig. 6. PDV 1% over a GE-Link between two Single-stage CE Switches

4 Analytical Evaluation of PDV in Tree-Structured Networks

In this section we propose a closed-form solution for the p2pPDV in a tree-structured network where PTP packets have highest priority. Figure 7 shows the packet flows in a tree-structured backhaul network (Figure 1): All downlink traffic is generated at the PTP master and radio controller. The background traffic may leave the route of the PTP packet at any one of the switches on the way to the PTP slave, but no background traffic enters the network at an intermediate switch. This implies that no new background packets will appear just in front of an arbitrary PTP packet. While this scenario might seem limited, it corresponds to the typical case of dedicated backhaul networks, which usually do not carry traffic other than that between the radio controller and the base stations.

In this scenario, the PTP Sync messages and the background traffic flow are aligned on the first link, and, consequently, the arrival times of background and PTP Sync packets are no longer statistically independent on the following links, where the packets arrive one after the other. If a PTP Sync message has to wait in the first node until the background packet is completely transmitted over the first link (remainder-of-packet delay), it will certainly have to wait in the second node again unless the data rate of the second link is much higher than that of the first link. The reason for this is that virtually all packet switching equipment operates in the store-and-forward mode, i.e. a packet has to be completely received on the ingress interface before its transmission on the egress interface may start. And since the PTP Sync message is comparatively small with a correspondingly low transmission delay, it typically catches up the larger background packet in the next node. This effect has also been observed in [22].

The probability that an arbitrary PTP packet encounters remainder-of-packet delay on the first link depends on the background traffic data rates $R_n^{bt}, n = 0, \dots, N$ and is equal to the background traffic load on the first link:

$$P_r = \frac{\sum_{n=1}^N R_n^{bt}}{R_{l1}},$$

where R_{l1} is the data rate of the first link.

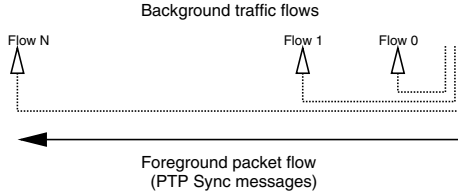


Fig. 7. Traffic flows in a tree-structured backhaul network

4.1 Closed-Form Expression for p2pPDV

We now derive a closed-form expression for the peak-to-peak PDV (p2pPDV) in a tree-structured network. Recall that p2pPDV is defined as the difference between the packet transmission delay of the slowest packet, PTD_{max} , and the packet transmission delay of the fastest packet, PTD_{min} (Section 2). We thus need to compute PTD_{max} and PTD_{min} . Since we are interested in worst-case behaviour, we assume a background traffic stream where all packets have the maximum packet size supported on the links.

If a PTP packet has to wait on the first link, the remainder-of-packet delay t_{r1} is uniformly distributed between zero and the transmission time of a complete background packet with size S_{bp} ,

$$0 \leq t_{r1} \leq \frac{S_{bp}}{R_{l1}},$$

where R_{l1} is the data rate of the first link.

In case the PTP packet was delayed by a background packet on the first link, it has to wait for the same packet to be transmitted on the second link. As the PTP packet only enters the second switch after being processed in the first switch, the second switch will already have started processing the background packet. The remainder-of-packet delay t_{r2} on the second link is then:

$$t_{r2} = \left(\frac{S_{bp}}{R_{l2}}\right) - \left(\frac{S_{PTP\ Sync}}{R_{l1}}\right) \tag{1}$$

with $S_{PTP\ Sync}$ being the size of the PTP Sync message. This kind of remainder-of-packet delay further accumulates over all links down to the PTP slave clock in the BS.

If we assume that all links have equal data rate we can simplify the notation. In order to simplify the notation we now assume that all links have equal data rate. Note that the derivation can be performed without this restriction, but

requires heavier notation and is omitted here for brevity. Let $u := S_{bp}/R_l$ denote the transmission time of the background packet and $p := S_{PTP\ Sync}/R_l$ be the transmission time of the PTP packet. In the worst case, the PTP Sync packet has to wait for transmission of a full background packet at the first link, then adding its own transmission time. At all subsequent links the PTP Sync message needs to wait for the remaining transmission time of the (usually larger) background packet. Finally, the transmission time of the PTP Sync message across all links needs to be added. In the best case, the PTP Sync message never needs to wait for remaining transmission delay of a background packet. Consequently, the maximum packet transfer delay PTD_{max} and the minimum packet transfer delay PTD_{min} of a PTP Sync message across N links are

$$\begin{aligned} PTD_{max} &= u + p + (N - 1) \max(0, (u - p)) + \\ &\quad (N - 1)p = p + Nu \\ PTD_{min} &= Np \end{aligned} \tag{2}$$

We can distinguish two cases. Either the background packet is larger than or of equal size as the PTP Sync message (case 1)) or vice versa (case 2)):

- 1) $u \geq p$,
- 2) $u < p$.

Only the first case is of practical interest and therefore we can omit the maximum in the definition of PTD_{max} in what follows. Peak-to-peak packet delay variation $p2pPDV$ is defined as the difference between PTD_{max} and PTD_{min} . For convenience and without limiting generality let us assume that the size of the background packet is an integer multiple of the size of the PTP Sync message $u = ip$, where $i \geq 1$ in order to respect case 1) and i can be any real-valued number. We thus obtain the following closed-form expression for the p2pPDV of a PTP packet over N links:

$$p2pPDV = PTD_{max} - PTD_{min} = p(1 + (i - 1)N). \tag{3}$$

4.2 Reducing PDV in Tree-Structured Networks

Packet delay variation is reduced if the PTP Sync messages can be prevented from catching up the background packets, i.e. t_{r2} has to be zero. Using (1) for the calculation of t_{r2} one obtains (for the case of two links):

$$t_{r2} = \frac{S_{bp}}{R_{l2}} - \frac{S_{PTP\ Sync}}{R_{l1}} = 0 \tag{4}$$

$$\frac{S_{PTP\ Sync}}{R_{l1}} = \frac{S_{bp}}{R_{l2}} \tag{5}$$

$$S_{PTP\ Sync} = S_{bp} \frac{R_{l1}}{R_{l2}}. \tag{6}$$

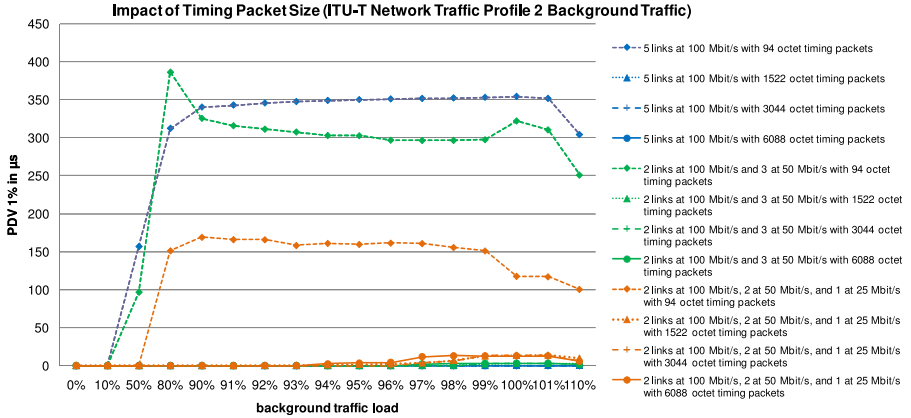


Fig. 8. Impact of Timing Packet Size on the 1% quantile of the packet-delay distribution. The plots for larger packet sizes overlap close to zero.

This means that the optimal size of the PTP Sync messages depends on the size of the background packets and the data rate of the ingoing and outgoing link to a switch. In most cases, the PTP Sync messages have to be enlarged. The optimal size depends on the PDV the ToP implementation can accept, the background traffic load and packet size distribution, and the number of links and their data rates.

Using Equation 3 we can generalise this result to N links when all links have the same data rate: The value of i that minimises the delay variation and obeys all restrictions is $i = 1$. For the optimal value of i the PTP Sync packet has the same size as the (largest) background packet and then the peak-to-peak packet delay variation reduces to

$$p2pPDV_{opt} = p.$$

For the second case ($u < p$) the delay variation is always p . The reasoning for links with different data rate is analogous.

4.3 Evaluation

We evaluate our approach using the simulation models proposed in Section 3. We simulate a chain of 5 links at different link speeds and observe PDV at different background load levels and different PTP packet sizes. We use the ITU-2 model [9], which describes a mix of packets of different sizes. This model represents realistic traffic conditions in a network. We increase the traffic load from 0% to 110%. Increasing the size of PTP packets from 94 bytes to 6088 bytes, we observe the 1% quantile of the PDV distribution for each combination of the parameters.

The results are presented in Figure 8. It is immediately obvious that larger PTP Sync messages reduce the 1% PDV quantile considerably. Using large PTP

Sync packets, the packet delay variation across 5 links stays below $200\ \mu\text{s}$ and hence the PDV requirements for ToP are fulfilled. The same cannot be said for the default packet size of 94 bytes, where the 1% quantile of PDV is much larger.

5 Discussion

Several authors have recognised challenges with PTP syntonisation over packet-switched networks and proposed solutions. [23,24] showed that IEEE 1588 can be expected to work well in small networks under low load. These assumptions are very restrictive and often unrealistic. The importance of queueing effects has been recognised in several works, e.g. [25]. The authors of [25] propose to predict the queueing delay by sending probe messages and evaluating their timing behaviour. The performance of this method significantly decreases with the network load. Other authors suggest to block the background traffic regularly in order to guarantee undelayed delivery of timing packets [22]. Both approaches do not seem to be likely candidates for deployment. The authors in [14] state that increasing the transmission rate of timing packets reduces the PDV. They propose the doubtful management rule of increasing the transmission rate of timing packets under high background load, while reducing the number of timing packets with low background traffic load.

In contrast, the methods we suggest are both simple and non-intrusive and can work for large networks comprising paths of up to 20 links. In particular, increasing PTP packets by padding performs especially well as the load increases, which is the interesting and critical case.

6 Conclusion

In this paper we proposed two approaches to evaluating the suitability of backhaul networks for precise syntonisation of base-station clocks using IEEE 1588 PTP. We showed that it is possible to capture PDV characteristics of networking equipment in simulation models. This enables the efficient evaluation of the suitability of arbitrary backhaul networks with respect to PTP using state-of-the-art discrete-event simulation, thus alleviating the need for expensive measurement studies. On the other hand, we also found that highly-detailed simulation models are required in order to correctly reflect the behaviour of common hardware. The level of detail required is typically beyond that provided in manufacturers' data sheets. We addressed the problem of excessive simulation times by a hybrid approach employing phase-type distributions for approximating packet-delay distributions, and demonstrated that the hybrid models achieved tremendous efficiency improvements, while still providing sufficient accuracy. Our second approach provides a closed-form expression for the peak-to-peak PDV in tree-structured networks, where PTP packets may be subjected to successive remainder-of-packet delays.

Employing the simulation and analytical approaches we derived and evaluated two solutions for reducing PDV in backhaul networks. While the leaky-bucket

egress shaper can be applied in networks of any topology, increasing the size of PTP packets is only suitable in tree-structured networks. On the other hand, implementation of the egress shaper requires changes to the switches themselves and reduces available data rate, while increasing the size of PTP packets implies only minimal changes to the master and slave clocks.

In future work we will address simulation of network technologies like packet over TDM and technologies like DSL. Furthermore, we are studying the effectiveness of enlarging PTP packets when the backhaul network carries a limited amount of partial cross traffic.

References

1. 3GPP, TS 25.104, V9.5.0 (2010-09) Base station (BS) radio transmission and reception (FDD), <http://www.3gpp.org/ftp/specs/html-info/25104.htm>
2. IEEE, Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4579760
3. Cosart, L.: Studying network timing with precision packet delay measurements. In: Proc. 40th Annual Precise and Time Interval (PTTI) Meeting (2008)
4. Cosart, L.: Packet network timing measurement and analysis using an iee 1588 probe and new metrics. In: International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS 2009, pp. 1–6 (October 2009)
5. Holmeide, O., Skeie, T.: Time Synchronization in Switched Ethernet. OnTime Networks, Tech. Rep., Unknown publication date, http://www.westermo.com/dman/Document.phx/Manuals/Manuals+for+UK+only/OnTime/Articles/Time_Sync.pdf?folderId=%2FManuals%2FManuals+for+UK+only%2FOnTime%2FArticles&cmd=download (last seen February 14, 2011)
6. Jacobs, A., Wernicke, J., Gordon, B., George, A.: Characterization of quality of service in commercial ethernet switches for statistically bounded latency in aircraft networks. High Performance Computing and Simulation (HCS) Research Lab, Department of Electrical and Computer Engineering, University of Florida, Tech. Rep. (2004), http://www.hcs.ufl.edu/~jacobs/rockwell_qos.doc (last seen February 14, 2011)
7. Zarick, R., Hagen, M., Bartos, R.: The impact of network latency on the synchronization of real-world iee 1588-2008 devices. In: 2010 International IEEE Symposium on Precision Clock Synchronization for Measurement Control and Communication, ISPCS 2010, pp. 135–140 (October 2010)
8. Jobert, S.: About the control of pdv using qos mechanisms and the applicability of proposed pdv metrics (mafe and mintdev), France Telecom, Tech. Rep. (2009)
9. ITU, G.8261/Y.1361 (04/2008) Timing and synchronization aspects in packet networks (April 2008), <http://www.itu.int/rec/T-REC-G.8261-200804-I>
10. ITU, Recommendation Y.1540 – IP packet transfer and availability performance parameters (November 2002), <http://www.itu.int/itudoc/itu-t/aap/sg13aap/history/y1540/y1540.html>
11. Demichelis, C., Chimento, P.: IP Packet Delay Variation Metric for IP Performance Metrics (IPPM). RFC 3393 (Proposed Standard), Internet Engineering Task Force (November 2002), <http://www.ietf.org/rfc/rfc3393.txt>

12. Dobinson, R.W., Haas, S., Korcyl, K., Levine, M.J., Lokier, J., Martin, B., Meirosu, C., Saka, F., Vella, K.: Testing and modeling ethernet switches and networks for use in atlas high-level triggers. *IEEE Trans. Nucl. Sci.* 48, 607–612 (2001)
13. Burch, J., Green, K., Nakulski, J., Vook, D.: Verifying the performance of transparent clocks in ptp systems. In: *International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS 2009*, pp. 1–6 (October 2009)
14. Bui, D.T., Dupas, A., Le Pallec, M.: Packet delay variation management for a better ieee1588v2 performance. In: *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication, Brescia*, pp. 1–6 (October 2009)
15. IEEE, IEEE 802.3: LAN/MAN CSMA/CDE (Ethernet) Access Method, <http://standards.ieee.org/getieee802/802.3.html>
16. Various authors, The Network Simulator ns-2, <http://www.isi.edu/nsnam/ns/> (last seen May 11, 2010)
17. Neuts, M.F.: *Matrix-Geometric Solutions in Stochastic Models. An Algorithmic Approach*. Dover Publications, Inc., New York (1981)
18. Horváth, A., Telek, M.: PhFit: A General Phase-Type Fitting Tool. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) *TOOLS 2002*. LNCS, vol. 2324, pp. 82–91. Springer, Heidelberg (2002)
19. Reinecke, P., Wolter, K., Bodrog, L., Telek, M.: On the Cost of Generating PH-distributed Random Numbers. In: Horváth, G., Joshi, K., Heindl, A. (eds.) *Proceedings of the Ninth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS 2009)*, Eger, Hungary, September 17–18, pp. 16–20 (2009)
20. Reinecke, P., Telek, M., Wolter, K.: Reducing the cost of generating APH-distributed random numbers. In: Müller-Clostermann, B., Echtele, K., Rathgeb, E.P. (eds.) *MMB&DFT 2010*. LNCS, vol. 5987, pp. 274–286. Springer, Heidelberg (2010)
21. Tanenbaum, A.S.: *Computer Networks*, 3rd edn. Prentice Hall, Englewood Cliffs (1996)
22. Mochizuki, B., Hadžić, I.: Improving ieee 1588v2 clock performance through controlled packet departures. *Comm. Letters* 14(5), 459–461 (2010)
23. Vallat, A., Schneuwly, D.: Clock synchronization in telecommunications via ptp (ieee 1588). In: *IEEE International Frequency Control Symposium, 2007 Joint with the 21st European Frequency and Time, Forum*, pp. 334–341 (June 2007)
24. Tu, K.-Y., Liao, C.-S., Lin, S.-Y.: Remote frequency control via ieee 1588. *IEEE Transactions on Instrumentation and Measurement* 58(4), 1263–1267 (2009)
25. Murakami, T., Horiuchi, Y.: Improvement of synchronization accuracy in ieee 1588 using a queuing estimation method. In: *International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, ISPCS 2009*, pp. 1–5 (October 2009)

Light-Weight Parallel I/O Analysis at Scale

Steven A. Wright, Simon D. Hammond,
Simon J. Pennycook, and Stephen A. Jarvis

Performance Computing and Visualisation
Department of Computer Science
University of Warwick, UK
{saw,sdh,sjp,saj}@dcs.warwick.ac.uk

Abstract. Input/output (I/O) operations can represent a significant proportion of the run-time when large scientific applications are run in parallel. Although there have been advances in the form of file-format libraries, file system design and I/O hardware, a growing divergence exists between the performance of parallel file systems and compute processing rates.

In this paper we utilise RIOT, an input/output tracing toolkit being developed at the University of Warwick, to assess the performance of three standard industry I/O benchmarks and mini-applications. We present a case study demonstrating the tracing and analysis capabilities of RIOT at scale, using MPI-IO, Parallel HDF-5 and MPI-IO augmented with the Parallel Log-structured File System (PLFS) middle-ware being developed by the Los Alamos National Laboratory.

Keywords: Input/Output, Message Passing Interface, Parallel I/O, Parallel Log-structured File System, Profiling.

1 Introduction

The substantial growth in supercomputer machine size – over two orders of magnitude in terms of processing element count since 1993 – has created machines of extreme computational power and scale. As a result, users of these machines have been able to create increasingly sophisticated and complex computational simulations, advancing scientific understanding across multiple domains. Historically, industry and academia have focused on the development of scalable parallel algorithms – the cornerstone of large parallel applications. ‘Performance’ has become a measure of the number of calculation operations that can be performed each second.

One of the consequences of this focus has been that some of the vital contributors to application run-time have been developed at a much slower rate. One such area has been that of input/output (I/O) – typically seen as somewhat of a black-box which creates a need to read data at the start of a run and write state information on completion.

As well as reading and writing state information at the beginning and end of computation, the use of checkpointing is becoming common-place – where the system state is periodically written to persistent storage so that, in the case of

an application fault, the computation can be reloaded and resumed. The cost of checkpointing is therefore a slowdown at specific points in the application in order to achieve some level of resilience. As we look to the future, the size of multi-petaflop clusters looks set to bring reliability challenges from just the sheer number of components – checkpointing will become a vital initial tool in addressing these problems. Understanding the cost of checkpointing and what opportunities might exist for optimising this behaviour presents a genuine opportunity to improve the performance of parallel applications at significant scale.

The Message Passing Interface (MPI) has become the *de facto* standard for managing the distribution of data and process synchronisation in parallel applications. The MPI-2 [13] standard introduced MPI-IO, a library of functions designed to standardise the output of data to the file system in parallel. The most widely adopted MPI-IO implementation is ROMIO [21] which is used by both OpenMPI [9] and MPICH2 [10], as well as a number of vendor-based MPI solutions [1,5].

In addition to the standardisation of parallel I/O through MPI, many file format libraries exist to further abstract low-level I/O operations such as data formatting from the application. Through the use of libraries such as HDF-5 [11], NetCDF [17] and Parallel NetCDF [12], information can be shared between multiple applications using the standardised formatting they create. Optimisations can also be made to a single library, creating improvements in the data throughput of many dependant applications. Unfortunately this has, in part at least, created a lack of responsibility on the part of the code designers to investigate the data storage operations used in their applications. The result has been poor I/O performance that does not utilise the full potential of expensive parallel disk systems.

In this paper we utilise the RIOT input/output toolkit (referred to throughout the remainder of this paper by the recursive acronym RIOT) introduced in [26] to demonstrate the I/O behaviours of three standard benchmarks at scale. RIOT is a collection of tools specifically designed to enable the tracing and subsequent analysis of application input/output activity. This tool is able to trace parallel file operations performed by the ROMIO layer and relate these to their underlying POSIX file operations. We note that this style of low-level parameter recording permits analysis of I/O middleware, file format libraries and application behaviour, all of which are assessed in a case study in Section 4.

The specific contributions of this work are the following:

- We extend previous work in [26] to demonstrate RIOT working at scale on a 261.3 TFLOP/s Intel Westmere-based machine located at the Open Computing Facility (OCF) at the Lawrence Livermore National Laboratory (LLNL);
- We apply our tracing tool to assessing the I/O behaviour’s of three standard industry benchmarks – the block-tridiagonal (BT) solver application from NASA’s Parallel Benchmark Suite, the FLASH-IO benchmark from the University of Chicago and the Argonne National Laboratory and IOR, a high-performance computing (HPC) file system benchmark which is used

during procurement and file system assessment [19,20]. The variety of configurations (including the use of MPI-IO, parallel HDF-5 [11] and MPI-IO utilising collective buffering hints) available makes the results obtained from our analysis of interest to a wider audience;

- We utilise RIOT to produce a detailed analysis of HDF-5 write behaviour for the FLASH-IO benchmark, demonstrating the significant overhead incurred by data read-back during parallel writes;
- Finally, through I/O trace analysis, we provide insight into the performance gains reported by the Parallel Log-structured File System (PLFS) [4,16] – a novel I/O middleware being developed by the Los Alamos National Laboratory (LANL) to improve file write times. This paper builds upon [26] to show how file partitioning reduces the time POSIX write calls spend waiting for access to the file system.

The remainder of this paper is structured as follows: Section 2 outlines previous work in the fields of I/O profiling and parallel I/O optimisation; Section 3 describes how parallel I/O is performed with MPI and HDF-5, and how RIOT captures the low-level POSIX operations; Section 4 contains a case study describing the use of RIOT in assessing the parallel input/output behaviours of three industry I/O benchmarking codes; finally, Section 5 concludes the paper and outlines opportunities for future work.

2 Related Work

The assessment of file system performance, either at procurement or during installation and upgrade, has seen the creation of a number of benchmarking utilities which attempt to characterise common read/write behaviour. Notable tools in this area include the BONNIE++ benchmark, developed for benchmarking Linux file systems, as well as the IOBench [25] and IOR [19] parallel benchmarking applications. Whilst these tools provide a good indication of potential maximum performance, they are rarely indicative of the true behaviour of production codes due to the subtle nuances that production grade software contains. For this reason, a number of mini-application benchmarks have been created which extract file read/write behaviour from larger codes to ensure a more accurate representation of performance. Examples include the Block Tridiagonal solver application from NASA’s Parallel Benchmark Suite [2] and the FLASH-IO [18] benchmark from the University of Chicago – both of which are used in this paper.

Whilst benchmarks may provide a measure of file system performance, their use in diagnosing problem areas or identifying optimisation opportunities within large codes is limited. For this activity profiling tools are often required which can record read/write behaviour in parallel. One approach, which aims to ascertain the performance characteristics of production-grade scientific codes, is to intercept communications between the application and the underlying file system. This is the approach taken by RIOT, Darshan [6], developed at the Argonne

National Laboratory, and the Integrated Performance Monitoring (IPM) suite of tools [8], from the Lawrence Berkeley National Laboratory (LBNL).

Darshan has been designed to record file accesses over a prolonged period of time, ensuring each interaction with the file system is captured during the course of a mixed workload. [6] culminates in the intention to monitor I/O activity for a substantial amount of time on a production BlueGene/P machine in order to generate analysis which may help guide developers and administrators in tuning the I/O back-planes used by large machines.

Similarly, IPM [22] uses an interposition layer to catch all calls between the application and the file system. This trace data is then analysed in order to highlight any performance deficiencies that exist in the application or middleware. Based on this analysis, the authors were able to optimise two I/O intensive applications, achieving a four-fold improvement in run-time. In contrast to both Darshan and IPM, RIOT focuses only on the POSIX function calls that are a direct result of using MPI-IO functions. As a result of this restriction, the performance data generated relates only to the parallel I/O operations performed, allowing users to obtain a greater understanding of the behaviour of various I/O intensive codes, as well as assessing the inefficiencies that may exist in any given middleware or MPI implementation.

As a result of previous investigations, a variety of methods have been introduced to improve the performance of existing codes or file systems. The development of middleware layers such as the Parallel Log-structured File System (PLFS) [4] and Zest [14] has, to an extent, bridged the gap between processor and I/O performance. In these systems multiple parallel writes are written sequentially to the file system with a log tracking the current data. Writing sequentially to the file system in this manner offers potentially large gains in write performance, at the possible expense of later read performance [15].

In the case of Zest, data is written sequentially using the fastest path to the file system available. There is, however, no read support in Zest; instead, it serves to be a transition layer caching data that is later copied to a fully featured file system at a later non-critical time. The result of this is high write throughput but no ability to restart the application until the data has been transferred and rebuilt on a read capable system.

In a similar vein to [23] and [24], in which I/O throughput is vastly improved by transparently partitioning a data file (creating multiple, independent, I/O streams), PLFS uses file partitioning as well as a log-structured file system to further improve the potential I/O bandwidth. An in-depth analysis of PLFS is presented in Section 4.5.

3 Warwick RIOT

The enclosed area in Figure 1 represents the standard flow of parallel applications. When conducting a parallel I/O operation using MPI, the application will make a call to the MPI-IO library, which will then provide inter-process communication and issue POSIX file operations as needed. When using an I/O

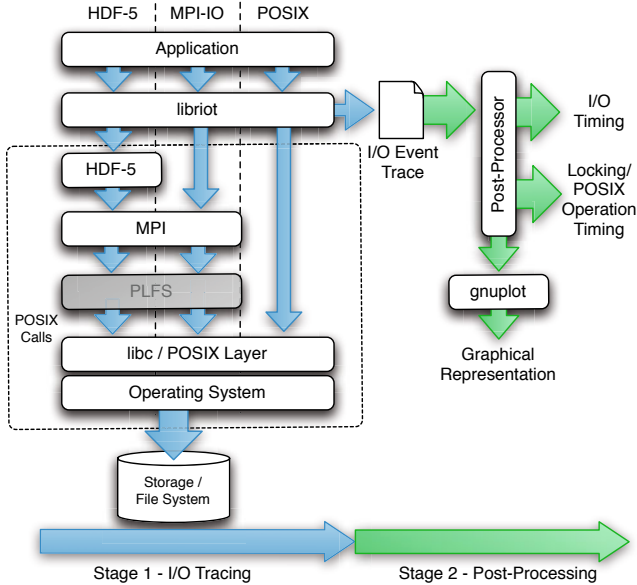


Fig. 1. RIOT tracing and analysis workflow

middleware such as HDF-5, the application first calls the HDF-5 library functions, which will in turn call a collection of MPI functions. PLFS can be configured to be used as a ROMIO file system driver, which will sit between the MPI library and the POSIX library in the application stack.

Tracing of I/O behaviour in RIOT is conducted using a two stage process. In the first stage (shown on the left in Figure 1), the tracing library is dynamically loaded and linked immediately prior to execution by the operating systems linker. `libriot` then overloads and intercepts calls to MPI-IO and POSIX file functions. The captured events are then performed by the library, where each function is timed and logged for later processing. The library makes use of function interposition to trace activity instead of requiring code modification or application recompilation. RIOT is therefore able to operate on existing application binaries and remain compiler or implementation language agnostic.

When the application being traced has completed, RIOT uses an operating system-level termination hook to write traced I/O information to disk – the delay of logging (by storing events in memory as opposed to flushing to disk) helps to prevent any distortion of application I/O behaviour which may result through the output of information whilst the application is being observed.

In the second stage, a post-execution analysis of the I/O trace is conducted (shown on the right in Figure 1). At this point I/O events are processed with aggregated statistics such as file operation count, total bytes written/read, number of locks *etc.* being generated.

Throughout this paper we make a distinction between effective MPI-IO and POSIX bandwidths – in these studies, “MPI-IO bandwidths” refer to the data throughput of the MPI functions on a per MPI-rank basis, “POSIX bandwidths” relate to the data throughput of the POSIX read/write operations as if performed serially, called directly by the MPI middleware.

4 Case Study

We previously reported on the use of RIOT using a maximum of 256 processing elements on the Minerva cluster located at the Centre for Scientific Computing at the University of Warwick [26]. In this paper we utilise the Sierra supercomputer located at LLNL. Sierra is a 261.3 TFLOP/sec machine consisting of 1,849 nodes each comprising of dual hex-core Intel X5660 “Westmere-EP” processors running at 2.8 GHz. The machine offers 22,188 processor-cores each with a minimum of 2 GB of system memory per-core. Three storage paths are provided, each utilising the Lustre parallel file system with between 755 TB and 1.3 PB of storage available. Bandwidth to each of the file systems ranges from 10 GB/sec up to a maximum of 30 GB/sec. All applications and code compiled for this case study were built using the GNU 4.3.4 compiler and OpenMPI 1.4.3. Both IOR and FLASH-IO utilise the parallel HDF-5 version 1.6.9 library.

4.1 Input/Output Benchmarks

We provide analysis for three I/O benchmarks utilising a range of configurations using the Sierra supercomputer. First we demonstrate the bandwidth tracing ability of RIOT, providing commentary on the divergence between MPI-IO and POSIX effective bandwidths. This difference in bandwidth represents the significant overhead incurred when using blocking collective write operations from the MPI library. We then analyse one simple solution to poor write performance provided by the ROMIO file system layer. Collective buffering creates an aggregator on each node, reducing on-node contention for the file system by sending all data through one process. We then monitor two middleware layers to show how they affect the write bandwidth available to the application. The benchmarks used in this study (described below) have been selected since they all have input/output behaviour which is either extracted directly from a larger parallel application (as is the case with FLASH-IO and BT) or have been configured to be representative of the read/write behaviour used by several scientific applications.

The applications used in this study are:

- **IOR** [19,20]: A parameterised benchmark that performs I/O operations through both the HDF-5 and MPI-IO interfaces. In this study it has been configured to write 256 MB per process to a single file in 8 MB blocks. Runs have been performed on a range of configurations, utilising between 1 and 128 compute nodes. Its write performance through both MPI-IO and HDF-5 are assessed.

- **FLASH-IO**: This benchmark replicates the checkpointing routines found in FLASH [7,18], a thermonuclear star modelling code. In this study we use a $24 \times 24 \times 24$ block size per process, causing each process to write approximately 205 MB to disk through the HDF-5 library.
- **BT** [2,3]: An application from the NAS Parallel Benchmark (NPB) suite has also been used in this study, namely the Block-Tridiagonal (BT) solver application. There are a variety of possible problem sizes but for this study we have used the C and D problem classes, writing a data-set of 6 GB and 143 GB respectively. This application requires the use of a square number of processes (*i.e.*, 4, 9, 16), therefore we have executed this benchmark on 1, 4, 16, 64, 256, 1,024 and 4,096 processors. Performance statistics were collected for BT using MPI-IO with and without collective buffering enabled, and also using the PLFS ROMIO layer.

Five runs of each configuration were performed, and the data contained throughout this paper is the mean from each of these runs, in an effort to reduce the influence of other users jobs.

4.2 MPI-IO and POSIX Bandwidth Tracing

First we demonstrate the MPI-IO and POSIX bandwidth traces for the three selected benchmarks. Figure 2 shows the significant gap between MPI bandwidth and POSIX bandwidth for each of the three benchmarks when using MPI-IO and parallel HDF-5. It is important to note that *effective* bandwidth refers to the total amount of data written divided by the total time spent writing as if done serially. Since the MPI-IO functions used are collective blocking operations, we can assume they are executed in parallel, therefore the *perceived* bandwidth is the effective bandwidth multiplied by the number of processor cores. As the POSIX write operations are performed in a non-deterministic manner, we cannot make any assumption about the perceived bandwidth; it suffices to say it is bounded by effective bandwidth multiplied by the processor count and the MPI perceived bandwidth.

Figure 2 also demonstrates a large performance gap between the BT mini-application and the other two benchmarks. This is largely due to the use of collective buffering ROMIO hints utilised by BT. The performance of HDF-5 in both IOR and FLASH-IO is also of interest. The POSIX write performance is much close to the MPI-IO performance in both IOR with HDF-5 and FLASH-IO. Despite this, the overall MPI-IO performance is lower when using HDF-5. The performance gap between HDF-5 and MPI-IO is analysed in Section 4.4. For the remainder of this study we concentrate on both BT and FLASH-IO as these better emulate the write behaviour found in other scientific applications.

4.3 Collective Buffering in ROMIO

As stated previously, BT makes use of the collective buffering ROMIO hint. This causes MPI to assign a set of “aggregator” processes that perform the I/O required. This is shown in Figure 3, where each process writes to the file system

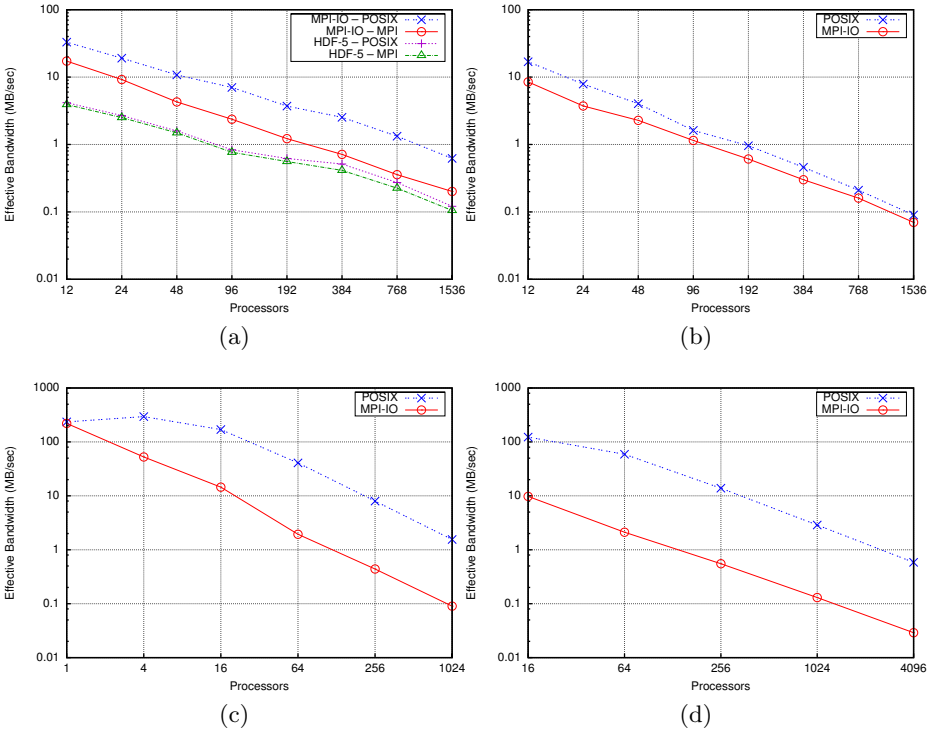


Fig. 2. Effective MPI and POSIX Bandwidths for (a) IOR (with both MPI-IO and HDF5), (b) FLASH-IO, (c) BT class C, and (d) BT class D

in (a) and each process communicates the data to an aggregator in (b). Using an aggregator on each node reduces the on-node contention for the file system and allows the file server to perform node-level file locking, instead of requiring on-node POSIX file locking, as is the case without collective buffering.

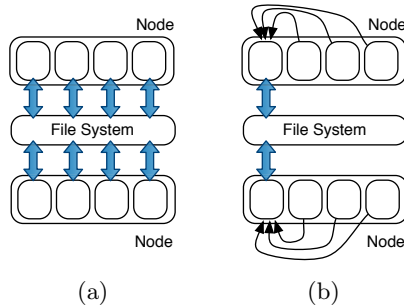
Table 1 demonstrates the bandwidth achieved by BT with and without collective buffering enabled. It is interesting to note the POSIX performance on 4 processor cores with and without collective buffering is very similar. As soon as the run is increased to 16 processes, the POSIX bandwidth is significantly reduced as there is now more on-node competition for the file system. Furthermore, the operating system cannot buffer the writes effectively as the behaviour of the second compute node is unknown.

