UKPEW 2014

30th Annual UK Performance Engineering Workshop

Newcastle University

19th September 2014

Programme

Welcome and registration 9.15

Session 1: 9.30-10.30

- Energy-Aware Profiling for Cloud Computing Environments, Ibrahim Alzamil, Karim Djemame, Django Armstrong, Richard Kavanagh (University of Leeds)
- A case study in inspecting the cost of security in cloud computing, Said Naser Said Kamil and Nigel Thomas (Newcastle University)

Coffee 10.30-11

Session 2: 11-12.30

- Efficient Reliability and Performance Analysis of Layered Queueing Models, Juan F. Perez and Giuliano Casale (Imperial College London)
- Performance Analysis of Collective Adaptive Behaviour in Time and Space, Cheng Feng, Marco Gribaudo and Jane Hillston (University of Edinburgh and Politecnico di Milano)
- Development of a Smart Grid Simulation Environment, J. Delamare , B. Bitachon, Z. Peng, Y. Wang, B.R. Haverkort, M.R. Jongerden, (University of Twente)

Lunch 12.30-1.30

Session 3: 1.30-3 PM

- Validation of automatic vehicle location data in public transport systems, Stephen Gilmore and Daniel Reijsbergen (University of Edinburgh)
- Dynamic subtask dispersion reduction in heterogeneous parallel queueing systems, Tommi Pesu and William Knottenbelt (Imperial College London)
- A Hybrid Simulation Framework for the Analysis of Queueing Networks and Petri Nets, Esmaeil Habibzadeh, Demetres D. Kouvatsos (University of Bradford) and Guzlan M.A. Miskeen (University of Sebha, Libya)

Afternoon tea 3-3.30 PM

Session 4: 3.30-5 PM

- A Case Study in Capacity Planning for PEPA Models with the PEPA Eclipse Plug-in, Christopher D. Williams and Allan Clark (University of Edinburgh)
- Performance Modelling and Evaluation of Secure Dynamic Group Communication Systems in RANETs, D.D. Kouvatsos, E. Habibzadeh (University of Bradford) and Guzlan M.A. Miskeen (University of Sebha, Libya)
- Operating policies for energy efficient dynamic server allocation, Thai Ha Nguyen, Matthew Forshaw and Nigel Thomas (Newcastle University)

Energy-Aware Profiling for Cloud Computing Environments

Ibrahim Alzamil, Karim Djemame, Django Armstrong and Richard Kavanagh

School of Computing
University of Leeds
Leeds, UK
E-mail: {sc11iaa, K.Djemame, scsdja, scsrek}@leeds.ac.uk

Abstract

Cloud Computing has changed the way in which people use the IT resources today. Now, instead of buying their own IT resources, they can use the services offered by Cloud Computing with reasonable costs based on a "pay-per-use" model. However, with the wide adoption of Cloud Computing, the costs for maintaining the Cloud infrastructure have become a vital issue for the vendors, especially with the large input of energy costs to underpin these resources. Thus, this paper proposes a system architecture that can be used for profiling the resources usage in terms of the energy consumption. From the profiled data, the application developers can enhance their energy-aware decisions when creating or optimising the applications to be more energy efficient. This paper provides initial exploratory insight into Key Performance Indicators (KPIs) that the proposed system architecture will make use of.

Keywords: Cloud Computing, Energy Efficiency, Energy-Aware Profiling, Energy Efficiency Metrics.

1 Introduction

The radical adoption of Cloud Computing technology has exposed a significant overhead in maintaining its infrastructure, which has become a major issue for the Cloud providers due to the associated high operational costs, such as energy consumption. It has been stated that a data centre may consume about 100 times more energy than a typical office of the same size [1]. So, efficiently managing the power consumed by the servers would improve the overall consumption; in the sense that as the servers consume less power, the heat generated by these servers would be reduced, which would then reduce the need for cooling resources that consume large amount of energy as well.

Improving the energy efficiency of Cloud Computing has been an attractive research topic for both academia and industry as it has become gradually significant for the future of Information and Communication Technology (ICT) [2]. Many researchers have investigated new ways for managing the Cloud infrastructure as a means of enhancing the energy efficiency. A number of techniques have been already

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

proposed and deployed for better resource management. For example, Data Voltage and Frequency Scaling (DVFS) and Virtual Machines (VMs) allocation have been widely studied and deployed to manage the resources more efficiently. Nonetheless, there is still a need to make the whole stack of Cloud Computing more energy-aware and not only focusing on the resource management.

There are a large number of different Cloud applications with different requirements of resources; some of them are data-intensive whereas others are compute-intensive. So, depending on the taxonomy of the application, the energy consumption of the resources that underpin these different applications can vary. Thus, this research is aimed to add value to the Cloud Computing energy efficiency by investigating energy efficiency modelling in terms of energy-aware profiling and energy efficiency metrics. Energy-aware profiling is studied in order to understand how the energy is consumed by the infrastructure components, like CPUs, when the application is in operation. Further, it would investigate the introduction of new energy efficiency metrics to better understand how energy efficient the running application is and provide overall Key Performance Indicators (KPIs) of that application.

Thus, the output measurements of energy-aware profiling and energy efficiency metrics will be combined to form KPIs for the running application. Also, these KPIs will be further analysed and used to facilitate the decision-making of application developers with better energy-aware programming.

The main contributions of this paper include:

- Proposed system architecture for profiling and assessing the energy efficiency of Cloud infrastructure resources.
- Supporting analysis of how the output of the proposed system architecture can enhance the decision making of Cloud application developers when creating and configuring new applications.

2 Energy Efficiency in Cloud Computing

For the Cloud Computing stack, energy efficiency has been extensively studied in the literature and has focused on a large number of different topics, like virtualisation, requirement engineering, programming models, and resource management.

In terms of virtualisation, a number of studies proposed different approaches for allowing resource utilisation, server consolidation and live migration of virtual machines [3,4,5], which all can offer significant energy and costs savings [6].

With the advancement of software-intensive systems to self-adaptive systems to meet the growing needs for autonomic computing [7], requirements engineering for self-adaptive software systems ensuring energy aspects has received less attention [8]; as that can be justified with the challenges to encounter when dealing with uncertainties associated with the operating environment [9].

In terms of programming models, there are a number of platforms used for the development and deployment of Cloud applications and services, like Hadoop [10], Windows Azure [11], Microsoft Daytona [12], Twister [13], Manjrasoft Aneka [14], and Google App Engine [15]. Yet, these platforms lack consideration for energy efficiency, whereas a work presented in [16] proposed a general-purpose programing

environment to simplify and help the developers make energy-efficient decisions for constructing energy-aware applications.

Most of the attention in the literature has focused on enhancing the energy efficiency of Cloud Computing through better resource management to avoid some issues like excessive power consumption and SLAs violation reliability [17]. Therefore, many developments have been introduced like, DVFS and Dynamic Power Management (DPM) techniques to control the power consumption of servers in accordance with the workload [18], virtual machine consolidation policies to optimise the hosts by migrating VMs from one host to another [17], some models for better prediction of the power consumption for the servers [19], task consolidation model for maximising resource utilisation [20], a holistic framework called Mistral for optimising the power consumption for the physical hosts [21], a CPU re-allocation algorithm that combines both DVFS and live migration techniques to reduce the energy consumption and increase the performance in Cloud datacentres [22].

However, there is a lack of research that tackles the issue of properly ensuring energy awareness from the design stage and not only through resource management of the Cloud Infrastructure. So, there is still a need for modelling the energy efficiency of Cloud infrastructures to gain a better understanding of energy efficiency and to feed the decision-making at the service design stage, which will be discussed in the following section.

3 Energy Efficiency Modelling

It is important to model energy profiling techniques and introduce new metrics to inform the providers how energy efficient their infrastructure is to make strategic decisions, such as creating and configuring energy-aware application and forming new energy-aware pricing mechanism, accordingly.

3.1 Profiling

Having such tools that would help understand how the energy has been consumed in a system is essential in order to facilitate software developers to make energy-aware programming decisions. Schubert et al [23] state that the developers lack the tools that indicate where the energy-hungry sections are located in their code and help them better optimize their code for enhancing energy consumption more accurately instead of just relying on their own intuitions. In their work, they proposed eprof, which is a software profiler that narrates energy consumption to code locations; therefore, it would also help developers make better energy-aware decisions when they re-write their code [23]. For example, with storing data on a disk, software developers might choose between storing the data in an uncompressed format or a compressed format, which would require more CPU resources. Compressed data has been commonly suggested as a way to reduce the amount of I/O needed to be performed and therefore reducing the energy based on the hypothesis that the CPU can process the task of compression and decompression with less energy than the task of transferring large data from and to the disk [24]. However, that would depend on the data being processed. In fact, some conducted experiments in [23] with eprof profiling tool show that the process of compressing and decompressing

the data consumes significantly more energy than the process of transferring large amount of uncompressed data because the former would use more CPU resources than the latter. So, it can be a controversial issue depending on the application domain. Thus, having such tools identifying where the energy has been consumed would help software developers to make more energy-aware decisions.

Moreover, a new framework called Symbolic Execution and Energy Profiles (SEEP) has been recently introduced in [25] as an approach to help software developers make well informed decisions for energy optimisation from early stages at the code level. To illustrate, SEEP is designed to provide the developers with energy estimations to make them more energy-aware while they are programming.

3.2 Metrics

Energy efficiency in Clouds can be assessed by different metrics. In terms of Cloud infrastructure, the well-known Power Usage Effectiveness (PUE) metric has been introduced by the Green Grid organisation to help the providers assess and improve the energy efficiency of their data centres [26]. However, despite the fact that the PUE metric has been successful and widely used, Grosskop [27] argues that it is restricted as an indicator for energy efficiency to the infrastructure management only and not considering the optimisation at the software levels to enhance the efficiency of the whole stack. Also, as stated by Wilke et al [28], analysing software's energy consumption is considered an important requirement for such optimisations. So, Grosskop proposed a new metric called the Consumption Near Sweet-Spot (CNS) that identifies how well the system's energy efficiency optimum and its utilisation are allied by calculating the ratio between the average consumption and optimum consumption for a system to deliver a particular unit of work [27].

Moreover, other works have looked at other metrics for energy efficiency measurements, like utilisation percentage and SLA violation percentage. For example, in the work conducted by Beloglazov et al [17], they evaluate the efficiency and performance of their proposed algorithms by using some metrics, namely the total energy consumption, the average number of SLA violations, and the number of VM migrations.

Recently, some work has started to measure the energy consumption in more details, like measuring energy consumption for each VM in a physical machine. Research conducted by [29] introduces a VM power model to measure the estimated power consumption of VM with using performance events counter. They argue that the results of their proposed model can get on average about 97% accuracy.

Nonetheless, as mentioned earlier, there is a lack of metrics to measure the energy efficiency of Clouds from different layers other than the infrastructure only. In terms of fine-grain measurement, there are a limited number of researches conducted to map the energy consumption for each single VM in a server, which indicates the need to fill this gap by introducing new suitable metrics for measuring and mapping the energy consumption to each VM.

4 Energy-Aware Profiling

Ensuring energy efficiency from different layers in Cloud Computing has become inevitable, especially with the unstable energy costs. We propose in this paper to have energy-aware profiling for the Cloud infrastructure to better understand how the energy has been consumed and assess its energy efficiency in order to help the software developers from the application layer enhance their decision-making in terms of energy-awareness when optimising their applications and services. The proposed system architecture will be discussed in the following subsection.

4.1 Proposed System Architecture

The scope of this proposed system architecture would be in the IaaS layer where the operation of services takes place. The main components of this model consist of Resource Monitoring Unit (RMU), Energy Profiling Unit (EPU), Reporting and Analysis Unit, as can be shown in Figure. 1.

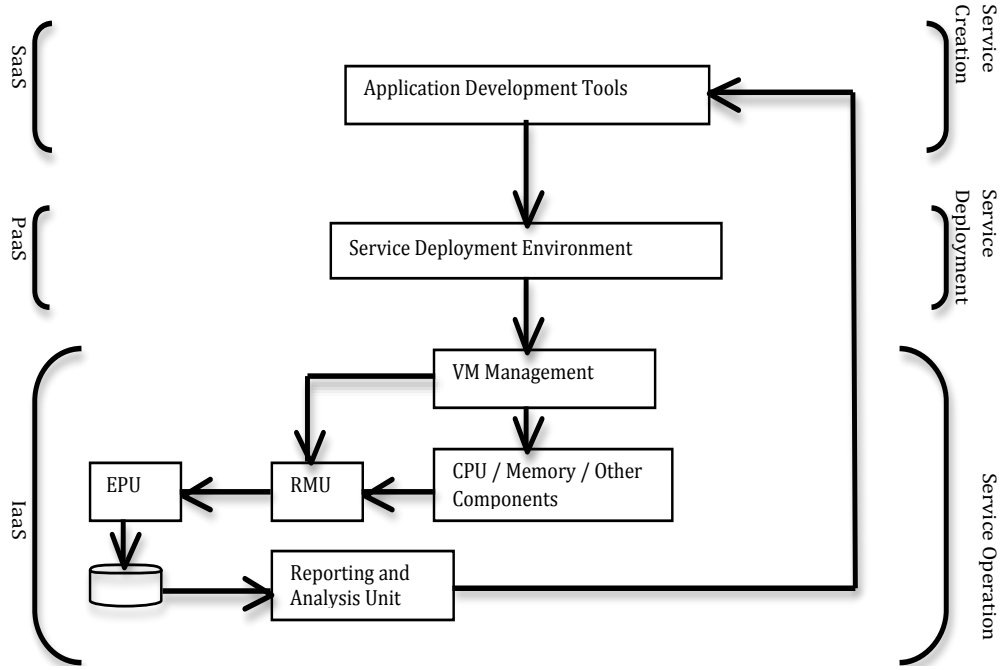


Fig. 1. Proposed System Architecture

This proposed system architecture would have the RMU to dynamically collect the energy consumed by the hardware components and observe the number of assigned VMs. After that, EPU would have appropriate algorithms to calculate the energy consumed by each VM and hardware components, and it would then profile and populate these measurements as KPIs to a database. This data can be further analysed by the Reporting and Analysis Unit to provide the software developers energy-aware reports in order to enhance their awareness of the energy consumption when making programming decisions. For example, it might be interesting to know whether the CPU or the memory of the hardware component would consume more energy, so that the developer can create applications that would depend more

on components with less energy consumption, without compromising performance.

4.2 Illustration

An application can run on the Cloud to deliver services for the end users. These services can consist of a number of tasks, like data-intensive or computation-intensive tasks. Data-intensive tasks would depend more on using disk storage for processing large amounts of data and data retrieval or update, which would require high disk I/O bandwidth to maintain performance, whereas computation-intensive tasks would depend more on using the processors to perform more computation [30].

When the service is configured in the application development tool with descriptions of the allocated software and hardware resources, and is deployed in the service deployment environment and goes through VM management, the proposed system would then start with the RMU to capture and monitor the energy consumed by the infrastructure that underpins and operates that service. The captured data (as input to the system) will be dynamically collected by the EPU for appropriate measurements and profiling in terms of energy efficiency. Next, EPU would populate the profiled data as KPIs to a database. Hence, these KPIs (as output of the system) can be further analysed and reported in a meaningful format to the application developers to enhance their energy-aware decisions when making and configuring new services.

The next section provides early exploration into the energy efficiency measurements that will be supported by the proposed architecture.

5 Early Exploration

5.1 Experiment 1

The overall objective of this experiment is to evaluate the variation of the energy consumption when running memory-intensive software and CPU-intensive software to find out whether CPU or memory would consume more energy.

Currently, the experiment is conducted using Windows 7 OS with 2GB of RAM and Intel Core 2 Duo 2.26GHz processor. Also, WattsUp? meter [31] is used and set to get actual measurement of energy consumption per second. The used measurements include energy consumption in watts, memory utilisation, and CPU utilisation. This experiment is limited that it has not been set in a real Cloud environment, but it still can give an indication of how the CPU and memory would consume energy.

Memory-intensive software has been created and used to generate high workload and keep the memory fully occupied by initiating an array of string and continually adding values to that array and setting the thread to sleep continually to reduce the load on the CPU. CPU-intensive software has been created and used to generate high workload to keep the CPU busy by doing pi computation continually.

Each one has been set to run for a minute to have enough time for observing the behaviour of the energy consumption.

The implementation of the experiment is designed to firstly start with collecting measurements (energy consumption in watts, CPU utilisation, and memory utilisa-

tion) for 20 seconds in order to observe how the system would normally consume energy just before running any software. Then, it runs memory-intensive software for a minute and then goes into idle again for 20 seconds to set everything back in normal state. Then, it runs cpu-intensive software for a minute, and finally it goes into idle again for 20 seconds and ends the execution. The following Figure. 2, Figure. 3, and Figure. 4 show the results of this experiment.

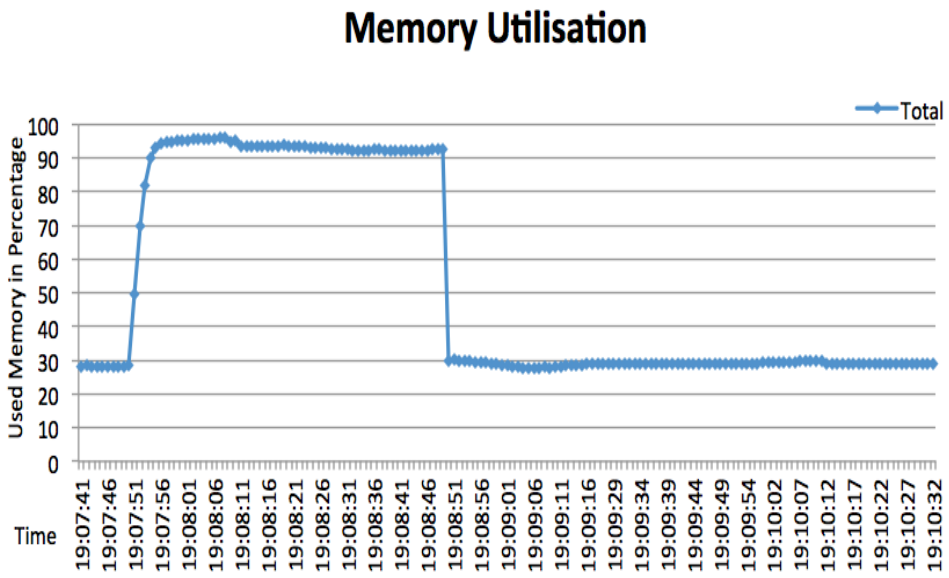


Fig. 2. Memory Utilisation

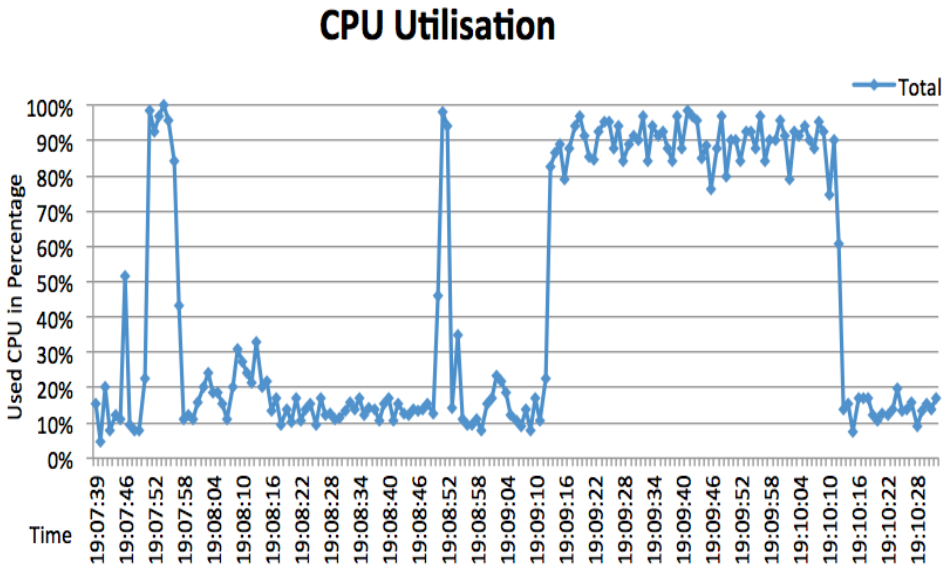


Fig. 3. CPU Utilisation

As depicted on Figure. 2 and Figure. 3, during the execution of memory-intensive software, the memory utilisation increases to the maximum, and the CPU mostly varies with low utilisation and gets very high only when starting and ending that software, which can be justified as the CPU is processed only to start and end

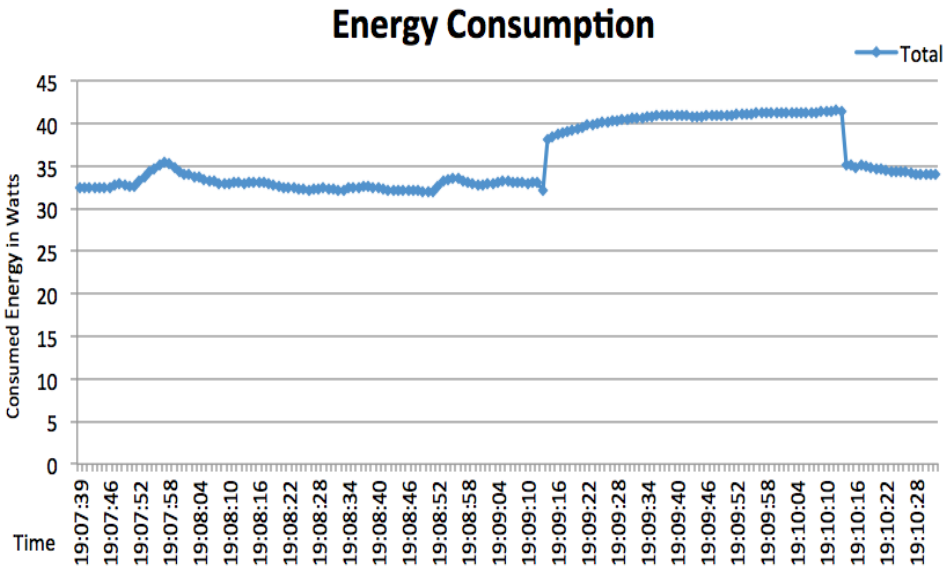


Fig. 4. Energy Consumption

that execution. When executing CPU-intensive software, the CPU utilisation gets to the maximum and memory utilisation does not increase, as the memory is not doing any job for that execution.

As shown in Figure. 4, the energy consumption during the execution of the memory-intensive software does not increase and mostly is stabilised around 33 watts. But, there is a slight increase when starting the execution for the memory-intensive software at 19:07:48, and when ending that execution at 19:08:48, and that can be justified with the increase of CPU utilisation during these times. During the execution of CPU-intensive software, the energy consumption increases dramatically and stabilise around 41 watts.

The important finding of this experiment shows that the energy consumption correlates with the CPU utilisation. It increases when the CPU is used regardless of memory usage. Hence, CPU-intensive application would consume more energy than memory-intensive application. In other words, the software developers can reduce the energy consumption by creating and configuring applications that would depend more on the memory rather than CPU.

5.2 Experiment 2

The overall objective of this experiment is to evaluate the variation of the energy consumption when running both memory-intensive and CPU-intensive software and when running only CPU-intensive software to find out which one would consume more energy.

This experiment is conducted using the same environment used in Experiment 1. Memory and CPU-intensive software has been created and used to generate high workload and keep the memory fully occupied by initiating an array of string and continually adding values to that array without setting thread to sleep in order to keep the CPU busy as well. The same CPU-intensive software used in Experiment 1 is used again here to generate high CPU-workload.

This experiment is designed and executed in the same way as Experiment 1,but it differs only with the first executed software, which is in this experiment both memory and CPU-intensive rather than memory only.