4.4 Analysis of HDF-5

In [26] we demonstrated the performance of FLASH-IO on the Minerva cluster located at the Centre for Scientific Computing at the University of Warwick. Minerva utilises two GPFS servers, backed by an array of 2 TB hard drives, connected through QLogic 4X QDR Infiniband. Table 2 demonstrates a similar

Table 1. MPI-IO and POSIX write performance (MB/sec) for BT on class C, with and without collective buffering

	Processor Cores			
	4	16	64	256
MPI-IO				
With collective buffering	70.86	23.63	4.45	1.24
Without collective buffering	9.90	3.46	1.34	0.54
POSIX				
With collective buffering	490.08	293.30	109.71	21.94
Without collective buffering	329.69	6.63	4.85	3.49

**Fig. 3.** (a) Each process writes to the file system and (b) each process communicates with a single aggregator process on each node

analysis for FLASH-IO at increased scale on the Sierra supercomputer. Whilst we previously reported a significant locking overhead, this becomes negligible when using the Lustre file system. However, there remains a large read-back overhead when using HDF-5 through MPI-IO. In the worst case, POSIX reads account for nearly half the total MPI write time. As the problem is scaled, POSIX reads still make up 20% of the overall MPI write time.

Whilst HDF-5 allows easy application integration, due to the standardised formatting, it creates a significant overhead for parallel writes. When data is being written periodically for checkpointing or visualisation purposes, this overhead creates a significant slow-down in application performance. This analysis motivates opportunities for HDF-5 optimisations, including reducing or eliminating the read-back behaviour and enabling some form of node-level write aggregation to reduce locking overheads.

4.5 Analysis of PLFS Middleware

Whilst HDF-5 has been shown to decrease parallel write performance, the Parallel Log-structured File System from LANL has been demonstrated to improve performance in a wide variety of circumstances, including for production applications from LANL and the United Kingdom's Atomic Weapons Establishment (AWE) [4]. PLFS is an I/O interposition layer designed primarily for checkpointing and logging operations.

Table 2. Detailed breakdown of MPI-IO and POSIX timings for FLASH-IO writes

	Processor Cores							
	12	24	48	96	192	384	768	1536
MB written	2812.48	5485.68	11037.27	23180.15	43191.78	86729.59	179202.26	365608.30
MPI write calls	636	1235	2436	4836	9634	19233	38432	76832
POSIX write calls	6050	11731	23584	49764	91707	184342	382608	783068
POSIX read calls	5824	11360	22856	48000	89328	179437	370904	757060
Locks requested	12000	24000	48000	96000	192000	384000	768000	1536000
MPI write time (sec)	331.14	1471.30	4832.43	20215.00	70232.87	288633.74	1145889.19	4942746.26
POSIX write time (sec)	166.54	696.58	2737.14	14298.17	44869.00	190337.91	845969.92	3909161.40
POSIX read time (sec)	139.40	503.34	1462.00	5850.12	16782.82	66895.21	235866.21	908076.69
Lock time (sec)	5.95	14.02	28.80	57.80	172.80	385.65	1176.30	4328.96

PLFS works by intercepting MPI-IO calls through a ROMIO file system driver, and translates the operations from n -processes writing to 1 file, to n -processes writing to n -files. The middleware creates a view over the n -files, so that the calling application can view and operate on these files as if they were all concatenated into a single file. The use of multiple files by the PLFS layer helps to significantly improve file write times as multiple, smaller files can be written simultaneously. Furthermore, improved read times have also been demonstrated when using the same number of processes to read back the file as were used in its creation [16].

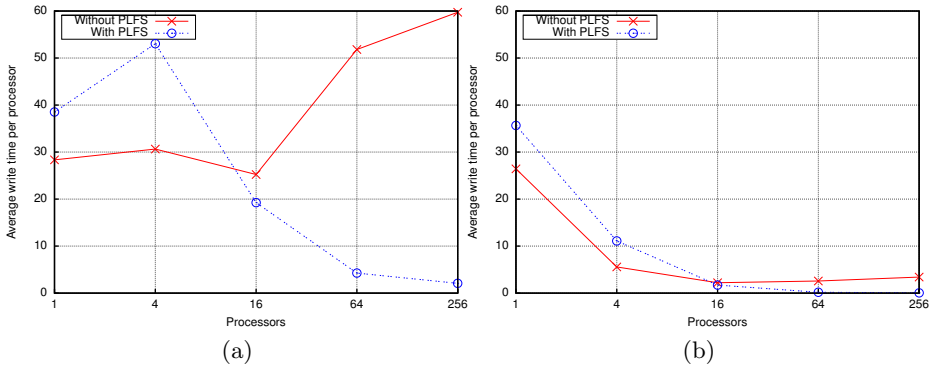
Figures 4(a) and 4(b) present the average total MPI-IO and POSIX write time per process for the BT benchmark when running with and without the PLFS ROMIO file system driver. Note that as previously, POSIX bandwidth in this table refers to the bandwidth of POSIX operations called from MPI-IO and hence are higher due to the additional processing required by MPI. It is interesting to note that the write time when using PLFS is generally larger or comparable with the same run not utilising PLFS when using a single node (as is the case with 1 and 4 processes) due to the on-node contention for multiple files.

The effective MPI-IO and POSIX bandwidths are shown in Table 3. Whilst previously we have seen POSIX write bandwidth decrease at scale, PLFS partially reverses this trend as writes can be flushed to the cache as they are not waiting on surrounding writes. The log structured nature of PLFS also increases the bandwidth as data can be written in a non-deterministic sequential manner, with a log file keeping track of the data ordering. For a BT class C execution on 256 processors, PLFS increases the bandwidth from 115.2 MB/sec perceived bandwidth up to 3,118.08 MB/sec, representing a 27-fold increase in write performance.

Figure 5 demonstrates that during the execution of BT on 256 processors, concurrent POSIX write calls wait much less time for access to the file system. As each process is writing to its own unique file, each process has access to its own unique file stream, reducing file system contention. This results in each POSIX write call completing much more quickly as the data can be flushed to the cache. For non-PLFS writes we see a stepping effect where all POSIX writes are queued and complete in a non-deterministic order. Conversely, PLFS writes do not exhibit this stepping behaviour as the writes are not waiting on other processes to complete.

Table 3. Effective MPI-IO and POSIX bandwidth (MB/sec) for BT class C using MPI-IO and PLFS

	Processor Cores				
	1	4	16	64	256
MPI-IO					
Standard	220.64	52.58	13.89	1.97	0.45
PLFS	164.21	29.81	21.07	23.72	12.18
POSIX					
Standard	235.51	294.80	169.56	40.78	7.98
PLFS	177.34	142.51	235.44	538.13	437.88

**Fig. 4.** (a) MPI-IO write time per processor and (b) POSIX write time per processor

The average time per POSIX write call is shown in Table 4. As the number of processes writing increases, data written in each write decreases. However when using standard MPI-IO, the time to write the data increases due to contention. When using the PLFS ROMIO file system driver, the write time decreases due to the transparent partitioning of the data files.

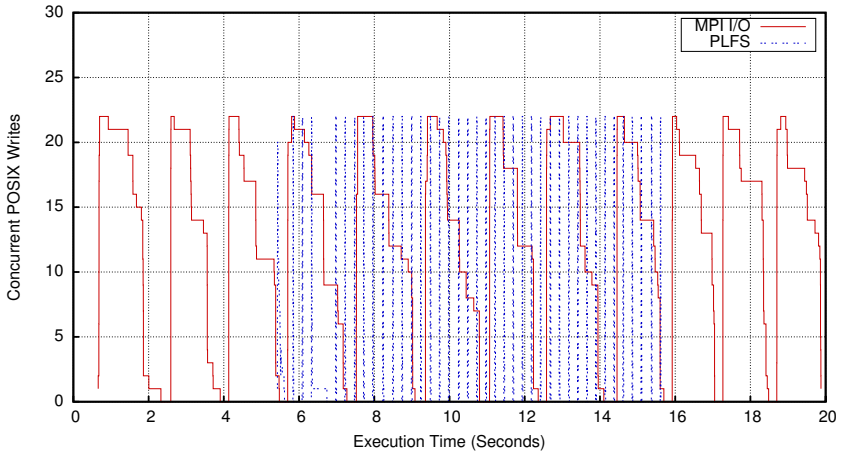
5 Conclusions

Parallel I/O operations continue to represent a significant bottleneck in large-scale parallel scientific applications. This is, in part, because of the slower rate of development that parallel storage has witnessed when compared to that of micro-processors. However, other causes relate to limited optimisation at code level as well as the use of complex file formatting libraries. The situation is that contemporary applications can often exhibit poor I/O performance because code developers lack an understanding of how their code utilises I/O resources and how best to optimise for this.

In this paper we utilise the RIOT toolkit to intercept, record and analyse information relating to file reads, writes and locking operations within three standard industry I/O benchmarks and mini-applications. We presented a case study demonstrating RIOT's ability to:

Table 4. Total time in POSIX writes and average time per POSIX write for BT class C

	Processor Cores				
	1	4	16	64	256
Number of POSIX writes	13040	440	480	480	880
Standard MPI-IO					
Total time in POSIX write (s)	27.617	22.264	38.459	159.348	816.373
Time per write (s)	0.002	0.051	0.080	0.332	0.928
MPI-IO with PLFS					
Total time in POSIX write (s)	35.669	44.383	27.554	12.055	14.815
Time per write (s)	0.003	0.101	0.057	0.025	0.017

**Fig. 5.** Concurrent POSIX write calls for BT through MPI-IO and PLFS

- Calculate effective MPI-IO write bandwidths as well as produce bandwidths for POSIX file system calls originating from MPI-IO at increased scale. The comparison of these two figures demonstrates the slow-down in per-process write speed which results from the use of MPI. Typically these overheads arise because of file system contention, collective negotiation between MPI ranks for lock ownership and the calculation of offsets at which reads or writes take place;
- Provide detailed write behaviour analysis through the interception of read, write and locking operations included aggregated read/write time and the time spent obtaining or releasing per-file locks. Our results demonstrate the significant overhead incurred when performing HDF-5-based writes in the FLASH-IO benchmark. The nature of this analysis allows light-weight, non-intrusive detection of potential bottlenecks in I/O activity providing a first point at which application designers can begin optimisation;
- Compare low-level file system behaviour. In the last section of our case study we were able to investigate the low-level improvement which results in the use of PLFS middleware when executing the BT benchmark. PLFS is de-

signed specifically to reduce file system and parallel overheads through the interposition of MPI file operations to re-target n -to-1 operations to n -to- n operations. Through the tracing of these runs, RIOT was able to demonstrate an improvement in MPI-IO bandwidth due to the improvement in parallel POSIX bandwidth.

5.1 Future Work

This work builds upon preliminary work in [26] to show RIOT operating at scale on the 261.3 TFLOP/s Sierra supercomputer. Future studies are planned, applying this method of I/O tracing to larger, full-science applications. We expect these to exhibit increased complexity in their read/write behaviour resulting in increased contention and stress on the parallel file system. Further work with our industrial sponsors is also expected to use RIOT in the on-going assessment of parallel file system software and I/O-related middleware including the use of Lustre, GPFS, PLFS and alternatives.

Furthermore, future work is expected to include in-depth analysis of various I/O configurations, such as that utilised by BlueGene systems. It is expected that the performance characteristics seen on these systems will introduce additional complexities in measuring and visualising the I/O behaviour.

Acknowledgements. This work is supported in part by The Royal Society through their Industry Fellowship Scheme (IF090020/AM). Access to the LLNL Open Computing Facility is made possible through collaboration with the UK Atomic Weapons Establishment under grants CDK0660 (The Production of Predictive Models for Future Computing Requirements) and CDK0724 (AWE Technical Outreach Programme). We are grateful to Scott Futral, Todd Gamblin, Jan Nunes and the Livermore Computing Team for access to, and help in using, the Sierra machine located at LLNL. We are also indebted to John Bent and Meghan Wingate at the Los Alamos National Laboratory for their expert PLFS advice and support.

References

1. Almási, G.S., Archer, C., Castaños, J.G., Erway, C.C., Heidelberger, P., Martorell, X., Moreira, J.E., Pinnow, K., Ratterman, J., Smeds, N., Steinmacher-burow, B., Gropp, W.D., Toonen, B.: Implementing MPI on the BlueGene/L Supercomputer. In: Danelutto, M., Vanneschi, M., Laforenza, D. (eds.) Euro-Par 2004. LNCS, vol. 3149, pp. 833–845. Springer, Heidelberg (2004)
2. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrisnan, V., Weeratunga, S.K.: The NAS Parallel Benchmarks. *International Journal of High Performance Computing Applications* 5(3), 63–73 (1991)
3. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, L., Fatoohi, R.A., Fineberg, S., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrisnan, V., Weeratunga, S.K.: The NAS Parallel Benchmarks. Tech. Rep. RNR-94-007, NASA Ames Research Center (March 1994)

4. Bent, J., Gibson, G., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., Wingate, M.: PLFS: A Checkpoint Filesystem for Parallel Applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC 2009) (2009)
5. Bull: BullX Cluster Suite Application Developer's Guide (April 2010)
6. Carns, P., Latham, R., Ross, R., Iskra, K., Land, S., Riley, K.: 24/7 Characterization of Petascale I/O Workloads. In: Proceedings of the IEEE International Conference on Cluster Computing and Workshops (CLUSTER 2009), pp. 1–10 (September 2009)
7. Fryxell, B., Olson, K., Ricker, P., Timmes, F.X., Zingale, M., Lamb, D.Q., MacNeice, P., Rosner, R., Truran, J.W., Tufo, H.: FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series* 131(1), 273 (2000)
8. Fuerlinger, K., Wright, N.J., Skinner, D.: Effective Performance Measurement at Petascale Using IPM. In: Proceedings of the IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS 2010), pp. 373–380 (December 2010)
9. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B.W., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: Open MPI: Goals, concept, and design of a next generation MPI implementation. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J. (eds.) EuroPVM/MPI 2004. LNCS, vol. 3241, pp. 97–104. Springer, Heidelberg (2004)
10. Gropp, W.D.: MPICH2: A New Start for MPI Implementations. In: Kranzlmüller, D., Kacsuk, P., Dongarra, J., Volkert, J. (eds.) PVM/MPI 2002. LNCS, vol. 2474, pp. 7–42. Springer, Heidelberg (2002)
11. Koziol, Q., Matzke, R.: HDF5 – A New Generation of HDF: Reference Manual and User Guide. Tech. rep., National Center for Supercomputing Applications, Champaign, Illinois, USA (1998)
12. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., Zingale, M.: Parallel netCDF: A High-Performance Scientific I/O Interface. In: Proceedings of the ACM/IEEE International Conference on Supercomputing, SC 2003 (November 2003)
13. Message Passing Interface Forum: MPI2: A Message Passing Interface Standard. *High Performance Computing Applications* 12(1-2), 1–299 (1998)
14. Nowoczynski, P., Stone, N., Yanovich, J., Sommerfield, J.: Zest Checkpoint Storage System for Large Supercomputers. In: Proceedings of the 3rd Annual Workshop on Petascale Data Storage (PDSW 2008), pp. 1–5 (November 2008)
15. Polte, M., Simsa, J., Tantisiriroj, W., Gibson, G., Dayal, S., Chainani, M., Uppugandla, D.K.: Fast Log-based Concurrent Writing of Checkpoints. In: Proceedings of the 3rd Annual Workshop on Petascale Data Storage (PDSW 2008), pp. 1–4 (November 2008)
16. Polte, M., Lofstead, J., Bent, J., Gibson, G., Klasky, S.A., Liu, Q., Parashar, M., Podhorski, N., Schwan, K., Wingate, M., Wolf, M.: And Eat It Too: High Read Performance in Write-Optimized HPC I/O Middleware File Formats. In: Proceedings of the 4th Annual Workshop on Petascale Data Storage (PDSW 2009), pp. 21–25 (November 2009)
17. Rew, R.K., Davis, G.P.: NetCDF: An Interface for Scientific Data Access. *IEEE Computer Graphics and Applications* 10(4), 76–82 (1990)

18. Rosner, R., Calder, A., Dursi, J., Fryxell, B., Lamb, D.Q., Niemeyer, J.C., Olson, K., Ricker, P., Timmes, F.X., Truran, J.W., Tuelo, H., Young, Y., Zingale, M., Lusk, E., Stevens, R.: Flash Code: Studying Astrophysical Thermonuclear Flashes. *Computing in Science & Engineering* 2(2), 33–41 (2000)
19. Shan, H., Antypas, K., Shalf, J.: Characterizing and Predicting the I/O Performance of HPC Applications using a Parameterized Synthetic Benchmark. In: *Proceedings of the ACM/IEEE International Conference on Supercomputing, SC 2008* (November 2008)
20. Shan, H., Shalf, J.: Using IOR to Analyze the I/O Performance for HPC Platforms. In: *Cray User Group Conference (CUG 2007)*, Seattle, WA, USA (May 2007)
21. Thakur, R., Lusk, E., Gropp, W.: ROMIO: A High-Performance, Portable MPI-IO Implementation. Tech. Rep. ANL/MCS-TM-234, Mathematics and Computer Science Division, Argonne National Laboratory (1997)
22. Uselton, A., Howison, M., Wright, N.J., Skinner, D., Keen, N., Shalf, J., Karavanic, K.L., Oliner, L.: Parallel I/O Performance: From Events to Ensembles. In: *Proceedings of the IEEE International Symposium on Parallel Distributed Processing (IPDPS 2010)*, pp. 1–11 (April 2010)
23. Wang, Y., Kaeli, D.: Source Level Transformations to Improve I/O Data Partitioning. In: *Proceedings of the 1st International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI 2003)* (September–October 2003)
24. Wang, Y., Kaeli, D.: Profile-guided I/O Partitioning. In: *Proceedings of the 17th Annual International Conference on Supercomputing (ICS 2003)*, pp. 252–260 (June 2003)
25. Wolman, B., Olson, T.: IOBENCH: A System Independent IO Benchmark. *ACM SIGARCH Computer Architecture News* 17(5), 55–70 (1989)
26. Wright, S.A., Pennycook, S.J., Hammond, S.D., Jarvis, S.A.: RIOT – A Parallel Input/Output Tracer. In: *Proceedings of the 27th Annual UK Performance Engineering Workshop (UKPEW 2011)*, pp. 25–39 (July 2011)

Can Linear Approximation Improve Performance Prediction ?

Vlastimil Babka and Petr Tůma

Department of Distributed and Dependable Systems
Charles University in Prague, Faculty of Mathematics and Physics
Malostranské náměstí 25, Prague 1, 118 00, Czech Republic
{vlastimil.babka, petr.tuma}@d3s.mff.cuni.cz

Abstract. Software performance evaluation relies on the ability of simple models to predict the performance of complex systems. Often, however, the models are not capturing potentially relevant effects in system behavior, such as sharing of memory caches or sharing of cores by hardware threads. The goal of this paper is to investigate whether and to what degree a simple linear adjustment of service demands in software performance models captures these effects and thus improves accuracy. Outlined experiments explore the limits of the approach on two hardware platforms that include shared caches and hardware threads, with results indicating that the approach can improve throughput prediction accuracy significantly, but can also lead to loss of accuracy when the performance models are otherwise defective.

Keywords: Performance modeling, resource sharing, linear models.

1 Introduction

Software performance modeling has long relied on the ability of relatively simple models to predict the performance of relatively complex systems with often surprising accuracy. To pick an early example, [17] recalls over 40 years old result, where a simple analytical model was used to predict the performance of a time sharing operating system. In spite of many simplifying assumptions, such as exponential distribution of execution times and first come first served scheduling, the mean response time prediction error was typically below 10% and always below 25%.

Interestingly, simple models of complex systems achieve surprising accuracy even today. In [15], a QPN model¹ with 14 places and 19 transitions describes a distributed computing application with a load balancer, four application server nodes running the WebLogic container, and a multiprocessor database node running the Oracle Database engine. Even though the size of the model is tiny compared to the size of the system, and even though the model still assumes

¹ Queueing Petri Nets combine Queueing Networks with Petri Nets, making it possible to integrate a service queue into each network place.

exponential service demands and first come first served scheduling, it is shown to predict mean response times with error typically below 10%.

With systems getting ever more sophisticated, the complexity gap between the model and the implementation is growing. This puts the research community into an increasingly uncomfortable situation – on one hand, reports illustrate surprisingly accurate performance predictions for very complex systems, but on the other hand, our understanding of the systems suggests that the models are not really capturing potentially relevant aspects of system behavior.

A prominent example of this situation are the effects of memory caches in contemporary multiprocessor systems. Although these effects have been well documented by measurement [6,1,14,21], they are typically ignored in software performance modeling. And although functional models of memory caches do exist [10,20,25], their complex inputs and other features make application in software performance modeling difficult [5,4]. Yet another example are the effects of garbage collection in managed environments, where the performance impact is pronounced [9,19], but where reasonably precise models still do not exist [18].

Completing the picture are increasingly frequent observations that, in many research studies, the true complexity of the system performance is not really understood or appreciated, leading to potentially misleading results [11].

In this context, we investigate the possibility of approximating the effects of sharing selected resources in contemporary multiprocessor systems by a linear function of the overall system utilization. Specifically, we focus on the processor execution units, memory caches, buses and other architectural elements that are shared by processor-intensive workloads and not represented explicitly in software performance models. We call these *resources* for short, but emphasize that our use of the term excludes elements already captured in software performance models, such as whole idealized processors, storage devices, connections or mutexes.

Our extension to the software performance models targets situations where a full model of the performance effects is too complex or requires unknown inputs, or where the spectrum of performance effects to be modeled is not even known. Main features of the extension are three. (1) We adjust service demands based on system utilization, our work is therefore applicable to a wide class of models that accept service demands among their inputs and provide system utilization among their outputs. (2) Rather than trying to find a complex function that fits particular empirical observations, we are trying to see how well a simple function works in general. This makes our results less precise but more applicable. (3) Instead of selecting and modeling a single resource, we check whether and to what degree we can approximate effects due to simultaneous sharing of multiple resources.

Our work shares rationale with other software performance modeling methods that approximate empirical observations when modeling the system structure is not feasible. In this, we can liken our approach to that of [13], which uses statistical regression to express service demands as functions of workload parameters. Going beyond [13], we assume that identifying and modeling the functions and

the parameters is also not necessarily feasible or efficient, and instead examine an approximation that uses a linear function of the overall system utilization, motivated by some past observations that have often revealed linear patterns [4,1].

We emphasize that we do not see our main contribution in the extension to the software performance models, which is deliberately simple, but in the evaluation of the potential that this extension has. Notably, this evaluation uses our tool for generating synthetic software applications [7].

The paper proceeds by introducing the synthetic software applications and the software performance models that we utilize in our observations, in Section 2. Next, we describe experiments that assess the maximum potential accuracy of the models, in Section 3, and experiments that assess the accuracy with realistic inputs, in Section 4. We conclude with analyzing the limitations of the method in Section 5.

2 Performance Models and Synthetic Applications

We consider software performance models derived from the *architecture* and *behavior* descriptions of a system – the former tells what software components the system consists of, the latter tells how the software components invoke each other when processing a particular workflow. These models are often derived from other software design artefacts using model driven transformations [23], and are used for example in KLAPER [12] or Palladio [8].

An essential input to the models are the durations of operations that the software components perform, typically represented as service demands in the model. We assume predictive performance modeling in late stages of development, where service demands of individual software components can be measured rather than estimated, as is done for example in some experiments in [15].

2.1 Random Software Applications

To test the accuracy of the models using empirical observations, we need systems to model and observe. Unfortunately, both modeling and observation of a software system generally requires significant amount of manual work, such as creating and calibrating the models or instrumenting and measuring the systems. This would make the testing too expensive and lead to using only a few systems, which would in turn make our results difficult to generalize – since our models do not capture system behavior in detail, we can only generalize if we test on a wide enough set of systems.

To overcome this obstacle, we use our tool for generating synthetic software applications [7]. The tool assembles modules taken from industry standard benchmarks [22] and other sources [6] into applications whose architecture is random and whose workload exhibits the effects of resource sharing. The object and activity diagrams of one such application are on Figure 1.

Using a wide range of synthetic software applications makes our results more general in that our observations are not collected on only a few systems, where inadvertent bias, experiment tuning or even plain luck can distort the conclusions

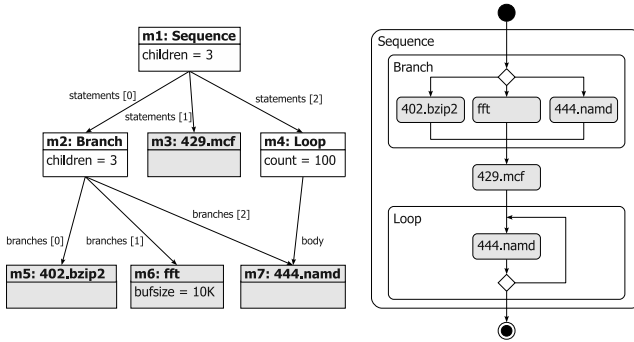


Fig. 1. Example object and activity diagrams of a generated application

significantly. For a more involved discussion of the representativeness of the applications, we refer the reader to [7] – here, we limit ourselves to asserting that our conclusions should be reasonably valid for concurrent systems running industry-standard processor-intensive workloads.

Another aspect of result generalization concerns the hardware platforms used in our observations. In this paper, we use two servers with modern Intel Xeon processors as representatives of common hardware platforms.² Measurements on more hardware platforms would represent a straightforward extension of the work presented here and, given the amount of experiments involved, are deemed out of scope for this paper.

2.2 Queueing Petri Net Models

The synthetic software applications are generated together with their architectural and behavioral model, which can be converted into multiple modeling formalisms. For this paper, we use the Queueing Petri Nets as implemented by the SimQPN tool [16]. The details of the mapping from the architectural and behavioral model into the performance model are given in [7,3]. Briefly, the mapping uses queueing places to represent processors and clients. Token colors encode computation state, transitions model control flow by removing tokens of a color corresponding to one particular program state and depositing tokens of a color corresponding to the subsequent program state. Helper places are used to represent thread pools, thread synchronization, branching and looping. Service

² For experiments without hardware multithreading, we use a Dell PowerEdge 1955 system with two Quad-Core Intel Xeon processors (Type E5345, Family 6, Model 15, Stepping 11, Clock 2.33 GHz), 8 GB DDR2-667 memory, Intel 5000P memory controller, Fedora Linux 8, gcc-4.1.2-33.x86 64, glibc-2.7-2.x86 64. For experiments with hardware multithreading, we use a Dell PowerEdge M610 system with two Quad-Core (Eight-Thread) Intel Xeon processors (Type E5540, Family 6, Model 26, Stepping 5, Clock 2.53 GHz (2.8 GHz TurboBoost)), 48 GB DDR3-1066 memory, ArchLinux 2.6.38.6, gcc-4.2.4-8, glibc-2.13-5.

demands use normal distribution, which we found to be more appropriate than the more common exponential distribution [7], loop counts are approximated with geometric distribution.

The service demands, which represent the operation durations of the individual software components, are measured. In a real system, the operation durations depend on the amount of resource sharing that occurs during the operation execution. We therefore focus on the way in which the service demands can be measured:

- Measurement in isolation. This option assumes that the measured component is executing alone, driven by an artificial workload generator harness. Since no other components execute, resource sharing is minimized.
- Measurement under resource sharing. This option assumes that the measured component is executing as a part of the modeled application. Here, resource sharing not only occurs, but occurs to the degree and with the effects that are characteristic for this particular application.

Obviously, software performance models that do not themselves capture the resource sharing effects will achieve better accuracy with service demands measured under resource sharing [7]. This, however, is not always realistic – as described, measurement under resource sharing requires having the modeled application, which is not the case in predictive performance modeling. Still, the approach serves well for comparison purposes – when we investigate a method that adjusts the service demands to reflect resource sharing, then measuring the service demands under resource sharing corresponds to a perfect adjustment. Hence, we start by comparing the accuracy of models that use service demands measured in isolation with models that use measurement under resource sharing.

As the next step, we approximate the effects of resource sharing by expressing the service demands as linear functions of the overall system utilization. With $t_{isolated}$ being a particular service demand measured in isolation, u representing the system utilization³ and δ approximating the sensitivity of this service demand to resource sharing, the adjusted service demand $t_{adjusted}$ is simply:

$$t_{adjusted} = t_{isolated} \times (1 + \max(0, u - 1) \times \delta) \quad (1)$$

The adjustment to the service demands is made in two steps. First, the model is solved with each service demand set to the appropriate $t_{isolated}$, yielding the estimate of utilization u and the adjusted service demands $t_{adjusted}$. These are then used to solve the model again. Multiple ways of determining the values of δ are described later.

The motivation for using the linear approximation stems from some past observations that have often revealed a roughly linear dependency between some parameters of the workload, such as the cache miss rate or the range of traversed addresses, and the operation durations [4,1]. The choice of system utilization as the argument of the linear function is based on the assumption that higher utilization means more system activity, which in turn means more opportunities for generating cache misses, traversing addresses, or other forms of resource sharing. The reader will note that none of these assumptions is, of course, valid in the

³ We use 0 for idle system and 1 for system with single processor fully utilized.

general sense – we are not looking for a perfect approximation of the operation durations, we are checking how far a simple approximation can go.

3 Determining Maximum Accuracy

To evaluate the accuracy, a number of synthetic software applications has been generated and compiled, and for each application, workloads of four to six different intensities have been measured and modeled. The accuracy of each combination of application and workload used is expressed as the prediction error $e = (x_{predicted} - x_{measured}) / x_{measured}$, with $x_{predicted}$ being the value of response time or throughput predicted by the model and $x_{measured}$ being the measured value, respectively.

The graphs that illustrate the accuracy contain a single data point reporting the accuracy of the model for each combination of application and workload used. The results of the *adjusted* models, where the service demands were set depending on the particular experiment, are plotted against the results of the baseline models, where the service demands were measured in isolation.

Special care has to be taken when interpreting the results of experiments that use the synthetic software applications. Since we are developing a simplified model for situations where a complete model of the performance effects is not known, we have to assume that the performance effects can depend on potentially unknown properties of the measured systems.⁴ When these properties are not known, we also cannot tell whether they would be represented among synthetic and real applications with the same frequencies. Hence, the fact that the performance models reach certain degree of accuracy in certain percentage of synthetic applications does not necessarily imply that the same degree of accuracy would be reached in the same percentage of real applications.

In this context, the *number* of synthetic applications where the baseline model exhibits particular prediction error is not relevant, but the *difference* in accuracy between the baseline model and the adjusted model is. This interpretation is explained in a verbose form with the first graph described below and applies analogously to the other graphs.

We use over 750 applications for the experiments without hardware multi-threading and over 500 applications for the experiments with hardware multi-threading, these numbers being picked somewhat arbitrarily – there would be no technical problem in using more applications, however, the results do not seem to indicate such need. The size of the individual applications ranges from 5 to 20 workload components glued together with additional control components, the typical execution time for each application is in the order of seconds. Common precautions against initialization effects were taken by discarding warmup measurements, leaving 220 measurements per application and configuration on average.

⁴ It is not that we could not provide many examples of properties that impact performance under resource sharing [1], we just cannot assume knowledge of the role of particular properties in particular workloads.

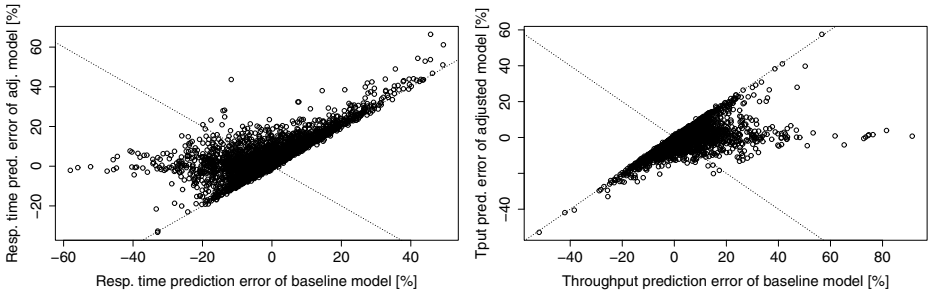


Fig. 2. Prediction error for response time (left) and throughput (right) of models adjusted with measurements under resource sharing. Platform without hyperthreading.

3.1 Perfect Service Demand Adjustment

To assess the best possible accuracy that any modeling method based on adjusting the service demands can achieve, we compare the accuracy of the baseline models with adjusted models that use measurement under resource sharing. Since thus measured service demands reflect the effects of resource sharing exactly as it happens in the executing application, they represent the perfect adjustment – which, however, is often not available in practice when constructing performance models.

By talking about the perfect adjustment, we do not mean that the model will produce results that are as close to measured overall performance as possible. It is possible that other service demands would lead to a result that is closer to measurement – however, rather than representing the true behavior of the application, such service demands would artificially compensate for other deficiencies in the model.

Figure 2 plots the accuracy of the adjusted models against the accuracy of the baseline models for response time and throughput predictions. The fact that all the response time observations are above the diagonal and all the throughput observations below the diagonal corresponds to the fact that service demands measured under sharing are higher than service demands measured in isolation, which leads to higher response time and lower throughput estimates. The observations near the origin point are not very useful, since they denote situations where both the baseline model and the adjusted model achieve reasonable accuracy. Other observations can be classified based on their location in the graph:

- Observations on the diagonal but not near the origin point. These observations suggest some causes of prediction error are not related to service demands, even the perfect adjustment therefore does not impact accuracy in those cases.
- Observations towards the nine o’clock direction for response time and the three o’clock direction for throughput. These observations concern situations where the baseline model predicts with a significant error that the adjusted model rectifies.

- Observations towards the twelve o'clock direction for response time and six o'clock direction for throughput. These observations concern situations where the baseline model predicts with a reasonable accuracy that the adjusted model lacks.

The gains in accuracy are more pronounced with throughput than with response time. We attribute this to the fact that various approximations in the model, especially the approximations of loop counts with geometric distributions, impact response time more than throughput. The observations indicate that the perfect adjustment can improve accuracy in situations where the baseline is 60% optimistic on response time and 90% optimistic on throughput into almost precise prediction on both response time and throughput.

The losses in accuracy are more pronounced with response time than with throughput, pointing out that the baseline models sometimes work only because the service demands measured in isolation compensate for other deficiencies in the model.

As explained, the number of observations for synthetic applications in each graph location does not necessarily indicate the number of real applications whose observations would fall in the same location. We can, however, refer to our earlier measurements in [2], which show that out of 29 SPEC CPU2006 benchmarks, 15 exhibit at least 50% slowdown and all exhibit at least 10% slowdown under some modes of resource sharing. Since the baseline models do not capture this slowdown and are therefore necessarily optimistic, these measurements would fall to the nine o'clock direction for response time and three o'clock direction for throughput in our graphs, that is, locations where the adjusted models improve accuracy. To the degree that the SPEC CPU2006 benchmarks represent real applications, we can therefore claim that over half of real applications can exhibit significant slowdown under resource sharing that the adjusted models predict better than the baseline models.

Figure 3 again plots the accuracy of the adjusted models against the accuracy of the baseline models, but the platform with hardware multithreading was used for the measurements. In general, the observations are in line with the expectation that hardware multithreading will contribute to performance effects due to resource sharing.

The plots again indicate that some causes of prediction error are not related to service demands. The plots also show an even more pronounced set of observations in the nine o'clock direction for response time and three o'clock direction for throughput, indicating that the perfect adjustment can improve accuracy in situations where the baseline is up to 90% optimistic on response time and 1000% optimistic on throughput into about 30% pessimism on response time and about 20% optimism on throughput.⁵

⁵ Note that this can include the effects of clock boosting as a mechanism not captured by the performance model, but the maximum performance change should not exceed 10%.

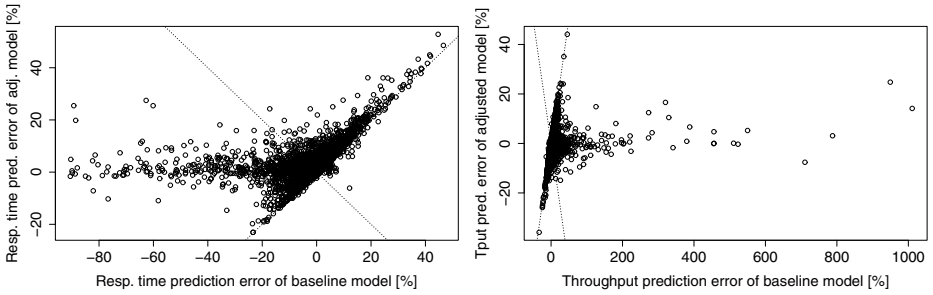


Fig. 3. Prediction error for response time (left) and throughput (right) of models adjusted with measurements under resource sharing. Platform with hyperthreading.

3.2 Linear Service Demand Adjustment

The models that use service demands measurement under resource sharing provide us with an estimate of the best possible accuracy that adjusting the service demands can achieve. Unfortunately, such measurements are not always available in practice – for example, the modeled application might still be in a design stage, an installation specifically for measurement purposes might be too expensive, or collecting enough observations of a rarely executed operation might just take too long. Which is why, as the next step, we approximate the service demands with simple linear functions of the overall system utilization using equation 1.

In this section, we take the service demands measured in isolation and under sharing as observations of $t_{isolated}$ and $t_{adjusted}$, respectively, together with the system utilizations u predicted using the isolated times, and derive the slowdown factors δ that yield the least square error for the response variable $t_{adjusted}$ using linear regression, for each type of operation.⁶ Again, this requires measurement under resource sharing and is therefore not practical, however, the experiment serves to indicate the accuracy achievable with the linear approximation. In the next section, we will assess the accuracy under realistic conditions.

To avoid the systematic error of deriving the slowdown factors from the systems whose prediction accuracy is evaluated, we use a variant of cross-validation based on random sub-sampling [24]. We repeatedly use random choice to split the set of all synthetic software applications into two subsets. The first of the subsets contains 90% of the applications and is used to derive the slowdown factors for all operations, the second of the subsets is then used to assess the prediction accuracy.

Compared to Figure 2, Figure 4 indicates that the linear approximation leads to lower prediction accuracy. Still, where the baseline models are 60% optimistic on response time and 90% optimistic on throughput, the adjusted models are about 20% optimistic on response time and about 20% optimistic on throughput.

⁶ Only systems with utilization higher than 1 are used in the regression.

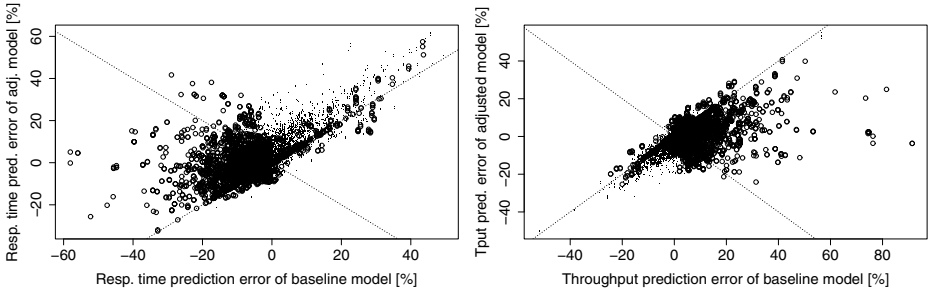


Fig. 4. Prediction error for response time (left) and throughput (right) of models adjusted with estimated service demands. Platform without hyperthreading.

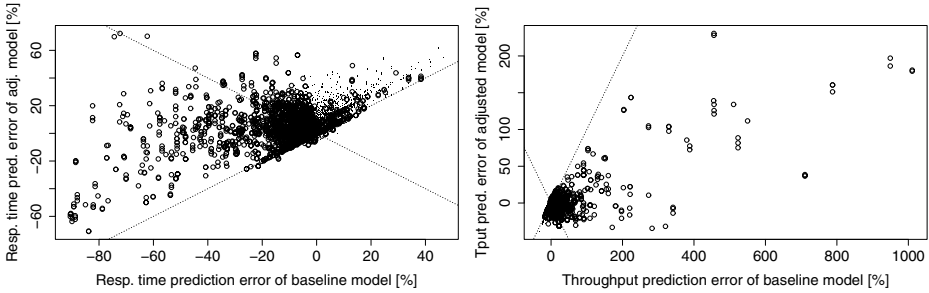


Fig. 5. Prediction error for response time (left) and throughput (right) of models adjusted with estimated service demands. Platform with hyperthreading.

For the platform with hardware multithreading, Figure 5 should be compared to Figure 3. Where the baseline models are 90 % optimistic on response time and 1000 % optimistic on throughput, the adjusted models are about 60 % optimistic on response time and about 200 % optimistic on throughput.

The observations where the accuracy of the adjusted models was already worse than the accuracy of the baseline models on Figures 2 and 3 are plotted with dots rather than circles. These are the cases where even using precise service demands does not improve accuracy, suggesting that these models are otherwise deficient and that adjusting service demands is not a way to improve their accuracy.

More interesting are the cases where using precise service demands does improve accuracy, but using linear adjustment does not – these correspond to the circles in the twelve o’clock direction for response time and six o’clock direction for throughput. This is an unavoidable occurrence given the linear approximation. Importantly, the observed gains in accuracy for the cases where the adjustment does help are larger than the observed losses for the cases where the baseline models are better.

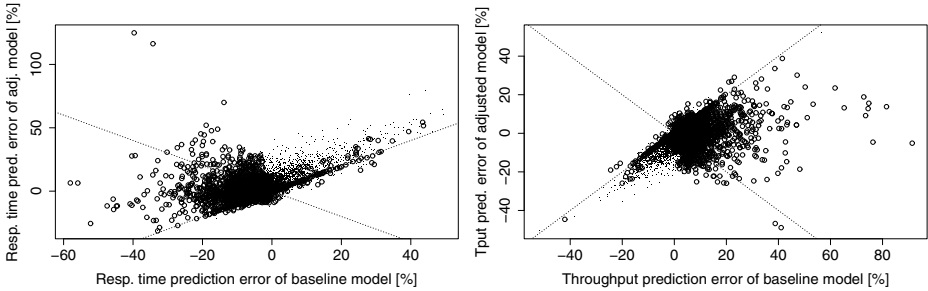


Fig. 6. Prediction error for response time (left) and throughput (right) of models adjusted with service demands estimated from measurements under sharing with SPEC CPU2006 benchmarks. Platform without hyperthreading.

4 Assessing Realistic Accuracy

To use the linear adjustment under realistic conditions, we have to estimate the slowdown factors δ without measuring the service demands in the modeled application. We do that by running each software component whose service demands are to be measured in parallel with the SPEC CPU2006 benchmarks, and calculating the average slowdown factor for each operation from these measurements. The SPEC CPU2006 benchmarks thus substitute the modeled application in creating conditions for resource sharing, even if not necessarily quite to the degree or with quite the same effects that are characteristic for the modeled application.

Our hardware platforms allow multiple ways of deploying the measured software components and the benchmarks on the available processors, depending on whether the processor package and the memory cache is shared. On the platform without hardware multithreading, we have decided to run the two workloads so that they share the package but not the cache on the platform without hardware multithreading. This was chosen as a middle option among the other choices after quick experiments.

Compared to Figures 2 and Figure 4, Figure 6 indicates that the realistic adjustment works about as well as the adjustment based on linear regression from the previous section. Where the baseline models are 60 % optimistic on response time and 90 % optimistic on throughput, the adjusted models are almost precise on response time and throughput.

On the down side, the adjusted models have worse precision than the baseline models for some cases where adjusting the service demands could have helped. In the worst case, the adjusted models are 120 % pessimistic on response time and 40 % pessimistic on throughput where the baseline models are 40 % optimistic on response time and 40 % pessimistic on throughput. This confirms our earlier observation that some approximations used in the model impact response time more than throughput.

On the platform with hardware multithreading, there are even more ways of deploying the measured software components and the benchmarks on the available processors. One extreme would be deploying on threads that share the same core, another extreme would be deploying on threads that do not share the same package. Unfortunately, our measurements indicate that such simple deployment options would yield poor slowdown factor estimates – and while including a more complex evaluation of the deployment options is possible, we believe that a better way of obtaining the slowdown factor estimates is by using application specific workloads, discussed in the next section.

5 Limitations

The measurements in Sections 3 and 4 indicate that adjusting service demands using a linear function of system utilization increases prediction accuracy in cases where the baseline models are too optimistic. Many of the cases where the adjustment decreases accuracy were shown to suffer from deficiencies not related to the service demands, however, one limitation of the adjustment is that it can, in general, also decrease accuracy.

Whether the potential decrease of accuracy is important in practice depends on many considerations – especially for throughput prediction, but to a degree also for response time prediction, the observed accuracy gains tended to be larger than losses. Unfortunately, our validation approach based on synthetic software applications does not permit us to state that the gains would occur more frequently than the losses in real applications.

Also evident is the fact that the method is never more optimistic than the baseline. This can be important for studies that concern system dimensioning – using the adjustment, such studies are less likely to lead to systems with insufficient performance.

Some improvement can be achieved by deriving the slowdown factor δ from measurements that use application specific workloads, rather than the SPEC CPU2006 benchmarks as we did. For example, when modeling an audio processing application, the slowdown factors would be measured against an audio processing workload, other workloads would be used for other applications. Our results for perfect adjustment from Section 3 show what accuracy can be expected in this case – also, we show that this adjustment does not always lead to accuracy gain. Nonetheless, this seems to be the most practical option for further improvement.

Besides the slowdown factor δ , the accuracy of the adjustment is also related to the system utilization, which is used in calculating the service demands. We obtain the system utilization estimate from the baseline model and use it to construct the adjusted model, as described in Section 2 – however, if the adjusted model is generally more accurate than the baseline model, it would also give us a better system utilization estimate.

It might seem that we could obtain the system utilization estimate through iteration, creating a new adjusted model in each iteration step using the adjusted

model from the previous iteration step. With potential for positive feedback and unclear stopping conditions, however, we have decided against using iteration.

Taking a broader view, the proposed adjustment assumes that performance effects of resource sharing can be modeled as increase in service demands. This holds reasonably well for many resources, such as execution units or memory caches, but does not hold in general. One notable exception is the sharing of managed heap, which typically leads to more frequent and potentially longer garbage collection pauses, rather than gradual increase in service demands.

Finally, our results are necessarily limited to the hardware platforms that we have used in the experiments.

6 Conclusion

The goal of this paper was to investigate whether and to what degree a simple linear adjustment of service demands in software performance models can improve accuracy.

The motivation for the goal stems from the fact that for many contemporary systems, it is difficult to just identify all the performance interactions, let alone model them accurately. When that is the case, current software performance models simply ignore the interactions, which is not satisfactory. Compared to that, the linear adjustment is not much better in terms of sophistication, however, it gives the models a chance of capturing at least some of the interactions and thus improving accuracy.

The linearity of the adjustment is based on the assumption that many performance interactions are due to sharing of resources such as execution units or memory caches, which generally slows execution down. More complex adjustments might approximate the performance interactions more precisely, however, they would also be more difficult to calibrate using limited amount of measurements.

Our results indicate that the proposed adjustment of service demands does improve prediction accuracy, however, we also reveal situations where performance models achieve reasonable accuracy only when populated with imprecise service demands. For those models, even correct service demands tend to worsen accuracy.

The presented experiments were carried out on two multiprocessor platforms and included potential for performance interactions through clock boosting and sharing of execution units, memory caches and memory buses. The results of the experiments indicate that the adjustment can improve especially throughput predictions, with the best examples including a correction of a 90% throughput overestimation into an almost precise throughput prediction. On the downside, the adjustment can also decrease accuracy, with the extreme example being a miscorrection of a 40% response time underestimation into a 120% response time overestimation.

The experiments required a large number of software applications to measure and model. Given that state of the art performance modeling tools still do not allow completely automated construction of performance models from legacy software, we have used a specialized tool that synthesizes software applications

together with models. Since the tool does not allow us to make claims on the relative frequency with which particular synthetic applications resemble particular real applications, we cannot say whether the (in)frequent observations in the experiments would also be (in)frequent in practice. We can, however, observe that over half of the workloads from the SPEC CPU2006 suite exhibits at least 50% slowdown under resource sharing that is not captured by current software performance models – and for cases where baseline models were predicting response time with at least 50% optimism, adjusted models decreased the prediction error to about 10%.

To conclude, we have shown that linear adjustment of service demands based on system utilization is a valid approach to improving prediction accuracy, especially when the baseline models have a tendency towards optimism. This result can be used to parametrize models with load dependent service times.

Acknowledgments. Complete measurements and software required to reproduce the results is available at <http://d3s.mff.cuni.cz/software/rpg>. This work was partially supported by the Czech Science Foundation projects GACR P202/10/J042 and GACR 201/09/H057.

References

1. Babka, V., Bulej, L., Decky, M., Kraft, J., Libic, P., Marek, L., Seceleanu, C., Tuma, P.: Resource Usage Modeling, Q-ImPrESS Project Deliverable D3.3 (2008), <http://www.q-impress.eu/>
2. Babka, V.: Cache Sharing Sensitivity of SPEC CPU 2006 Benchmarks, Tech. Rep. No. 2009/3, Dep. of SW Engineering, Charles University in Prague (June 2009), <http://d3s.mff.cuni.cz/>
3. Babka, V., Bulej, L., Ciancone, A., Filieri, A., Hauck, M., Libic, P., Marek, L., Stammel, J., Tuma, P.: Prediction Validation, Q-ImPrESS Project Deliverable D4.2 (2010), <http://www.q-impress.eu/>
4. Babka, V., Libič, P., Tůma, P.: Timing Penalties Associated with Cache Sharing. In: Proceedings of MASCOTS 2009. IEEE, Los Alamitos (2009)
5. Babka, V., Marek, L., Tůma, P.: When Misses Differ: Investigating Impact of Cache Misses on Observed Performance. In: Proceedings of ICPADS 2009, pp. 112–119. IEEE, Los Alamitos (2009)
6. Babka, V., Tůma, P.: Investigating Cache Parameters of x86 Family Processors. In: Kaeli, D., Sachs, K. (eds.) SPEC Benchmark Workshop 2009. LNCS, vol. 5419, pp. 77–96. Springer, Heidelberg (2009)
7. Babka, V., Tůma, P., Bulej, L.: Validating Model-Driven Performance Predictions on Random Software Systems. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) QoSA 2010. LNCS, vol. 6093, pp. 3–19. Springer, Heidelberg (2010)
8. Becker, S., Koziolok, H., Reussner, R.: The Palladio Component Model for Model-driven Performance Prediction. *J. Syst. Softw.* 82(1) (2009)
9. Blackburn, S.M., Cheng, P., McKinley, K.S.: Myths and Realities: The Performance Impact of Garbage Collection. *SIGMETRICS Perform. Eval. Rev.* 32(1) (2004)
10. Chandra, D., Guo, F., Kim, S., Solihin, Y.: Predicting Inter-Thread Cache Contention on a Chip Multi-Processor Architecture. In: Proceedings of HPCA 2005. IEEE CS, Los Alamitos (2005)

11. Click, C.: Evaluate 2010 Keynote (October 2010), <http://evaluate2010.inf.usi.ch/>
12. Grassi, V., Mirandola, R., Randazzo, E., Sabetta, A.: KLAPER: An Intermediate Language for Model-Driven Predictive Analysis of Performance and Reliability. In: Rausch, A., Reussner, R., Mirandola, R., Plášil, F. (eds.) *The Common Component Modeling Example*. LNCS, vol. 5153, pp. 327–356. Springer, Heidelberg (2008)
13. Happe, J., Westermann, D., Sachs, K., Kapová, L.: Statistical Inference of Software Performance Models for Parametric Performance Completions. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) *QoSA 2010*. LNCS, vol. 6093, pp. 20–35. Springer, Heidelberg (2010)
14. Kalibera, T., Bulej, L., Tůma, P.: Benchmark Precision and Random Initial State. In: *Proceedings of SPECTS 2005*, pp. 853–862. SCS (June 2005)
15. Kounev, S.: Performance Modeling and Evaluation of Distributed Component-Based Systems Using Queueing Petri Nets. *IEEE Trans. Software Eng.* 32(7) (2006)
16. Kounev, S., Buchmann, A.: SimQPN: A Tool and Methodology for Analyzing Queueing Petri Net Models by Means of Simulation. *Perform. Eval.* 63(4) (2006)
17. Lavenberg, S.S., Squillante, M.S.: Performance Evaluation in Industry: A Personal Perspective. In: Reiser, M., Haring, G., Lindemann, C. (eds.) *Dagstuhl Seminar 1997*. LNCS, vol. 1769, pp. 3–13. Springer, Heidelberg (2000)
18. Libič, P., Tůma, P.: Towards Garbage Collection Modeling, Tech. Rep. No. 20011/1, Dep. of Distributed and Dependable Systems, Charles University in Prague (January 2011), <http://d3s.mff.cuni.cz/>
19. Libič, P., Tůma, P., Bulej, L.: Issues in Performance Modeling of Applications with Garbage Collection. In: *Proceedings of QUASOSS 2009*, pp. 3–10. ACM, New York (2009)
20. Liu, F., Guo, F., Solihin, Y., Kim, S., Eker, A.: Characterizing and Modeling the Behavior of Context Switch Misses. In: *Proceedings of PACT 2008*. ACM, New York (2008)
21. Mytkowicz, T., Diwan, A., Hauswirth, M., Sweeney, P.F.: Producing Wrong Data Without Doing Anything Obviously Wrong! In: *Proceedings of ASPLOS 2009*, pp. 265–276. ACM, New York (2009)
22. Standard Performance Evaluation Corporation: SPEC CPU 2006 Benchmark, <http://www.spec.org/cpu2006/>
23. The Q-ImPrESS Project Consortium: Quality Impact Prediction for Evolving Service-oriented Software, <http://www.q-impress.eu/>
24. Wasserman, L.: *All of Statistics: A Concise Course in Statistical Inference*. Springer, Heidelberg (2004)
25. Xu, C., Chen, X., Dick, R.P., Mao, Z.M.: Cache Contention and Application Performance Prediction for Multi-core Systems. In: *Proceedings of ISPASS 2010*. IEEE, Los Alamitos (2010)

TWOEAGLES: A Model Transformation Tool from Architectural Descriptions to Queueing Networks

Marco Bernardo¹, Vittorio Cortellessa², and Mirko Flamminj²

¹ Dipartimento di Scienze di Base e Fondamenti, Università di Urbino, Italy

² Dipartimento di Informatica, Università dell'Aquila, Italy

Abstract. We present the implementation of a methodology for the modeling, analysis, and comparison of software architectures based on their performance characteristics. The implementation is part of a software tool that is called TWOEAGLES, which extends the architecture-centric tool TwoTowers – based on the stochastic process algebraic description language *ÆMILIA* – and integrates it into Eclipse. The extension consists of a Java-coded plugin that we have called *AEmilia_to_QN*. This plugin transforms *ÆMILIA* descriptions into queueing network models expressed in the XML schema *PMIF*, which can then be rendered via the *QN_Editor* tool or analyzed by multiple queueing network solvers that can be invoked through the Weasel web service.

1 Introduction

The importance of an integrated view of functional and nonfunctional characteristics in the early stages of software development is by now widely recognized. This is a consequence of the awareness of the risks arising either from considering those two classes of characteristics on two different classes of system models that are not necessarily consistent with each other, or from examining nonfunctional features at later stages of the development cycle. This has resulted in the extension of numerous semi-formal and formal notations with nonfunctional attributes yielding quantitative variants of logics, automata, Petri nets, process calculi, and specification languages as well as suitable UML profiles, many of which are surveyed, e.g., in [3].

The assessment of nonfunctional characteristics is not only instrumental to enhancing the quality of software systems. As an example, a number of alternative architectural designs may be developed for a given system, each of which is functionally correct. In that case, we need to establish some criteria for deciding which architectural design is more appropriate and hence is the one to implement. As it turns out, performance requirements and constraints are certainly among the most influential factors that drive architecture-level design choices.

In order to address the various issues mentioned above, in [2] a methodology has been proposed for predicting, improving, and comparing the performance of software architectures. This methodology, called *PERFSEL* in [1], consists of a number of phases at the end of which typical performance indices are assessed in different scenarios for the various architectural designs both at the system level and at the component level. On the basis of those indices, it can be decided to discard some designs, improve others, or select the one to be implemented.

Although the PERFSEL methodology is independent to a large extent from the notation in which architectural designs are expressed, as in [2,1] here we focus on *ÆMILIA*. This is an architectural description language based on stochastic process algebra that enables functional verification via model checking or equivalence checking, as well as performance evaluation through the numerical solution of continuous-time Markov chains or discrete-event simulation.

On the analysis side, PERFSEL instead employs queueing networks [10]. A main motivation of this choice is that, in contrast to continuous-time Markov chains – which are flat performance models – queueing networks are structured performance models providing support for establishing a correspondence between their constituent elements and the components of architectural descriptions. Moreover, some families of queueing networks, like product-form queueing networks [4], are equipped with efficient solution algorithms that do not require the construction of the underlying state space when calculating typical average performance indices at system level or component level, such as response time, throughput, utilization, and queue length. Therefore, the transformation of *ÆMILIA* models into queueing networks enables a wider set of performance analysis techniques on the same architectural model.

Starting from a number of alternative architectural designs – which we assume to be functionally correct – of a software system to be implemented, PERFSEL requires the designer to formalize each such design as an *ÆMILIA* description, which is subsequently transformed into a queueing network model. Whenever one of these models is not in product form, the model itself is replaced by an approximating product-form queueing network model. The possibly approximate product-form queueing network model associated with each architectural design is then evaluated in order to derive the typical average performance indices both at the system level and at the component level. The evaluation is done in several different scenarios of interest and the obtained performance figures are interpreted on the various *ÆMILIA* descriptions.

On the basis of those figures, for each alternative a decision has to be made as to whether the design is satisfactory, should be discarded, or may be improved. When the predict-improve cycle is terminated for all the survived architectural designs, a comparison among them takes place in the various scenarios according to the average performance indices. The selected architecture is finally checked against the specific performance requirements of the system under construction.

This final check is necessary for two reasons. Firstly, the selection is made by relying on general performance indices, which are not necessarily connected in any way to the specific performance requirements. Secondly, the product-form queueing network model associated with the selected architecture may have been subject to approximations. Although the perturbation of the average performance indices introduced by the approximations cannot be easily quantified, we recall from [11] that queueing network models are in general robust, in the sense that even their approximate analysis is in any case helpful to get useful insights into the performance of the systems they represent.

The key point of PERFSEL is the combined use of the two above mentioned formalisms: *ÆMILIA* for component-oriented modeling purposes and queueing networks for component-oriented performance analysis purposes. As observed in [2,1], the two formalisms are quite different from each other. On the one hand, *ÆMILIA* is

a completely formal, general-purpose architectural description language handling both functional and performance aspects, whose basic ingredients are actions and behavioral operators. On the other hand, queueing networks are instances of a queue-based graphical notation for performance aspects only, in which some details like the queueing disciplines are usually expressed in natural language. Another important feature to take into account is the different level of granularity of the models expressed in the two formalisms. In particular, it turns out that the components of an *ÆMILIA* description cannot be precisely mapped to the customer populations and the service centers of a queueing network model, but on finer parts called queueing network basic elements that represent arrival processes, buffers, service processes, fork processes, join processes, and routing processes. Therefore, not all *ÆMILIA* descriptions can be transformed into queueing network models, but only those satisfying certain constraints specified in [2,1].

This paper presents an implementation of *PERFSEL* and is organized as follows. In Sect. 2, we define the model transformation carried out by the plugin *ÆMILIA_to_QN* within *TWOEAGLES*. In Sect. 3, we introduce the plugin *ÆMILIA_to_QN* itself. In Sect. 4, we describe the architecture of *TWOEAGLES* and we show how it interoperates with other tools via *ÆMILIA_to_QN*. In Sect. 5, we illustrate by means of an automated teller machine example the adequacy of the model transformation and the higher degree of scalability achieved by *TWOEAGLES* in the performance evaluation of software architectures. Finally, in Sect. 6 we report some concluding remarks.

2 The Transformation from *ÆMILIA* to Queueing Networks

In this section, we present the transformation from *ÆMILIA* descriptions to queueing network models that we have implemented. More precisely, after recalling the transformation source (Sect. 2.1) and the transformation target (Sect. 2.2), we give an idea of how the transformation works in accordance with the queueing network basic elements identified in [2,1] (Sect. 2.3). Finally, we detail the transformation through a hierarchy that we have specifically developed for our implementation of *PERFSEL*, which is composed of an action classification, a behavioral pattern classification, pattern combination rules, and connectivity rules for the queueing network basic elements (Sect. 2.4).

2.1 The Transformation Source: *ÆMILIA*

ÆMILIA [1] is an architectural description language based on stochastic process algebra. An *ÆMILIA* description represents an architectural type, which is a family of software systems sharing certain constraints on the observable behavior of their components as well as on their topology. As shown in Table 1, the textual description of an architectural type in *ÆMILIA* starts with its name and its formal parameters (initialized with default values), then comprises an architectural behavior section and an architectural topology section.

The first section defines the overall behavior of the system family by means of types of software components and connectors, which are collectively called architectural element types. The definition of an AET, which starts with its name and its formal parameters, consists of the specification of its behavior and its interactions.

Table 1. Structure of an *ÆMILIA* textual description

ARCHI_TYPE	<i><name and initialized formal parameters></i>
ARCHI_BEHAVIOR	
⋮	⋮
ARCHI_ELEM_TYPE	<i><AET name and formal parameters></i>
BEHAVIOR	<i><sequence of stochastic process algebraic equations built from stop, action prefix, choice, and recursion></i>
INPUT_INTERACTIONS	<i><input synchronous/semi-synchronous/asynchronous uni/and/or-interactions></i>
OUTPUT_INTERACTIONS	<i><output synchronous/semi-synchronous/asynchronous uni/and/or-interactions></i>
⋮	⋮
ARCHI_TOPOLOGY	
ARCHI_ELEM_INSTANCES	<i><AEI names and actual parameters></i>
ARCHI_INTERACTIONS	<i><architecture-level AEI interactions></i>
ARCHI_ATTACHMENTS	<i><attachments between AEI local interactions></i>
END	

The behavior of an AET has to be provided in the form of a sequence of behavioral equations written in a verbose variant of stochastic process algebra allowing only for the inactive process (rendered as `stop`), the action prefix operator supporting possible boolean guards and value passing, the alternative composition operator (rendered as `choice`), and recursion. Every action represents an activity and is described as a pair composed of the activity name and the activity duration. On the basis of their duration, actions are divided into exponentially timed (duration `exp(τ)`), immediate (duration `inf(1, w)`), and passive (duration `_(1, w)`).

The interactions of an AET are actions occurring in the stochastic process algebraic specification of the behavior of the AET that act as interfaces for the AET itself, while all the other actions are assumed to represent internal activities. Each interaction has to be equipped with three qualifiers, with the first qualifier establishing whether the interaction is an input or output interaction.

The second qualifier represents the synchronicity of the communications in which the interaction can be involved. We distinguish among synchronous interactions which are blocking (default qualifier `SYNC`), semi-synchronous interactions which cause no blocking as they raise an exception if prevented (qualifier `SSYNC`), and asynchronous interactions which are completely decoupled from the other parties involved in the communication (qualifier `ASYNC`). Every semi-synchronous interaction is implicitly equipped with a boolean variable usable in the architectural description, which is automatically set to true if the interaction can be executed, false if an exception is raised.

The third qualifier describes the multiplicity of the communications in which the interaction can be involved. We distinguish among uni-interactions which are mainly involved in one-to-one communications (qualifier `UNI`), and-interactions guiding inclusive one-to-many communications like multicasts (qualifier `AND`), and or-interactions

guiding selective one-to-many communications like in a server-clients setting (qualifier OR). It can also be established that an output or-interaction depends on an input or-interaction, in order to guarantee that a selective one-to-many output is sent to the same element from which a selective many-to-one input was received (keyword DEP).

The second section of an *ÆMILIA* description defines the topology of the system family. This is accomplished in three steps. Firstly, we have the declaration of the instances of the AETs – called AEIs – which represent the actual system components and connectors, together with their actual parameters. Secondly, we have the declaration of the architectural (as opposed to local) interactions, which are some of the interactions of the AEIs that act as interfaces for the whole systems of the family. Thirdly, we have the declaration of the architectural attachments among the local interactions of the AEIs, which make the AEIs communicate with each other. An attachment is admissible only if it goes from an output interaction of an AEI to an input interaction of another AEI. Moreover, a uni-interaction can be attached only to one interaction, whereas an and/or-interaction can be attached only to uni-interactions. Within a set of attached interactions, at most one of them can be exponentially timed or immediate.

The semantics for *ÆMILIA* is given by translation into stochastic process algebra. Basically, the semantics of every AEI is the sequence of stochastic process algebraic equations defining the behavior of the corresponding AET. Then, the semantics of an entire architectural description is the parallel composition of the semantics of the constituent AEIs, with synchronization sets determined by the attachments. From the state-transition graph underlying the resulting stochastic process term, a continuous-time Markov chain can be derived for performance evaluation purposes, provided that there are no transitions labeled with passive actions (performance closure) and all the transitions labeled with immediate actions are suitably removed.

2.2 The Transformation Target: Queueing Networks

A queueing network (see, e.g., [10,11]) is a collection of interacting service centers that represent resources shared by classes of customers, where customer competition for resources corresponds to queueing into the service centers. In contrast to continuous-time Markov chains, queueing networks are structured performance models because they elucidate system components and their connectivity.

This brings a number of advantages in the architectural design phase. Firstly, typical average performance indices like throughput, utilization, mean queue length, and mean response time can be computed both at the level of an entire queueing network and at the level of its constituent service centers. Such global and local indicators can then be interpreted back at the level of an entire architectural description and at the level of its constituent components, respectively, in order to obtain diagnostic information. Secondly, there exist families of queueing networks that are equipped with fast solution algorithms that do not require the construction of the underlying state space. Among those families, we mention product-form queueing networks [4], which can be analyzed compositionally by solving each service center in isolation and then combining their solutions via multiplications. This provides support for a performance analysis that scales with respect to the number of components in architectural descriptions. Thirdly,

the solution of a queueing network can be expressed symbolically in the case of certain topologies. This feature is useful in the early stages of the software development cycle, since the actual values of system performance parameters may be unknown at that time.

2.3 The Transformation at a Glance

As mentioned in Sect. 1, the transformation source and target are quite different from each other. In particular, the respective models have different levels of granularity. As a consequence, the AEIs of an *ÆMILIA* description cannot be precisely mapped to the customer populations and the service centers of a queueing network model. For this reason, in [2,1] a number of finer parts called queueing network basic elements (QNBE for short) have been identified together with suitable syntactical restrictions that establish when an AEI can be transformed into one of those elements and when the AEIs from which those elements have been derived are connected in a way that yields a well-formed queueing network. The various QNBES are shown in Fig. 1, where f (resp. r) denotes the number of alternative destinations (resp. sources), h denotes the number of customer classes, and interarrival and service times are expressed through phase-type distributions, i.e., suitable combinations of exponential distributions.

An arrival process is a generator of arrivals of customers of a certain class. While a single arrival process is enough in the case of an unbounded population, an instance of the arrival process is necessary for each customer in the case of a finite population, with the return of the customer being explicitly modeled. A buffer is a repository of customers of different classes that are waiting to be served according to some queueing discipline that we assume to be first-come-first-served. In the case of a bounded buffer, incoming customers of class i can be accommodated only if the buffer capacity c_i for that class is not exceeded. A service process is a server for customers of various classes, whose service times can be different for each class. When a service center is composed of multiple servers, it is necessary to represent each of them through an instance of the service process. A fork process splits requests coming from customers of a certain class into subrequests directed to different service centers, which are then recombined together by a join process. Finally, a routing process simply forwards customers of a certain class towards different destinations.

2.4 A Hierarchical Approach to the Transformation

We now describe the hierarchical approach that we have developed for implementing the transformation. As sketched in Sect. 2.3, we need to build a mapping between *ÆMILIA* elements and the QNBES depicted in Fig. 1. Due to the notational gap between these two modeling languages, in the mapping implementation we have followed a bottom-up approach that starts from small-grained *ÆMILIA* elements and ends up to assemblies of QNBES. In particular, this section presents: the *ÆMILIA* action classification, the *ÆMILIA* behavioral pattern classification, the *ÆMILIA* pattern combination rules to make QNBES, and the connectivity rules for QNBES.

For the sake of readability, in the remainder of this section actions are *italicized*, behavioral patterns are *typewritten*, and QNBES are **bolded**.

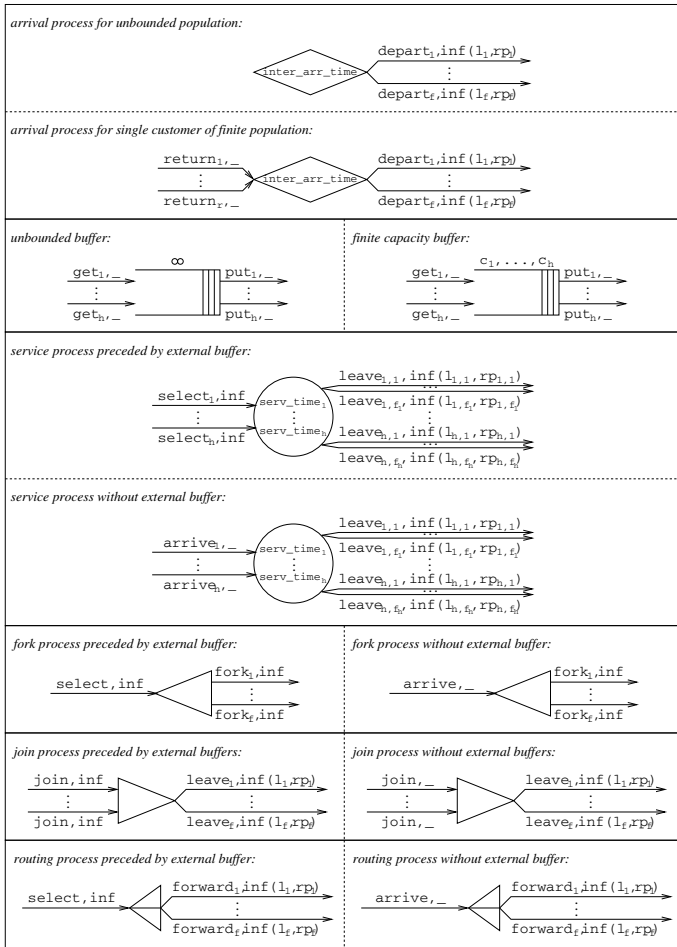


Fig. 1. Queuing network basic elements

Action Classification. In Table 2, the classification of *ÆMILIA* actions is illustrated. Actions are placed on the table rows. The first column is used to partition the rows into three groups of actions, which are: input interactions, output interactions, and internal actions. The other columns represent respectively: the name given to the action, the action duration (i.e., exponential, immediate, passive), the connection multiplicity of the action in the case that it is an interaction (i.e., uni, and, or), and the QNBES that need such an action within their behavioral description (see Fig. 1).

For example, the first row of Table 2 specifies that a passive input uni-interaction might represent a *return* action in a **Single Client Arrival Process** of a queuing network. Similarly, an immediate internal action named *pre-exit* can belong to many different QNBES such as **Arrival** and **Service Processes**.