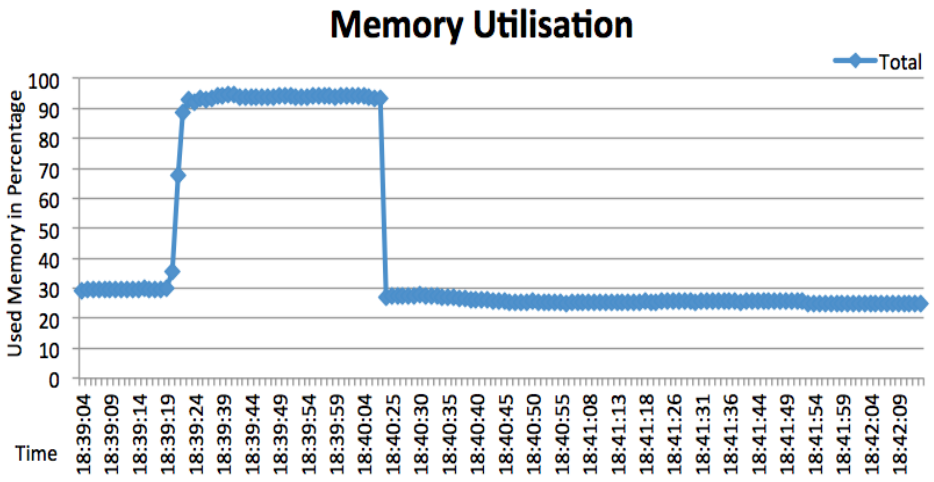


Fig. 5. Memory Utilisation

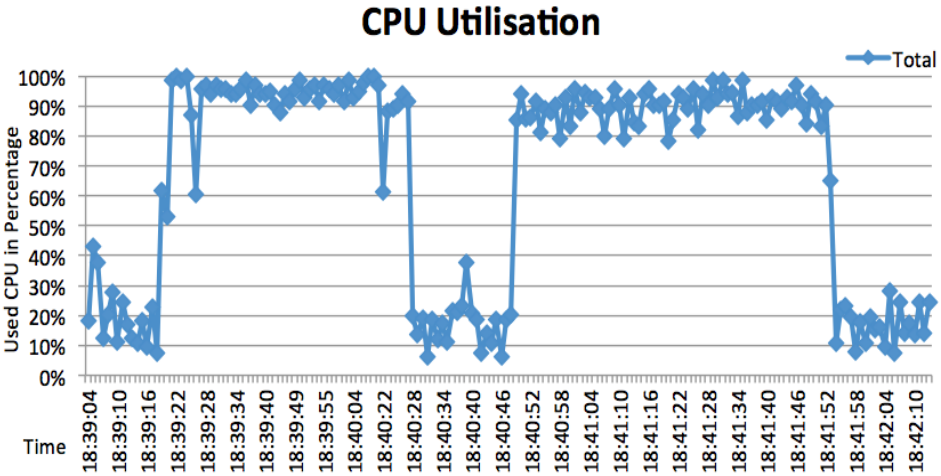


Fig. 6. CPU Utilisation

As shown on Figure. 5, and Figure. 6, during the execution of memory and CPU-intensive software, the memory utilisation gets to the maximum, and the CPU utilisation also gets to the maximum but it varies when starting and ending that software. When executing CPU-intensive software, the CPU utilisation gets to the maximum and memory utilisation does not increase.

As shown in Figure. 7 , the energy consumption during the execution of the memory and CPU-intensive software increases over time and then it is mostly stabilised around 41 watts. When the CPU-intensive software start executing, the energy consumption increments dramatically and stabilise around 43 watts.

The findings of this experiment show that the energy consumption correlates with the CPU utilisation. Even though both the memory and CPU gets to the maximum utilisation, memory and CPU-intensive software interestingly consumes

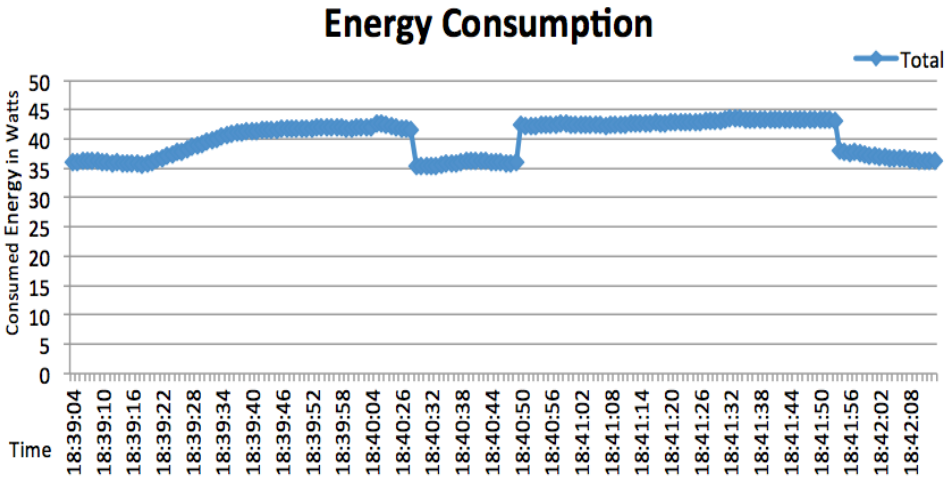


Fig. 7. Energy Consumption

about 41 watts, which is slightly less than the CPU-intensive only software that consumes about 43 watts. Thus, creating and configuring applications that balance between using both memory and CPU rather than depending only the CPU can reduce the energy consumption.

6 Conclusion and Future Work

This paper has proposed system architecture for Cloud Computing environment. This system architecture can help software developers understand how their applications are using the infrastructure resources and consuming energy in order to enhance their decision-making when creating and configuring new applications.

The conducted experiments in section V. revealed that the CPU is an energy-hungry component that consumes more energy compared to memory. Hence, applications should be created and configured in a way that depend on memory or balances between using memory and CPU rather than CPU only in order to gain some energy savings.

Future work will overcome the limitation of the conducted experiments to be implemented on a Cloud environment that consists of a number of machines linked together over a network and runs many different Hypervisor technologies. It would include using real Cloud application that is memory-intensive and another that is CPU-intensive. Also, performance measurement will be considered and included to show a trade-off between performance and energy consumption.

References

[1] P. Scheihing. Creating Energy-Efficient Data Centers. In *Data Center Facilities and Engineering Conference*, Washington, DC, 18th May., 2007.

[2] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis. Energy-Efficient Cloud Computing. *The Computer Journal*, 53(7):1045–1051, August 2009.

[3] K. Ye, D. Huang, X. Jiang, H. Chen, and S. Wu. Virtual Machine Based Energy-Efficient Data Center Architecture for Cloud Computing: A Performance Perspective. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 171–178, 2010.

[4] J. Hardy, L. Liu, N. Antonopoulos, W. Liu, L. Cui, and J. Li. Assessment and Evaluation of Internet-Based Virtual Computing Infrastructure. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on*, pages 39–46, 2012.

[5] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, 2008.

[6] T. Arthi and H. Hamead. Energy aware cloud service provisioning approach for green computing environment. In *Energy Efficient Technologies for Sustainability (ICEETS), 2013 International Conference on*, pages 139–144, 2013.

[7] H. Giese, H. Muller, M. Shaw, and R. De Lemos. 10431 Abstracts Collection – Software Engineering for Self-Adaptive Systems. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.

[8] N. Qureshi and A. Perini. Engineering adaptive requirements. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, volume 2009, pages 126–131. IEEE, May 2009.

[9] J. Camara and R. De Lemos. Evaluation of resilience in self-adaptive systems using probabilistic model-checking. In *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 53–62. IEEE, June 2012.

[10] Welcome to Apache, Hadoop! Available at http://hadoop.apache.org/, 2011.

[11] Windows Azure: Microsoft's Cloud Platform — Cloud Hosting — Cloud Services. Available at http://www.windowsazure.com/en-us/, 2012.

[12] Daytona - Microsoft Research. Available at http://research.microsoft.com/en-us/projects/daytona/, 2013.

[13] J. Ekanayake. Twister: Iterative MapReduce. Available at http://www.iterativemapreduce.org/, 2009.

[14] Manjrasoft - Products. Available at http://www.manjrasoft.com/products.html, 2008.

[15] Google App Engine - Google Developers. Available at https://developers.google.com/appengine/?hl=en, 2014.

[16] C. Xian, Y. Lu, and Z. Li. A programming environment with runtime energy characterization for energy-aware applications. In *Low Power Electronics and Design (ISLPED), 2007 ACM/IEEE International Symposium on*, pages 141–146, 2007.

[17] A. Beloglazov, J. Abawajy, and R. Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, 28(5):755–768, May 2012.

[18] D. Kliazovich, P. Bouvry, and S. Khan. DENS: Data Center Energy-Efficient Network-Aware Scheduling. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 69–75. IEEE, December 2010.

[19] R. Basmadjian, F. Niedermeier, and H. De Meer. Modelling and analysing the power consumption of idle servers. *Sustainable Internet and ICT for Sustainability (SustainIT), 2012*, pages 1–9, 2012.

[20] Y. Lee and A. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, March 2010.

[21] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 62–73, 2010.

[22] W. Chawarut and L. Woraphon. Energy-aware and real-time service management in cloud computing. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2013 10th International Conference on*, pages 1–5, 2013.

[23] S. Schubert, D. Kostic, W. Zwaenepoel, and K. Shin. Profiling software for energy consumption. *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, pages 515–522, 2012.

[24] A. Kansal and F. Zhao. Fine-grained energy profiling for power-aware application design. *ACM SIGMETRICS Performance Evaluation Review*, 36(2):26, August 2008.

[25] T. Honig, C. Eibel, W. Schroder-Preikschat, and B. Kapitza. Proactive Energy-Aware System Software Design with SEEP. *In Porceedings of 2nd Workshop on Energy-Aware Software-Engineering and Development (EASED@BUIS)*, 2013.

[26] The Green Grid. Harmonizing Global Metrics for Data Center Energy Efficiency. Technical report, The Green Grid, 2012.

[27] K. Grosskop. PUE for end users-Are you interested in more than bread toasting? *In Porceedings of 2nd Workshop on Energy-Aware Software-Engineering and Development (EASED@BUIS)*, 2013.

[28] C. Wilke, S. Gotz, and S. Richly. JouleUnit: A Generic Framework for Software Energy Profiling and Testing. In *Proceedings of the 2013 Workshop on Green in/by Software Engineering*, GIBSE '13, pages 9–14, New York, NY, USA, 2013. ACM.

[29] W. Chengjian, L. Xiang, Y. Yang, F. Ni, and Y. Mu. System Power Model and Virtual Machine Power Metering for Cloud Computing Pricing. In *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*, pages 1379–1382, 2013.

[30] F. Chen, J. Schneider, Y. Yang, J. Grundy, and Q. He. An energy consumption model and analysis tool for Cloud computing environments. *2012 First International Workshop on Green and Sustainable Software (GREENS)*, pages 45–50, June 2012.

[31] Watts Up? Plug Load Meters. Available at www.wattsupmeters.com.

A case study in inspecting the cost of security in cloud computing

Said Naser Said Kamil Nigel Thomas
School of Computing Science
Newcastle University, UK.

***Abstract*— The growing demands of business and the competition in the provision of services has led to many enterprises outsourcing IT provision using cloud computing to handle business processes and data management. Ideally, the benefits that are offered by cloud computing technology can accommodate the rapidly increased demands of the organizations and individual customers. However, the usage of cloud computing has potential security concerns. The proposed research will put forward an approach for investigating the cost of security in cloud computing. The proposed method is based on a multi-level security model, which uses the distribution of partitioned workflows upon hybrid clouds. Furthermore, the PEPA Eclipse plug-in tool will be used to create a cost model that uses the valid deployment choices that generated by the multi-level security model. The outcomes that can be obtained by means of implementing this research will used to describe the evaluation of the performance. Thus, predictions of the performance can be derived as well as the potential cost of deployment under different options and scenarios.**

Index Terms — Cloud computing, Cost model, Cloud computing security, PEPA.

I. INTRODUCTION

Over the last few years cloud computing has become a valuable option for a considerable number of organizations. The reasons behind this move are the growing capability of outsourced solutions and the high cost of buying and maintaining infrastructure. This allows organizations to exploit the advantages offered by cloud computing such as: high performance, availability, scalability and the low cost (i.e. pay on demand).

According to NIST[1] [3], the definition of cloud computing is "*a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*" . These features are attracting organizations to migrate toward cloud computing to cope with their rapidly increased processes and related data.

Services offered by an organization are often business processes presented as workflows. The deployment of workflows on internal resources (i.e. private cloud) can affect the performance of services where the resources are limited [4]. On the other hand, cloud computing (i.e. public cloud) can overcome the limited resources problem that is facing enterprises and can offer high performance with cost-saving.