Table 2. ÆMILIA action classification

	Action Name	Action Duration	Action Multiplicity	QNBEs
Input Interactions	<i>return</i> <i>get</i> <i>select</i>	passive passive immediate	uni uni, or uni	Single Client Arrival Process (Finite or Infinte Capacity) Buffer Buffered Service Process, Buffered Fork Process, Buffered Routing Process Unbuffered Service Process, Unbuffered Fork Process, Unbuffered Routing Process (Buffered or Unbuffered) Join Process
	<i>arrive</i>	passive	uni, or	
	<i>join</i>	immediate	and	
Output Interactions	<i>exit</i>	passive, immediate	uni, or	(Single Client or Infinite) Arrival Process, (Buffered or Unbuffered) Service Process, (Buffered or Unbuffered) Join Process, (Buffered or Unbuffered) Routing Process (Finite or Infinte Capacity) Buffer (Buffered and Unbuffered) Fork Process
	<i>put</i> <i>fork</i>	passive immediate	uni, or and	
Internal Actions	<i>pre-exit</i>	immediate	N/A	(Single Client or Infinite) Arrival Process, (Buffered or Unbuffered) Service Process, (Buffered or Unbuffered) Join Process, (Buffered or Unbuffered) Routing Process (Single Client or Infinite) Arrival Process, (Buffered or Unbuffered) Service Process (Single Client or Infinite) Arrival Process, (Buffered or Unbuffered) Service Process
	<i>exp-phase</i>	exponential	N/A	
	<i>pre-phase</i>	immediate	N/A	

This classification helps to restrict the focus on the 11 actions that have been listed in Table 2. Behavioral patterns of interest for the transformation are built only using these actions. Therefore, such actions represent the alphabet to build words, which are the behavioral patterns introduced in next paragraph to model QNBE behaviors.

Behavioral Pattern Classification. Following our bottom-up approach, we build up on the actions classified in Table 2 to obtain patterns that typically describe (partial) behaviors of QNBEs. The result is illustrated in Table 3, where we have identified 8 behavioral patterns. The table has an organization similar to the one of Table 2. Behavioral patterns are on the table rows and they are grouped into input, output, and internal behaviors. A name is assigned to each pattern, then the pattern is described in the third column of the table, and the QNBEs where such pattern occurs are listed in the fourth column. Parameters are associated with names of patterns that depend on their values (e.g., `uncond-get(i, n)`).

For example, the sixth row of Table 3 (i.e., the first output behavior) specifies that an `exit` behavior describes a job exiting a QNBE and routing somewhere else. Such a behavior can manifest itself in two ways: either as a set of unconditioned alternative behaviors, where each behavior has an `exit` action, or as a single `exit` action.

These behavioral patterns represent the words that can be used to build sentences representing QNBEs, as it will be illustrated in the next paragraph.

Combination Rules for Behavioral Patterns. The last step to obtain the mapping illustrated in Fig. 1 between ÆMILIA constructs and QNBEs is accomplished by introducing rules to combine the previously identified behavioral patterns into QNBEs¹. When parsing an ÆMILIA description, the ÆMILIA_to_QN component of Fig. 1 looks

¹ All behaviors occurring in ÆMILIA descriptions are assumed to be tail recursive.

Table 3. ÆMILIA behavioral pattern classification

	Pattern Name	Pattern Description	QNBEs
Input Behaviors	return	The return of a job can be described in two different ways: (i) two or more alternative processes, each made of an unconditioned <i>return</i> action; (ii) a process represented by a <i>return</i> action	Single Client Arrival Process
	uncond-get(<i>i, n</i>)	An unconditioned get process for the <i>i</i> -th class of clients is made of an unconditioned <i>get</i> action, and a behavioral call with actual parameters pa_1, pa_2, \dots, pa_n that satisfy the following constraints: $pa_j = p_j + 1$ for $j = i$ and $pa_j = p_j$ for $j \neq i$, where p_j is the current number of clients of <i>j</i> -th class	Infinite Capacity Buffer
	cond-get(<i>i, n, N_i</i>)	A conditioned get process for the <i>i</i> -th class of clients is made of: the condition $p_i < N_i$ (where N_i is the buffer capacity for the <i>i</i> -th class of clients), a <i>get</i> action, and a behavioral call with actual parameters pa_1, pa_2, \dots, pa_n that satisfy the following constraints: $pa_j = p_j + 1$ for $j = i$ and $pa_j = p_j$ for $j \neq i$, where p_j is the current number of clients of <i>j</i> -th class	Finite Capacity Buffer
	select	A selection behavior is made of one or more alternative processes, where each process starts with an unconditioned <i>select</i> action	Buffered Service Process, Buffered Fork Process, Buffered Routing Process
	arrive	An arrival behavior is made of one or more alternative processes, where each process starts with an unconditioned <i>arrive</i> action	Unbuffered Service Process, Unbuffered Fork Process, Unbuffered Routing Process
Output Behaviors	exit	A job exit (and routing) can be described in two different ways: (i) two or more alternative processes, each made of an unconditioned <i>pre-exit</i> action followed by an <i>exit</i> action; (ii) a process represented by an <i>exit</i> action	(Buffered or Unbuffered) Service Process, (Buffered or Unbuffered) Join Process, (Buffered or Unbuffered) Routing Process, (Single Client or Infinite) Arrival Process
	put(<i>i, n</i>)	A put process for the <i>i</i> -th class of clients is made of: the condition $p_i > 0$, a <i>put</i> action, and a behavioral call with actual parameters pa_1, pa_2, \dots, pa_n that satisfy the following constraints: $pa_j = p_j - 1$ for $j = i$ and $pa_j = p_j$ for $j \neq i$, where p_j is the current number of clients of <i>j</i> -th class	(Finite or Infinite Capacity) Buffer
Internal Behaviors	phase	The behavior of a phase-type distribution is an arbitrary combination of <i>exp-phase</i> and <i>pre-phase</i> actions that determine a set of alternatives, where each alternative terminates with a non-phase behavior	(Single Client or Infinite) Arrival Process, (Buffered or Unbuffered) Service Process

for such combinations of behavioral patterns in order to identify QNBEs within an ÆMILIA description and to generate them.

These rules are defined in Table 4. The order of QNBEs in the table is the same as the one in Fig. 1. Each row in Table 4 represents a QNBE, where the second column provides the combination rules for behavioral patterns that define the QNBE behavior, whereas the third column represents additional assumptions that have to be verified before generating the QNBE itself. Note that where behavioral patterns have been numbered (e.g., **Infinite Arrival Process**), it means that a simple sequencing rule has to be applied to those patterns. In some cases, such as **Buffered Fork Process**, rules have to be applied to simple actions beside behavioral patterns.

For example, a **Single Client Arrival Process** is defined as a sequence of three behavioral patterns (i.e., *phase*, *exit*, and *return*), with the additional conditions

Table 4. Combining behavioral patterns into QNBES

QNBE	Combination rules of behavioral patterns	Additional assumptions
Infinite Arrival Process	1. phase 2. exit	a. output interactions must only be <i>exit</i> b. no input interactions
Single Client Arrival Process	1. phase 2. exit 3. return	a. output interactions must only be <i>exit</i> b. input interactions must only be <i>return</i>
Infinite Capacity Buffer	· Parameters p_1, p_2, \dots, p_n have to be declared · Parameters have to be initialized to non-negative numbers · $2n$ alternative processes have to be defined: n <i>uncond-get</i> (i, n) patterns and n <i>put</i> (i, n) patterns	a. output interactions must only be <i>put</i> (with different names) b. input interactions must only be <i>get</i> (with different names)
Finite Capacity Buffer	· Parameters p_1, p_2, \dots, p_n have to be declared · Parameters have to be initialized to non-negative numbers and have to be all declared as intervals of integers · Each parameter p_i has to fall within the $[0, N_i]$ interval · $2n$ alternative processes have to be defined: n <i>cond-get</i> (i, n) patterns and n <i>put</i> (i, n) patterns	a. output interactions must only be <i>put</i> (with different names) b. input interactions must only be <i>get</i> (with different names)
Buffered Service Process	1. select 2. phase 3. exit	a. output interactions must only be <i>exit</i> b. input interactions must only be <i>select</i>
Unbuffered Service Process	1. arrive 2. phase 3. exit	a. output interactions must only be <i>exit</i> b. input interactions must only be <i>arrive</i>
Buffered Fork Process	1. <i>select</i> 2. <i>fork</i>	a. the only output interaction is the <i>fork</i> action b. the only input interaction is the <i>select</i> action
Unbuffered Fork Process	1. <i>arrive</i> 2. <i>fork</i>	a. the only output interaction is the <i>fork</i> action b. the only input interaction is the <i>arrive</i> action
Buffered or Unbuffered Join Process	1. <i>join</i> 2. exit	a. output interactions must only be <i>exit</i> b. the only input interaction is the <i>join</i> action
Buffered Routing Process	1. <i>select</i> 2. exit	a. output interactions must only be <i>exit</i> b. the only input interaction is the <i>select</i> action
Unbuffered Routing Process	1. <i>arrive</i> 2. exit	a. output interactions must only be <i>exit</i> b. the only input interaction is the <i>arrive</i> action

that the process must only have *exit* output interactions and *return* input interactions. As another example, a **Join Process** is defined as a sequence of a *join* action and an *exit* behavioral pattern, with the additional conditions that the only input interaction is the *join* action and the process must only have *exit* output interactions.

Connectivity Rules for Queueing Network Basic Elements. Finally, we introduce several connectivity rules that allow QNBES to be assembled in semantically valid queueing network models:

- An arrival process can be followed only by a service or fork process, possibly preceded by a buffer.
- A buffer can be followed only by a service, fork, join, or routing process.
- A service process can be followed by any QNBE.
- A fork process can be followed only by a service process or another fork process, possibly preceded by a buffer.

- A join process can be followed by any QNBE.
- A routing process can be followed by any QNBE.

3 The Eclipse Plugin `ÆMILIA_to_QN`

In order to enable the application of `PERFSEL`, we have developed `ÆMILIA_to_QN`, a Java-coded Eclipse plugin for transforming `ÆMILIA` descriptions into queueing networks. As shown in Sect. 2, the model transformations realized by `ÆMILIA_to_QN` rely on two queue-driven classifications – one for actions and one for behavioral patterns built from actions and process algebraic operators that can occur in `ÆMILIA` descriptions, respectively – which hierarchically formalize most of the syntactical restrictions of [2,1]. These two classifications are then complemented by a number of rules establishing which combinations of behavioral patterns result in QNBEs (combination rules) and, in turn, how QNBEs should be connected to each other in order to yield well-formed queueing networks (connectivity rules).

Given an `ÆMILIA` description, `ÆMILIA_to_QN` parses the behavioral part of the `ÆMILIA` representation of each AEI to search for occurrences of the previously identified action classes and queue-like behavioral patterns. Whenever this search is successful on all AEIs and the combination rules are respected, then `ÆMILIA_to_QN` transforms each AEI into the corresponding QNBE. Afterwards, `ÆMILIA_to_QN` checks the topological part of the `ÆMILIA` description for compliance with the previously established connectivity rules of QNBEs. If this check succeeds too, `ÆMILIA_to_QN` transforms the entire `ÆMILIA` description into a queueing network model.

The queueing network models produced by `ÆMILIA_to_QN` are stored in PMIF format [13]. The reason is that this is an XML schema that acts as an interchange format and hence makes it possible to pass those models as input to queueing network tools.

4 The Architecture of `TWOEAGLES`

`ÆMILIA_to_QN` can be launched from within `TWOEAGLES`. This is a new version of `TwoTowers` [5] – a software tool for the functional verification, performance evaluation, and security analysis of software architectures described with `ÆMILIA` – which is entirely integrated in the Eclipse framework.

The software architecture of `TWOEAGLES` is depicted in Fig. 2, where the provided interfaces of components are represented by lollipops whereas required interfaces by dashed arrows. In the box labeled with `TWOEAGLES`, only the components that strictly belong to the tool have been included, which are: `ÆMILIA_to_QN`, `TT_GUI`, and `TwoTowers`. The remaining components of Fig. 2 are either external tools (i.e., `NuSMV` and `Eclipse`) or in-house built components that support the `TWOEAGLES` task but do not belong to the tool (i.e., `QN_Editor` and `QN_Solver`).

In order to embed `TwoTowers` in `Eclipse`, `TWOEAGLES` relies on a new graphical user interface for `TwoTowers` – the `TT_GUI` component in Fig. 2 – which has been developed as an Eclipse plugin. `TT_GUI` tailors the Eclipse environment to offer all the original `TwoTowers` functionalities to users, along with the model transformation introduced in this paper. This wrapping of `TwoTowers` has allowed its implementation to be

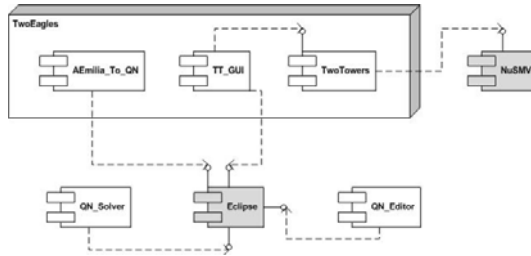


Fig. 2. Composition and environment of TWOEAGLES

kept basically unchanged – including the use of the external model checker NuSMV [6] – whereas its interfaces can be invoked, as they are, through the TT_GUI component.

In addition to TT_GUI, there are other three Eclipse plugins in Fig. 2. The first one, *ÆMILIA*_to_QN, has already been described in the previous section. Since it relies on PMIF, it acts as a bridge between TwoTowers on one side and the next two plugins on the other side. The second one, QN_Editor, allows PMIF-based queueing networks to be imported and edited in Eclipse and supports their graphical visualization. The third one, QN_Solver, is the client side of a web service called Weasel [15], which can be exploited to invoke several existing queueing network solvers.

A typical scenario for the architecture in Fig. 2 is the following. TWOEAGLES starts and TT_GUI, within Eclipse, is ready to accept user commands. For example, the user opens the *ÆMILIA* editor, which is part of TwoTowers, and enters an *ÆMILIA* description. The user may then run any (functional, security, or performance) analysis technique provided by the original TwoTowers release, including the NuSMV model checker. In addition, due to the extension presented in this paper, the user may decide to invoke a model transformation that generates a queueing network from the *ÆMILIA* description (i.e., the *ÆMILIA*_to_QN component in Fig. 2). The output queueing network, which can also be modified with the QN_Editor, is represented in PMIF and can be rendered (i) in a textual XML format through the standard Eclipse XML editor or (ii) in a graphical format through the QN_Editor component shown in Fig. 2. The latter is able to import and export queueing networks in PMIF format and to graphically represent them within Eclipse. Finally, the user can invoke the queueing network solver (i.e., QN_Solver in Fig. 2), which is a web service able to invoke different existing solvers. The solution results are represented in the standard Eclipse text editor.

5 TWOEAGLES at Work: An Automated Teller Machine

In this section, we illustrate TWOEAGLES at work on an automated teller machine (ATM), a system made of a certain number of distributed client terminals from where it is possible to require services to a central server. Common types of service requests are: withdrawal, deposit, and balance. Terminals only enable clients to perform I/O operations, whereas large part of computation is performed on the central server. The experiments that we illustrate on this example are aimed at: (i) showing the tool usability, (ii) validating the transformation from *ÆMILIA* descriptions to queueing network models

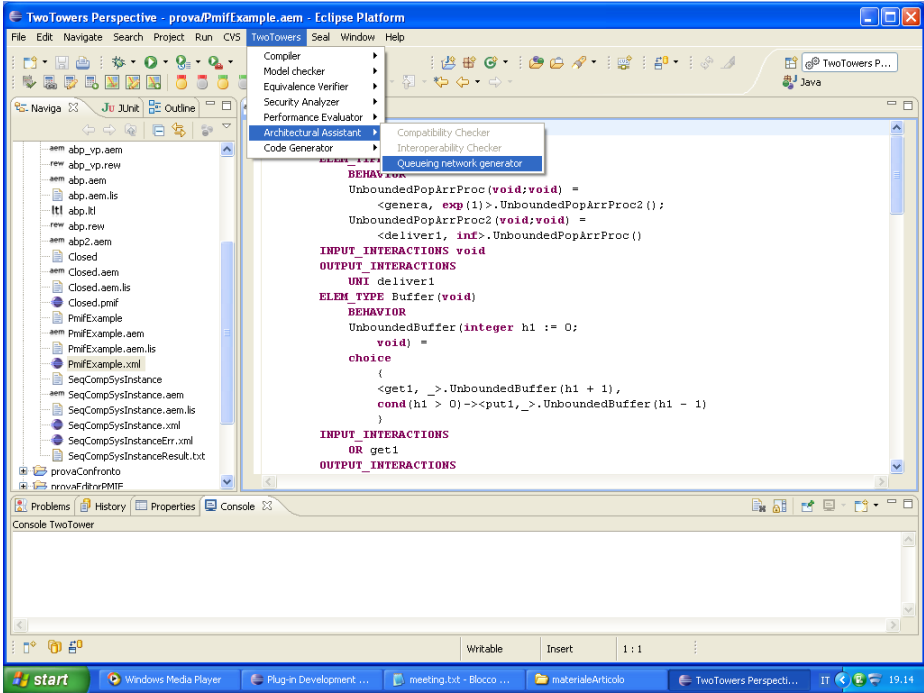


Fig. 3. Eclipse user interface for TWOEAGLES

on the basis of numerical results, and (iii) showing the larger scalability of queueing network solvers with respect to TwoTowers traditional performance evaluator.

The application of $\mathcal{A}EMILIA_{to_QN}$ (see Fig. 3) to the $\mathcal{A}EMILIA$ description of the ATM system (which is not shown here due to lack of space) results in a queueing network formed by four QNBES (corresponding to as many AEIs): *ThinkDevice*, which is the workload generator for the whole network, together with *CPU*, *DISKS*, and *VIDEO*, whose service times are 0.5 ms, 0.5 ms, and 1 ms, respectively (these values are only approximations of real scenarios, as we are more interested in the validation of the transformation and the analysis of pros and cons of our approach, rather than in the numbers themselves). Jobs originated from *ThinkDevice* are delivered to *CPU*. On the basis of interaction rates, *CPU* decides whether sending jobs to *DISKS* and/or *VIDEO*. Thereafter, jobs are sent back to *ThinkDevice*. Being a closed queueing network, the parameters that drive our performance analysis are: the number N of client terminals and the thinking time Z of each client.

In Fig. 4, we have reported the throughput (on the left) and the utilization (on the right) that we have obtained for the main ATM devices, which are *CPU* and *DISKS*, while varying the number of clients N in the system, with a fixed thinking time of $Z = 1$ s. Four curves are shown because for each device we have represented both the values obtained with the TwoTowers performance solver and the ones obtained with the external queueing network solver after the $\mathcal{A}EMILIA$ description has been transformed into a queueing network. As can be seen, the two solvers obtain exactly the same

numerical results for both considered devices, and this supports the correctness of the transformation of the *ÆMILIA* description into the queueing network model. However, the TwoTowers solver, whose results are labeled as TwoTowers in the figure, is unable to solve models with more than 7 clients. This is due to the state space explosion phenomenon encountered when the solver handles the continuous-time Markov chain model. In contrast, the queueing network solver, whose results are labeled as PMVA in the figure, is able to solve larger models in few seconds. This is due to the product form [10] of the resulting queueing network model, which allows polynomial-time solution algorithms such as Mean Value Analysis (MVA) to be applied.

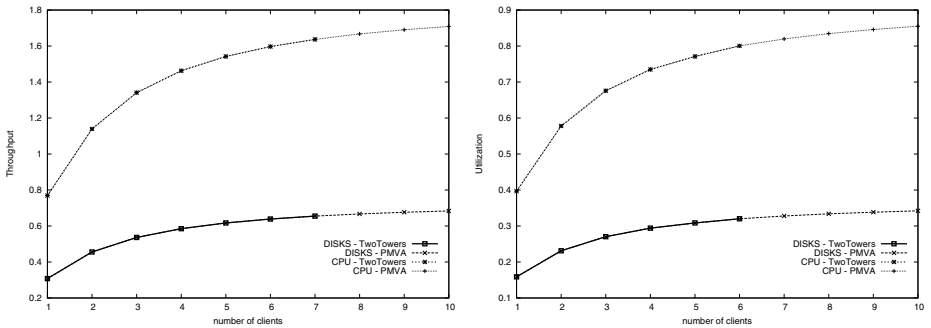


Fig. 4. ATM throughput (left) and utilization (right)

6 Conclusions and Future Work

In this paper, we have presented a tool integrated into Eclipse that allows *ÆMILIA*-based architectural descriptions to be transformed into queueing network models and hence supports the PERFSEL methodology of [2,1]. As shown by the ATM example, the tool TWOEAGLES improves on TwoTowers because the possibility of exploiting queueing network solvers makes the performance evaluation process faster and applicable to larger software architectures with respect to continuous-time Markov chain solvers.

Many approaches have been introduced in the last decade to transform architectural models into performance models [3], but very few of them have been implemented in working tools and rely on structured models like queueing networks. Moreover, most implementations are based on UML, whereas in TWOEAGLES we consider a fully fledged, formally defined architectural description language as source notation.

With regard to future work, we intend to strengthen the transformation implemented in TWOEAGLES by moving from a general-purpose programming language like Java to model transformation languages like ATL [9] and QVT [12]. Moreover, we would like to investigate whether results relating stochastic process algebras and queueing networks [8,7,14] can be exploited in our architectural framework.

Acknowledgment. Work funded by MIUR-PRIN project *PaCo – Performability-Aware Computing: Logics, Models, and Languages*.

References

1. Aldini, A., Bernardo, M., Corradini, F.: A Process Algebraic Approach to Software Architecture Design. Springer, Heidelberg (2010)
2. Balsamo, S., Bernardo, M., Simeoni, M.: Performance Evaluation at the Software Architecture Level. In: Bernardo, M., Inverardi, P. (eds.) SFM 2003. LNCS, vol. 2804, pp. 207–258. Springer, Heidelberg (2003)
3. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. *IEEE Trans. on Software Engineering* 30, 295–310 (2004)
4. Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, G.: Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *JACM* 22, 248–260 (1975)
5. Bernardo, M.: TwoTowers 5.1 User Manual (2006), <http://www.sti.uniurb.it/bernardo/twotowers/>
6. Cavada, R., Cimatti, A., Olivetti, E., Pistore, M., Roveri, M.: NuSMV 2.1 User Manual (2002)
7. Harrison, P.G.: Compositional Reversed Markov Processes, with Applications to G-Networks. *Performance Evaluation* 57, 379–408 (2004)
8. Hillston, J., Thomas, N.: Product Form Solution for a Class of PEPA Models. *Performance Evaluation* 35, 171–192 (1999)
9. Jouault, F., Allilaire, F., Bezivin, J., Kurtev, I.: ATL: A Model Transformation Tool. *Science of Computer Programming* 72, 31–39 (2008)
10. Kleinrock, L.: *Queueing Systems*. John Wiley & Sons, Chichester (1975)
11. Lazowska, E.D., Zahorjan, J., Scott Graham, G., Sevcik, K.C.: *Quantitative System Performance: Computer System Analysis Using Queuing Network Models*. Prentice Hall, Englewood Cliffs (1984)
12. OMG, Query/View/Transformation, formal/08-04-03
13. Smith, C.U., Llado, C.M.: Performance Model Interchange Format (PMIF 2.0): XML Definition. In: Proc. of QEST 2004, pp. 38–47. IEEE-CS Press, Los Alamitos (2004)
14. Thomas, N., Zhao, Y.: Mean Value Analysis for a Class of PEPA Models. In: Bradley, J.T. (ed.) EPEW 2009. LNCS, vol. 5652, pp. 59–72. Springer, Heidelberg (2009)
15. Zallocco, S.: Web service for Analyzing queueing networks with multiple solvers (2006), <http://sealabtools.di.univaq.it/weasel/>

A Tool Suite for Modelling Spatial Interdependencies of Distributed Systems with Markovian Agents

Davide Cerotti¹, Enrico Barbierato², and Marco Gribaudo³

¹ Dipartimento di Informatica, Università di Torino, Torino, Italy
cerotti@di.unito.it

² Dip. Informatica, Università Piemonte Orientale, Alessandria, Italy
enrico.barbierato@mf.n.unipmn.it

³ Dip. Elettronica ed Informazione, Politecnico di Milano, Italy
gribaudo@elet.polimi.it

Abstract. Distributed systems are characterized by a large number of similar interconnected objects that cooperate by exchanging messages. Practical application of such systems can be found in computer systems, sensor networks, and in particular in critical infrastructures. Though formalisms like Markovian Agents provide a formal support to describe these systems and evaluate related performance indices, very few tools are currently available to define models in such languages, moreover they do not provide generally specific functionalities to ease the definition of the locations of the interacting components. This paper presents a prototype tool suite capable of supporting the study of the number of hops and the transmission delay in a critical infrastructure.

1 Introduction

The 2006 European Programme for Critical Infrastructure Protection (EPCIP) stated that “The security and economy of the European Union as well as the well-being of its citizens depends on certain infrastructure and the services they provide. The destruction or disruption of infrastructure providing key services could entail the loss of lives, the loss of property, a collapse of public confidence and moral in the EU” [12].

Among the eight critical infrastructures identified were telecommunications, electrical power, gas and oil storage, transportation and water supply. These infrastructures share some common aspects: i) their components are controlled by communication networks and ii) their physical infrastructure can be represented by graphs. For instance in a power grid, telecontrol buildings, like the SCADA control centers, or the electrical stations can be represented as node of a graph. Instead the links can model the communication channel or the high-voltage lines over which the current travels.

The representation of networks as a graphs pervades the study of critical infrastructures and more generally of distributed systems, in [24] the authors propose a graph-based model to represent and to analyze interdependencies among

critical infrastructures; in [20] structural analysis of graphs is performed to support disaster vulnerability assessment, moreover in [3,5,6] the reliability of a power grid controlled by a SCADA communication network was computed.

The number of hops and transmission delays are well-known concepts in data-communication networks. These parameters have proved to be useful as approximated performance metrics in different application contexts. The occurrence of a critical event (e.g. fault in an electrical line) in a power grid, its location or its distance from the station, as well as the total delay from the instant of the occurrence need to be signaled to a central station. When a critical event is detected, the alarm signal is propagated along a path of intermediate nodes. Assuming that the distances between the nodes are known or estimated, the number of hops provides a measure of the distance between the critical event and the central station.

Though formalisms like *Markovian Agents* (MAs) [16] provide a support to describe these systems and evaluate related performance indices, very few tools are currently available to define models in such languages, moreover they do not provide generally specific functionalities to ease the definition of the locations of the interacting components. In Section 4 a new tool for the definition of graph-based interconnection network is presented.

Section 5 shows a formal stochastic model, based on MAs, to compute the number of hops between a source and a destination in addition to the transmission time in a multi-hop routing. More precisely, the model represents a system of interconnected nodes where each node is represented by a MA that can transmit messages directly or through intermediate nodes to a specific destination. Transmitted messages carry the number of hops, incrementing this value at each step so that the mean hop count and the mean time needed by a message to reach its destination are computed.

2 Related Works

Beside the work on MAs, that will be considered in depth in Section 3, several other spatial models have been introduced in the literature. One of the more mature is Cellular automata (CAs) [21]. They are simple models of a spatially extended decentralized systems composed by a set of individual entities (cells). The communication between cells is limited to local interaction, and each individual can be in a state which changes over time depending on the states of its local neighbors and on simple rules. In the area of performance evaluation, applications of CA can be found in biomedical [23], ecology [15,13] and geology [19]. All the models of this sort are usually studied by running several simulations of the CA, starting from different initial states and computing the desired performance indexes.

More recent works considering spatial models are for example *Spatial Process Algebra* (SPA) [14] where *Locations* are added to take into account spatial dependencies. In particular spatial concepts are added to the stochastic process algebra PEPA by allowing named locations to appear in process terms and on the labels of the transitions between them.

The need of automatic tools to include spatial aspects in performance models is considered in [1]. The authors derive a Generalised Stochastic Petri Net (GSPN) model from high-precision location tracking data traces, using clustering techniques. In particular, GSPNs places are used to model locations where an object spends a large amount of time, and timed transitions are used to model the movement among the locations.

3 Markovian Agents

MA models consist of a set of agents positioned into a space. The behavior of each agent is described by a continuous-time Markov chain (CTMC) with two types of transitions: *local transitions* that model the internal features of the MA, and *induced transitions* that account for interaction with other MAs. During local transitions, an MA can send messages to other MAs. The *perception function* $u(\cdot)$ regulates the propagation of messages, taking into account the agent position in the space, the message routing policy, and the transmittance properties of the medium.

MAs are scattered over a space \mathcal{V} , which can correspond either to a discrete number of locations, or to continuous n -dimensional space. Agents can be grouped in classes, and messages divided into different types. Formally a *Multiple Agent Class, Multiple Message Type Markovian Agents Model (MAM)* is a tuple:

$$MAM = \{\mathcal{C}, \mathcal{M}, \mathcal{V}, \mathcal{U}, \mathcal{R}\}, \quad (1)$$

where $\mathcal{C} = \{1 \dots C\}$ is the set of agent classes; $\mathcal{M} = \{1 \dots M\}$ is the set of message types; \mathcal{V} is the space (discrete or continuous) where Markovian Agents are spread; $\mathcal{U} = \{u_1(\cdot) \dots u_M(\cdot)\}$ is a set of M perception functions (one for each message type); $\mathcal{R} = \{\xi^1(\cdot) \dots \xi^C(\cdot)\}$ defines the density of agents, where each component $\xi^c(\mathbf{v})$ accounts either for the number or the density of class c agents in position $\mathbf{v} \in \mathcal{V}$.

Each agent MA^c of class c is defined by the tuple:

$$MA^c = \{\mathbf{Q}^c(\mathbf{v}), \mathbf{\Lambda}^c(\mathbf{v}), \boldsymbol{\pi}_0^c(\mathbf{v}), \mathbf{G}^c(m, \mathbf{v}), \mathbf{A}^c(m, \mathbf{v})\}. \quad (2)$$

$\mathbf{Q}^c(\mathbf{v}) = [q_{ij}^c(\mathbf{v})]$ is the infinitesimal generator matrix of the CTMC that models the local behavior of an agent of class c . An element $q_{ij}^c(\mathbf{v})$ represents the transition rate from state i to state j (with $q_{ii}^c(\mathbf{v}) = -\sum_{j \neq i} q_{ij}^c(\mathbf{v})$), and $\mathbf{\Lambda}^c(\mathbf{v}) = [\lambda_i^c(\mathbf{v})]$, is a vector containing the rates of *self-jumps* (i.e. the rate at which the Markov chain re-enters the same state). Using self-jumps, an agent can continuously send messages while remaining in the same state. $\mathbf{G}^c(m, \mathbf{v}) = [g_{ij}^c(m, \mathbf{v})]$ and $\mathbf{A}^c(m, \mathbf{v}) = [a_{ij}^c(m, \mathbf{v})]$ represent respectively the probability that an agent of class c generates a message of type m during a jump from state i to state j , and the probability that an agent of class c accepts a message of type m in state i , performing an immediate jump to state j . $\boldsymbol{\pi}_0^c(\mathbf{v})$ represents the initial state distribution. All the previous quantities depends on the location \mathbf{v} : this allows the agents to modify the rate at which they perform their activities as a function of the position in which they are located.

The perception function is defined as $u_m : \mathcal{V} \times \mathcal{C} \times \mathbb{N} \times \mathcal{V} \times \mathcal{C} \times \mathbb{N} \rightarrow \mathbb{R}^+$, and $u_m(\mathbf{v}, c, i, \mathbf{v}', c', i')$ represents the probability that an agent of class c , in position \mathbf{v} , and in state i , perceives a message m generated by an agent of class c' in position \mathbf{v}' in state i' .

MAs models can be analyzed solving a set of differential equations that compute the density $\rho_i^c(t, \mathbf{v})$ of agents in class c , in state i in position \mathbf{v} at time t . In [8], a prototype tool, called *MASolver* from now on, was developed to analyse *MAs* models using conventional discretization techniques for both time and space. Since *MAs* models have no strict form of synchronisation, the whole state-space of the model is not built, avoiding the well-known state explosion problem. To consider the interaction among agents, we resort to an approximate technique based on mean-field theory [22]. The behavior of each agent depends on both its local behavior, represented by the infinitesimal generator matrix $\mathbf{Q}^c(\mathbf{v})$, and the behavior induced by the interactions with other *MAs*, computed as a mean-field. The whole behavior is represented by an infinitesimal generator matrix $\mathbf{K}^c(t, \mathbf{v})$ which changes in time and depends on the class and the position of the agent.

The evolution of the entire model can be studied by solving $\forall \mathbf{v}, c$ the following differential equations:

$$\rho^c(0, \mathbf{v}) = \xi^c(\mathbf{v})\pi_0^c(\mathbf{v}) \quad (3)$$

$$\frac{d\rho^c(t, \mathbf{v})}{dt} = \rho^c(t, \mathbf{v})\mathbf{K}^c(t, \mathbf{v}). \quad (4)$$

where $\rho^c(t, \mathbf{v}) = [\rho_i^c(t, \mathbf{v})]$.

The computation of the matrix $\mathbf{K}^c(t, \mathbf{v})$ represents the most expensive step in the solution algorithm, because it considers all the possible interactions among agents and messages, in every possible location. However, in most practical applications, the definition of the perception function confines the interaction of each *MA* to a limited number of neighboring *MAs*, significantly reducing the complexity of this step. Please refer to [8,7] for a more detailed description of Markovian Agents and the related analysis techniques.

3.1 Algorithmic Generation of Spatial Dependencies

The *MA* have been enhanced in order to model the geographical location of agents over the space. The modelling of complex spatial behaviors is based on a set of matrices that define the local agent behavior and the perception function depending on the location \mathbf{v} .

This approach requires however a larger set of parameters that needs to be specified: without appropriate tools or techniques, it is not possible to exploit such features. Another aspect to consider concerns the kind of topology. In case of very simple or special topologies, the spatial dependencies can be determined by using simple algorithms.

For example, in [11] a sensor network with a circular topology and the sink in the center were considered. The perception function was computed to take

into account a specific minimal number of hops routing policy. In [7] instead, a protocol based on ant colony optimization was studied. In that case the matrices were constants, and the only spatial dependency was on the location of the sinks. Sinks however were very limited in number, and their position were specified by manually indicating their coordinates. In [4] the motion of agents in a tunnel was considered. In that case, the parameters could be computed starting from broad characteristics like the curvature of the track.

In more complex cases, specific tools are required to automatically infer the model parameters from measured data, or to allow the modeler to design freely the interactions among agents.

3.2 The Image-Based Tools

In previous works about MAs, parameters were obtained using *Image-based tools*. These tools create the spatial dependencies starting from a bit-mapped image that color-codes the value of the different parameters. They are suitable for studying models defined over a geographical areas where parameters can be extrapolated from already available maps.

For example in [9], the propagation of a disaster such as an earthquake was studied starting from maps of the considered region such as the one presented in Figure 1. In that particular case, an application capable of performing discretization over a circular or elliptic grid was used. Figure 2 shows some screen-shots of GUI of such application. The application is capable of specifying both the center of the discretization that corresponds to the epicenter of the propagation, a semi-axis and a direction in case of elliptical propagation. The output of the tool consists in a list of cells, each characterized by its own parameters derived from the associated map.

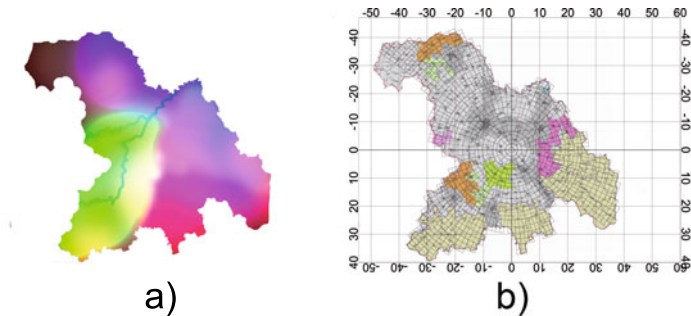


Fig. 1. Color coded maps of the model parameters: a) the source data; b) the discretized version

In [10], the propagation of fire over a region was considered. In that case a simple application that extracts the agent densities and the fire-extinction rate from the RGB channels of a satellite image, and that allows the user to specify the wind direction sampled by the meteorological stations was developed. Some

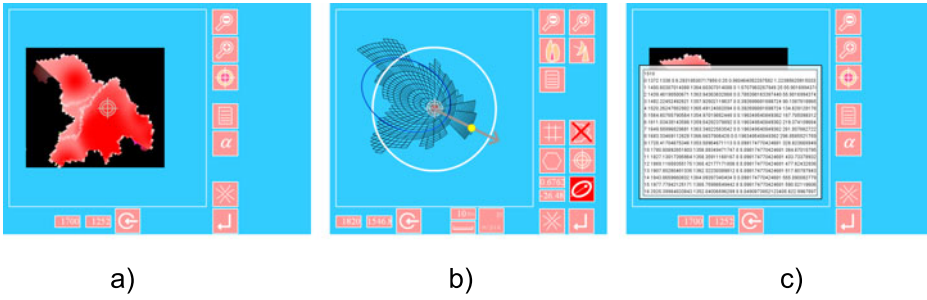


Fig. 2. The circular/elliptic discretization Tool screen-shot: a) importing the terrain data; b) defining the center and the form of the discretization; c) the tool output

screen-shots of the tool interface are shown in Figure 3. The tool divides the region into square cells of equal dimensions such that the burning properties and the wind direction inside each cell can be considered constant. Also in this case the tool produces a list of cells, each characterized by the property of the corresponding area under the map.