Using cloud computing has highlighted several security concerns, for instance where the organizations data will be stored and how ensure the confidentiality and privacy of information. Correspondingly, the security aspects are one of the main concerns for many organizations [5], [6]. Moreover, according to surveys that conducted by IDC [7] in 2008 and 2009, security is still the top challenge for cloud services. Thereby, some companies tend to use a combination of public and private clouds based on the sensitivity of data; private cloud as they perceive more secure and public clouds to gain the benefit of high performance, scalability and availability.

Corporations can obtain many significant improvements in security by using cloud computing [8]. Based on the multi-level security model presented by Watson [1], this research will investigate the cost of security for the deployment of the partitioned workflow in cloud competing. Moreover, the PEPA Eclipse plug-in tool will be used to create a cost model that relies on the valid selections generated by means of Watson's approach. Furthermore, a new cost model will be developed to evaluate the variety of distribution possibilities, where each option has its characteristic to deploy partitioned workflow on federated clouds.

The expected results will help to answer the following questions: how many resources are needed for specific deployment?; which better to deploy on private cloud or public clouds based on the sensitivity of data and the resources that organizations have? This work is taking into account the security requirements that should be met.

This paper is structured as follows. Section II presents a motivating example of a cloud workflow drawn from the literature. In section III we will discuss some other approaches to analysis. Section IV will introduce a set of PEPA models representing different cloud deployment options. In section V we will present some results and finally draw some conclusions and outline some further work.

II. MOTIVATING EXAMPLE

A considerable amount of research has addressed cloud computing and its associated concerns of security in addition to the trade-off of the cost of deployment on private cloud and public cloud and the partitioning of workflow onto clouds in terms of security concerns. Our research has adopted the multi-level security model of Watson [1] that originally extends the Bell-LaPadula security model [9]. The example in this paper is a Healthcare Data Analysis example that

[1] National Institute of Standards and Technolo

introduced by Watson. As can be seen in Figure 1. patient data that would be analyzed is shown as a workflow, which originally is a medical research application. In the first place, it reads data as input which includes the patient name and his/her measurements of heart rates. Then, the service *anonymize* takes off the name producing anonymized data. Consequently service named *analyze* starts to analyze the data. As a result, a summary of analyzed heart rates measurements will be created as output.

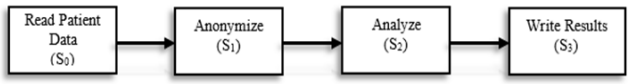


Figure 1: Workflow of health care data analysis [1].

The primary workflow that is shown in Figure 1 and introduced by Watson [1]; has been modelled as a directed graph, where data and services are represented as nodes and the data dependencies denoted as edges. The Bell-LaPadula security conditions (no read-up and no write-down) are applied on the workflows and extended to include the cloud properties. Furthermore, a security level is assigned for each service as well as data which are consumed and produced by services. The model is extended to include clouds that will accommodate service and data of workflow, where a security level is assigned to a cloud. Additionally, the security scheme of transferring data and services between clouds has been addressed, through insuring the security level assigned to the source besides the destination. As well as to this, a tool has been developed by Watson [1] to generate valid distribution choices automatically (see Figure 2) that complies with the security constraints mentioned earlier. Finally, a very simple cost model has been created to rank the deployment selections that produced by the tool based on cost. While the calculation of the cost in [1] is depending on three factories: Data storage (size and time), CPU and Data transfer in and out.

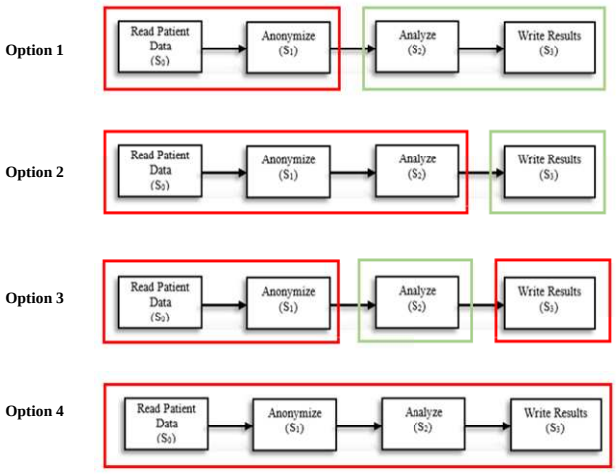


Figure 2: Valid Deployment Option, where boxes red and green referring to Private and Public Clouds respectively [1].

III. APPROACHES TO EVALUATION OF CLOUD COMPUTING

Pearson and Sander [10] introduced an approach that assisted customers to select the more appropriate cloud service provider through the decision support system that has been developed. An overview of this approach can be seen in Figure 3. Additionally, the outcomes of this system can give an assessment to threats that may possibly arise through the distribution of private information. As a consequence, the costs of the selection of service provider will be lowered. The approach of Pearson and Sander is concerned with the potential cloud security threats and to reduce the risk. However, our research will investigate the cost of the distribution of workflows over clouds by way of a different methodology with the aim of mitigate the cost.

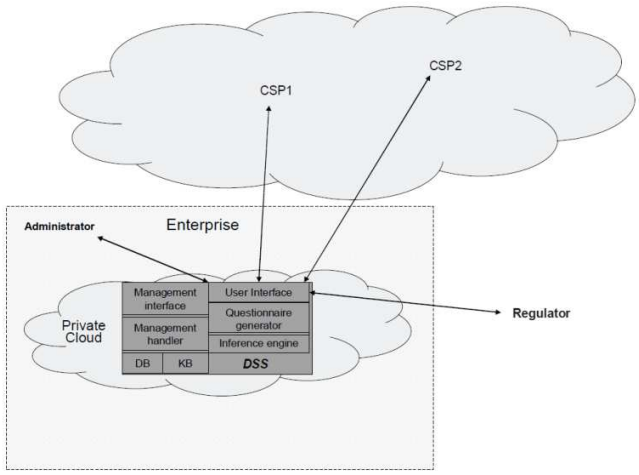


Figure 3: Overview of the approach of Pearson and Sander.

In a similar manner to the Pearson and Sander scheme, Wenge *et al* [11] proposed a method for assessing a cloud provider, particularly with respect security aspects and the collaboration with other cloud providers. This technique can be used to help customers to determine the most appropriate services providers as well as to the provision of a solution for the security risks associated.

Goettelmann *et al* [2] developed a methodology for partitioning a business process and deploying it on a set of clouds that had been taken into account security conditions. As can be seen in Figure 4, this method utilized a decentralization algorithm that proposed a mechanism to insure security constraints and quality of service. One limitation of this approach is that, the method encountered some problems in synchronization of messages.

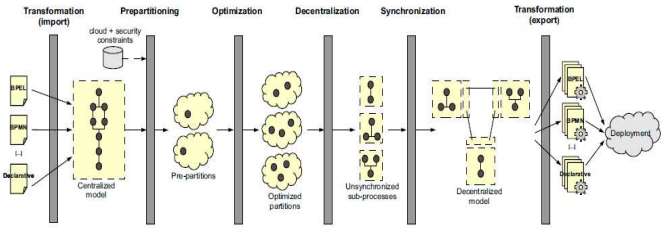


Figure 4: Decentralization approach [2].

It can be noticed that, there are some similarities between the Watson scheme and the Goettelmann *et al* method, through the manner of partitioning of workflows while meeting security requirements and distribution process over clouds. Nevertheless, the difference is that Watson considers

only sequential workflows tasks. Furthermore, [11] and [2] do not investigate the cost of deployment choices over clouds and [2] claims that the calculation of the cost will be time and resource consuming.

Mach and Schikuta [12] introduced a new cost model, where they stated that their economic cost model can provide a beneficial information for cloud provider and consumer based on the calculation of the server's energy consumption. They assume that a cloud cost model can be considered as a traditional production process that has production factors and produced goods. As well, the benchmark SPECpower ssj2008 that developed by SPEC[2] is used to test performance and power consumption. The produced information can be used to make right decisions for the business strategies that might be implemented and its impact. In spite of some similarities with Watson method as well as to the work presented by Pearson and Sander, Mach and Schikuta's work lies in operational cost that are related to power consumption, while our research is concerned with deployment cost in cloud environments.

A novel tool has been developed by [13] for dynamic exception handling, which extends the multi-level security model of [1]. The authors indicate that the tool can discover alternative partitions with low-cost paths to deal with exceptions that may occur during run time. The work uses dynamic exception handling for partitioned workflow on federated clouds and has adopted Watson's simple cost model of multi-level security model. This means that, they have not made changes to the method of calculation of deployment cost, but they consider exceptions are handled to achieve fault tolerance and to find another low cost paths for the successfully completion of a deployed process.

From the side of security, Capgemeni [8] described the cloud computing with associated risks, which are already associated with other types of using outsourcing. Capgemeni concentrates on multi-tenancy and required compliance, which is assumed to be more relevant to cloud computing. Capgemeni argued that, significant benefits in security can be gained on adoption of cloud computing over traditional outsourcing.

A further approach has been presented by Nada *et al* [15] for partitioning BPEL, a workflow description language. A program technique is used in order to partition automatically. This approach states that the distribution of data brings several improvements on performance for example reducing network traffic. In addition, the method of Nada *et al* has attempted to reduce the cost of communication between partitions.

From the consumers perspective Dillon *et al* [5] argue that, numerous models of cost can be raised specially with the use of hybrid cloud distribution model where, enterprise data needs to be transferred from its source (private cloud) to its destination (public cloud) and vice versa. The integration of consumer data with the cloud providers is shown to add additional cost. Likewise, Leinwand [16] also discusses the cost of using cloud computing. The charge of an application that generates a lot of bandwidth using Windows Azure Platform has been presented as an example. Additionally, it has been suggested that, if size of data in excess of 50 gigabytes the cloud consumer should buy his own bandwidth.

[2] Standard Performance Evaluation Council

A new methodology is presented by [17], where it evaluates the performance of clouds in order to ensure that a specific service get its proper level. Also, their methodology is an extension to ASTAR method, which defined as a method for evaluation of configurations in the context of service-oriented architectures [18]. An approach that has been developed to consist of five steps: identify benchmark, identify configuration, run tests, analyze and recommend. As a result of the analysis, the recommendations is given for a specific configuration, where the calculation of cost for each service is used for optimization.

An experimental methodology is introduced by Cerotti *et al* [19], where the performance of multi-core systems in the cloud have been evaluated. Several types of benchmarks such as: DaCapo[3] and IOzone[4] have been implemented on VirtualBox[5] and Amazon EC2[6] platforms. Hence, the obtained outcomes are used to acquire estimations of the response time. Although numerous of experiments are implemented on real platforms, the cost only mentioned from the provider side, whereas their findings show that the provision of many single-core machines is more cost-effective than single machines with many cores.

WORKFLOW MODELLING

The importance of modelling business processes is stated by Adriansyah *et al* [20], through an approach that provides an analysis of the performance. The techniques that have been used in their work starting with create a process model via YAWL language and then replay the developed event logs framework against the model. Open source tool have been used such as: Open Source framework process mining ProM[7] and extensible event stream XES[8]. Some similarities to our proposed approach can be observed in modelling the business process and performance analysis. However, the aim is different, as the proposed work will concentrate on the cost in cloud taking into account the importance of evaluation of performance of the modelled systems.

Workflows modelling languages acting essential role in abstracting business processes, modelling and analyzing. Therefore, several workflow modelling languages including YAWL and BPEL will be investigated in order to create a new cost model for the deployment of partitioned workflow over hybrid clouds. In addition to this, extensive experiments will be implemented to simulate the performance behaviour of completion of workflow activities. Also, a comparison between the results those will be obtained from the afore mentioned modelling languages will take place.

YAWL[9] is a workflow language that was developed to overcome the limitations of other workflow management systems. Petri Nets have been chosen as a starting point, due to their support to the most of the workflow patterns. The Petri Nets structure is then extended to add higher abstraction level

[3] http://dacapobench.org/
[4] http://www.iozone.org/
[5] https://www.virtualbox.org/
[6] http://aws.amazon.com/
[7] http://www.processmining.org
[8] http://www.xes-standard.org/
[9] http://www.yawlfoundation.org/

patterns [26], [27]. One advantage of YAWL is that it supports human actions. According to [28] the following are some of features offered via YAWL: it can discover the dependencies of control-flow; it can use XML, XQuery and XPath in order to handle data; it can support dynamic workflow, thus can deal with new changes; it provides an environment that can straightforwardly make changes to particular requirements. On the other hand, YAWL is not standard language and lacks industry support.