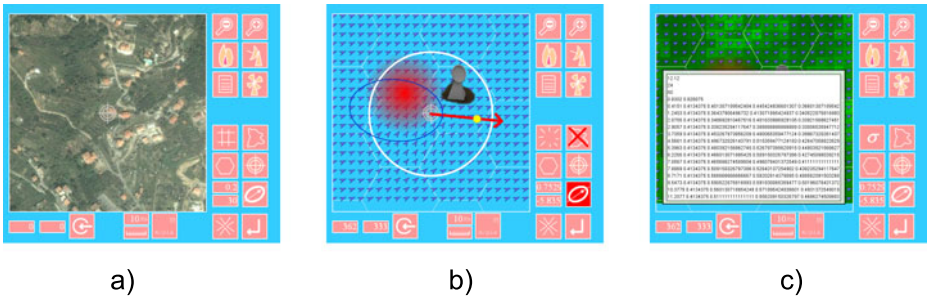


Fig. 3. Tool screen-shot: a) importing the terrain data; b) defining the wind intensity; c) exporting the output values

Both applications are written in *Adobe Flash*, and join with the *MASolver* can be considered as a part of a tool suite developed to allow the specification and analysis of spatial dependencies in MA models.

4 The Graph-Based Tool

In this work a new tool for the Graph-based specification of the interconnection among agents is introduced. This tool allows the user to define visually the interaction among the objects using a graphical user interface, and it is preferable over image-based tools when the number of objects is limited and the interactions among the components can be defined manually.

The Graph-based tool uses the *DrawNet*[17,18] framework as its Graphical User Interface. DrawNet is a framework supporting the design and solution of models represented in any graph-based formalism. It is based on an open architecture and uses an XML-based language to create new (multi)formalisms and model-based (multi)formalisms.

A new DrawNET formalism¹ called **Markovian Agents Routing Graph** has been implemented to define the interconnection graph of stochastic models based on multi-hop networks. The formalism allows the user to edit the following objects: i) **Location Nodes**, ii) the **Message Service Rates** and iii) a set of **Parameters** used to define the solution. Locations are used to define the positions of the agents in the model, and message service rates are used to specify the interconnections among the agents.

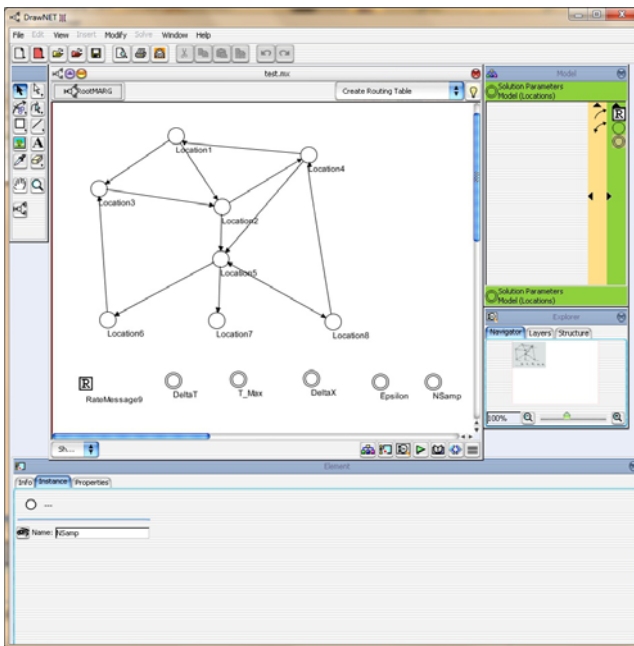


Fig. 4. A Markovian Agents Routing Graph model in DrawNet

An example of a model-based **Markovian Agents Routing Graph** is shown in Fig.4. The built model can be exported to a plain ASCII file and made available for further processing by other tools including the *MASolver*. The exported data includes the **Topology Incident Matrix**, the **Message Service Rates**, the **solution Parameters** and a **Distance Matrix**. The **Topology Incident Matrix** is a matrix M where, for each pair of indexes (i, j) , $M[i][j]$ is set to 1 if there is

¹ DrawNET addresses sets of graphical primitives that can be used to design a model as *formalisms*


```

#####
### Legend
# Location8 none
# Location7 none
# Location6 none
# Location5 none
# Location4 none
# Location3 none
# Location2 none
# Location1 none
#####
### Rate messages
1.0
#####
### Topology incidence matrix
0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
1.0 1.0 1.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 1.0 0.0 0.0 0.0 1.0
0.0 0.0 0.0 0.0 0.0 0.0 1.0 0.0
0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0
#####
### Distance matrix
0.0 0.0 0.0 190.2761151589973 249.6096953245206 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 194.4350791395421 0.0 0.0
190.2761151589973 91.19758768739445 189.707466985021982 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 201.8341162185698 0.0 0.0 0.0 197.9898987322333
0.0 0.0 0.0 0.0 0.0 0.0 183.99184764548673 0.0
0.0 0.0 0.0 77.02596964660685 149.37536610833797 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 138.13037319865406 125.0959631642844 0.0
#####
### Solution parameters:DeltaT, NSamp, T_max, DeltaX, Epsilon
#####
0.5
1.0
1.0
1.0
1.0

```

Fig. 5. Output produced by DrawNet

a directed arc from Location_{*i*} to Location_{*j*} (0 otherwise). The **Message Service Rates** and the **solution Parameters** are floating point numbers; finally, the **Distance Matrix** is a matrix **D** where, for each pair of indexes (*i*, *j*), $D[i][j]$ is calculated as the geometric distance between Location_{*i*} and Location_{*j*}. Fig.5 shows an example of output file for the previous model.

5 Computing the Number of Hops in Critical Infrastructures with MA

In this section we provide an application of the Graph-based tool join with the *MASolver* to define and analyse a case study of a simplified fault detection distributed system monitoring an electrical power grid.

In a power grid, the nodes monitor the status of the electrical lines and signal the presence of faults to the central station by sending messages each time they detect a faulty line. These messages are transferred to the central station by means of intermediate nodes according to a multi-hop routing. Moreover, each message includes the number of hops traversed in its path to the sink. In this way the central station can infer the distance from its position to the detected faulty line in terms of number of hops. Each time a node receives messages with different values of number of hops, it forwards only the message carrying the minimum value incremented by one. This is done in order to signal to the central station the minimum number of hops to reach the faulty line. This scenario is

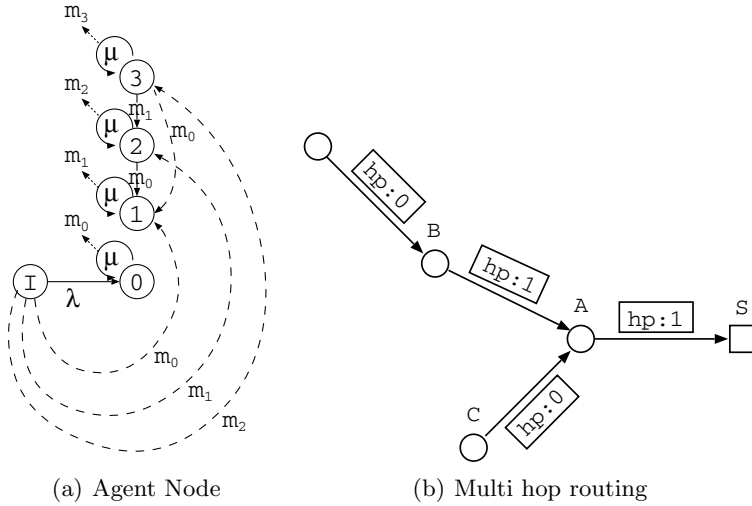


Fig. 6. Agent Node (a), choice of the minimum between different values of hop number (b)

illustrated in Figure 6(b): node *A* receives messages from both nodes *B* and *C* with a value of hops equal to one and zero respectively, and sends to the central station *S* a message where the number of hops is set to one.

Node Agent. Agents belonging to the node class represent the distance of a detected or signalled fault in their state space. It is assumed that the grid is deployed to signal the presence of fault at most at *M* hops from the central station, in such case each node maintains information of the presence of a fault at most at *M* hops. The node agent is therefore characterized by *M* + 2 states, and its state space is defined as $S_n = \{I, 0, 1, \dots, M\}$. Nodes are represented by a single agent MA^n shown in Figure 6(a) for *M* = 3. The meaning of the states is the following:

- I* - is the idle state: the node does not detect fault and it has not received messages signaling the presence of it;
- i* - are the detection states: state 0 means that the node has detected a local fault, state *i*, with $0 < i \leq M$, means that the node was informed by the other ones of the presence of a fault at *i* hops from its position.

The local transition at a rate λ from the idle state to the state 0 indicate the detection of a local fault.

The MA^n can emit and receive *M* + 1 types of messages (m_0, m_1, \dots, m_M) corresponding to the number of hops. The behavior of the MA^n agent at the reception of the messages is the following:

- m_i - the signal message; a message m_i is sent with probability 1 at the rate μ when the MA^n sojourns in state *i* (shown as a self-loop in Figure 6(a)); when

the MA^n is in states j and a message of type m_i is perceived, it induces a transition to state $i + 1$ with probability 1 only if $j = I$ or $i + 1 < j$, it is ignored otherwise. In such a way a node agent a , which is informed of a faulty line at i hops from its position, transfers one hop further this information at his neighbor node b which jumps to state $i + 1$. Such information is ignored if node b is already informed of a closer fault.

The nodes of the grid are connected to each other by communication links along which the messages are exchanged. This results in a communication network with a topology that can be represented by a graph $G = (V, E)$, where the elements in the set V are the vertices and the elements in set E are the edges of the graph. The perception function $u_{m_i}(\mathbf{v}, n, i, \mathbf{v}', n, i')$ is built to route messages of type m_i along the edges of such graph. To this end, the perception function is defined for all the node of class n and all the message of types m_i as:

$$u_{m_i}(\mathbf{v}, n, i, \mathbf{v}', n, j) = \begin{cases} 1 & \text{if } (\mathbf{v}', \mathbf{v}) \in E \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

Therefore, a node \mathbf{v} perceives messages incoming from a node \mathbf{v}' , if and only if node \mathbf{v}' is directly connected with \mathbf{v} .

5.1 Performance Evaluation

For each node of the network, the main measure of interest is the evolution of the mean value of the hop-number carried by the received incoming messages. This value is computed as:

$$\phi(t, \mathbf{v}) = \sum_{i=0}^M i \cdot \rho_i^n(t, \mathbf{v}) \tag{6}$$

For a given node \mathbf{v} of the network at time t , such index measures on average how far (in hop units) are the sources of the incoming messages received by the node.

A practical performance index is defined as the mean time needed by a message - originated at a distance equal to a number of i hops from a node \mathbf{v} - to be received. This index is denoted as $\bar{T}(\mathbf{v}, i)$. Another measure of interest can be derived by calculating how quickly the central station was informed of the detection of a fault in the electrical lines of the power grid and how quickly it could recover the fault. The value of $\bar{T}(\mathbf{v}, i)$ depends on both:

- the number of hop i needed by a message originated in the faulty section of the grid to reach the central station;
- the mean time needed to the message to perform a single hop in its path to the central station. Due to the exponential distribution of the time needed by a transition to be performed by an MA , this value is equal to $1/\alpha$.

The index can be computed as:

$$\bar{T}(\mathbf{v}, i) = \int_0^{+\infty} (1 - \rho_i^n(t, \mathbf{v})) dt \quad (7)$$

the derivation of Eq (7) can be found in [25].

5.2 Numerical Results

The indices defined in Eq (6) and (7) have been computed by means of the *MASolver* under different topologies of the communication network monitoring the power grid. The first set of experiments include two simple topologies shown in Figure 7. They are called direct ring (a) and direct ring with shortcut (b). In both cases it is assumed that the fault is originated in the section of the grid monitored by node n_7 . In the direct ring topology such node starts to send messages to its only neighbor node n_0 , n_0 forwards to n_1 and so on, instead in the direct ring with shortcut n_7 forwards its messages to both n_0 and n_3 , then n_0 forwards to n_1 and n_3 forwards to n_4 and so on. In all the experiments, $\lambda = \alpha = 10s^{-1}$ while the initial state of n_7 is set to 0, i.e. this node has detected a fault. The initial state of all other nodes is set to the idle state I .

Figure 8(a) plots the evolution in time of $\phi(t, \mathbf{v})$, the mean value of the hop-number carried by the messages received by the nodes n_3 , n_4 and n_5 . It can be observed that these values are equal to zero at $t = 0$ and then converge to the exact number of hops that separate them to the node n_7 . Set \bar{t} to the time of convergence, Figure 8(b) shows the node identifier n_i on the x-axis and the value of $\phi(\bar{t}, n_i)$ on the y-axis. For all the nodes, this value is equal to the distance in number of hops to the node n_7 , which confirms that the defined behavior of the MAs allows to compute correctly in a distributed way the number of hops between node n_7 and all the other nodes of the network. Finally, Figure 8(c) plots for each node of the network the value of $\bar{T}(n_i, j)$ with j equal to the exact distance in number of hops between the node itself and the node n_7 . This value is the time interval needed by a message originated from node n_7 to reach the other nodes n_i . The expected trend obtained in the figure is due to the topology of the network. This time interval is proportional to $m * K$, where m is the number of hops performed and K is a constant equal to the time needed to perform a single hop.

The second topology was designed to test whether the defined behavior of the MAs allows not only to compute the number of hops, but also the minimum value of them. To this end, it has been added a shortcut to the ring topology allowing the message originated from node n_7 to reach some nodes with less hops. Figure 9 plots for the ring with shortcut case the same set of results previously shown for the ring topology. The minimum value is computed correctly in this case as well. For example, node n_3 can be reached by node n_7 in two ways: along the path $n_7 \rightarrow n_0 \rightarrow n_1 \rightarrow n_2 \rightarrow n_3$ with four hops or directly through the shortcut with one hop. As shown in Figure 9(b), the last path with the minimum number of hop was correctly chosen.

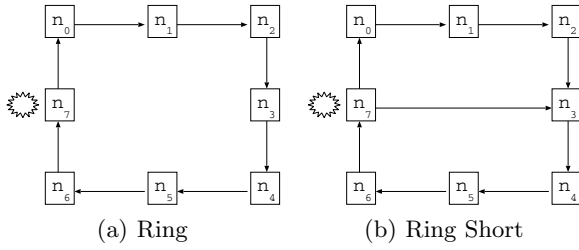


Fig. 7. Topologies: Direct Ring (a), Direct Ring with shortcut

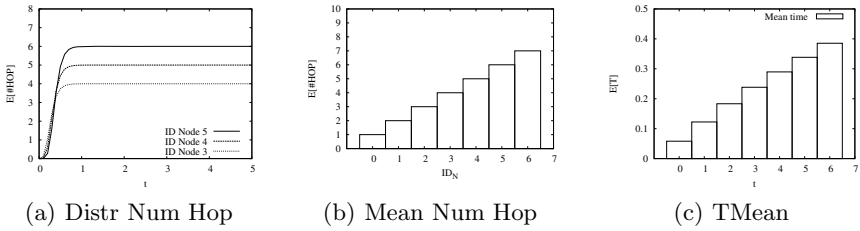


Fig. 8. Topology: ring. Time evolution of the mean hop number to reach each node (a), mean hop number to reach each node in steady state (b), mean time $\bar{T}(\mathbf{v}, i)$ (c).

A final case was used to test a complex topology. This included a random network - generated by using the functionality of the tool Pajek - and the computed value of $\phi(\bar{t}, n_i)$. Pajek [2] is a tool developed by the University of Ljubljana for the generation and analysis of large networks. Given a set of parameters (e.g. number of nodes, connectivity degree distribution, ecc...) this tool is able to generate automatically a random network with specific characteristics, moreover it can export the network in standard formats such as adjacency matrix, adjacency list or others. The network produced by Pajek is shown in Figure 10(a), whereas Figure 10(b) plots for each node n_i the value of $\phi(\bar{t}, n_i)$. Also, in complex random networks including bidirectional links, the results are computed correctly.

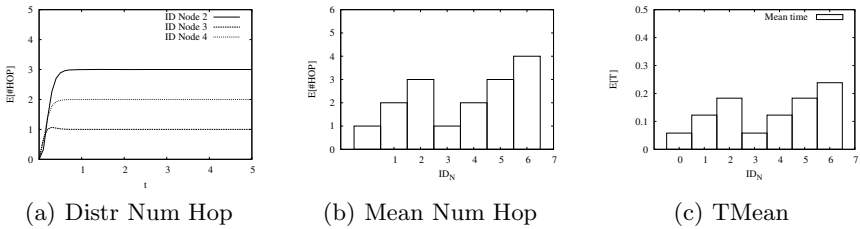


Fig. 9. Topology: ring with shortcut. Time evolution of the mean hop number to reach each node (a), mean hop number to reach each node in steady state (b), mean time $\bar{T}(\mathbf{v}, i)$ (c).

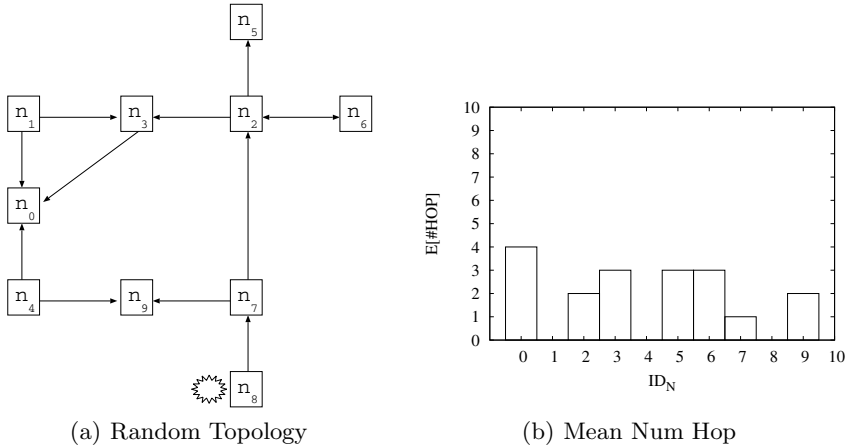


Fig. 10. Random topology (a), mean hop number to reach each node in steady state (b)

6 Conclusions

In this work the problem of considering spatial aspects and performance models in MAs has been considered. A review of some of the available tools has been provided, and a new tool for the definition of graph-based interactions has been introduced. To demonstrate how the tool works, an application including a power grid critical infrastructure has been presented.

The graph-based and the image-based tools considered in this work, although intended for MAs, can be applied also to other spatial formalisms (such as CAs or SPAs). Currently, the presented tools are still in an experimental phase; future works aim at integrating them into a multi-formalism environment to allow full analysis of MA and SPA models.

Acknowledgments. This work has been partially supported by Regione Piemonte within the framework of the “M.A.S.P.” project POR FESR 2007/2013 -Misura I.1.3 “Poli di Innovazione - Polo Information & Communication Technology”.

References

1. Anastasiou, N., Horng, T.-C., Knottenbelt, W.: Deriving generalised stochastic petri net performance models from high-precision location tracking data. In: Proceedings of the ValueTools 2011, ValueTools 2011. IEEE, Los Alamitos (2011)
2. Batagelj, V., Mrvar, A.: Pajek - analysis and visualization of large networks. In: Graph Drawing Software, pp. 77–103. Springer, Heidelberg (2003)
3. Bobbio, A., Bonanni, G., Ciancamerla, E., Clemente, R., Iacomini, A., Minichino, M., Scarlatti, A., Terruggia, R., Zendri, E.: Unavailability of critical SCADA communication links interconnecting a power grid and a telco network. Reliability Engineering and System Safety 95, 1345–1357 (2010)

4. Bobbio, A., Cerotti, D., Gribaudo, M.: Presenting dynamic markovian agents with a road tunnel application. In: MASCOTS 2009. IEEE-CS, Los Alamitos (2009)
5. Bobbio, A., Terruggia, R., Boellis, A., Ciancamerla, E., Minichino, M.: A tool for network reliability analysis. In: Saglietti, F., Oster, N. (eds.) SAFECOMP 2007. LNCS, vol. 4680, pp. 417–422. Springer, Heidelberg (2007)
6. Bonanni, G., Ciancamerla, E., Minichino, M., Clemente, R., Iacomini, A., Scarlatti, A., Zendri, E., Terruggia, R.: Exploiting stochastic indicators of interdependent infrastructures: the service availability of interconnected networks. In: Safety, Reliability and Risk Analysis: Theory, Methods and Applications, vol. 3. Taylor & Francis, Abington (2009)
7. Bruneo, D., Scarpa, M., Bobbio, A., Cerotti, D., Gribaudo, M.: Markovian agent modeling swarm intelligence algorithms in wireless sensor networks. Performance Evaluation (in press, corrected proof: 2011)
8. Cerotti, D.: Interacting Markovian Agents. PhD thesis, Università degli Studi di Torino (2010)
9. Cerotti, D., Gribaudo, M., Bobbio, A.: Disaster propagation in heterogeneous media via markovian agents. In: Setola, R., Geretschuber, S. (eds.) CRITIS 2008. LNCS, vol. 5508, pp. 328–335. Springer, Heidelberg (2009)
10. Cerotti, D., Gribaudo, M., Bobbio, A., Calafate, C.T., Manzoni, P.: A markovian agent model for fire propagation in outdoor environments. In: Aldini, A., Bernardo, M., Bononi, L., Cortellessa, V. (eds.) EPEW 2010. LNCS, vol. 6342, pp. 131–146. Springer, Heidelberg (2010)
11. Chiasserini, C.F., Gaeta, R., Garetto, M., Gribaudo, M., Manini, D., Sereno, M.: Fluid models for large-scale wireless sensor networks. Performance Evaluation 64(7–8), 715–736 (2007)
12. European commission (2006), <http://europa.eu/rapid/pressReleasesAction.do?reference=MEM0/06/477>
13. Dunn, A.: A model of wildfire propagation using the interacting spatial automata formalism. PhD thesis, University of Western Australia (2007)
14. Galpin, V.: Modelling network performance with a spatial stochastic process algebra. In: International Conference on Advanced Information Networking and Applications, pp. 41–49 (2009)
15. Di Gregorio, S., Rongo, R., Serra, R., Spataro, W., Spezzano, G., Talia, D., Villani, M.: Parallel simulation of soil contamination by cellular automata. In: Parcella, pp. 295–297 (1996)
16. Gribaudo, M., Cerotti, D., Bobbio, A.: Analysis of on-off policies in sensor networks using interacting markovian agents. In: PerCom, pp. 300–305 (2008)
17. Gribaudo, M., Raiteri, D.C., Franceschinis, G.: Drawnet, a customizable multi-formalism, multi-solution tool for the quantitative evaluation of systems. In: QEST, pp. 257–258 (2005)
18. DrawNet Project (2011), <http://www.drawnet.com>
19. Jimnez, A., Posadas, A.M.: A moore’s cellular automaton model to get probabilistic seismic hazard maps for different magnitude releases: A case study for greece. Tectonophysics 423(1-4), 35–42 (2006)
20. Matisziw, T.C., Murray, A.T.: Modeling s-t path availability to support disaster vulnerability assessment of network infrastructure. Comput. Oper. Res. 36, 16–26 (2009)
21. Neumann, J.V.: Theory of Self-Reproducing Automata. University of Illinois Press, Champaign (1966)
22. Opper, M., Saad, D.: Advanced Mean Field Methods: Theory and Practice. MIT University Press, Cambridge (2001)

23. Siregar, P., Sinteff, J.P., Chahine, M., Lebeux, P.: A cellular automata model of the heart and its coupling with a qualitative model. *Comput. Biomed. Res.* 29(3), 222–246 (1996)
24. Svendsen, N.K., Wolthusen, S.D.: Graph Models of Critical Infrastructure Interdependencies. In: Bandara, A.K., Burgess, M. (eds.) *AIMS 2007. LNCS*, vol. 4543, pp. 208–211. Springer, Heidelberg (2007)
25. Trivedi, K.S.: *Probability and statistics with reliability, queuing and computer science applications*, 2nd edn., pp. 215–217. John Wiley and Sons Ltd., Chichester (2002)

A Grid Broker Pricing Mechanism for Temporal and Budget Guarantees

Richard Kavanagh and Karim Djemame

School of Computing, University of Leeds,
Leeds, LS2 9JT, UK

{screk, karim.djemame}@leeds.ac.uk

<http://www.comp.leeds.ac.uk>

Abstract. We introduce a pricing mechanism for Grid computing, with the aim of showing how a broker can accept the most appropriate jobs to be computed on time and on budget. We analyse the mechanism's performance via discrete event simulation, and illustrate its viability, the benefits of a new admission policy and to how slack relates to machine heterogeneity.

Keywords: Grid, Economic Model, Time-Cost Constrained Brokering, Job Admission Control.

1 Introduction

Grids [8,7] enable the execution of large and complex programs in a distributed fashion. It is however, common that resources are provisioned in a best effort approach only, with no guarantees placed upon service quality. It has also been known for some time that guaranteed provision of reliable, transparent and quality of service (QoS) oriented resources is an important step for successful commercial use of Grid systems [2,24,15,20].

The idea of introducing QoS to aid the uptake of Grids is not new and there is a significant trend towards the provision of QoS. This is because of a widely held belief that QoS is the major issue that keeps Grids from substantial uptake outside academia [15,16,22,9,21].

In real world commercial and time-critical scientific settings guarantee that computation is going to be completed on time are required. Best-effort service limits the economic importance of Grids because users will be reluctant to pay or contribute resources if the service they receive is not going to return computed results at the time they are required [16].

We present two motivational scenarios that illustrate this need for time guarantees. The first is a commercial scenario where computation is required to generate results which are relied upon as part of the business process, such as animation, where it is useful to have all frames computed overnight before the animation team arrive, where partial completion of the work delays or stops the team starting the next days work [1]. The second scenario is in an academic

environment where it is common before conferences for Grids to become overloaded [10]. It therefore makes sense to prioritise particular compute jobs based upon when the results are required. In order that prioritisation is provided correctly an economic approach is used to ensure users truthfully indicate their priorities [4,17].

To provide QoS the nature of the work that is being performed upon Grids has to be understood. A substantial part of Grid workloads are formed from Bag of Task/Parameter sweep based applications [13,10]. These are formed by sets of tasks that execute independently of one another without communication and are typically considered to be “embarrassingly parallel” [19]. Parameter sweeps are formed by the same application been executed, while having either different parameters or datasets to process.

In [15] it is made clear that uncharacterised, unguaranteed resources have no value and that if poor QoS is provided that can be precisely characterized then the value of the service may be defined. We therefore introduce an economic model for brokering that ensures jobs/a bag of tasks only hold value to the broker if they are completed in a timely fashion. Discrete event simulation is performed to ensure we can characterise how the pricing model behaves, thus ensuring that time and budget guarantees made by the broker can be achieved.

This paper’s main contributions are:

- A viable pricing mechanism that binds time and economic requirements of a job together.
- Recommendations upon how to use slack, in relation to the reference processors speed to mitigate machine heterogeneity within the proposed economic model.
- a new job admission control mechanism, that deals with due date and deadline based job requirements.

The remaining structure of the paper is as follows: The next section discusses the pricing mechanism that has been formulated. The third section covers experimental setup. The fourth discusses the simulation results. The fifth discusses the related work and finally the last section discusses future work and conclusions.

2 The Pricing Mechanism

In this section we introduce the pricing mechanism that is intended to be used in a broker aiming for temporal and budgetary guarantees. We first describe the sequence of events that occur in this brokering service (see Figure 1).

1. Job requirements are sent to the broker. This includes the job’s *resource requirements*, a *budget* that the user has assigned for the completion of a job, a notion of priority that the broker will use to define its *markup* for the service and finally its temporal requirements. These will be shown as a *due date* by which the job should be completed by and a *deadline* by when it must be completed.

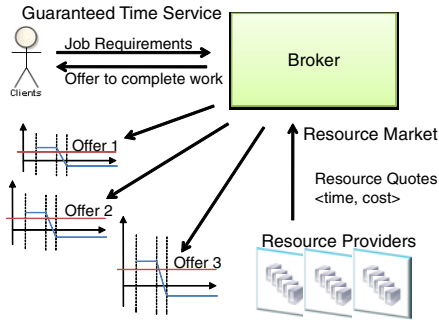


Fig. 1. The pricing mechanism

2. The broker then contacts the various resource providers that are available to the user. It requests a cost and time estimate for completing the work.
3. The resource providers perform initial scheduling of the tasks within a job so they can derive an estimate for completion time and price. This is based upon their local resource markets.
4. The resource providers then present offers to complete the work to the broker.
5. The broker from the offers that have been returned, ranks them and filters out poor offers i.e. sort by earliest and filter out unprofitable offers.
6. The broker then asks the user if it acceptable to proceed with the job at a given price.
7. If the offer made to the user is acceptable the broker then submits the bag of tasks that make up the job to the winning provider.

It is eventually expected that negotiations will occur between the end user and the broker, given the current market conditions and properties of the job to be submitted.

The resource providers generate offers to complete jobs from resource markets. These offers are evaluated by the broker which generates a service market, this market structure was previously seen in SORMA [21]. In this paper we focus upon the broker's service market and the evaluation of offers made by the resource providers. After the broker has received the offers, it is required to evaluate them, which is illustrated in Figure 2.

From the resource providers a cost for running the job upon their resources is provided. This is illustrated as a single line that is used as a reference point in the service market. The broker needs to establish a markup which ensures the broker has incentive for participating in the market (individual rationality) while also lending itself towards budget balance.

Individual rationality requires the utility caused by participation in the Grid market has to increase and budget balance means that participants have to ensure that the income and expenditure balance as long term deficits must be subsidised making them unfeasible [25,26].

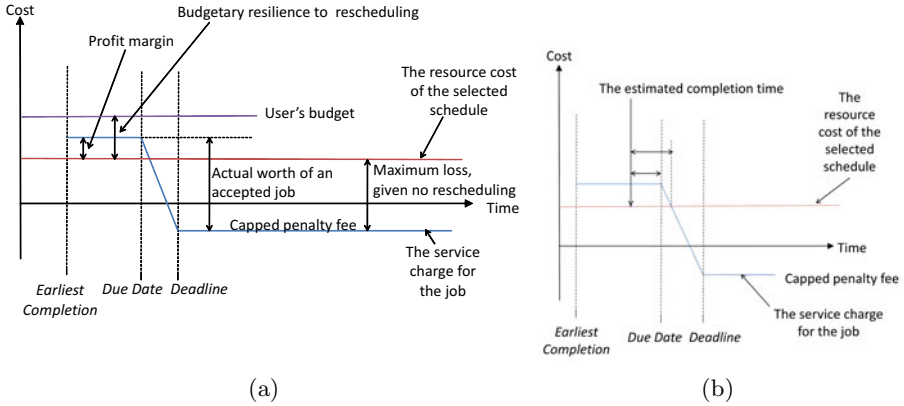


Fig. 2. The fiscal and temporal nature of an offer

The broker has a service charge that it will get paid by a user, which is denoted by the z-shaped line. This intersects the resource cost which ensures the broker will aim to complete the work on time.

In the initial stage before the due date the broker will make a profit. This will be equal to the markup \times resource cost. The next point of interest is that of the deadline, at this point a cap upon the penalty fee is set. This lends itself to budget balance as it limits the maximum possible loss. The rate at which the service charge is reduced is defined to be a linear decent from the due date and the maximum/original value to the deadline and the cap upon the penalty fee.

The penalty fee will be set to zero as this generates further useful properties. If it is set to zero, the point where the broker breaks even is independent of the resource cost and is a fixed percent of the due date to deadline period. The breakeven point is located at: $\frac{\text{markup}}{100+\text{markup}} \times 100$ percentage through the due date to deadline period i.e. if the markup was 50% then $\frac{50}{100+50} \times 100 = 33\%$. If this is not the case then the profit at this same position, where any given resource cost will have the same profit is further defined as $\frac{\text{markup}}{100+\text{markup}} \times \text{cap-on-penalty-fee}$.

The markup in this pricing mechanism will also be used to differentiate between jobs in terms of priority. The higher the priority/markup the faster the budget is consumed upon a given resource, but the more likely the broker is to accept the job due to higher profits. Higher markups also lead to a greater time before the breakeven point when the due date and deadline are not equal i.e. when not using a hard deadline.

In Figure 2a the final two concepts indicated are the budget resilience and the potential loss for accepting work given no rescheduling. These are both numeric figures relating to risk. The budget resilience indicates the gap between the budget and the resource cost. If this is small the broker is more likely to make a loss or use money it would have reserved for profit. The potential loss for accepting the work is the economic risk the broker accepts when scheduling work and asserting guarantees to the end user about its completion.

In Figure 2b there are two important temporal factors marked. The first is the difference between the estimated completion time and the breakeven point. This takes into account the markup assigned to a job. This is because the higher the markup the larger this gap becomes, which leads to the broker having a greater preference for accepting higher markup jobs. Hence this value relates to both temporal and budgetary risk factors i.e. if this gap is small there is little spare time before a loss is made. The second factor is the slack, which is the difference between the due date and the estimate of completion time. This is a purely temporal interpretation of the spare time available to complete the job.

Pricing mechanisms also need admission policies to ensure they are not swamped. We propose the following admission policy:

```

FOR EACH (Offer) {
  Sort the offers based upon the ranking mechanism chosen
  IF Completion_Time <= Due_Date AND
  Service_Price <= Budget {
    Accept Offer;
    BREAK;
  } ELSE {
    Take the last n accepted offers and find the average rate
    at which profit accumulates and establish the going rate.
    IF Current_offer_profit_rate >=
    (going_rate - acceptable_deviation_below_going_rate) {
      Accept Offer;
      BREAK;
    }
  }
}

```

If the due date or budget does not effect the service price then the first most profitable offer is accepted as it must be close to the best possible to accept. If the service price has been affected by the due date or the jobs budget then the broker must determine how much this has affected the job. The offer is therefore only accepted if it is within a lower bound/threshold of the going rate. This ensures that if a job was completed just after the due date that it still may be acceptable if enough profit is gained. This ensures each job that has its service price affected by the constraints is evaluated with respect to the competition for resources within the job's given lifespan i.e. had the job been competitive it would be completed before the due date with money to spare. If it is not competitive then there may still be room to accept the job so long as it does not affect the next job's chance of returning full profit to the broker.

3 Experimental Setup

The simulation was performed in a bespoke simulator that was written in Java and tested using static code analysis¹ and JUnit testing. The simulator works in the following fashion. At the start of the simulation a bucket of jobs is

¹ <http://findbugs.sourceforge.net/>

constructed. These are then released at a set rate determined by the simulator's core. The broker submits jobs to the local schedulers/providers, which in turn calculate the machine to task allocations that they will make in their offer. The broker then receives offers from the providers. These offers are then ranked and filtered. If a winning offer exists it is accepted and the work is scheduled. Once the initial bucket is empty the simulation is then allowed to finish the remaining scheduled work after which the simulation ends.

The simulator used is highly configurable, the parameters that remain fixed throughout the experiment are hence going to be discussed in the remainder of this section. The duration of the experimentation was not capped but 1,000 jobs were to be processed.

A simple local scheduler was used for the experimentation, it was setup to assign work based upon the machines workload, it took the least loaded first and assigned tasks in order of workload (largest task first).

8 providers were used, each with 144 machines. A range of both machine speed and cost values was chosen. The machine cost was set such that the faster the machine the more it costs, though there will be no set distribution shape for this and values will be set by hand.