BPEL is a workflow modelling language designed to support the representation of business process specifically with web services aspects and it is based on XML language. BPEL is an orchestration language (i.e. identifies an executable process, which involves exchange message with other systems). BPEL is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) and supported by numerous organizations such as: IBM, Microsoft and Oracle. According to [29] BPEL can be described in two ways: firstly, abstract business process, secondly executable business process. In addition, different versions of BPEL are available for instance BPEL4WS and WS-BPEL. However, the BPEL language does not offer enough support to patterns for example: Synchronization and Advanced Branching [29].

IV. The Model

Four simple PEPA models have been derived based on the four main distribution choices that adopted from [1] and illustrated in Figure 2. ***Option 1*** is modelled as sequential components of services. Additionally, the activities *read* and *anonymize* will be distributed on a private cloud whereas *analyze* and *write* will be deployed on public clouds. Also, the co-ordination equation is modelled. ***Option 2*** is modelled as sequential components of services similar to option 1. However, the sequential components of clouds are different, where only *write* activity will be deployed on the public clouds. Also, the system equation will be same as option 1. Identically, ***Option 3*** is modelled as a sequential components of services same as option 1 and option 2. Nevertheless, the consecutive components of clouds are different, where only *analyse* activity will be deployed onto the public clouds. Likewise, system equation will be same as option 1 and option 2. Even though ***option 4*** has similar representation to the previous workflow activities, it differs from the former options, where will be only positioned over a private cloud.

In each model there is a component which represents the workflow and components that represent the public and private cloud services. The difference between each model is only in the services offered by the public and private clouds. By considering each option and investigating different capacities of public cloud servers, it is possible to explore which configuration offers the best overall performance.

The workflow component, which is identical in each model, is specified as a simple sequential flow, as follows:

$$Service_0 \stackrel{def}{=} (readData, r).Service_1$$

$$Service_1 \stackrel{def}{=} (anonymize, s).Service_2$$

$$Service_2 \stackrel{def}{=} (analyze, t).Service_3$$

$$Service_3 \stackrel{def}{=} (writeResult, \underline{r}).Service_0$$

The private and public cloud components are then specified for each model as follows.

- Option 1

$$Private \stackrel{def}{=} (readData, r).Private + (anonymize, s).Private$$

$$Public \stackrel{def}{=} (analyze, t).Public + (writeResult, r).Public$$

- Option 2

$$Private \stackrel{def}{=} (readData, r).Private + (anonymize, s).Private + (analyze, t).Private$$

$$Public \stackrel{def}{=} (writeResult, r).Public$$

- Option 3

$$Private \stackrel{def}{=} (readData, r).Private + (anonymize, s).Private + (writeResult, r).Private$$

$$Public \stackrel{def}{=} (analyze, t).Public$$

- Option 4

$$Private \stackrel{def}{=} (readData, r).Private + (anonymize, s).Private + (analyze, t).Private + (writeResult, r).Private$$

The system equation for options 1 to 3 is given as,

$$System \stackrel{def}{=} Service_0 [N_1] \underset{L}{\bowtie} (Private [N_2] \,||\, Public [N_3])$$

Where the cooperation set $L = \{readData, anonymize, analyze, writeResult\}$. For option 4, where the entire workflow is deployed only on the private cloud, the system equations is given by,

$$System \stackrel{def}{=} Service_0 [N_1] \underset{L}{\bowtie} Private [N_2]$$

It is important to note here that the rates of each action are specified in both the workflow component ($Service_i$) and the corresponding cloud component (*Private* or *Public*). In the terminology of PEPA, this is referred to as active-active cooperation. The reason for specifying the rates in this way is because we wish to exploit the definition of the apparent rate in such a way to investigate the scalability of the cloud provision. In this way the apparent rate of the action *anonymize*, for example, will be given by the product of *s* multiplied by the minimum of the number of $Service_1$ and *Private* components currently enabled. That is, an individual instance of the *anonymize* action will never be be served at a rate greater than *s*, regardless of how many instances there are or how many servers are available. This condition might not always be preserved if we were to use passive actions in our model.

V. Experimental Results

The models introduced above were analysed using the PEPA Eclipse Plug-in. We assume that the *analyze* action will be the most computationally expensive (hence the rate *t* is relative small) and that *readData* and *writeResults* are relatively fast. We also assume that the rates of *readData* and *writeResults* actions are the same (specified as *r* in the models above). Thus we have focused our experiments in a number of scenarios whose rates for *readData*, *anonymize*, *analyze* and *writeResults*, are given in Table 1. In each of the models representing options 1, 2 and 3 were have further considered different capacities within the public cloud by varying N_2,

which represents the number of servers. For ease of comparison, the number of servers in the private cloud, N_3, has been fixed as 1 and the number of workflow instances, N_1, has been fixed at 20.

Table 1: Activities rates (r, s and t) by using variety of assumptions

	Rates		
Assumption	***r***	***s***	***t***
1	1	0.1	0.01
2	1	0.1	0.001
3	1	0.1	0.0001
4	1	0.01	0.0001
5	1	0.01	0.00001

In the first set of experiments we compute throughput based on the direct continuous time Markov chain steady state solution in the PEPA Eclipse Plug-in. This gives rise to a set of models each with 1771 states whose results are shown in Figures 5-9.

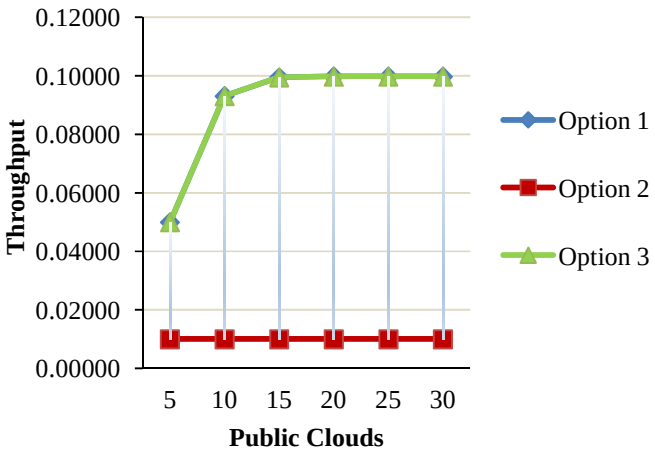


Figure 5: Throughput of options (1, 2 and 3) by using rates of assumption 1 from table 1.

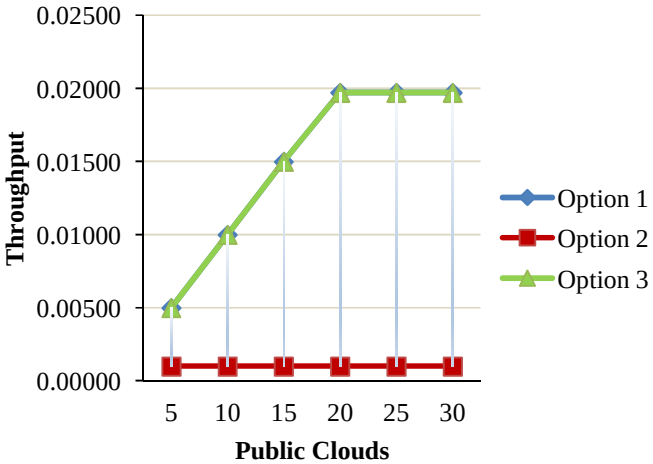


Figure 6: Throughput of options (1, 2 and 3) by using rates of assumption 2 from table 1.

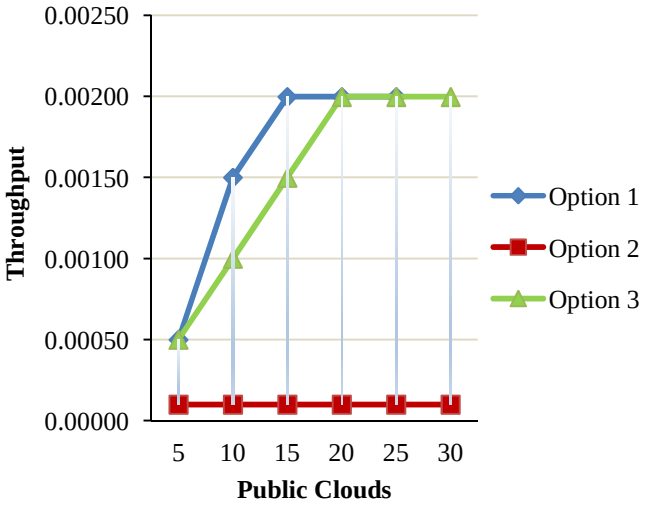


Figure 7: Throughput of options (1, 2 and 3) by using rates of assumption 3 from table 1.

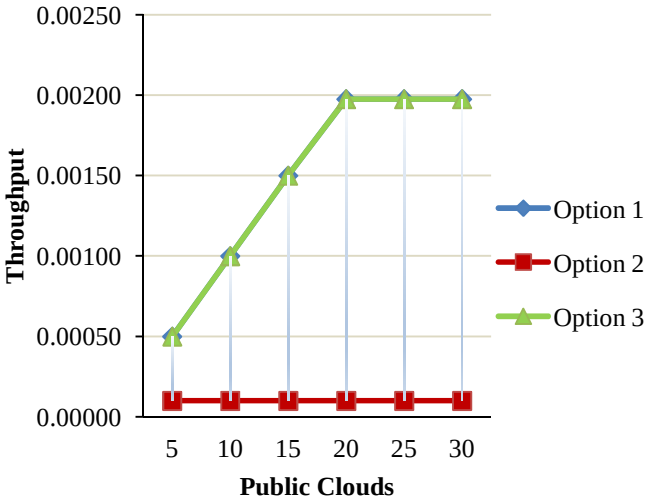


Figure 8: Throughput of options (1, 2 and 3) by using rates of assumption 4 from table 1.

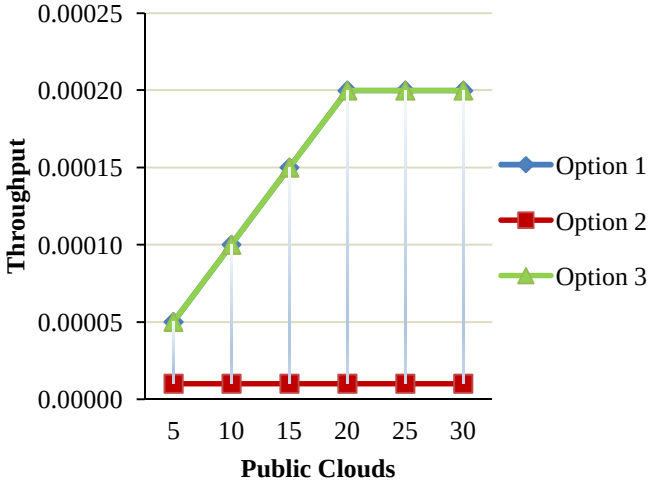


Figure 9: Throughput of options (1, 2 and 3) by using rates of assumption 5 from table 1.

The first observation to make from this set of experiments is that option 1 and option 3 perform almost identically in each case. This is not surprising given that these options differ only in whether the relatively fast *writeResults* action is performed in the public or private cloud. A further observation is that option 2 has a consistently lower throughput than either of the other options. This is unsurprising given that the relatively slow *analyze* action is being performed in the (non-scaled) private cloud in option 2, whereas it is performed in the (scalable) public cloud in options 1 and 3. In each case there is no increase in throughput beyond 20 servers, because there are only 20 workflow instances. In some cases the maximum throughput is reached with fewer servers as the low capacity of the private cloud means that actions performed there become a bottleneck in the system.

Figure 10 shows the corresponding results for option 4, where all actions are deployed on the single instance of the private cloud. Here it is obvious that the throughput is the same as the rate of the *analyze* action, as this becomes the bottleneck.