The task workload will be placed into three categories: small, medium and large. The categories will denote the average task length. They will be set at the equivalent of 2hrs, 4hrs and 8hrs worth of work on the average machine of a provider. The average speed of the machines is 16,333 units of work per second. Thus the amount of work for a medium sized task is 240,000,000 operations. Similarly this has been done for the small (120,000,000) and large (480,000,000) categories. The workload for a task will have a normal distribution with a mean of 2.73 and a standard deviation of 6.1 following the average task runtime indicated in [13]. The durations were chosen because [11] gives the average time to run a task in a group submission/job as 14,181s and [12,10] indicates that most runtimes are shown to be between a fraction of a minute and 1,000 min (1,000min = 16.6hrs). The runtime variability between tasks belonging to the same BoT [13] is indicated to follow a Weibull distribution with a shape of 2.05 and a scale parameter of 12.25 hence this variability will also be accounted for.

The task per job will be separated into three categories: small, medium and large. A Weibull distribution will be chosen with a shape parameter set to 1.76 as per [13], the scale parameter will be adjusted in a fashion that creates the appropriate range desired. Iosup et al. [13] give an average BoT size in their traces as been between 5 and 50, while the maximum BoT size can be on the order of thousands. The authors in [10] indicate that the average number of tasks in a bag in various different Grid traces investigated ranges between 2 and 70 and that most averages for the traces examined fall between 5 and 20. The ranges for the amount of tasks in a bag are therefore indicated in table 1.

The inter-arrival time (i.e. the time between arrivals of jobs) will follow the Weibull distribution and will have a shape parameter of 4.25 as per [13]. The inter-arrival time for job submissions is indicated to be between 1 and 1,000s in [10,12], with most been below 100s.

Table 1. The minimum and maximum tasks per job

Bag Size	Min and Max Values	Mean
Small	2-10	4.99
Medium	11-39	21.49
Large	40-70	51.23

A selection of inter-arrival times have therefore chosen to reflect this prior work. Though a wide spread of inter-arrival times has been tested a more select range which provides high levels of competition for resources will be discussed in this paper. Inter-arrival rates will be below 334.14s with an inter-arrival time of 210.14s where it is not otherwise indicated.

Each simulation will be run 10 times and where shown a confidence interval of 95.4% will be indicated.

4 Evaluation

In this section we discuss the results of our experimentation. We initially focus upon the effects of a range of parameters upon jobs accepted and its relationship to slack. We then focus upon results highlighting the behaviour of the admission policy introduced in section 2. In examining the profit made over the life of the simulation we can then assess the long term viability of the broker while ensuring as many users as possible have their requirements satisfied.

In Figure 3 the due date and deadline are set to be the same time. The inter-arrival time is set at 210.14s where is not subject to change as part of the experiment. This arrival rate has been determined to give a high level of competition between tasks. The markup is fixed at 10% and a medium workload per task and medium count of tasks per job were chosen. The broker's selection policy sorts the offers by earliest first then accepts the topmost offer.

In Figure 3a two markups are shown, it can be seen that there is little distinguishing difference between them. The profit made which is not shown here differs in that the higher markup when compared with the lower markup makes more profit. The lack of difference between markups is placed down to the permissive admission policy, used by this first experiment.

A clear transition between 250% and 300% is observed. This is considered to be an aspect of competition. Firstly this experiment is conducted with due date and deadline been equal. It can therefore be said that at 100% (i.e. just in time without any slack provided) there would have to be no start delay present, i.e. no effective competition is present. Given higher levels of slack we can consider that some delay can be tolerated and that this delay will be directly related to other jobs in the system and hence the competition between them.

The arrival rate influences the amount of jobs accepted, but as shown in Figure 3b this only occurs after the transition point. It is noted that the 86.14 inter-arrival time trace is levelling out and is tending towards a maximum permissible amount of acceptable jobs, while 210.14 and 243.7 observe a much more linear relationship

after this transition period. Finally the 334.14 trace has reached the maximum amount of jobs submitted after this transition. The effects shown here are because jobs have a fixed time window in which they must be completed, so high arrival rates ensure these windows overlap more and once the slack is increased more freedom of job placement permitted and fewer jobs have to be rejected.

We now show that the position of the transition point is not related to the task count (Figure 3c) or task size (Figure 3d).

In Figure 3c it is observed that before 2.5x slack the amount of tasks in a job distinguishes the traces, which is not seen in the other figures. The differences in gradients seen after the 3.0x point is also of interest as jobs with a lower amount of tasks per job are accepted much more readily, though this also relates to their simply being less work. Argument can still be made that if the slack is small then it helps if the job contains fewer tasks, especially before the transition point. This distinction occurs as there is a finite amount of machines over the average speed of the provider. The average speed was chosen as the mechanism for estimating completion time. Hence when more tasks are present more competition exists between the tasks, as there is a finite set of machines that are fast enough to complete the work in time, especially with low levels of slack.

Figure 3d like Figure 3c changes the amount of workload per job; it must therefore be asked why Figure 3d before the transition does not distinguish between the low, medium and high traces. The reason for this is placed down to the fact the duration a job relates to the amount of work needing completing. In Figure 3d this therefore makes no difference as the size of the slack is related to the estimate of completion time and hence task size.

The start delay relative to the task length during the experimentation has been very limited. The initial slack therefore often limits the issues that competition and start delay causes. In Figure 3e we see an alternative use for slack and the reason for the sudden transition. Figure 3e shows that the reference processor for estimating the duration is responsible. Three separate reference speeds were chosen, namely the fastest possible machine 25,000, the average 16,333 and a slow machine 14,000. The choice of reference machine ensures tasks cannot be placed on machines slower than the reference until the slack is sufficient to compensate for the extra time taken. If the slack is always chosen to compensate for the heterogeneity in machines then job acceptance rates will be high given the temporal constraints. If the end user requests a small amount of slack then it can be assumed that they will require machines that are at least as fast as the reference machine.

The fastest/average/slowest machines per provider where 25,000/16,333 and 6,000 ops/sec respectively. In table 2 the reference machine's speed is compared to the slowest machine and in doing so indicates the location on the x axis of the transition point.

In the experimentation shown by Figure 4 the inter-arrival time is set at 210.14s. The aim of this experimentation it to show the effects of increasing the gap between the due date and deadline. The offers from providers are ordered by earliest completion first. In Figure 4a and 4b the results are shown for several

Table 2. The relative speed difference between the reference machine and the slowest machine

Calculation	Value
Average/Slowest	2.72
Fastest/Slowest	4.17
Slow/Slowest	2.33

markups. The filter/admission policy being used takes the first offer where a profit is made, which is similar to related work such as [23], given that faster processors in our experimentation cost more. In Figure 4c and 4d the proposed admission policy is used instead. The results shown are for 50% markup which highlights the benefits of our admission policy the most. The last 50 accepted jobs are used to establish the going rate.

In Figure 4a the tendency is to follow a straight line equation, 10% and 50% have been marked on 4a to highlight this. Confidence intervals on 10, 40 and 50% markup runs have been added though for 20% and 30%'s error bars have been omitted but are broadly similar.

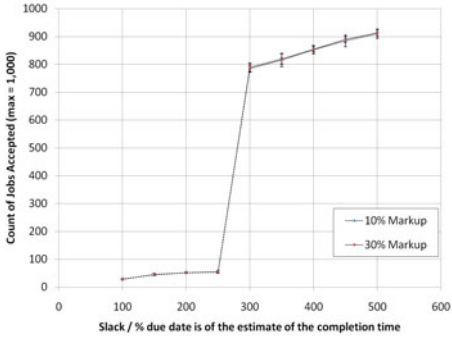
It can be seen in Figure 4a when the deadline is set to be 5x (500%) that of the due date, that all jobs with 50% markup are accepted. This illustrates how higher markup encourages job acceptance and exacerbates the effect of the increased gap between due date and deadline. This occurs because the breakeven point is later in the 50% markup than it is with the 10% markup and in general the 50% markup run is more likely to make a profit on a given job.

The detrimental effects of accepting any profitable offers is shown in Figure 4b. The profit drops as the gap between due date and deadline increases. This is due to more jobs being accepted (Figure 4a) which makes it more likely that the next job to arrive will overshoot the due date. The increase in gap size effectively gives more time for job to complete, which generates an increase in competition for the finite resources available as more jobs can plausibly complete before the deadline. It can also be noted that when the gap between due date and deadline is small the rate at which the service price drops quickly, hence providing the steepness of the curve.

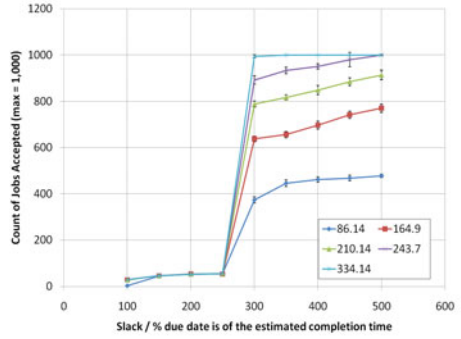
This downward trend in profit is eventually reversed as the gap increases. This is due to the broker having more time to complete the work before the breakeven point as the drop in the service price is slower and is aided by the acceptance of more jobs.

In the Figure 4c we can see fewer jobs were accepted when the threshold below the going rate was less than zero i.e. when all jobs after the due date must have a higher going rate than the average for the last 50 jobs.

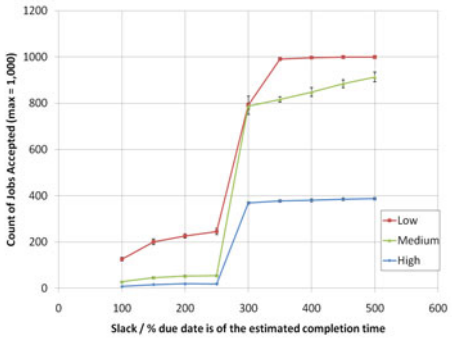
The broker's profit can be seen to increase in Figure 4d as the difference in rates is set so that jobs after the completion time have to have a higher rate of return. There remains some loss relative to not allowing any gap, but this has largely been mitigated. The due date to deadline period may also be used solely as a recovery mechanism/a way to deal with completion time uncertainty, at the expense of never accepting work that is expected to complete between the due date and deadline.



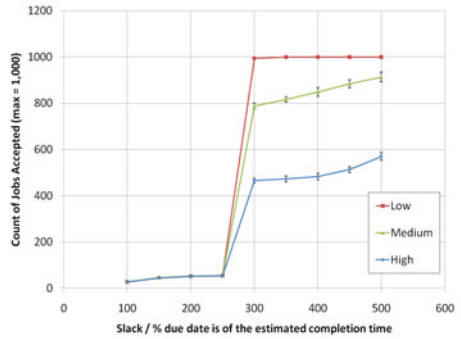
(a)



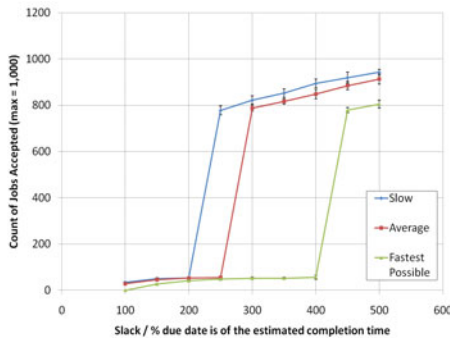
(b)



(c)



(d)



(e)

Fig. 3. The effects of slack on job acceptance

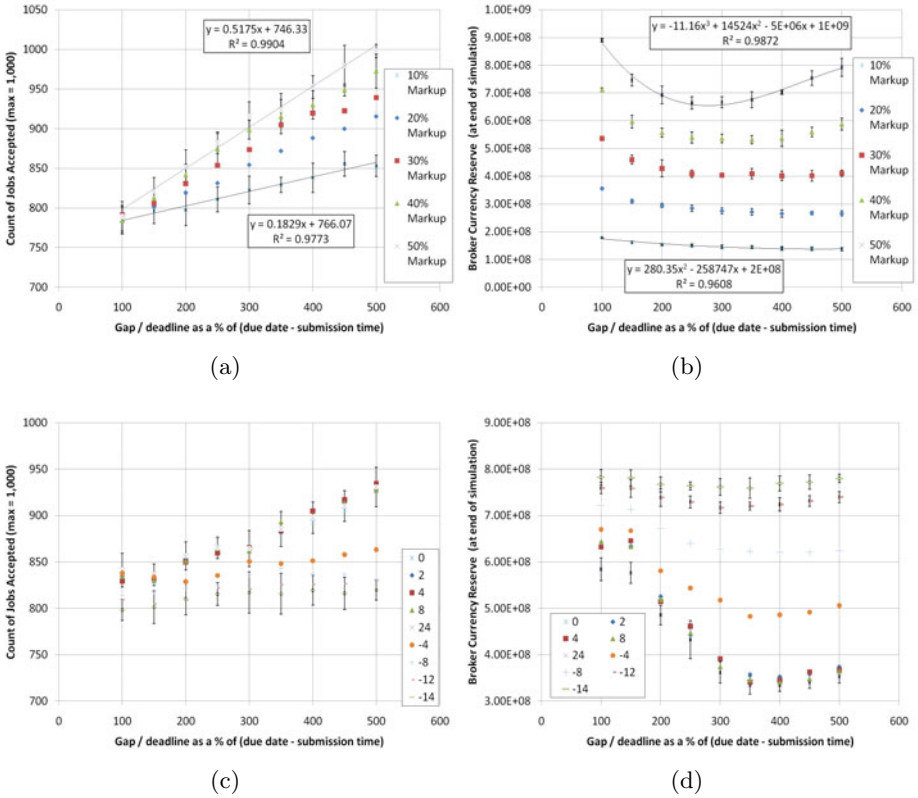


Fig. 4. The effects of intelligent filtering

5 Related Work

A good introduction to pricing may be found in [30,21]. The SORMA project introduced two pricing schemes namely k-pricing [26,3] and the GREEDEX [28] clearing mechanism, though the most similar pricing mechanisms can be found in [6,23,14,5,1] where a service charge that diminishes once a job has passed a specified time is also used.

In [6,23,14,5] the focus is upon the minimum time for completion/slowdown and not upon the user preference for a completion time. It seems more suitable when considering user satisfaction to aim for a due date and hence when the user wants/needs the results rather than the minimum possible runtime for a given job. The concept of slowdown is also made unclear by various machine speeds across the Grid, as it removes the clear reference point by which slowdown may be judged. This therefore reflects the real world better as it simply stops the need for reference to a dedicated compute resource and focuses on when the results are needed.

In [6,23,14,5,1] a completion time and rate of loss of a jobs economic value is used. Our approach uses a due date and deadline which is seen as more user friendly than asking for a rate by which a job loses its economic worth.

In [6] currency is provided to users at periodic intervals into accounts with a finite limit. This acts as a control mechanism to ensure arbitrarily large bids cannot be made. [17] describes why hoarding money introduces predictability and resource starvation issues, which is reasonable justification for using a control mechanism that has upper limits on the account size.

Like our approach Aggregate Utility [23] FirstProfit and FirstOpportunity [23] takes a bound approach to penalty fees. In First Reward and Risk Reward [14] it is chosen to be only potentially bound and in LibraSLA [5] there is no bound at all. This unlimited penalty has issues as pricing mechanisms should have properties such as budget balance and individual rationality which unbound penalties preclude [25,26].

The difference between the due date and the runtime of a job gives a notion of slack, derived from the user's preference or willingness to accept delays in a jobs completion time. This concept of slack is first seen in [1], but a rate of decay is still considered, rather than seeing the decay rate as a product of a due date and deadline.

In LibraSLA [5] a deadline based approach is taken, in which deadlines are classified as either hard or soft. Soft deadlines cause the price of jobs to diminish at a given rate. Our approach differs in that it lacks the distinction between deadline types as hard deadlines merely have the due date and deadline been equal.

In [1] an aggregate utility function is used so the user may specify their sensitivity to not having all the tasks in a job been completed. This is seen by the authors to be beneficial in preventing the service provider from cherry picking only the most profitable jobs.

In [14] the concept of net present value is used so future work is discounted with the intent of ensuring that long jobs are penalised. They are seen as being potentially more risky as profit can only be achieved after they complete. The long duration also ensures that they may potentially block a more profitable job in the meantime. This therefore allows the scheduler to be configured to accept only smaller jobs as the returns are likely to be made earlier and the job is less likely to block further more profitable jobs. This feature is worthy of note and may be considered in our own model in the future. It is however not clear how to continually adjust the discount rate to ensure jobs that will not have an detrimental effect upon the next job are accepted, especially in low competition based circumstances.

The admission control was shown during both our own and others experimentation to be key [14,23] to a good brokering system. In early work such as First Price [6] no mention is made. In First Reward and Risk Reward [14] the time before the perceived utility drops below zero is used. This utility derives from using a discount rate that depreciates the perceived value of a job the based on the time it takes to complete. The assumption is that short jobs are

less likely to block future more profitable jobs. Our mechanism avoids this and only penalises jobs that do not reach full profit and are likely to delay the next job. In [23] profit is used as admission criteria for FirstProfit, FirstOpportunity, LJF (longest job first) and SJF (shortest job first). For FirstOpportunityRate The ProfitRate (profit divided by schedule duration) is used. In [1] profit rate is again used though it is complicated by the aggregate utility function which allows unprofitable tasks should it ensure the contract mechanism in place for the amount of tasks to complete provides an appreciable gain. Finally LibraSLA [5] uses a “return value” which is derived from the job’s value/runtime/deadline. This return value is then summed across all nodes and tasks. The new candidate schedule is then compared with the previous schedule.

6 Conclusion and Future Work

We have illustrated a pricing model that is aimed at ensuring the broker that is been developed as part of our work will perform work on time and on budget[29]. It ensures the broker aims towards a due date and deadline provided by the end user, given their cost constraints/priority for their job. We have made recommendations upon how slack can be used in relationship to machine heterogeneity and a reference machine which is used for estimating job completion times. We have also shown a admission policy that removes the significant losses made by accepting too many jobs. This focused upon the due date to deadline period and ensured the pricing mechanism’s viability by making sure the broker in accepting additional work still maintains its reason for participating in the Grid i.e incentive compatibility.

In the future we plan to consider dynamic pricing at the resource provider level. This ensures resource providers are not going to either undervalue or overvalue resources, which in turn can lead to unrealised profit or utility [17,21]. The models in [23,14,5,1] all make little attempt to cope with data transfer requirements, which has been shown by [27,18] to be a substantial problem for the scalability of algorithms. We intend to add pricing based on data transfer to our model and to account for transfer times and costs. We also intend to investigate the effects of underestimates of task lengths and the effect of events and slowdowns that require rescheduling.

References

1. AuYoung, A., Grit, L., Wiener, J., Wilkes, J.: Service contracts and aggregate utility functions. In: 15th IEEE International Symposium on High Performance Distributed Computing (HPDC-15). IEEE, New York (2005)
2. Battre, D., Hovestadt, M., Kao, O., Keller, A., Voss, K.: Planning-based scheduling for sla-awareness and grid integration. In: Bartk, R. (ed.) PlanSIG 2007 The 26th Workshop of the UK Planning and Scheduling Special Interest Group, vol. 1, p. 8. Prague, Czech Republic (2007)

3. Becker, M., Borrisov, N., Deora, V., Rana, O.F., Neumann, D.: Using k-pricing for penalty calculation in grid market. In: Proceedings of the 41st Annual Hawaii International Conference on System Sciences, p. 97 (2008)
4. Buyya, R., Abramson, D., Venugopal, S.: The grid economy. Proceedings of the IEEE 93(3), 698–714 (2005)
5. Chee Shin, Y., Buyya, R.: Service level agreement based allocation of cluster resources: Handling penalty to enhance utility. In: IEEE International Cluster Computing, pp. 1–10 (2005)
6. Chun, B.N., Culler, D.E.: User-centric performance analysis of market-based cluster batch schedulers. In: 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid 2002, p. 30 (2002)
7. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. In: Tennessee, U.C.B.L. (ed.) Workshop on Clusters and Computational Grids for Scientific Computing, vol. 15, pp. 200–222. Sage Publications Inc., Chateau Faverges (2000)
8. Foster, I.: What is the grid? a three point checklist. Grid Today 1(6), 22–25 (2002)
9. Hovestadt, M.: Fault tolerance mechanisms for sla-aware resource management. In: Proceedings of 11th International Conference on Parallel and Distributed Systems 2005, vol. 2, pp. 458–462 (2005)
10. Iosup, A., Epema, D.: Grid computing workloads. IEEE Internet Computing 15(2), 19–26 (2011)
11. Iosup, A., Jan, M., Sonmez, O.O., Epema, D.H.J.: The characteristics and performance of groups of jobs in grids. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 382–393. Springer, Heidelberg (2007)
12. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.J.: The grid workloads archive. Future Generation Computer Systems 24(7), 672–686 (2008)
13. Iosup, A., Sonmez, O., Anoep, S., Epema, D.: The performance of bags-of-tasks in large-scale distributed systems. In: Proceedings of the 17th International Symposium on High Performance Distributed Computing. ACM, Boston (2008)
14. Irwin, D.E., Grit, L.E., Chase, J.S.: Balancing risk and reward in a market-based task service. In: Proceedings of 13th IEEE International Symposium on High Performance Distributed Computing 2004, pp. 160–169 (2004)
15. Kenyon, C., Cheliotis, G.: Architecture requirements for commercializing grid resources. In: Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing, HPDC-11 2002, pp. 215–224 (2002)
16. Kokkinos, P., Varvarigos, E.A.: A framework for providing hard delay guarantees and user fairness in grid computing. Future Generation Computer Systems 25(6), 674–686 (2009)
17. Lai, K.: Markets are dead, long live markets. SIGecom Exch. 5(4), 1–10 (2005)
18. Lee, Y.C., Zomaya, A.Y.: Data sharing pattern aware scheduling on grids. In: International Conference on Parallel Processing, ICPP 2006, pp. 365–372 (2006)
19. Lee, Y.C., Zomaya, A.Y.: Practical scheduling of bag-of-tasks applications on grids with dynamic resilience. IEEE Transactions on Computers 56(6), 815–825 (2007)
20. Liu, C., Baskiyar, S.: A general distributed scalable grid scheduler for independent tasks. Journal of Parallel and Distributed Computing 69(3), 307–314 (2009)
21. Neumann, D., Stober, J., Weinhardt, C., Nimis, J.: A framework for commercial grids - economic and technical challenges. Journal of Grid Computing 6(3), 325–347 (2008)

22. Nou, R., Kounev, S., Julia, F., Torres, J.: Autonomic qos control in enterprise grid environments using online simulation. *Journal of Systems and Software* 82(3), 486–502 (2009)
23. Popovici, F.I., Wilkes, J.: Profitable services in an uncertain world. In: *Proceedings of the ACM/IEEE Conference on Supercomputing, SC 2005*, pp. 36–36 (2005)
24. Priol, T., Snelling, D.: Next generation grids: European grid research 2005–2010 (2003)
25. Schnizler, B., Neumann, D., Veit, D., Weinhardt, C.: A multiattribute combinatorial exchange for trading grid resources. In: *Proceedings of the 12th Research Symposium on Emerging Electronic Markets, RSEEM* (2005)
26. Schnizler, B., Neumann, D., Veit, D., Weinhardt, C.: Trading grid services - a multi-attribute combinatorial approach. *European Journal of Operational Research* 187(3), 943–961 (2008)
27. da Silva, F.A.B., Senger, H.: Improving scalability of bag-of-tasks applications running on master-slave platforms. *Parallel Computing* 35(2), 57–71 (2009)
28. Ster, J., Neumann, D.: GreedEx a scalable clearing mechanism for utility computing. *Electronic Commerce Research* 8(4), 235–253 (2008)
29. The University of Leeds: ISQoS Project Homepage (2010), <http://www.comp.leeds.ac.uk/CollabSysAndPerf/html/ISQOS.php>
30. Wilkes, J.: Utility functions, prices, and negotiation. In: Buyya, R., Bubendorfer, K. (eds.) *Market Oriented Grid and Utility Computing*. Wiley Series on Parallel and Distributed Computing, pp. 67–88. John Wiley & Sons, Inc., Chichester (2008)

Visualisation for Stochastic Process Algebras: The Graphic Truth

Michael J.A. Smith¹ and Stephen Gilmore²

¹ Department of Informatics and Mathematical Modelling
Danmarks Tekniske Universitet, Lyngby, Denmark
mjas@imm.dtu.dk

² Laboratory for Foundations of Computer Science
University of Edinburgh, Edinburgh, United Kingdom
Stephen.Gilmore@ed.ac.uk

Abstract. There have historically been two approaches to performance modelling. On the one hand, textual language-based formalisms such as stochastic process algebras allow compositional modelling that is portable and easy to manage. In contrast, graphical formalisms such as stochastic Petri nets and stochastic activity networks provide an automaton-based view of the model, which may be easier to visualise, at the expense of portability. In this paper, we argue that we can achieve the benefits of both approaches by *generating* a graphical view of a stochastic process algebra model, which is synchronised with the textual representation, giving the user has two ways in which they can interact with the model.

We present a tool, as part of the PEPA Eclipse Plug-in, that allows the components of models in the Performance Evaluation Process Algebra (PEPA) to be visualised in a graphical way. This also provides a natural interface for labelling states in the model, which integrates with our interface for specifying and model checking properties in the Continuous Stochastic Logic (CSL). We describe recent improvements to the tool in terms of usability and exploiting the visualisation framework, and discuss some of the general features of the implementation that could be used by other tools. We illustrate the tool using an example based on a model of a financial web-service application.

1 Introduction

It is often said that seeing is believing. Even though we know from biology that the eye can be tricked in all manner of ways, most people will agree that being able to see — or *visualise* — something with their own eyes adds great weight to their belief in it. This is true in performance modelling, just as much as in the real world. Unlike a computer program, which implements a specification and therefore can be tested for correctness, a performance model is often a specification in and of itself. This leads to a big problem — how do we *convince* ourselves that the model we have written is really the same as the model we intended to write?

There are many approaches to performance modelling, but the use of *language-based* formalisms such as *stochastic process algebras* [16, 19] have been particularly successful. In addition to being natural for computer scientists, who are used to programming

in linear, text-based languages, they have the advantage of portability — we do not require a special tool to view or edit the model. A disadvantage, however, is that it can be difficult to visualise the behaviour of the model — for example, a small typo in the model can lead to strange and unintended behaviour, but can easily go unnoticed.

An alternative approach is to use *graphical* formalisms for performance modelling, such as stochastic Petri nets [3] and stochastic activity networks [20]. Since these are automata-based formalisms, it is easy to visualise the structure and behaviour of components in the model. Whilst several highly successful tools make use of such formalisms — for example PIPE [4] and Möbius [11] — they suffer from some limitations. Most notably, *portability* of the model between tools, and *flexibility* of the tool, since the interface may be too restrictive or cumbersome for advanced users, compared to the freedom of a text editor.

The contribution of this paper is to bring these two approaches together, in a tool that supports *two different views* of the same model. We present an extension to the PEPA Eclipse Plug-in [30] that allows performance models in the Performance Evaluation Process Algebra (PEPA) [16] to be presented *graphically*. This is useful not only for visualising the model, but also as an intuitive interface for *abstracting* it, and for specifying *performance properties* we would like to verify.

Since we have already presented a summary of the analysis features of our tool in [24], implementing the compositional abstractions developed in [25, 26], it is important to clarify the purpose of this paper. Our focus here is not on the back-end of the tool, but on the *novel user interfaces* that we have developed. We present some significant improvements in features and usability compared with [24], and moreover describe the implementation details of our front-end, to allow the principles to be applied to other tools based on stochastic process algebras. Figure 1 shows a screenshot of the plug-in.

We begin in Section 2 by introducing the PEPA language, along with a running example based on a financial web service case study. We then motivate the need for visualisation of PEPA models in Section 3, before introducing the visualisation features of the PEPA Eclipse Plug-in. In Section 4, we describe how PEPA models can be visualised in a graphical way, along with how this interface can be used for specifying abstractions of the model, and for labelling states. In Section 5, we then present the interface for constructing performance properties (in the Continuous Stochastic Logic (CSL) [2]), which ensures that the user can only enter syntactically valid properties. We discuss implementation details in Section 6, before considering related work in Section 7 and concluding in Section 8.

2 Modelling in PEPA

The Performance Evaluation Process Algebra (PEPA) [16] is a widely used language for performance modelling and analysis, which allows models to be built compositionally. PEPA models are built out of *components*, which run in parallel and can perform *activities*. An activity (a, r) is a pair consisting of an *action type* $a \in \mathcal{A}$, and a *rate* $r \in \mathbb{R}_{\geq 0} \cup \{\top\}$. The rate parameterises an exponential distribution that describes the duration of the activity. The special rate \top denotes a *passive rate*, meaning that another component must determine the rate of the activity. The syntax of PEPA is as follows:

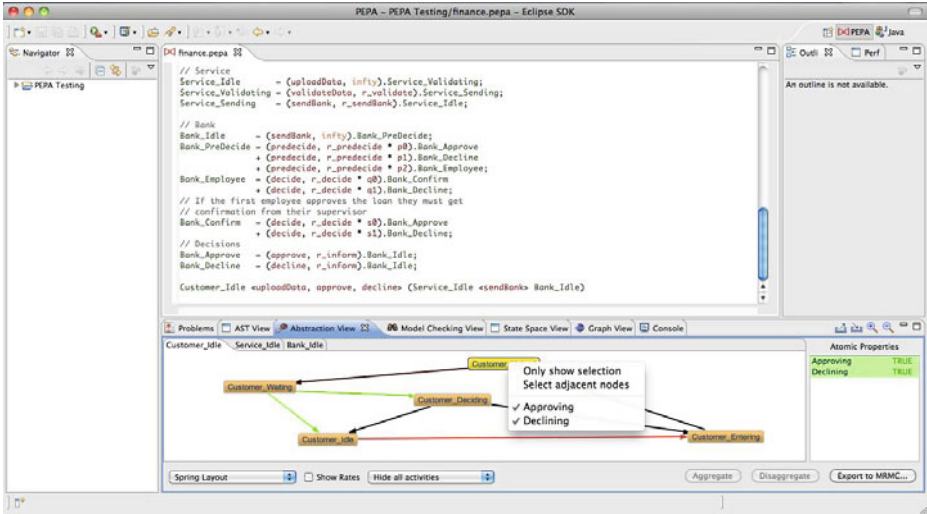


Fig. 1. The PEPA Eclipse Plug-in, showing the editor and the abstraction view

$$\begin{aligned}
 C_S &:= (a, r).C_S \mid C_S + C_S \mid A \\
 C_M &:= C_S \mid C_M \underset{L}{\bowtie} C_M \mid C_M/L
 \end{aligned}$$

Here, we call C_S *sequential components*, and C_M *model components*. A PEPA model is constructed by defining a collection of sequential components, along with a model component called the *system equation*, which describes the initial configuration of the model. The PEPA combinators are as follows:

- Prefix** $(a, r).C$ The component can carry out an activity (a, r) to become C .
- Choice** $C_1 + C_2$ The component may behave as either C_1 or C_2 , according to the first that completes an activity (the race condition).
- Cooperation** $C_1 \underset{L}{\bowtie} C_2$ C_1 and C_2 synchronise over the actions in L (the cooperation set). For activities whose type is not in L , the two components proceed independently. Otherwise, they must perform the activity together, at the rate of the slowest component.
- Hiding** C/L The component behaves as C , except that activities with an action type in L are hidden, and cannot be synchronised over.
- Constant** $A \stackrel{def}{=} C$ The name A refers to component C .

PEPA has an operational semantics, which maps a model onto a labelled multi-transition system, from which a continuous-time Markov chain (CTMC) is derived [16]. If we want to operate on the underlying CTMC of a model in a *compositional* way, however, it is more useful to use an alternative semantics based on a *Kronecker representation*. This was first introduced in [17], and developed further in [25, 26]. It was proven in [25] that the reachable state space of the CTMC given by the Kronecker semantics is isomorphic to that given by the original semantics of PEPA in [16].

To specify the CTMC of a PEPA model in a compositional way, we first need to define the notion of a *CTMC component*:

Definition 1. A CTMC component is a tuple (S, r, \mathbf{P}, L) , where S is a finite non-empty set of states, $r : S \rightarrow \mathbb{R}_{\geq 0} \cup \{\top\}$ assigns a rate (or \top) to each state, $\mathbf{P} : S \times S \rightarrow [0, 1]$ assigns a probability distribution over S to each state $s \in S$, and $L : S \rightarrow AP$ is a labelling function (AP is a finite set of atomic properties). We require for all $s \in S$ that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$.

Note that if all the rates are active, a CTMC component is just a standard CTMC.

We construct a PEPA model by *composing* CTMC components. To do this, we use two composition operators, \otimes and \odot , which correspond to synchronised and independent parallel composition respectively. For CTMC components $M_1 = (S_1, r_1, \mathbf{P}_1, L_1)$ and $M_2 = (S_2, r_2, \mathbf{P}_2, L_2)$, these are defined as:

$$M_1 \otimes M_2 = (S_1 \times S_2, \min\{r_1, r_2\}, \mathbf{P}_1 \otimes \mathbf{P}_2, L_1 \times L_2)$$

$$M_1 \odot M_2 = (S_1, r_1, \mathbf{P}_1, L_1) \otimes (S_2, r'_\top, \mathbf{I}, L_2) + (S_1, r'_\top, \mathbf{I}, L_1) \otimes (S_2, r_2, \mathbf{P}_2, L_2)$$

where we define $\min\{r_1, r_2\}(s_1, s_2) = \min\{r_1(s_1), r_2(s_2)\}$, $(L_1 \times L_2)(s_1, s_2) = L_1(s_1) \cap L_2(s_2)$. $r_\top(s) = \top$ for all s , and \mathbf{I} is the identity matrix ($\mathbf{I}(s_1, s_2) = 1$ if $s_1 = s_2$ and 0 otherwise). \otimes is the standard Kronecker product of two matrices [21].

For two CTMC components $M_1 = (S, r_1, \mathbf{P}_1, L)$ and $M_2 = (S, r_2, \mathbf{P}_2, L)$ with the same state space S and labelling function L , the addition operator used in the previous equation is defined as follows:

$$M_1 + M_2 = \left(S, r_1 + r_2, \frac{r_1}{r_1 + r_2} \mathbf{P}_1 + \frac{r_2}{r_1 + r_2} \mathbf{P}_2, L \right)$$

where we define $(r_1 + r_2)(s) = r_1(s) + r_2(s)$, $\frac{r_i}{r_1 + r_2}(s) = \frac{r_i(s)}{r_1(s) + r_2(s)}$, $i \in \{1, 2\}$, and $(r\mathbf{P})(s_1, s_2) = r(s_1)\mathbf{P}(s_1, s_2)$.

The Kronecker semantics of PEPA is as follows. For a PEPA sequential component C , we use the operational semantics in [16] to derive a CTMC component: $\llbracket C \rrbracket^{PEPA} = (S, r, \mathbf{P}, L)$. Technically, the labelling function L is not given as part of the model, but we will show how to define it using the PEPA Eclipse Plug-in, in Section 4. We can similarly define $\llbracket C \rrbracket_a^{PEPA} = (S, r_a, \mathbf{P}_a, L)$ to be the CTMC component over the same state space S , where only the contribution of activities of action type a is considered (if a state s cannot perform an activity of type a , $r_a(s) = 0$).