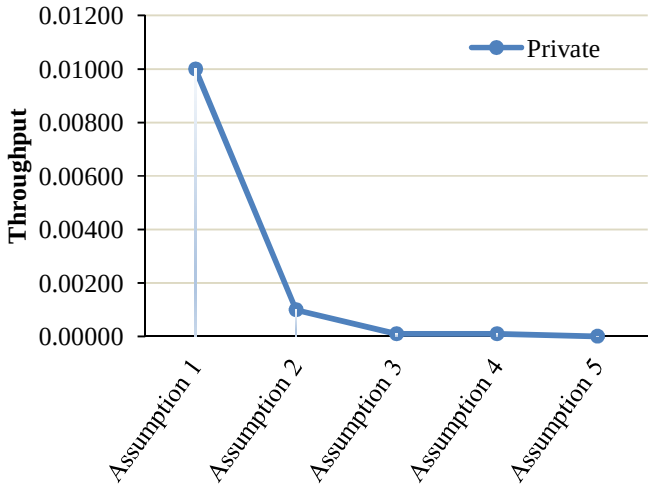


Figure 10: Throughput of option 4 on a private cloud

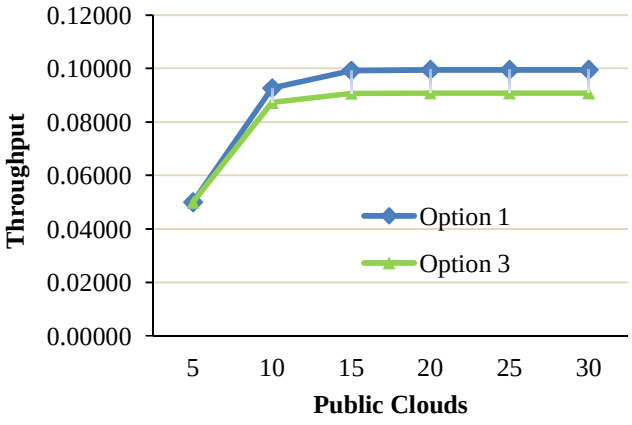


Figure 11: Throughput of options 1 and 3, where r=1, s=0.1, t=0.01 and w=0.1.

In Figure 11 we further investigate the behaviour of options 1 and 3 by introducing a different rate for the *writeResults* action, which has been given the label w. We investigated a number of different combinations of rates and in many situations there was very little difference between the performance of each option. However, under certain circumstances, such as that shown in Figure 11, the overhead of *writeResults* in the private cloud is sufficient to limit the throughput of option 3.

Clearly the experiments presented here are on a limited scale and in many practical circumstances we might wish to consider greater volumes of both workload and service capacity. In general this becomes problematic when using a direct solution of a continuous time Markov chain as the underlying state space rapidly becomes prohibitively large. A state space in the order of 1 million states can be solved directly with standard methods given sufficient memory to store the large sparse matrices required, although such a capacity is well beyond most users of the PEPA Eclipse Plug-in. Hence if we wish to investigate larger systems than the one explored above, we need to utilise other methods.

The PEPA Eclipse Plug-in is equipped with two scalable analysis approaches, stochastic simulation (via Gillespie's method) and fluid flow approximation by means of ordinary differential equations [25]. These techniques not only allow much larger systems to be considered, but they also facilitate transient analysis which provides further insight into the system behaviour.

Figures 12 and 13 show the transient evolution of the system under option 1 before steady state is reached. In both instances the number of workflow instances, N_1, is 20 and the rates employed are those given by assumption 1 in Table 1. In Figure 12 the number of public instances, N_3, is 5 and in Figure 13 the number of public instances is 10. The graphs show the populations of each service type, i.e. the number of workflows that are doing each action at any given time. The two graphs look fairly similar and certainly the populations of $Service_0$ and $Service_3$ appear indistinguishable. However, the populations of $Service_1$ and $Service_2$ are subtly altered by the increase in service capacity. In Figure 12 the population of $Service_1$ is decreasing steadily over time as *anonymize* actions are completed, with the population of $Service_2$ increasing correspondingly. Recall from Figure 5 that the throughput here is fairly low, so very few workflow instances are completing. However, when the service capacity is doubled in Figure 13 the situation changes due to the significant increase in throughput. Now the $Service_3$ population is levelling off as more workflow instances complete. Thus the system is approaching steady state much more rapidly than in Figure 12.

A more dramatic effect can be observed if we increase the number of workflow instances to 2000, as in Figure 14 and 15. Here we observe the evolution of the system under option 3 over a much longer period. Even with a long period of observation the small capacity system in Figure 14 (N_3=5) fails to reach steady state. However, when we double the service capacity in Figure 15 (N_3=10), steady state is reached relatively quickly. The importance of these observations in a practical setting is to illustrate that increasing the service capacity not only has a significant effect on throughput, but also means that the deployment is more stable and the predicted (steady-state) performance is more likely to be observed at any given instant of time.

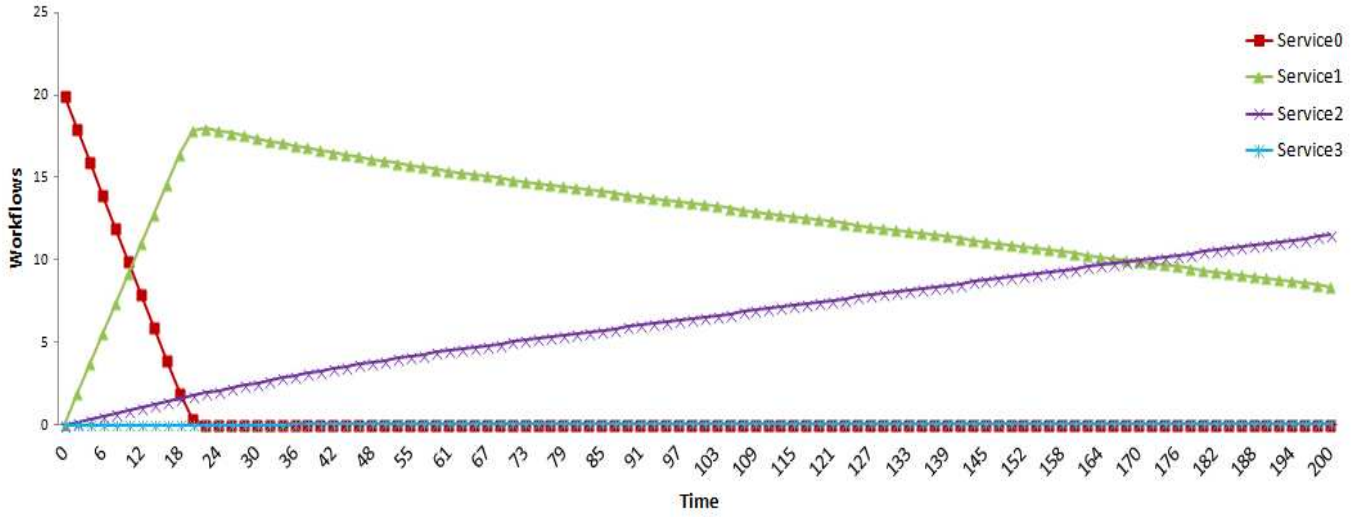


Figure 12: ODE transient analysis for 20 workflows instances in option 1 model using assumption 1 from Table 1 with 5 public instances.

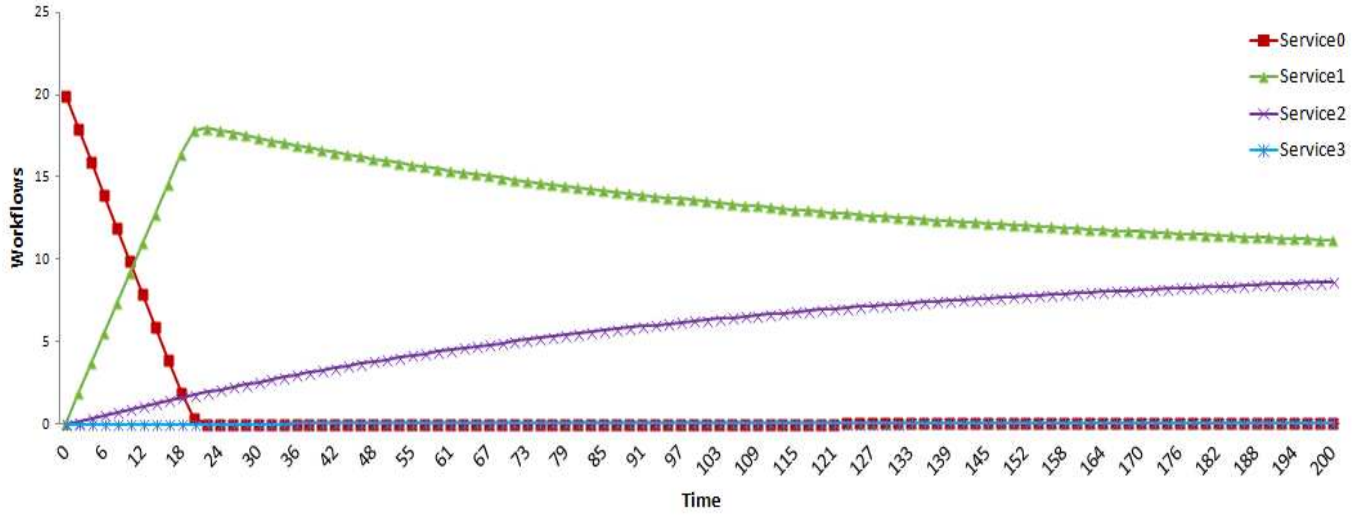


Figure 13: ODE transient analysis for 20 workflows instances in option 1 model using assumption 1 from Table 1 with 10 public instances.

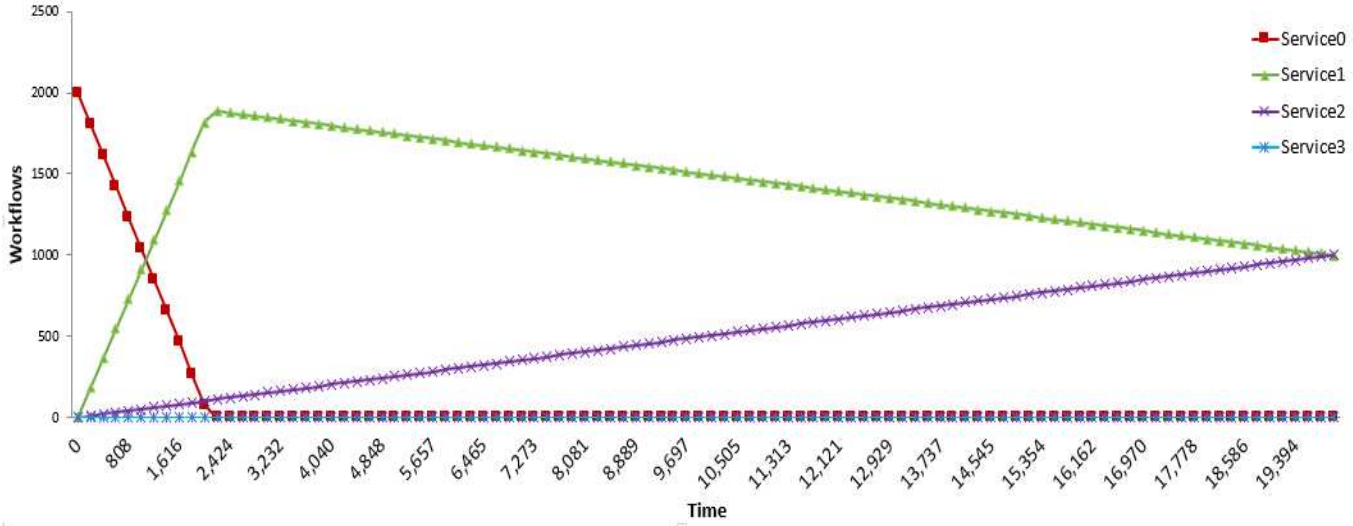


Figure 14: ODE analysis for 2000 workflows instances in option 3 model using assumption 1 from table 1 with 5 public instances