Definition 2. The CTMC induced by a PEPA model C is:

$$\llbracket C \rrbracket = \sum_{a \in Act(C)} \llbracket C \rrbracket_a$$

where $Act(C)$ is the set of all action types that occur in C (both synchronised and independent), and $\llbracket C \rrbracket_a$ is as follows:

$$\llbracket C \rrbracket_a = \llbracket C \rrbracket_a^{PEPA} \quad \text{if } C \text{ is a sequential component}$$

$$\llbracket C_1 \bowtie_{\mathcal{C}} C_2 \rrbracket_a = \begin{cases} \llbracket C_1 \rrbracket_a \otimes \llbracket C_2 \rrbracket_a & \text{if } a \in \mathcal{L} \\ \llbracket C_1 \rrbracket_a \odot \llbracket C_2 \rrbracket_a & \text{if } a \notin \mathcal{L} \end{cases}$$

$$\begin{aligned}
Customer_Idle &= (request, r_{request}).Customer_Entering \\
Customer_Entering &= (enterData, r_{enter_data}).Customer_Upload \\
Customer_Upload &= (uploadData, r_{upload}).Customer_Waiting \\
Customer_Waiting &= (approve, r_{inform}).Customer_Idle \\
&+ (decline, r_{inform}).Customer_Deciding \\
Customer_Deciding &= (reapply, r_{reapply} \times t_0).Customer_Entering \\
&+ (reapply, r_{reapply} \times t_1).Customer_Idle \\
\\
Service_Idle &= (uploadData, r_{upload}).Service_Validating \\
Service_Validating &= (validateData, r_{validate}).Service_Sending \\
Service_Sending &= (sendBank, r_{sendBank}).Service_Idle \\
\\
Bank_Idle &= (sendBank, r_{sendBank}).Bank_PreDecide \\
Bank_PreDecide &= (predecide, r_{predecide} \times p_0).Bank_Approve \\
&+ (predecide, r_{predecide} \times p_1).Bank_Decline \\
&+ (predecide, r_{predecide} \times p_2).Bank_Employee \\
Bank_Employee &= (decide, r_{decide} \times q_0).Bank_Confirm \\
&+ (decide, r_{decide} \times q_1).Bank_Decline \\
Bank_Confirm &= (decide, r_{decide} \times s_0).Bank_Approve \\
&+ (decide, r_{decide} \times s_1).Bank_Decline \\
Bank_Approve &= (approve, r_{inform}).Bank_Idle \\
Bank_Decline &= (decline, r_{inform}).Bank_Idle \\
\\
Customer_Idle &\quad \boxtimes_{\{uploadData, approve, decline\}} \left(Service_Idle \quad \boxtimes_{\{sendBank\}} Bank_Idle \right)
\end{aligned}$$

Fig. 2. A PEPA model of a financial web service application

Example: As a running example for the remainder of this paper, consider the PEPA model in Figure 2. This is a model of a financial services case study from the SENSORIA project — a five-year EU-funded project on software engineering for service-oriented computing [23]. The project brought together a large number of European universities and research centres together with four industrial partners, one of whom was a European bank engaged in business-to-business operation. The bank explained the process by which loans are awarded to businesses: the workflow must be reliable, to guard against fraud, and also meet legal constraints on fiscal and monetary transactions.

The PEPA model presented here describes this workflow in terms of a customer using a service portal. Through this, the customer interacts with the bank, where employees approve or decline loans subject to managerial approval. Structurally, the model is a typical idiomatic PEPA model, with a small number of sequential components, which may be replicated to make larger instances of the problem. These sequential components are brought together in a parallel composition, requiring them to co-operate on shared activities (such as *uploadData*) and to proceed independently on other activities (such as the *decide* activity, which approves the loan request).

Some components have a relatively complex workflow with multi-way branching and loops to different entry points in the workflow. Each component is cyclic so that the model has a meaningful steady state solution, and describes an unending process with infinite behaviour. The visualisation capabilities of the PEPA Eclipse Plug-in were

very helpful in enabling us to communicate the meaning of the model to partners in the project who were not familiar with process calculi and stochastic processes.

A more complete description of the SENSORIA Finance Case Study appears in [9].

3 The Argument for Visualisation

The approach adopted in this paper for visualising PEPA models differs from the approach taken in graphical modelling formalisms such as Petri nets and Stochastic Activity Networks (SANs), where the visual representation and layout of the model is central. In these formalisms, the modeller most often creates a manual layout of the model — this is the case for the PIPE Petri net editor [4] and the SAN editor of Möbius [11]. Other, mostly textual, representations of the model, such as XML or program source code, are generated from the graphical representation. Some tools for stochastic process algebras — most notably CASPA [22] and Aemilia [6] — have also used this approach to provide a graphical formalism as an alternative to the textual language.

In contrast to this, we wanted the *textual representation* of the PEPA model to be considered as the model source — having primary importance — and for graphical representations to be automatically derived from this and have secondary importance. The idea is to use automatic layout algorithms to generate a first attempt at a layout, which can then be manually improved by the user according to their aesthetic sensibilities and tastes. The manually improved version is automatically saved so that it does not need to be redone after every edit. We believe that this is a good pragmatic compromise between a fully manual and a fully automatic approach to visualisation.

It is important to us that the graphical representation of the PEPA model should be based on the components that PEPA uses to structure large models. The visualisation of the model is only helpful if the user can meaningfully interpret the visualisation. From earlier work [8], we have seen that each component of the model is a coherent unit of behaviour, and we believe that using this to structure the visualisation is more comprehensible than looking at the underlying CTMC. In other words, if we show the synchronisation between components by expanding out the model, then the graphical representation becomes too large, and difficult to understand. Finding a good solution to this problem is an important aspect of future work.

Given this component-level perspective, it is important that we visualise both the structural and stochastic information in each component. This means that rates must play an essential part in the visual representation, so that we can detect errors in rate definitions, such as placing the decimal place in an unintended position. Similarly, we should have some way of noticing if we incorrectly declare an activity to be passive rather than active. Our solution is to use different colours to distinguish active and passive activities, and different shades of colour to distinguish fast and slow activities.

4 Visualisation of PEPA Models

The PEPA Eclipse Plug-in allows us to edit and analyse PEPA models using the popular Eclipse framework [12]. This separates the user interface into two main parts. The *editor* window is where the PEPA model is displayed, and can be edited. This is basically a

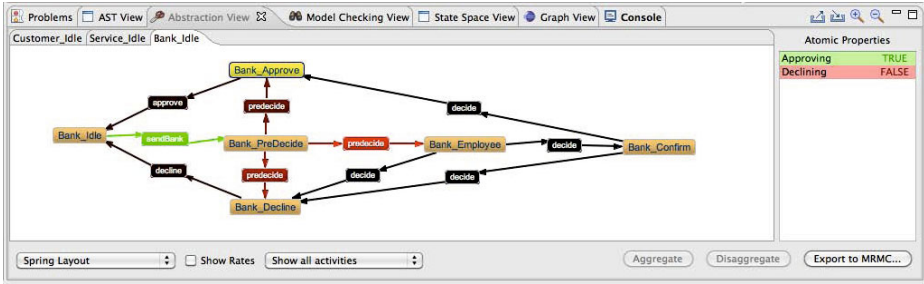


Fig. 3. The abstraction view

text editor, with additional features such as syntax highlighting and identification of parse errors. Alongside the editor are a number of *views*, which are used to display information about the model, and as an interface for invoking analyses of the model. This can be seen in Figure 1, where the editor is positioned centrally, with the views below it and to the right. In this section, we will look at the **abstraction view** (shown at the bottom of the screen), which has a graphical interface for viewing PEPA models, built on top of the Eclipse Graphical Editing Framework (GEF) and the Zest toolkit.

The idea of visualising PEPA models is not a new one, and was first proposed in [29]. Since PEPA is a compositional language, we can view each component in the model independently, as an automaton whose transitions are labelled by activities. In [29] it was suggested that we can do this by displaying the derivation graph of a component. We use a slightly different approach based on the Kronecker representation described in the Section 2 — this is entirely equivalent in terms of what the user sees, but the underlying data structure is more versatile, as we shall discuss in Section 6.

Figure 3 shows the abstraction view, displaying the *Bank* component from Figure 2. There are four essential features of this view:

1. **Visualising:** In the main panel of the view is a series of tabs, which display an automaton for each sequential component in the PEPA model. This allows a fast visualisation of the component as described, so that certain errors in the model can be seen immediately — for example, if the component is supposed to cycle between a number of states, we expect to see a cyclic automaton.

Since an individual PEPA component typically has a small number of states, its automaton will usually be small enough to display clearly. However, for larger or more complicated components, we have a feature to display only certain states. Either we can manually select the states we are interested in, right-click, and select ‘Only show selection’, or we can right-click on a blank area and select ‘Choose states to select...’, which offers a dialog box where we can select states by name.

Active and passive transitions have different colours (red and green respectively) so that they can be more easily distinguished. Moreover, we display active transitions in varying shades of red, ranging from bright red (for the fastest transitions) to black (for the slowest transitions). From a drop-down box, we can select whether to label the transitions with their activities — either for all transitions or for only certain transitions (such as the passive ones, or the fastest active ones).

2. **Labelling:** On the right-side of the view is a list of atomic properties, which can be referred to in performance properties. These can be thought of as *labels*, which identify a set of states in the model using a more human-readable name.

To define a new label, we first select the states that we want to label, and then right-click in the atomic properties list, and select ‘New property’. We can change which properties a state is labelled with by right-clicking on it, which gives a list of properties that can be selected, or deselected. Clicking on a state will display (in the atomic properties list) which properties are true or false, and clicking on a property selects all the states with that label. We specify atomic properties compositionally.

3. **Abstracting:** In order to analyse the model, we may want to reduce its size by first *aggregating* certain states. In the back-end of the tool, compositional abstraction techniques based on abstract Markov chains [25] and stochastic bounds [26] are used to construct an abstract model that is passed to the model checker for analysis. The front-end interface for this is very simple — the user simply has to select the states they want to aggregate, and click the ‘Aggregate’ button. These states can subsequently only be selected as a unit, and moved (or labelled) together.
4. **Exporting:** We allow the model to be exported to the input format of the MRMC model checker [18]. If the model has not been abstracted, the output is a CTMC (consisting of a `.lab` and `.tra` file), otherwise the output is a CTMDP (a `.lab` and a `.ctmdp` file). This means that MRMC can be used as an alternative to the in-built model checker in the plug-in. Note, however, that MRMC currently only supports time-bounded reachability properties for CTMDPs, whereas the in-built model checker supports all the CSL operators¹.

When we create a new PEPA model, or load a new model that we have not worked with before, the abstraction view uses an automated layout algorithm from the Zest toolkit. The idea is to provide a rough initial layout that can be changed by the user to one of their liking. Activity labels are automatically placed so that multiple transitions between two states do not overlap with one another. The layout information and defined properties are *automatically saved* (to an XML format) by the tool, so that if we return to a model in the future, the abstraction view looks precisely as we left it.

An important feature of the abstraction view is that it is *robust* with regard to minor changes to the model. For example, if we add a new component in the system equation, the layout information for the existing components is preserved. If we add a new state to a sequential component, the node appears in the default location (the top-left corner of the view), but the other nodes remain in their correct position. If the tool detects a new state for which it has no information, it displays a warning, and sets all atomic propositions to be true for that state, by default.

5 Constructing Performance Properties

There are a number of ways to describe and analyse performance properties of a PEPA model. Sometimes, we want to directly analyse a simple property, such as the steady state probability of a set of states, or the throughput of a given action. In this case, the

¹ Except the time-bounded next operator, since this is not preserved after uniformisation.

plug-in provides a simple interface for obtaining such information. We often want to ask more sophisticated questions, however, and so we need a more powerful language to describe it. The PEPA Eclipse Plug-in supports two ways of doing this: stochastic probes [10], which use a regular-expression syntax to query the passage time distribution between events, and Continuous Stochastic Logic (CSL) [2], described here.

There are two types of CSL properties: state properties Φ , which concern a state in the model, and path properties φ , which concern a sequence of states (a path) in the model. Together, these allow the logic to specify many useful performance properties, which can be analysed using a model checker. The plug-in has a built-in CSL model checker but, as described in the previous section, it is also possible to export to MRMC.

The syntax of CSL supported by the PEPA Eclipse Plug-in is as follows (where we include derived operators):

$$\begin{aligned} \Phi &::= \text{tt} \mid \text{ff} \mid a \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \Phi \Rightarrow \Phi \mid \neg\Phi \mid \mathcal{S}_{\leq p}(\Phi) \mid \mathcal{P}_{\leq p}(\varphi) \\ \varphi &::= X \Phi \mid \Phi U^I \Phi \mid F^I \Phi \mid G^I \Phi \end{aligned}$$

where $\leq \in \{\leq, \geq\}$, $a \in AP$, $p \in [0, 1]$, and $I = [a, b]$ is a non-empty interval over the reals, such that $a, b \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, and $a \leq b$.

Rather than give the formal semantics of CSL in this paper, we will consider the five types of state property that we most commonly construct:

Steady State	$\mathcal{S}_{\geq p}(\Phi)$	In the long-run behaviour of the model, does Φ hold with probability at least p ?
Next	$\mathcal{P}_{\geq p}(X \Phi)$	In the next state, does Φ hold with probability at least p ?
Until	$\mathcal{P}_{\geq p}(\Phi_1 U^{\leq t} \Phi_2)$	Will Φ_2 become true no later than time t , and Φ_1 hold at all times until this point, with probability at least p ?
Eventually	$\mathcal{P}_{\geq p}(F^{\leq t} \Phi)$	Will Φ become true no later than time t with probability at least p ?
Globally	$\mathcal{P}_{\geq p}(G^{\leq t} \Phi)$	Will Φ always be true until time t with probability at least p ?

At the top level, we also support *quantitative* CSL properties, of the form $\mathcal{S}_{=?}(\Phi)$ and $\mathcal{P}_{=?}(\varphi)$, which return the actual probability of the given steady state or path property, rather than comparing it to a fixed value. These are very useful in practice.

In general, since we support model checking of abstracted models (i.e. CTMDPs in addition to CTMCs), we use a three-valued variant of CSL. That is to say, a property can be true, false, or *maybe*. Similarly, a quantitative property returns an *interval* of probabilities — so, if we get $[0.1, 0.3]$, we know that the probability in the original model is between 0.1 and 0.3. If a qualitative property returns ‘maybe’, or a probability interval is too wide, we should experiment with different abstractions to achieve better results. The theory underlying the tool [25, 26] guarantees the accuracy of the model checker, but the precision depends on the choice of abstraction.

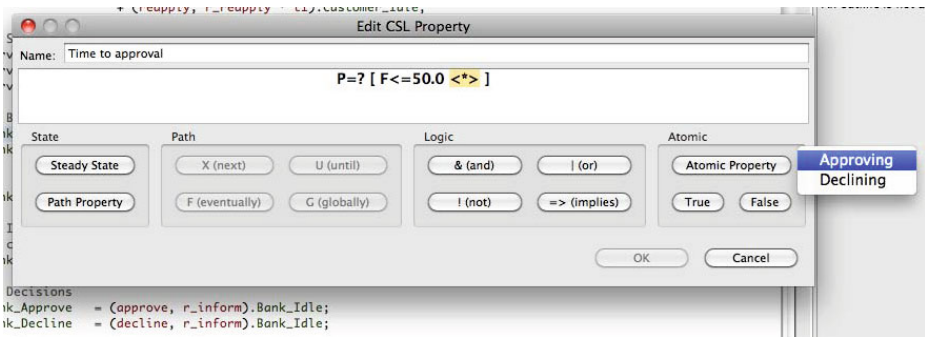


Fig. 4. The CSL property editor

To specify CSL properties and perform model checking, the plug-in provides a **model checking view**. This is basically a table of CSL properties, which can be verified using the internal model checker. We allow saving and loading of properties, in an XML format, so that the same properties can be shared between different models. The most interesting feature, however, is our novel interface for constructing CSL properties.

There are a number of approaches when it comes to helping a user specify a performance property. One approach is to provide a graphical, user-friendly language for specifying properties, such as performance trees [28]. Another is to have a simple dialog box where the user types in the property in a logic like CSL, such as in PRISM [19]. Our approach lies somewhere in between, in that we *do* expect the user to be expert enough to understand CSL, but we *do not* expect them to know the specific syntax of the tool — the interface supports the user by providing them with the correct syntax.

Figure 4 shows the CSL editor, where the property $P_{=?}(F^{\leq 50}$ “Approving”) is being entered for the example in Figure 2 — querying the probability that a loan is approved within 50 time units. We cannot type the property by hand, but instead are guided by the enabled buttons, corresponding to CSL terms that can be used in the current position (there are keyboard shortcuts for experienced users). When we click on part of the property, the editor determines which term we clicked on, and highlights the entire term. We then have the option to substitute it with another term — if no term has yet been entered, or we delete a term, the placeholder ‘<*>’ is seen. Numerical parts of the property can be edited directly, but will only be accepted if syntactically correct.

The most useful feature of the editor is that it is linked to the abstraction view. When we click on the ‘Atomic Property’ button, we see a list of all the labels that have been defined for the model. This avoids us having to switch back and forth between the two views. Because the underlying data structures are shared between the two views, if we change the name of an atomic property in the abstraction view, it is immediately updated in the model checking view. The plug-in prevents us from deleting an atomic property if it is in use, and displays an error message if we try to do so.

Since we can load CSL properties, the plug-in also has a built-in parser for CSL. Only syntactically correct properties will be loaded, and if any atomic properties are used that were not defined, a warning is given and they are replaced with ‘true’.

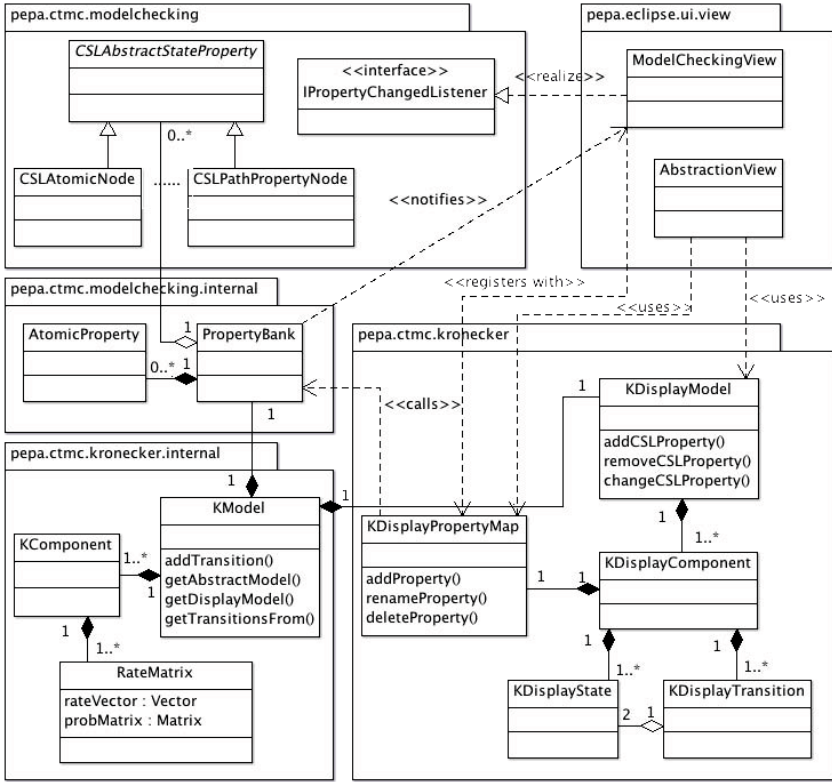


Fig. 5. A UML class diagram of the main classes involved in visualisation in the plug-in

6 Implementation Details

The key idea behind the implementation of the visualisation, abstraction, and model checking functionality of the plug-in is the Kronecker representation described in Section 2. We *partially* derive the state space of the PEPA model when it is parsed, constructing the CTMC component for each sequential component and action type. We only derive the state space of the model when we perform model checking on it — it should be noted that we do this by exploring the reachable state space, and *not* by explicitly performing the matrix operations described in Section 2!

Figure 5 gives an overview of the important classes used by the visualisation part of the plug-in. The most important point is that there are *two* data structures describing the PEPA model. `KModel` is the back-end data structure, used by the abstraction and model checking engines, and stores the Kronecker representation of the model. A `KModel` contains a number of `KComponent`s, which in turn contain a number of `RateMatrix` objects — one for each action type in the model. A `RateMatrix` consists of a vector of rates and a matrix of probabilities. Essentially, it corresponds to a CTMC component, although we have to be a little careful about the mapping between states in the component and indices in the matrix (omitted from the UML diagram).

`KDisplayModel` is the front-end data structure, used by the abstraction and model checking views in Eclipse. A `KDisplayModel` object contains a number of components (`KDisplayComponent` objects), which contain themselves a collection of states (`KDisplayState`) and transitions (`KDisplayTransition`)—an implicitly-linked graph data structure, as opposed to a matrix. This makes it easier to map onto a `Zest Graph` object in order to actually draw the graph in the abstraction view.

To understand this separation, it is necessary to explain the structure of the tool. The PEPA Eclipse Plug-in is quite a complicated piece of software, but the most important functionality is separated into two modules (called OSGi bundles) — `pepa`, which contains all the back-end functionality such as parsing, model checking, and CTMC solvers, and `pepa.eclipse.ui`, which contains the front-end Eclipse functionality, including the PEPA editor and the various views. Only certain classes in the `pepa` bundle are made externally visible, to minimise the coupling between different bundles. This means that `pepa.eclipse.ui` only has access to the classes it actually needs, and is unaware of the internal data structures for the Kronecker representation.

This separation of concerns differs from the standard model-view-controller design pattern, in that the two representations of the model are static. Whenever we modify the PEPA model, we need to parse it again, and this creates a new `KModel` and `KDisplayModel`. This is necessary, because a small change in the source file can result in a radically different model. It does *not*, however, mean that all the information from the previous version of the model is lost — information such as labels and the layout of the graphical view are stored by the plug-in in an XML file, which is then re-loaded so that the data can be re-attached to the model as closely as possible.

Atomic properties are managed through a `KDisplayPropertyMap` object, which is associated with a `KDisplayComponent`. A new atomic property is created by a request to the `PropertyBank`. This creates a new `AtomicProperty` object, which records which states (in each component) are labelled with the property². This is hidden from the abstraction view, and must be accessed through a `KDisplayPropertyMap`.

Because CSL properties are managed at the level of the entire model (rather than for each component), they are created and modified through `KDisplayModel`. Again, the `PropertyBank` is responsible for storing the properties (which are instances of a subclass of `CSLAbstractStateProperty`), and keeps a link between the abstract syntax of the CSL property, and the actual atomic properties. The abstraction and model checking views in Eclipse register with the `PropertyBank` (through `KDisplayModel`) as a *listener* (implementing `IPropertyChangeListener`). This ensures that they are notified of any changes, so that the abstraction and model checking views remain synchronised with one another.

7 Related Work

Our visualisation has taken the textual representation of a PEPA model as the primary source in order to be compatible with other modelling and analysis tools that process PEPA models, such as IPC [5] and GPA [27]. The PEPA language has enjoyed a wide

² Atomic properties are compositional, which means that a state in the model satisfies an atomic property if and only if each sequential component is in a state that is labelled with the property.

range of tool support from the PEPA Workbench [15] to PRISM [19] and the PEPA Eclipse Plug-in, which influences decisions about matters such as visualisation.

The PEPA Workbench contained a single-step navigator, which provided a visualisation of the PEPA model for behavioural debugging (i.e. finding deadlocks or dead code where actions of the model can never fire). This did not show any quantitative information, however, and so did not help modellers to work out why a probabilistic model-checking formula fails to hold, when they think it should.

Other attempts to add a graphical dimension to PEPA have included the DrawNET editor, which allowed the user to create a PEPA model by editing it graphically [14]. DrawNET provides graphical editors for both the parallel composition language of PEPA and the sequential component sub-language. It has also been used to build a graphical interface for the process algebra Aemilia [6].

Other stochastic process algebras also have graphical editors. CASPA [22] uses the Eclipse Graphical Modelling Framework (GMF) [13], which is built on top of GEF, to provide an interface for constructing and editing models. This allows the textual representation of the model to be exported from, and imported into the graphical representation, but the two representations are stored separately. This is in contrast to the PEPA Eclipse Plug-in, where we *only* store the textual representation of the model — generating the graphical representation whenever the model is loaded — which removes the problem of keeping the two representations synchronised. It should be noted that our interface is built on top of GEF directly, and not GMF — this is so that we can make the interface cleaner and simpler, since we do not currently support editing.

A more general modelling tool with a graphical front-end is TAPAs [7]: a didactic tool for the analysis of process algebra. The idea was to develop a framework in which new process algebras can be easily added. At present TAPAs implements CCSP (a process algebra with features of both Milner's CCS and Hoare's CSP) and PEPA.

8 Conclusions

Both textual and graphical performance modelling formalisms have their advantages and disadvantages, but the use of one does not necessarily have to preclude the other. PEPA is a highly successful language-based approach to performance modelling, and yet there are great benefits from being able to *visualise* a model in a more graphical way. To this end, we have created a novel interface for visualising PEPA models, as part of the PEPA Eclipse Plug-in. The latest version is available for download from <http://www.dcs.ed.ac.uk/pepa/tools/plugin>.

In future work, there are many additional ways in which we can increase the functionality of the plug-in. One idea would be to allow the model to be edited via the graphical interface, rather than just being viewed passively — for example, by making use of the editing functionality of GMF. This would require us to continually maintain the synchronisation between the editor and the abstraction view, but would lead us in the direction of a truly combined textual/graphical approach to modelling. Furthermore, there is a great deal of scope for advanced visualisation, such as illustrating how components interact with one another, and animating the *dynamic behaviour* of the model.

To summarise, visualisation is a powerful tool in performance modelling, even for experienced modellers, as it allows a better understanding of the model — particularly

in the face of sophisticated transformations such as state-space aggregation and other abstraction techniques. To the best of our knowledge, this is a unique feature of our modelling tool, and we hope that it will be a benefit to the performance modelling community, and provide inspiration for similar features in other modelling tools. Seeing may not always be believing, but it certainly helps in understanding!

Acknowledgements. A prototype implementation of some improvements to the visualisation of PEPA models in was completed by Nan Ai in his Master's thesis [1]. This was very helpful for allowing us to generate and assess different ideas related to visualisation. The implementation of the visualisation, abstraction, and model checking features of the plug-in was supported initially by a Microsoft Research European PhD Scholarship, and subsequently by the Danish Research Council (FTP grant 09-073796). The PEPA case study example included in this paper was created by Allan Clark while working on the EU FET-IST Global Computing 2 project SENSORIA ("Software Engineering for Service-Oriented Overlay Computers" (IST-3-016004-IP-09)).

References

1. Ai, N.: An Enhanced Abstraction View for the PEPA Eclipse Plug-in. Master's thesis, School of Informatics, The University of Edinburgh (2010)
2. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
3. Balbo, G.: Introduction to stochastic petri nets. In: Brinksma, E., Hermanns, H., Katoen, J.-P. (eds.) EEF School 2000 and FMPA 2000. LNCS, vol. 2090, pp. 84–155. Springer, Heidelberg (2001)
4. Bonet, P., Llado, C.M., Puijaner, R., Knottenbelt, W.J.: PIPE v2.5: A Petri Net Tool for Performance Modelling. In: 23rd Latin American Conference on Informatics (2007)
5. Bradley, J.T., Dingle, N.J., Gilmore, S.T., Knottenbelt, W.J.: Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler. In: Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, pp. 344–351. IEEE Press, Los Alamitos (2003)
6. Calvarese, F., Di Marco, A., Malavolta, I.: Building graphical support for Aemilia ADL. Technical Report TRCS 008/2007, University of L'Aquila (2007)
7. Calzolari, F., De Nicola, R., Loreti, M., Tiezzi, F.: TAPAs: A tool for the analysis of process algebras. In: Jensen, K., van der Aalst, W.M.P., Billington, J. (eds.) Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, pp. 54–70. Springer, Heidelberg (2008)
8. Canevet, C., Gilmore, S., Hillston, J., Prowse, M., Stevens, P.: Performance modelling with UML and stochastic process algebras. IEE Proceedings: Computers and Digital Techniques 150(2), 107–120 (2003)
9. Cappelletto, L., Clark, A., Gilmore, S., Latella, D., Loreti, M., Quaglia, P., Schivo, S.: Quantitative analysis of services. In: Rigorous Software Engineering for Service-Oriented Systems. Springer, Heidelberg (2011)
10. Clark, A., Gilmore, S.: State-aware performance analysis with eXtended stochastic probes. In: Thomas, N., Juiz, C. (eds.) EPEW 2008. LNCS, vol. 5261, pp. 125–140. Springer, Heidelberg (2008)
11. Daly, D., Deavours, D.D., Doyle, J.M., Stillman, A.J., Webster, P.G., Sanders, W.H.: Möbius: An extensible framework for performance and dependability modeling. In: Multi-Workshop on Formal Methods in Performance Evaluation and Applications (1999)

12. The Eclipse platform, <http://www.eclipse.org>
13. The Eclipse Graphical Modeling Framework (GMF), <http://www.eclipse.org/modeling/gmf/>
14. Gilmore, S., Gribaudo, M.: Graphical modelling of process algebras with DrawNET. In: Proceedings of the Tools Appendix to the International Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems (2003)
15. Gilmore, S., Hillston, J.: The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In: Haring, G., Kotsis, G. (eds.) TOOLS 1994. LNCS, vol. 794, pp. 353–368. Springer, Heidelberg (1994)
16. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, Cambridge (1996)
17. Hillston, J., Kloul, L.: An efficient kronecker representation for PEPA models. In: de Luca, L., Gilmore, S. (eds.) PROBMIV 2001, PAMP-PROBMIV 2001, and PAMP 2001. LNCS, vol. 2165, pp. 120–135. Springer, Heidelberg (2001)
18. Katoen, J.-P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems (QEST), pp. 243–244. IEEE Press, Los Alamitos (2005)
19. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
20. Movaghar, A., Meyer, J.F.: Performability modelling with stochastic activity networks. In: Proceedings of 1984 Real-Time Symposium, pp. 8–40 (1984)
21. Plateau, B.: On the stochastic structure of parallelism and synchronization models for distributed algorithms. SIGMETRICS Performance Evaluation Review 13(2), 147–154 (1985)
22. Riedl, M., Schuster, J., Siegle, M.: Recent extensions to the stochastic process algebra tool CASPA. In: Proceedings of the 5th International Conference on the Quantitative Evaluation of Systems (QEST), pp. 113–114. IEEE Press, Los Alamitos (2008)
23. SENSORIA Web site. SENSORIA: Software engineering for service-oriented overlay computers (2011), <http://www.sensoria-ist.edu>
24. Smith, M.J.A.: Abstraction and model checking in the PEPA plug-in for Eclipse. In: Proceedings of the 7th International Conference on the Quantitative Evaluation of Systems (QEST), pp. 155–156. IEEE Press, Los Alamitos (2010)
25. Smith, M.J.A.: Compositional abstraction of PEPA models for transient analysis. In: Aldini, A., Bernardo, M., Bononi, L., Cortellessa, V. (eds.) EPEW 2010. LNCS, vol. 6342, pp. 252–267. Springer, Heidelberg (2010)
26. Smith, M.J.A.: Compositional abstractions for long-run properties of stochastic systems. In: Proceedings of the 8th International Conference on the Quantitative Evaluation of Systems (QEST). IEEE Press, Los Alamitos (2011)
27. Stefanek, A., Hayden, R.A., Bradley, J.T.: GPA — Tool for rapid analysis of very large scale PEPA models. In: Proceedings of the 26th UK Performance Engineering Workshop (UKPEW), pp. 91–101 (2010)
28. Suto, T., Bradley, J.T., Knottenbelt, W.J.: Performance trees: A new approach to quantitative performance specification. In: MASCOTS 2006, 14th International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 303–313. IEEE Press, Los Alamitos (2006)
29. Thomas, N., Munro, M., King, P., Pooley, R.: Visual representation of stochastic process algebra models. In: Proceedings of the 2nd International Workshop on Software and Performance (WOSP), pp. 18–19. ACM, New York (2000)
30. Tribastone, M., Duguid, A., Gilmore, S.: The PEPA Eclipse plugin. SIGMETRICS Performance Evaluation Review 36(4), 28–33 (2009)

Efficient Experiment Selection in Automated Software Performance Evaluations

Dennis Westermann, Rouven Krebs, and Jens Happe

SAP Research, Karlsruhe, Germany

{dennis.westermann, rouven.krebs, jens.happe}@sap.com

Abstract. The performance of today's enterprise applications is influenced by a variety of parameters across different layers. Thus, evaluating the performance of such systems is a time and resource consuming process. The amount of possible parameter combinations and configurations requires many experiments in order to derive meaningful conclusions. Although many tools for automated performance testing are available, controlling experiments and analyzing results still requires large manual effort. In this paper, we apply statistical model inference techniques, namely Kriging and MARS, in order to adaptively select experiments. Our approach automatically selects and conducts experiments based on the accuracy observed for the models inferred from the currently available data. We validated the approach using an industrial ERP scenario. The results demonstrate that we can automatically infer a prediction model with a mean relative error of 1.6% using only 18% of the measurement points in the configuration space.

1 Introduction

Performance engineering is a crucial discipline throughout development and hosting of enterprise applications. However, the sheer size and complexity of software systems and development processes hinders the application of performance engineering in many cases. Especially in large enterprise applications, the performance of a system is affected by a variety of parameters. Understanding their influences (and capturing them in a performance model) requires a huge number of experiments and "what-if" analyses in order to draw meaningful conclusions.

State-of-the-art performance engineering research approaches [14] use architectural information and detailed performance behavior descriptions in order to build prediction models. In most cases, the performance models are a combination of simulation models built using domain-specific languages and measurements to calibrate, validate or extend the models [3,9,12,20]. In industrial practice, performance measurements are, for example, used to benchmark systems, customize configuration settings, or test the quality of a new release before shipment [26,28]. In both cases, the amount of possible parameter combinations and configurations makes the measurement process time and resource consuming. While many tools provide automation for generating load and getting monitoring information there is still a lot of manual effort remaining to analyze the measured data and to decide how many and which measurements to conduct in order to reach a certain goal (e.g., finding a performance-optimized configuration).

In this paper, we present a fully automated approach that (i) selects and conducts experiments, (ii) uses statistical inference techniques to derive a prediction model based

on the measured data, (iii) validates the prediction model, and (iv) iteratively determines new experiments that maximise the information gain and thus increase the accuracy of the model. The statistical inference techniques that we use in our experiments are Multivariate Adaptive Regression Splines (MARS) [7] and Kriging [29]. MARS has already been successfully applied for software performance analyses [3,6,10]. Kriging is a geostatistical interpolation technique that has been applied to various research areas dealing with spatial data. However, to the best of our knowledge Sacks et al. [25] are the only ones that applied Kriging to analyze data measured in computer system experiments. The strength of both methods is that they provide robust predictions and do not require any prior knowledge about the underlying dependencies in the data (e.g., in contrast to simple linear regression).

The contributions of this paper are (a) the description and comparison of three automated experiment selection methodologies for the efficient derivation of statistical performance prediction models and (b) the application of the Kriging interpolation technique for software performance analyses.

We validate our approach in two case studies. The results demonstrate that adaptive experiment selection can yield accurate prediction models with a significantly reduced amount of measurements. Moreover, we show that the geostatistical interpolation technique Kriging can be applied for the analysis of performance measurements. In fact, Kriging outperforms MARS for some problem classes.

The remainder of this paper is organized as follows. Section 2 gives an overview of our ongoing research and brings this paper into line with our overall motivation. In Section 3, we discuss related research approaches. Section 4 provides basics of statistical model inference using MARS and Kriging. In Section 5, we describe the three experiment selection algorithms that we apply in our approach. A real-world case study as well as detailed validation results are illustrated in Section 6. Finally, Section 7 concludes the paper.

2 Motivation and Overview

In this section, we give an overview of our overall approach for performance predictions of enterprise applications. Applying Software Performance Engineering (SPE) in practice is still a challenging task. In most cases, software vendors built their applications on a large basis of existing components such as middleware, legacy applications, or third-party services. Software architects are facing questions like "How does middleware A affect the performance of my application?", or "Will the application under development meet the performance requirements?". A software as a service provider wants to know, for example, "What happens to the performance of my system if I double the amount of underlying virtual machines?" or "What happens to performance if the number of users increases?". Existing model-driven performance engineering approaches mainly realise a pure top down prediction approach. Software architects have to provide a complete model of their system in order to conduct performance analyses. Measurement-based performance evaluations, by contrast, depend on the availability of the application and can only be applied in late development cycles. Our approach aims at integrating model-driven and measurement-based performance predictions in order to build practical performance models of enterprise applications.

2.1 Software Performance Curves

The main idea of our approach is to apply goal-oriented, systematic measurements to already existing parts of a system. The result of the systematic measurements is a quantification of the dependencies between the system's usage (workload and parameters) and performance (timing behavior, throughput, and resource utilization). We refer to the statistical models describing these dependencies as software performance curves. Formally, a performance curve describes the performance \mathcal{P} (response time, throughput, and resource utilisation) of a system in dependence on a set of input parameters A_1, \dots, A_n with $n \in \mathbb{N}$. It is a function $f: A_1 \times A_2 \times \dots \times A_n \rightarrow \mathbb{R}$, where each input parameter A_i is a number ($\subset \mathbb{R}$), an enumeration, or a boolean value. The function's result represents the performance metric of interest. The benefit of these statistically inferred models is that they do not require specific knowledge on the internal structure of the system under study (e.g., in contrast to other approaches that use statistical inference to estimate parameters of queuing networks [15,17,22]). Thus, the software performance curves are a black-box description of the performance behaviour of the system under test. The derivation of the performance curves requires the execution of many measurements in different settings. Therefore, we developed a framework called Software Performance Cockpit [33,32] that encapsulates best practices and allows for separation of concerns regarding the different aspects of a performance evaluation. Using this framework we can automatically control measurements, trigger analyses and export results [34].

2.2 Integrating Software Performance Curves and Model-Driven Performance Analyses

In order to use the software performance curves to decide on design alternatives, plan capacities, or identify performance critical system configurations we propose to integrate the curves with model-driven performance engineering approaches (such as surveyed in [1] and [14]). Figure 1 illustrates the approach.

For those parts of a system that are under development we apply an existing approach for model-driven performance engineering [2]. Software architects specify the system's components, behaviour, deployment, and usage (*System Modelling*). This activity results in a *System Model* that describes the newly developed parts as well as its usage. In order to consider the effect of existing parts in a performance analysis, we need to include them in the prediction model. The *Measurements* described above result in *Performance Data* of the system. Such data can be used for *Model Inference*. The resulting software performance curves consider the effect of system external parts on performance, these models have to be integrated with or made available in model-driven prediction approaches (*Integration*). This step merges both model types and creates a common basis for further performance analysis (*Prediction*). Based on the *Performance Predictions*, software architects and performance analysts can decide about design alternatives, plan capacities, or identify critical components.

In this paper we focus on the *Measurement* and *Model Inference* parts. In our future work, we will describe how performance curves can be integrated with the Palladio Component Model (PCM) [2].

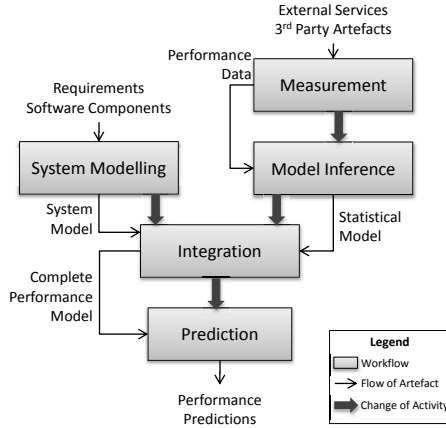


Fig. 1. Overview of integrating model-driven and measurement-based performance analysis

3 Related Work

In this section, we present related work in the area of measurement-based performance analysis. Various approaches explore the influence of different parameters on the performance of software applications. The authors focus on the instrumentation itself [5,13,19] or use the results to build performance models (or tests) [35,24,30,12] or detect errors [19,20].

Reussner et al. [24] introduce an approach to benchmark and compare different OpenMPI implementations. Their approach combines performance metrics with linear interpolation techniques to assess the implementation’s overall performance behaviour. To maximise the information gain of subsequent experiments, they identify those points with the (potentially) largest error in the current prediction model. While this approach presents one of the starting points of our work, it is limited to the evaluation of a single parameter and simple linear interpolation techniques that are not suited for multi-dimensional scattered data. Another starting point for our work is the approach of Woodside et al. [35] and Courtois et al. [3]. They introduce a workbench for automated measurements of resource demands in dependence of configuration and input parameters. The results are fitted by different statistical methods resulting in so-called resource functions that capture performance metrics with respect to the given parameters. However, the authors did not compare different experiment selection methodologies or different analysis methods.

Denaro et al. [5] propose an approach for early performance testing of distributed applications. Their core assumption (similar to Gorton et al. [8]) is that the middleware is the determining factor of an application’s performance. However, the usage of middleware features (like transaction or persistence) is determined by the application. Therefore, Denaro et al. use architecture designs to derive performance test cases that can be executed and used to estimate the applications performance in the target environment. Gorton et al. also conduct measurements in the target environment but use the

results to calibrate a prediction model which is then used to predict the application's performance. Both approaches do not explicitly evaluate the influence of parameters of configurations. The measurements are focused on specific scenarios. While this is sufficient for the author's purposes, it is not enough to capture the influence of different configurations and input parameters on performance.

In [12], Jin et al. introduce an approach called BMM that combines **Benchmarking**, production system **Monitoring**, and performance **Modelling**. Their goal is to quantify the performance characteristics of real-world legacy systems under various load conditions. However, the measurements are driven by the upfront selection of a performance model (e.g layered queuing network) which is later on built based on the measurement results.

Miller et al. [19] propose Paradyn, a tool for the automatic diagnoses of performance problems. They apply dynamic instrumentation to control the instrumentation in search of performance problems. Paradyn starts looking for high-level problems for a whole application and, once the general problem is found, inserts further instrumentations to find more specific causes. Miller et al. focus on the detection of performance problems and do not measure parameter spaces systematically.

4 Statistical Model Inference

Statistical model inference is the process of learning from data [11]. A variety of methodologies have been developed in statistical science [11,18,21] in order to extract patterns and trends from data or to fit curves to the data. In this paper, we focus on so called supervised learning problems [11] where the goal is to predict the value of an observed metric based on a number of input parameters. The different statistical methods have their own characteristics that mainly differ in their degree of model assumptions. For example, linear regression makes rather strong assumptions on the model underlying the observations (they are linear) while the nearest neighbor estimator makes no assumptions at all. Most other statistical estimators lie between both extremes. Methods with stronger assumptions, in general, need less data to provide reliable estimates, if the assumptions are correct. Methods with less assumptions are more flexible, but require more data. In the course of this paper, we apply and compare two different methodologies, namely MARS and Kriging. Both methods are able to deal with the assumption that we have less or no knowledge about the structure of the data. MARS has already been successfully applied in software performance prediction [3,6,10]. Geostatistical interpolation methods, such as Kriging, are designed to analyse irregularly spaced set of data points in a three dimensional space [29]. Characteristics of geostatistical data are (i) high costs to get the value of interest for these points and (ii) that near measurements are more interrelated to each other than distant ones [31]. We assume that these characteristics are also true for measured performance data. Measuring a single configuration of an enterprise application often requires extensive effort. Moreover, in most cases a minor change in a configuration has less effect on the performance metric of interest than larger changes. Furthermore, the adaptive experiment selection method presented in Section 5.3 creates an irregularly spaced set of data points. For this reasons, we decided to investigate the use of Kriging to derive software performance curves. In the following, we briefly introduce the two methods.

4.1 Kriging

Kriging is a generic name for a family of spatial interpolation techniques using generalised least-squares regression algorithms [18]. It is named after Daniel Krige who applied the method to a mineral ore body [16]. Examples of Kriging algorithms are Simple, Ordinary, Block, Indicator, or Universal Kriging. In [18], the authors provide a comprehensive review of multiple Kriging algorithms as well as other spatial interpolation techniques. Generally, the goal of spatial interpolations is to infer a spatial field at unobserved sites using observations at few selected sites. According to [18], nearly all spatial interpolation methods share the same general estimation formula:

$$\hat{Z}(x_0) = \sum_{i=1}^n \lambda_i Z(x_i)$$

where the estimated value of an attribute at the point of interest x_0 is represented by \hat{Z} , the observed value at the sampled point x_i is Z , the weight assigned to the sampled point is λ_i , and the number of sampled points used for the estimation is represented by n . Furthermore, the semivariance (γ) of Z between two data points is an important concept in geostatistics. It is defined as:

$$\gamma(x_i, x_0) = \gamma(h) = \frac{1}{2} \text{var}[Z(x_i) - Z(x_0)]$$

where h is the distance between point x_i and x_0 and $\gamma(h)$ is the semivariogram (commonly referred to as variogram)[18].

Figure 2 shows an example variogram with an exponential variogram model. The *nugget* (or nugget effect) is a contribution to variability without spatial continuity [29]. The *range* is the distance where the model first flattens out and the *sill* is the value at which the variogram model reaches the range.

The Kriging implementation [23] that we applied in our experiments uses the Ordinary Kriging algorithm to estimate unknown points. As described above the estimated values are computed as simple linear weighted average of neighbouring measured data points. The weights are determined from the fitted variogram with the condition that they must add up to 1 which is equivalent to the process of reestimating the mean value at each new location [4].

4.2 MARS

Multivariate Adaptive Regression Splines (MARS) [7] is a non-parametric regression technique which requires no prior assumption as to the form of the data. The method fits functions creating rectangular patches where each patch is a product of linear functions (one in each dimension). MARS builds models of the form $f(x) = \sum_{i=1}^k c_i B_i(x)$, the model is a weighted sum of basis functions $B_i(x)$, where each c_i is a constant coefficient [7]. MARS uses expansions in piecewise linear basis functions of the form $[x - t]_+$ and $[t - x]_+$. The $+$ means positive part, so that

$$[x - t]_+ = \begin{cases} x - t, & \text{if } x > t \\ 0, & \text{otherwise} \end{cases} \quad \text{and} \quad [t - x]_+ = \begin{cases} t - x, & \text{if } x < t \\ 0, & \text{otherwise} \end{cases}$$

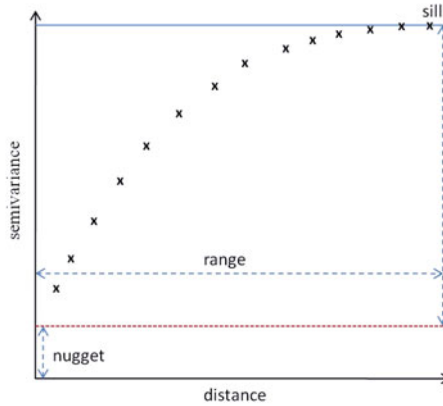


Fig. 2. Sample Variogram

The model-building strategy is similar to stepwise linear regression, except that the basis functions are used instead of the original inputs. An independent variable translates into a series of linear segments joint together at points called knots [3]. Each segment uses a piecewise linear basis function which is constructed around a knot at the value t . The strength of MARS is that it selects the knot locations dynamically in order to optimize the goodness of fit. The coefficients c_i are estimated by minimizing the residual sum-of-squares using standard linear regression. The residual sum of squares is given by $RSS = \sum_{i=1}^N (\hat{y}_i - \bar{y})^2$, where $\bar{y} = \frac{1}{N} \sum \hat{y}_i$, where N is the number of cases in the data set and \hat{y}_i is the predicted value.

5 Experiment Selection

In order to automatically derive a software performance curve with the least possible number of measurements, we need an iterative algorithm that (i) selects new experiments for each iteration, (ii) infers a statistical model based on the available data after each iteration, and (iii) is aware of the quality of the inferred model. In the context of this paper, an experiment (or configuration point) is defined as one configuration of all parameters (i.e., it corresponds to one point in the configuration space). The configuration space is spanned by the configuration parameters and their corresponding domains. In this section, we present three experiment selection methodologies that fulfill the requirements mentioned above by applying different strategies (see Figure 3). The random experiment selection strategy randomly selects a fixed number of new experiments. The equidistant experiment selection strategy splits the parameter space in equidistant areas. The adaptive experiment selection strategy selects new experiments in those areas of the parameter space that show the worst predictions. Each of the three methodologies can be combined with various model inference techniques.

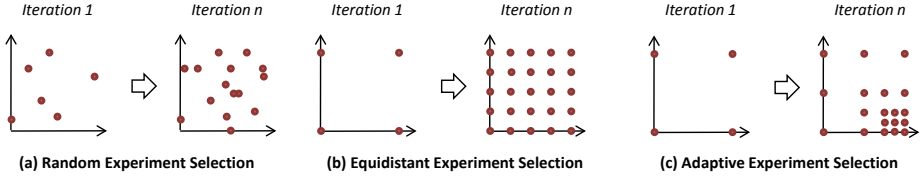


Fig. 3. Experiment Selection Methodologies

5.1 Random Experiment Selection Based on Global Prediction Error

The first algorithm randomly selects new experiments in order to minimize the global prediction error. In each iteration a fixed number of n randomly selected configuration points are measured (see Figure 3 (a)). Based on a set of validation points (measured before the first iteration starts), we calculate the mean relative error (MRE) of the prediction model. The algorithm terminates when the error is below a predefined threshold, a predefined number of measurements has been reached, or a predefined measurement time has expired.

5.2 Equidistant Experiment Selection Based on Global Prediction Error

The second algorithm determines the measurement points for the next iteration by stepwise equidistant splitting of the configuration space (see Figure 3 (b)). We define $P = \{x|x \in \mathbb{R}^i \wedge \forall x_i \in [0..1]\}$ as a set of all possible positions in the configuration space with normalized values. An element $p \in P$ describes one experiment or configuration point. Let the elements $p_1, p_2 \in P$ be two opposing positions that describe the multidimensional configuration space. Function $f_{center} : P \times P \rightarrow P$ returns the center of the two given points which is calculated by the element-wise arithmetic middle of the two vectors. Furthermore, function $f_{edges} : P \times P \rightarrow P^*$ returns the set of all edges of the embraced space defined by two configuration points. The function computes all possible element-wise combinations of the two given configuration points. We use $H_{i,n,z}$ as a set which helps to find the experiments for each iteration. H contains a set of tuples describing areas of the space to be measured, therefore every element in H consists two opposing positions. The process itself does continuously divide the space into equidistant areas based on previously computed areas. Thereby, the measurement iteration is expressed by the index i . The index n is used to iterate over the tuples stored in H . The third index $z \in [1, 2]$ defines which of the two positions stored in every element of H is referenced. Formally, the set of experiments to be executed is computed as follows:

$$\begin{aligned}
 H_0 &= f_{edges}(p_1, p_2) \times f_{center}(p_1, p_2) \\
 H_i &= \bigcup_{n=1}^{|H_{i-1}|} f_{edges}(H_{i-1,n,1}, H_{i-1,n,2}) \times f_{center}(H_{i-1,n,1}, H_{i-1,n,2}) \\
 Experiments_i &= \left(\bigcup_{n=1}^{|H_i|} H_{i,n,1} \cup H_{i,n,2} \right) \setminus Experiments_{i-1}
 \end{aligned}$$

The experiments that will be demanded in the next iteration include all the points within the tuples of H that have not yet been measured.

The termination criteria as well as the validation procedure are the same as those defined for the random selection methodology (see 5.1).

5.3 Adaptive Experiment Selection Based on Local Prediction Error

In contrast to the algorithms described in the previous sections, this algorithm takes the locality and the size of single prediction errors into account when determining experiments for the next iteration (see Figure 3 (c)). We assume that a new experiment at the area with the highest prediction error raises the accuracy of the overall model at most. Another difference to the previous experiment selection methodologies is that it does not include all determined points for an iteration in the training data, but uses a subset of these points for validation. Thus, this methodology does not require the creation of a validation set before the actual iteration starts. In the following, we describe the algorithm in detail. First, we introduce some basic data types, variables and functions followed by a listing of the algorithm. As in Section 5.2, we define $P = \{x | x \in \mathbb{R}^i \wedge \forall x_i \in [0..1]\}$ as a set of all possible positions in the configuration space with normalized values. Elements of P are declared as p . The elements p_1 and p_2 are opposing positions necessary to describe a multidimensional space. Function $f_{center} : P \times P \rightarrow P$ returns the center of the two given points which is calculated by the element-wise arithmetic middle of the two vectors. Furthermore, function $f_{edges} : P \times P \rightarrow P^*$ returns a set of all edges of the embraced space given by p_1 and p_2 .

In addition, let $e \in \mathbb{R}^+$ describe the error of the performance curve at a defined area or position and $S = \{p_1 \times p_2 \times e | p_1 \in P \wedge p_2 \in P \wedge e \in \mathbb{R}^+\}$. Three subsets of S control the measurement progress. A priority-controlled queue $Q \subset S$ contains tuples describing areas in the configuration space, where the error of the curve ran out of the acceptable threshold. The order of priority is based on e . The collection $V \subset S$ is the validation set which contains all the tuples describing areas where a good prediction has already been observed. $M \subset S$ is the training set which contains the measurement results used to create a performance curve. All subsets of S are mutually disjoint and it holds that $S = Q \cup V \cup M$. The function $predict_M : P \rightarrow \mathbb{R}$ creates a prediction results based on the given measurements M for a specific configuration point. The functionality of the method is based on the assumption, that the prediction error of the curve on $f_{center}(p_1, p_2)$ is representative for the error in the spatial field embraced by p_1 and p_2 . The parameter $threshold \in \mathbb{R}^+$ is predefined by the performance analyst and gives an option to control the accuracy and thus the runtime of the method.

- 1: $p_1 = (1, 1, \dots, 1)$
- 2: $p_2 = (0, 0, \dots, 0)$
- 3: $e = \infty$
- 4: $Q \leftarrow \{ \langle p_1, p_2, e \rangle \}$
- 5: **while** sizeof(Q) != 0 **do**
- 6: $T \leftarrow \emptyset$
- 7: $t_{tmp} \leftarrow dequeue(Q)$
- 8: $T \leftarrow T \cup t_{tmp}$

```

9:   while  $Q.first.e = t_{tmp}.e$  do
10:      $t_{tmp} \leftarrow dequeue(Q)$ 
11:      $T \leftarrow T \cup t_{tmp}$ 
12:   end while
13:   for all  $t$  in  $T$  do
14:     measure all points  $f_{edges}(t.p_1, t.p_2)$ , add results to  $M$ 
15:     measure value  $r_m$  at point  $f_{center}(t.p_1, t.p_2)$ 
16:      $r_p \leftarrow predict_M(f_{center}(t.p_1, t.p_2))$ 
17:      $e \leftarrow \frac{r_m}{|r_m - r_p|}$ 
18:     if  $e > threshold$  then
19:       for all  $p_{tmp}$  in  $f_{edges}(t.p_1, t.p_2)$  do
20:          $p_1 \leftarrow f_{center}(t.p_1, t.p_2)$ 
21:          $t_{tmp} \leftarrow \langle p_1, p_{tmp}, e \rangle$ 
22:         enqueue( $Q, t_{tmp}$ )
23:          $M \leftarrow M \cup \langle r_m, f_{center}(t.p_1, t.p_2) \rangle$ 
24:       end for
25:     else
26:        $V \leftarrow V \cup t$ 
27:     end if
28:   end for
29:   for  $t$  in  $V$  do
30:     measure value  $r_m$  at point  $f_{center}(t.p_1, t.p_2)$ .
31:      $r_p \leftarrow predict_M(f_{center}(t.p_1, t.p_2))$ 
32:      $e \leftarrow \frac{r_m}{|r_m - r_p|}$ .
33:     if  $e > threshold$  then
34:        $t.e \leftarrow e$ 
35:        $V \leftarrow V \setminus t$ 
36:        $Q \leftarrow Q \cup t$ 
37:     end if
38:   end for
39: end while
40: for all  $t$  in  $V$  do
41:   measure value  $r_m$  at point  $f_{center}(t.p_1, t.p_2)$ 
42:    $M \leftarrow M \cup \langle r_m, f_{center}(t.p_1, t.p_2) \rangle$ 
43: end for

```

Line 1-4 ensure the preconditions for the actual experiment selection which starts in line 5. The primary control structure is the loop over Q starting in 5. Lines 6-12 deal with the selection of all elements with highest error. Starting at line 13 the loop body executes measurements (line 14) in the area of each selected tuple. Furthermore, it calculates the error for these areas in line 15-17 and defines new subareas to be measured in further iterations (line 18-25) if the error is greater than the defined *threshold*. If the error is less than the *threshold* the current tuple is stored in V at line 26. To provide faster convergence against the underlying performance functions it brings significant advantages to execute this breadth-first approach over all elements with the same e . This ensures to step down in the area with the highest prediction faults. Since nearly

all interpolation or regression techniques cannot absolutely avoid the influence of new elements in M onto preliminary well predicted areas, the validation repository V is checked in line 30-32 for negative effects in areas that have been well predicted before the last modifications. If for any element in V the curve is still not accurate enough it is returned to Q at line 33-37 and thus measured in more detail in later iterations. We expect that the heuristic converges more efficient if a new measurement has only local effects onto the interpolation function. Finally, line 40-43 copies all elements from V to M as the positions where measured before and thus the data is available but not yet added to the training data of the model.

6 Case Study and Validation

In this section, we demonstrate the efficiency of the approach and the accuracy of the inferred prediction models. Moreover, we apply the software performance curves in a "real-world" scenario using a large enterprise application. For the evaluation we formulate the following research questions:

- **RQ1:** To which extent are the experiment selection methodologies presented in Section 5 more efficient (in terms of necessary measurements to create an accurate prediction model) (i) compared to measuring the full configuration space and (ii) compared to other approaches?
- **RQ2:** Are geostatistical interpolation techniques applicable in software performance analysis scenarios? Are they more efficient compared to multivariate regression?
- **RQ3:** Is the approach applicable to automatically create measurement-based performance models of large enterprise applications in a reasonable amount of time?

In the remainder of this section we present two case studies and a discussion of the results. Figure 4 summarizes the results of the two case studies. The table contrasts the different combinations of experiment selection method and analysis method. To determine the quality of the derived prediction model we compared the prediction for each measurement point in the configuration space with its actual value and calculated the mean relative error (MRE).

6.1 Communication Server Case Study

In this case study we applied our approach to a scenario described by Courtois and Woodside [3]. The authors applied a similar adaptive experiment selection approach combined with MARS to derive resource functions for a unicast-based multicast communications server. The basic components of the server are (i) a Supplier Handler that reads, packages, and enqueues incoming messages from the Supplier processes and (ii) a Consumer Router which dequeues a message and sends it to each of its Consumer processes (see [3] for details). The authors derived the following resource function for the Consumer Router component:

$$\begin{aligned}
\text{Consumer-Router-CPU} = & 1436.73 + 0.1314 * h(\text{msgsize} - 1) \\
& - 0.0159 * h(-(\text{consumers} - 9)) * h(\text{msgsize} - 1) \\
& + 808.082 * h(\text{consumers} - 9) - 149.399 * h(-(\text{consumers} - 9)) \\
& - 0.03 * h(\text{consumers} - 9) * h(-(\text{msgsize} - 21091)) \\
& - 0.0092 * h(\text{consumers} - 1) * h(-(\text{msgsize} - 10808)) \\
& + 0.0989 * h(\text{msgsize} - 4223) - 0.01 * h(-(\text{consumers} - 9)) * h(\text{msgsize} - 5424)
\end{aligned}$$

The domain of message sizes was set between 1 and 64K bytes and the number of consumers varied from 1 to 10. Thus, the full configuration space consists of 640 measurement points. For our case study, we tried to fit this function using the same domains for the two parameters. The results (see Figure 4) show that for this case study the combinations RandomSelection/MARS with 89 measurement points (#M) and an average prediction error of 4.1% and AdaptiveSelection/Kriging with 92 measurement points and an error of 2.3% performed best. Thus, the approaches required only 14% of the full configuration space to create a very good prediction model. Compared to the approach presented in [3] which required 157 measurements to build the prediction model (with an error of 8.58%) we saved 65 measurements (i.e., 41%). However, when comparing the two approaches one has to note that we fitted the simulated function and had not to deal with the real measurement data which might cause additional prediction error.

6.2 Enterprise Application Case Study

The goal of this case study is to demonstrate that the approach is applicable on real data measured on a large enterprise application. We address the problem of customizing an SAP ERP application to an expected customer workload. The workload of an enterprise application can be coarsely divided into batch workload (background jobs like monthly business reports) and dialog workload (user interactions like displaying customer orders). This workload is dispatched by the application server to separate operating system processes, called work processes, which serve the requests [28]. At deployment time of an SAP system the IT administrator has to allocate the available number of work processes (depending on the size of the machine) to batch and dialog jobs, respectively. With the performance curve derived in this case study, we enable IT administrators to find the optimal amount of work processes required to handle the dialog workload with the constraint that the average response time of dialog steps should be less than one second. The system under test consists of the enterprise resource planning application SAP ERP2005 SR1, an SAP Netweaver application server and a MaxDB database (version 7.6.04-07). The underlying operating system is Linux 2.6.24-27-xen. The system is deployed on a single-core virtual machine (2,6 GHz, 1024KB cache). To generate load on the system we used the SAP Sales and Distribution (SD) Benchmark. This standard benchmark covers a sell-from-stock scenario, which includes the creation of a customer order with five line items and the corresponding delivery with subsequent goods movement and invoicing. Each benchmark user has his or her own master data, such as material, vendor, or customer master data to avoid data-locking situations [27]. The dependent variable is the average response time of dialog steps (*AvgResponseTime*). The independent variables in this setup are (i) the number of

active users ($NumUser$) where the domain ranges from 60 to 150 and (ii) the number of work processes for dialog workload ($NumWP$) varied from 3 to 6. Thus, we are looking for the function $f(NumUser, NumWP) = AvgResponseTime$. The full configuration space consists of 360 measurement points. In order to get statistically stable results we repeated each measurement multiple times. All in all, the determination of a single measurement point takes approximately one hour which means that in the worst case the IT administrator has to measure 15 days in order to determine the optimal configuration. The results (see Figure 4) show that our adaptive experiment selection methodologies provides very good results with both analysis methods. The combination AdaptiveSelection/Kriging required only 64 measurement points ($\approx 18\%$ of the full configuration space) to derive a prediction model with a mean relative error of 1.6%. This reduces the time necessary to derive an optimal configuration from 15 to ≈ 2.5 days.

	Random				Equidistant				Adaptive			
	Kriging		Mars		Kriging		Mars		Kriging		Mars	
	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE	#M/Full	MRE
Communication Server	162/640	10,80%	89/640	4,10%	249/640	10,00%	147/640	1,20%	92/640	2,30%	105/640	3,40%
Enterprise Application	104/360	24,50%	90/360	14,20%	123/360	7,60%	122/360	8,10%	64/360	1,60%	67/360	8,30%

Fig. 4. Case Study Results

6.3 Discussion

The results of the two case studies (see Figure 4) show that the approach presented in this paper can significantly reduce the effort necessary to derive measurement-based performance models with high prediction accuracy. In both cases, our approach was able to derive a very good prediction model using only $\approx 15\%$ of the full configuration space (**RQ1 (i)**). Even the comparison with a similar approach proofed the efficiency of our methodology (**RQ1 (ii)**). The equidistant experiment selection strategy generated the worst results independent of the analysis strategy. The random strategy achieved good results especially in combination with MARS. However, the best results have been achieved by the adaptive experiment selection strategy independent of the analysis strategy. But, the Kriging predictions outperformed MARS in both scenarios which proofs the applicability of geostatistical interpolation techniques for software performance analyses (**RQ2**). The combination AdaptiveSelection/Kriging reduced the time necessary to find the optimal configuration of an enterprise application server from 15 days to ≈ 2.5 days which is an import reduction due to the fact that the time for the configuration of a system in the staging phase is often very limited (**RQ3**).

7 Summary

In this paper, we presented an approach for the automated and efficient selection of experiments in order to derive performance prediction models. We introduced three experiment selection methodologies and combined them with the statistical model inference techniques MARS and Kriging. Moreover, we applied the approach in two case

studies and compared the different combinations of experiment selection and analysis methods. In these case studies, the combination of adaptive experiment selection and Kriging achieved the best results. The proposed techniques support software architects and performance analysts to capture the effect of existing software systems on software performance and include these effects into further performance evaluations. In our future work, we will investigate further experiment selection strategies and analysis methods. We will address the curse of dimensionality [11] by applying the approach in case studies with more than two independent parameters. Furthermore, we are going to integrate the measurement-based performance models with model-driven approaches as described in Section 2.

References

1. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering* 30(5), 295–310 (2004)
2. Becker, S., Koziolok, H., Reussner, R.: The Palladio component model for model-driven performance prediction. *Journal of Systems and Software* 82, 3–22 (2009)
3. Courtois, M., Woodside, M.: Using regression splines for software performance analysis and software characterization. In: *Proceedings of the 2nd International Workshop on Software and Performance, WOSP 2000, September 17–20*, pp. 105–114. ACM Press, New York (2000)
4. De Smith, M.J., Goodchild, M.F., Longley, P.A.: *Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools*. Troubador Publishing
5. Denaro, G., Polini, A., Emmerich, W.: Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes* 29(1), 94–103 (2004)
6. Fioukov, A.V., Hammer, D.K., Obbink, H., Eskenazi, E.M.: Performance prediction for software architectures. In: *Proceedings of PROGRESS 2002 Workshop* (2002)
7. Friedman, J.H.: Multivariate adaptive regression splines. *Annals of Statistics* 19(1), 1–141 (1991)
8. Gorton, I., Liu, A.: Performance Evaluation of Alternative Component Architectures for Enterprise JavaBean Applications. *IEEE Internet Computing* 7(3), 18–23 (2003)
9. Groenda, H.: Certification of software component performance specifications. In: *Proceedings of Workshop on Component-Oriented Programming (WCOP) 2009*, pp. 13–21 (2009)
10. Happe, J., Westermann, D., Sachs, K., Kapová, L.: Statistical inference of software performance models for parametric performance completions. In: Heineman, G.T., Kofron, J., Plasil, F. (eds.) *QoSA 2010. LNCS*, vol. 6093, pp. 20–35. Springer, Heidelberg (2010)
11. Hastie, T., Tibshirani, R., Friedman, J.: *The Elements of Statistical Learning: Data mining, Inference, and Prediction*, 2nd edn. Springer Series in Statistics. Springer, Heidelberg (2009)
12. Jin, Y., Tang, A., Han, J., Liu, Y.: Performance evaluation and prediction for legacy information systems. In: *Proceedings of ICSE 2007*, pp. 540–549. IEEE CS, Washington (2007)
13. Jung, G., Pu, C., Swint, G.: Mulini: An Automated Staging Framework for QoS of Distributed Multi-Tier Applications. In: *ASE Workshop on Automating Service Quality* (2007)
14. Koziolok, H.: Performance evaluation of component-based software systems: A survey. *Performance Evaluation* (in press, corrected proof, 2009)
15. Kraft, S., Pacheco-Sanchez, S., Casale, G., Dawson, S.: Estimating service resource consumption from response time measurements. In: *Proc. of VALUETOOLS 2009*. ACM, NY (2009)

16. Krige, D.G.: A Statistical Approach to Some Basic Mine Valuation Problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa* 52(6), 119–139 (1951)
17. Kumar, D., Zhang, L., Tantawi, A.: Enhanced inferencing: Estimation of a workload dependent performance model. In: *Proceedings of VALUETOOLS 2009* (2009)
18. Li, J., Heap, A.D.: A review of spatial interpolation methods for environmental scientists. *Geoscience Australia, Canberra* (2008)
19. Miller, B.P., Callaghan, M.D., Cargille, J.M., Hollingsworth, J.K., Irvin, R.B., Karavanic, K.L., Kunchithapadam, K., Newhall, T.: The paradyn parallel performance measurement tool. *Computer* 28, 37–46 (1995)
20. Mos, A., Murphy, J.: A framework for performance monitoring, modelling and prediction of component oriented distributed systems. In: *WOSP 2002: Proc. of the 3rd International Workshop on Software and Performance*, pp. 235–236. ACM, New York (2002)
21. Motulsky, H.J., Ransnas, L.A.: Fitting curves to data using nonlinear regression: a practical and non-mathematical review (1987)
22. Pacifici, G., Segmuller, W., Spreitzer, M., Tantawi, A.: Dynamic estimation of cpu demand of web traffic. In: *Proc. of VALUETOOLS 2006*, page 26. ACM, New York (2006)
23. Pebesma, E.J.: Multivariable geostatistics in s: the gstat package. *Computers and Geosciences* 30, 683–691 (2004)
24. Reussner, R., Sanders, P., Prechelt, L., Müller, M.S.: SKaMPI: A detailed, accurate MPI benchmark. In: Alexandrov, V.N., Dongarra, J. (eds.) *PVM/MPI 1998*. LNCS, vol. 1497, pp. 52–59. Springer, Heidelberg (1998)
25. Sacks, J., Welch, W.J., Mitchell, T.J., Wynn, H.P.: Design and analysis of computer experiments. *Statistical Science* 4, 409–423 (1989)
26. Sankarasetty, J., Mobley, K., Foster, L., Hammer, T., Calderone, T.: Software performance in the real world: personal lessons from the performance trauma team. In: Cortellessa, V., Uchitel, S., Yankelevich, D. (eds.) *WOSP*, pp. 201–208. ACM, New York (2007)
27. SAP: SAP Standard Application Benchmarks (March 2011), <http://www.sap.com/solutions/benchmark>
28. Schneider, T.: *SAP Performance Optimization Guide: Analyzing and Tuning SAP Systems*. Galileo Pr. Inc., Bonn (2006)
29. Switzer, P.: *Kriging*. John Wiley and Sons Ltd., Chichester (2006)
30. Thakkar, D., Hassan, A.E., Hamann, G., Flora, P.: A framework for measurement based performance modeling. In: *WOSP 2008: Proceedings of the 7th International Workshop on Software and Performance*, pp. 55–66. ACM, New York (2008)
31. Tobler, W.: A computer movie simulating urban growth in the detroit region. *Economic Geography* 46(2), 234–240 (1970)
32. Westermann, D., Happe, J.: Performance Cockpit: Systematic Measurements and Analyses. In: *ICPE 2011: Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering*. ACM, New York (2011)
33. Westermann, D., Happe, J.: *Software Performance Cockpit* (March 2011), <http://www.softwareperformancecockpit.org/>
34. Westermann, D., Happe, J., Hauck, M., Heupel, C.: The performance cockpit approach: A framework for systematic performance evaluations. In: *Proceedings of the 36th EUROMICRO SEAA 2010*. IEEE CS, Los Alamitos (2010)
35. Woodside, C.M., Vetland, V., Courtois, M., Bayarov, S.: Resource function capture for performance aspects of software components and sub-systems. In: Dumke, R.R., Rautenstrauch, C., Schmietendorf, A., Scholz, A. (eds.) *WOSP 2000 and GWPESD 2000*. LNCS, vol. 2047, pp. 239–256. Springer, Heidelberg (2001)

Author Index

- Alcaraz, Salvador 14
Anastasiou, Nikolas 29
- Babka, Vlastimil 250
Barbierato, Enrico 280
Bernardo, Marco 265
Bradley, Jeremy T. 87
Braghetto, Kelly Rosa 42
Bušić, Ana 204
- Cerotti, Davide 280
Cortellessa, Vittorio 265
- Dao-Thi, Thu-Ha 189
de Boer, Pieter-Tjerk 133
Djemame, Karim 295
- Ferreira, João Eduardo 42
Flamminj, Mirko 265
Fokkink, Wan 72, 174
Fourneau, Jean-Michel 57, 189
- Ghassemi, Fatemeh 72
Gilly, Katja 14
Gilmore, Stephen 310
Gribaudo, Marco 280
Guenther, Marcel C. 87
- Habibi, Jafar 174
Hammond, Simon D. 148, 235
Happe, Jens 325
Haverkort, Boudewijn R. 133
Hlavacs, Helmut 102
Hyon, Emmanuel 204
- Jarvis, Stephen A. 148, 235
Juiz, Carlos 14
- Kavanagh, Richard 295
Knottenbelt, William 29
Kolesnichenko, Anna 133
Kounev, Samuel 10
Krebs, Rouven 325
- Marin, Andrea 29
Mittermaier, Alfons 219
Movaghar, Ali 72
Murphy, John 1
- Nussbaumer, Michael 102
- Pennycook, Simon J. 148, 235
Perks, Oliver 148
Puigjaner, Ramon 14
- Quessette, Franck 57
- Reinecke, Philipp 163, 219
Remke, Anne 133
- Shamsaie, Abolhassan 174
Smith, Michael J.A. 310
- Talebi, Mahmoud 72
Tran, Minh-Anh 189
Tůma, Petr 250
- van Moorsel, Aad 117
Vincent, Jean-Marc 42
- Westermann, Dennis 325
Wieczorek, Aleksander 204
Wolter, Katinka 163, 219
Wright, Steven A. 235
- Yassin Kassab, Rouaa 117