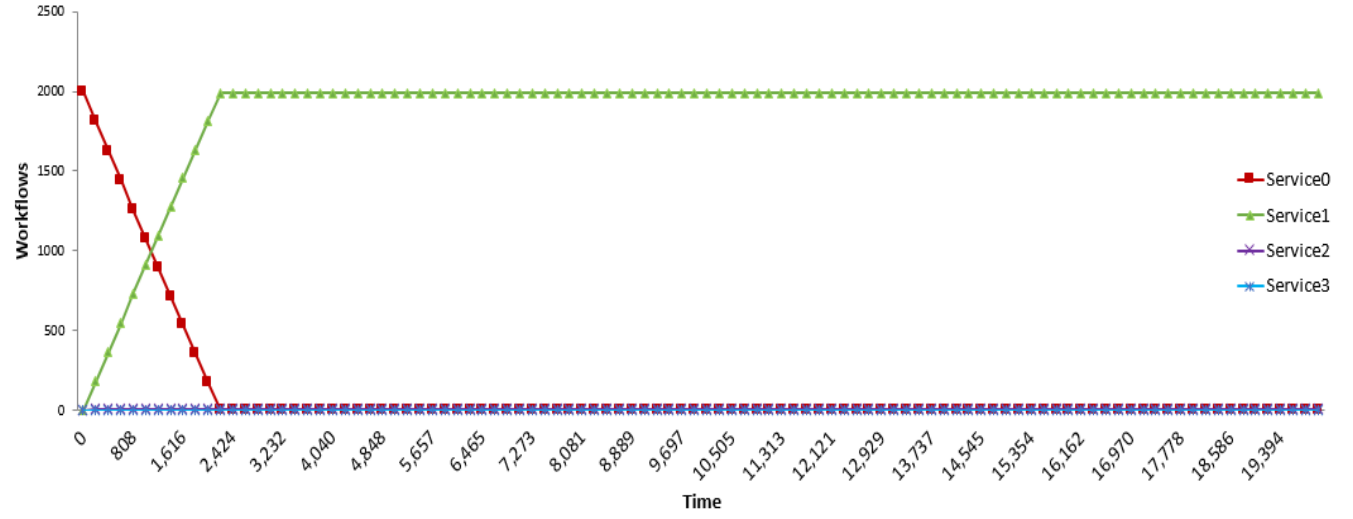


Figure 14: ODE analysis for 2000 workflows instances in option 3 model using assumption 1 from table 1 with 10 public instances

VI. Conclusions and Further Work

In this paper we have presented some initial work in modelling workflow deployment using PEPA. The aim of this work is to explore the costs associated with different security decisions. This paper has been motivated through an example of a healthcare application. We have shown that a simple form of model with standard analysis tools can provide insight into the behaviour of the system in question.

The system we have explored in this paper is clearly very simple. The application we have studied has a linear flow, whereby there is a fixed sequence of actions with no choice or deviation. It will therefore be interesting to explore more complex workflows, for example those involving choice and loops, to investigate model behaviours that may arise in such scenarios. In addition it would be interesting to consider different sets of public cloud servers deployed for different actions.

The model we have developed to study this application also has some limitations. The most significant of these is that we have not modelled any data transfer costs. Clearly it would be a simple matter to add some network delays between actions being undertaken in different locations. This would enable a clearer comparison between the performance of different deployment options in public and private clouds where different transfer costs will be evident.

We have considered servers to belong to a homogeneous set. A consequence of this assumption is that an action will occur at the same rate wherever it is executed (in the private cloud or on any of the public cloud servers). In practice there are a wide range of service levels which can be purchased within clouds and within those options there is wide range of performance that might be experienced. Predicting exactly what a provider will offer on any specification is not generally possible in the current evolution of cloud systems. However it would be relatively straight forward to give different performance characteristics for different systems (at least public and private) and compute the average performance at different service levels.

Clearly to have practical value approaches such as the one described here need to be validated against real implementations. Performing such experiments in a rigorous way is a difficult and time-consuming business. Our aim therefore is to further develop the modelling approach to show the potential of this line of investigation before undertaking work to validate the results.

VII. Acknowledgements

The authors would like to take this opportunity to thank Prof Paul Watson of Newcastle University for taking time to explain his approach and the example used in this paper, as well as providing the authors with a copy of his analysis tool.

References

[1] P. Watson, "A multi-level security model for partitioning workflows over federated clouds," *Journal of Cloud Computing*, vol. 1, pp. 1-15, 2012/07/28 2012.

[2] E. Goettelmann, W. Fdhila, and C. Godart, "Partitioning and Cloud Deployment of Composite Web Services under Security Constraints," in *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, 2013, pp. 193-200.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," U.S. Department of Commerce, National Institute of Standards and Technology2011.

[4] J. C. Mace, A. Van Moorsel, and P. Watson, "The case for dynamic security solutions in public cloud workflow deployments," in *Dependable Systems and Networks Workshops (DSN-W), 2011 IEEE/IFIP 41st International Conference on*, 2011, pp. 111-116.

[5] T. Dillon, W. Chen, and E. Chang, "Cloud Computing: Issues and Challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, pp. 27-33.

[6] Y. Chen, V. Paxson, and R. H. Katz, "What's New About Cloud Computing Security?," UCB/EECS-2010-5, 2010.

[7] F. Gens. (2009, 31 July). *New IDC IT Cloud Services Survey: Top Benefits and Challenges*. Available: http://blogs.idc.com/ie/?p=730

[8] Capgemeni, "Putting Cloud security in perspective," Tech. rep.,Capgemeni2010.

[9] D. E. Bell, L. J. La Padula, and C. Mitre, *Secure computer systems*. Bedford, Mass.; Springfield, Va.: Mitre Corp.; Distributed by National Technical Information Service, 1973.

[10] S. Pearson and T. Sander, "A mechanism for policy-driven selection of service providers in SOA and cloud environments," in *New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on*, 2010, pp. 333-338.

[11] O. Wenge, M. Siebenhaar, U. Lampe, D. Schuller, and R. Steinmetz, "Much Ado about Security Appeal: Cloud Provider Collaborations and Their Risks," in *Service-Oriented and Cloud Computing*. vol. 7592, F. Paoli, E. Pimentel, and G. Zavattaro, Eds., ed: Springer Berlin Heidelberg, 2012, pp. 80-90.

[12] W. Mach and E. Schikuta, "A Consumer-Provider Cloud Cost Model Considering Variable Cost," in *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, 2011, pp. 628-635.

[13] W. Zhenyu and P. Watson, "Dynamic Exception Handling for Partitioned Workflow on Federated Clouds," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, 2013, pp. 198-205.

[14] Y. Zhao and N. Thomas, "Efficient solutions of a PEPA model of a key distribution centre," *Performance Evaluation*, vol. 67, pp. 740-756, 8// 2010.

[15] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, *et al.*, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley UCB/EECS-2009-28, February 10 2009.

[16] A. Leinwand. (2009). *The Hidden Cost of the Cloud: Bandwidth Charges*. Available: http://gigaom.com/2009/07/17/the-hidden-cost-of-the-cloud-bandwidth-charges/

[17] V. Stantchev, "Performance Evaluation of Cloud Computing Offerings," in *Advanced Engineering Computing and Applications in Sciences, 2009. ADVCOMP '09. Third International Conference on*, 2009, pp. 187-192.

[18] V. Stantchev, *Architectural Translucency*. Berlin: GITO-Verlag, , 2008.

[19] D. Cerotti, M. Gribaudo, P. Piazzolla, and G. Serazzi, "End-to-End Performance of Multi-core Systems in Cloud Environments," in *Computer Performance Engineering*. vol. 8168, M. Balsamo, W. Knottenbelt, and A. Marin, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 221-235.

[20] A. Adriansyah, B. F. van Dongen, D. Piessens, M. T. D. Wynn, and M. Adams, "Robust Performance Analysis on YAWL Process Models with Advanced Constructs," *Journal of Information Technology Theory and Application (JITTA)*, vol. 12, 2012.

[21] J. Hillston, *A compositional approach to performance modelling*. Cambridge; New York: Cambridge University Press, 1996.

[22] M. Tribastone, A. Duguid, and S. Gilmore, "The PEPA eclipse plugin," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, pp. 28-33, 2009.

[23] J. Hillston and L. Kloul, "A Function-Equivalent Components Based Simplification Technique for PEPA Models," in *Formal Methods and Stochastic Models for Performance Evaluation*. vol. 4054, A. Horváth and M. Telek, Eds., ed: Springer Berlin Heidelberg, 2006, pp. 16-30.

[24] D. Radev, V. Denchev, and E. Rashkova. (May 2005, Steady-state solutions of Markov chains.

[25] J. Hillston, "Fluid flow approximation of PEPA models," in *Quantitative Evaluation of Systems, 2005. Second International Conference on the*, 2005, pp. 33-42.

[26] W. M. P. van der Aalst and A. H. M. ter Hofstede, "YAWL: yet another workflow language," *Information Systems*, vol. 30, pp. 245-275, 6// 2005.

[27] W. P. van der Aalst, L. Aldred, M. Dumas, and A. M. ter Hofstede, "Design and Implementation of the YAWL System," in *Advanced Information Systems Engineering*. vol. 3084, A. Persson and J. Stirna, Eds., ed: Springer Berlin Heidelberg, 2004, pp. 142-159.

[28] (6 June 2014). *YAWL (Yet Another Workflow Language)*. Available: http://www.yawlfoundation.org/

[29] M. Vasko and S. Dustdar, "A view based analysis of workflow modeling languages," in *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, 2006, p. 8 pp.

[30] Arthur H. M. Hofstede, Wil M. P. van der Aalst, Michael Adams, and N. Russell, "Modern Business Process Automation YAWL and its Support Environment," ed. Springer-Verlag Berlin Heidelberg 2010, pp. 241-242.

LINE: Efficient Reliability and Performance Analysis of Layered Queueing Models [Extended Abstract]

Juan F. Pérez, Giuliano Casale[1,2]

Department of Computing
Imperial College London
London, UK

In this paper we introduce Line, a software tool for evaluating Layered Queueing Network (LQN) models. LQNs are a class of stochastic models well-suited for the performance evaluation of complex software systems [3]. LQNs are particularly popular in model-driven engineering, where the generation of LQNs can be automated from high-level software specifications such as UML MARTE [6,8], PCM [1,5], or CBML [9]. LQNs provide great flexibility as they are able to represent a number of features relevant for software applications, such as resource pooling, synchronous calls between components, admission control, among others. Further, efficient methods and tools exist to automatically solve LQN models and compute relevant performance measures, e.g., the mean-value analysis approximations in the LQN solver (LQNS) [4].

In spite of these advantages, LQNs also have some limitations. For instance, the LQN framework lacks a mechanism to specify the reliability of the hardware and software resources used by a software system. In addition, the efficient solution of LQN models rely on methods that are very effective to compute *mean* performance measures, but do not provide information such as response time distributions or higher-order moments of performance and reliability metrics.

To overcome these limitations, we present Line, a tool that relies on a *fluid* multi-class queueing network (QN) model to efficiently evaluate LQNs. The advantage of relying on a fluid model lies in its ability to make the analysis of large-scale models tractable The fluid approach also provides approximations of the response-time distribution of requests processed by the application, a key metric to assess percentile-based service level objectives. In addition, Line offers the ability to ex-

[1] The research of Giuliano Casale and Juan F. Pérez leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no. 318484.

[2] Email: j.perez-bernal@imperial.ac.uk, g.casale@imperial.ac.uk.

plicitly model reliability aspects through the specification of a *random environment*, i.e., a stochastic model of the operational environment the software system evolves in. With Line it is thus possible to model software applications in the presence of resource failures, servers breakdown and repairs, start-up delays, and interference due to multi-tenancy, an increasingly important issue for cloud applications. Line also supports Coxian distributions for the processing times, a class of distributions that can be used to approximate empirical data more flexibly than with exponential distributions.

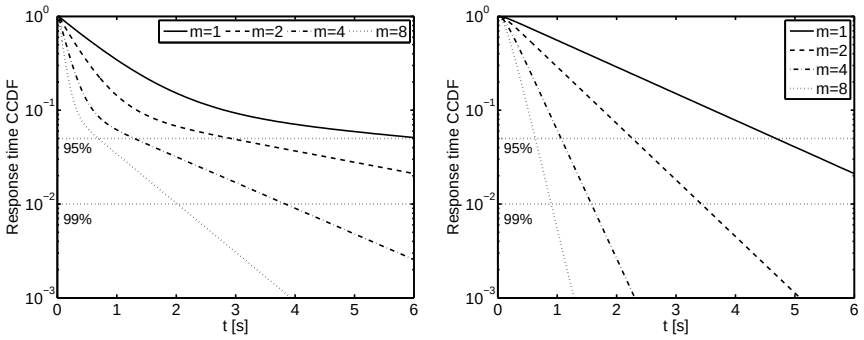
The modeling framework implemented in Line combines and generalizes early results obtained in [7] and [2] to assess the reliability and performance of complex software system subject to reliability risks. Compared to these early works, Line relaxes the assumptions on the processing times, allowing them to follow Coxian distributions. Further, Line extends the blending methodology, necessary to incorporate the random environments, to handle *multi-class* and *processor sharing* resources, instead of the *single-class* and *first-come first-serve* resources considered in the original method.

Line has been developed to allow different usage types. Being developed in MATLAB, it can be used as a library of methods to solve LQN models and compute performance metrics. Further, its binary distribution can directly interact with the Palladio Bench tool [1] to evaluate Palladio Component Models (PCMs). As a result, the use of Line from Palladio Bench only requires the user to specify Line as its LQN solver. Line is readily available for download at http://code.google.com/p/line/.

Example

We now illustrate the potential of Line with an example based on the Media Store application presented in [1]. A PCM of this application is provided as part of the Palladio Bench distribution. The application consists of a web GUI, a store manager component, a watermaking component, and a mySQL database. The database is deployed on a database server, while the other components are deployed on the application server. The main resources to consider are the CPU of both servers, and the disk (HDD) of the DB server. The application serves two classes of requests, Download and Upload, submitted by a closed population of N users. Compared to the PCM distributed with Palladio Bench, we have eliminated the connection pool associated to the database, as this creates a finite capacity region in the QN model that the current version of Line cannot handle. The QN model obtained from the Media Store PCM has 5 stations: a delay station that models the users think times, the App. Server CPU station, the DB CPU station, the DB HDD station, and a station that models the network between the App. Server and the DB Server.

To illustrate the effect of the processing time *distribution* on the request response time (RT) *distribution* we set all the processing times to be exponentially-distributed, except for the Upload requests at the DB HDD, where we assume a Coxian distribution with SCV equal to 5. The baseline case assumes all the stations have a single server, and its results are shown in Figure 1 with the label $m = 1$. The remaining cases are generated by increasing the number of servers in the DB HDD station to m, and at the same time reducing the mean think time of both request



(a) RT CCDF - Upload (b) RT CCDF - Download

Fig. 1. Response time Varying number of servers

types by a factor m, with $m = 2, 4, 8$. Figures 1(a) and (b) show how increasing the number of servers, while facing a similar load, reduces the RT faced by all the requests. Here we display the RT Complementary Cumulative Distribution Function (CCDF) for each request class, and mark the 95% and 99% percentiles for clarity.

Although the impact is significant for both Upload and Download requests, we observe that the actual RT distribution differs due to the different processing time distributions. The results reveal that while $m = 3$ servers is enough to offer a RT of less than to 2 s to 99% of the Download requests, this only guarantees a 4 s RT to the same proportion of the Upload requests. The RT distribution obtained with Line can thus be used to evaluate percentile-based service-level objectives. We further observe how the multi-class assumption in Line, with differentiated processing times, reveals different RT distributions for different request classes. This information allows the analysis at the request class level, which can be valuable as some design decisions can have a very different impact for different request classes.

References

[1] Becker, S., H. Koziolek and R. Reussner, *Model-based performance prediction with the palladio component model*, in: *Proceedings of the 6th WOSP*, 2007, pp. 54–65.

[2] Casale, G. and M. Tribastone, *Modelling exogenous variability in cloud deployments*, SIGMETRICS Performance Evaluation Review **40** (2013), pp. 73–82.

[3] Franks, G., T. Al-Omari, M. Woodside, O. Das and S. Derisavi, *Enhanced modeling and solution of layered queueing networks*, Software Engineering, IEEE Transactions on **35** (2009), pp. 148–161.

[4] Franks, G., P. Maly, M. Woodside, D. C. Petriu, A. Hubbard and M. Mroz, "Layered Queueing Network Solver and Simulator User Manual," Carleton University (2013).

[5] Koziolek, H. and R. Reussner, *A model transformation from the palladio component model to layered queueing networks*, in: *Performance Evaluation: Metrics, Models and Benchmarks*, Lecture Notes in Computer Science **5119**, Springer, 2008 pp. 58–78.

[6] Object Management Group (OMG), *A UML profile for MARTE: Modeling and analysis of real-time embedded systems, beta 2*, Technical Report OMG Document Number: ptc/2008-06-09, OMG (2008).

[7] Pérez, J. F. and G. Casale, *Assessing SLA compliance from palladio component models*, in: *Proceedings of the 15th SYNASC*, 2013.

[8] Tribastone, M., P. Mayer and M. Wirsing, *Performance prediction of service-oriented systems with layered queueing networks*, in: *Proceedings of ISoLA 2010*, 2010, pp. 51–65.

[9] Wu, X. and M. Woodside, *Performance modeling from software components*, in: *Proc. 4th International Workshop on Software and Performance (WOSP)*, 2004.

Performance Analysis of Collective Adaptive Behaviour in Time and Space

Cheng Feng[1,2]

School of Informatics
University of Edinburgh
Edinburgh, UK

Marco Gribaudo[3]

Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano
Milan, Italy

Jane Hillston[1,4]

School of Informatics
University of Edinburgh
Edinburgh, UK

Abstract

Many systems, both natural and engineered, exhibit collective adaptive behaviour. Natural examples are swarming animals and insects, and man-made examples include swarm robots and sensor networks. Indeed many systems have both human and ICT-based agents in play, distributed over some geographical region: an informatics environment as defined by Milner. Over recent years there has been increased interest in modelling these systems in order to understand their dynamic behaviour. Here we consider the subsequent issue of how to define useful performance measures for such systems, based on consideration of a simple, intuitive example.

Keywords: Spatio-temporal modelling, performance modelling, collective adaptive behaviour

1 Introduction

Systems which exhibit collective behaviour have many interesting properties. Examples from the natural world such as swarming animals and insects are often studied

[1] This work is partially supported by the QUANTICOL project 600708.

[2] Email: s1109873@sms.ed.ac.uk

[3] Email: marco.gribaudo@polimi.it

[4] Email: jane.hillston@ed.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

for their emergent behaviour, patterns which become apparent at the population-level but which were not readily apparent from the described behaviour of individuals. In engineered systems this emergent behaviour constitutes the *performance* of the system.

Increasingly large-scale, geographically distributed ICT systems are being developed to support human endeavour in a variety of ways. This can be considered to be the realisation of the *informatics environment* predicted by Milner and Weiser [10,13]. Such systems operate with human agents interacting almost transparently with computing elements. Examples include smart city applications such as smart transportation, smart grid and many modern automotive systems.

In such systems, their transparency and pervasiveness mean that it is perhaps more important than ever to investigate their behaviour from both a qualitative and quantitive point of view prior to deployment. Work is currently underway, for example in the QUANTICOL project [6] to develop modelling formalisms to capture the behaviour of these systems [5,9]. Here we start a complementary investigation into the types of measure that can be derived from spatio-temporal systems. Classic performance measures assume that there is a single locus of operation. When there is a (limited) spatial aspect to behaviour, state annotations are usually used to syntactically distinguish the different locations and regular performance measures are applied. We seek to take a more radical approach to support modelling in which space is modelled explicitly and exploited fully when characterising the behaviour of the system.

2 Leader and Follower Scenarios

We consider a simple scenario in which agents are moving in a two-dimensional grid, as shown in Figure 1. We assume that the grid is finite and that the boundaries are

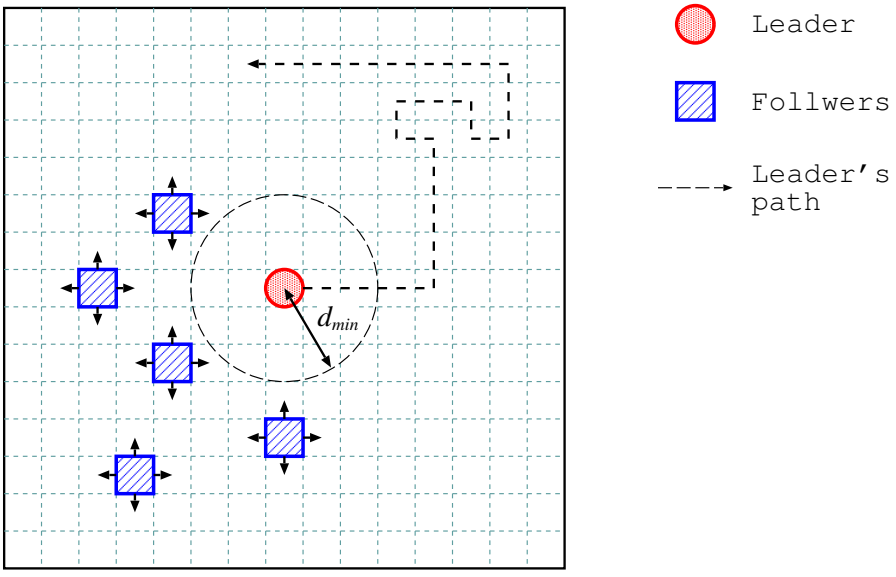


Fig. 1. The *Leader-Follower* scenario.

wrapped, meaning that essentially we are considering movement on a torus. Each

agent moves one step at a time and at each step can move in any direction: north, east, south or west. We assume that there is one distinguished agent, the *Leader* who moves autonomously, performing a random walk over the grid. Any other agent is a *Follower*. The objective of a *Follower* is to mimic the movement of the *Leader*. However, there is a restriction that the *Follower* should keep a minimum distance d_{min} from the *Leader* and should definitely avoid collisions.

In the following we consider a number of scenarios of increasing complexity to illustrate our points. In particular, the model will be specified in PALOMA [5], a new process algebra that is designed for the modelling of spatially distributed collective and adaptive systems. Before describing the scenarios, we first give a brief introduction to PALOMA. For more details the interested reader is referred to [5].

2.1 PALOMA

PALOMA is novel stochastic process algebra that allows the expression of models representing systems comprised of populations of agents distributed over space. In PALOMA each agent is a finite state machine and the language is *conservative* in the sense that no agents are spawned or destroyed during the evolution of a model (although they can cease to change state). The language has a two level grammar:

$$X(\ell) ::= !(\alpha, r).X'(\ell') \mid ?(\alpha, p).X'(\ell') \mid X(\ell) + X(\ell)$$
$$P ::= X(\ell) \mid P \parallel P$$

Agents are parameterised by a *location*, here denoted by ℓ. Agents can undertake two types of actions, *spontaneous* actions, denoted $!(\alpha, r)$, and *induced* actions, denoted $?(\alpha, p)$. When an agent performs a spontaneous action, it does so with a given rate r, which is taken to be the parameter of an exponential distribution, where $1/r$ is the expected duration of the action. Spontaneous actions are broadcast to the entire system, and can induce change in any other agent which enables an induced action with the matching type α. An induced action has an associated probability p, which records the probability that the agent responds to a spontaneous action of the same type. In the style of the Calculus of Broadcasting Systems [11], this can be thought of as the probability that the agent *listens* as opposed to simply *hearing*. Alternative behaviours are represented by the standard choice operator, $+$. A choice between spontaneous actions is resolved via the race policy, based on their corresponding rates. We assume that there is never a choice between induced actions of the same type.

A model, P, consists of a number of agents composed in parallel. There is no direct communication between agents, for example in the style of shared actions in PEPA [7]. Instead, all communication/interaction is via spontaneous/induced actions. When an action is induced in an agent the extent of its impact is specified by a perception function, $u(\alpha, \ell, X, \ell', X')$ where α is the action type, ℓ and X are the location and state of the receiver agent whereas ℓ' and X' are the location and state of the sender agent. This is a further probability which, given the locations of the two agents, their current states and action type involved, determines the likelihood that the induced action occurs. For example, the perception function might have value 1 when the two agents are within a communication radius r of

each other, but a value of 0 whenever the distance between them is greater than r. Obviously this gives a rich set of possible styles of interaction, but note that each agent with an induced action chooses independently whether to respond or not.

2.2 Scenario 1: Passive Followers

In this scenario, we assume the *Leader* can either choose to take a rest with rate r_{rest}, or to move a step along a random direction at the rate of r_{mv}. Moreover, we use p_n, p_s, p_w, p_e to represent the probability to move north, south, west and east, respectively. Thus, the *Leader* agent can be described as follows:

$$\begin{aligned} L(x,y) ::= &!(rest, r_{rest}).L(x,y) + !(n, r_{mv}p_n).L(x,y+1) + \\ &!(s, r_{mv}p_s).L(x,y-1) + !(w, r_{mv}p_w).L(x-1,y) + \\ &!(e, r_{mv}p_e).L(x+1,y) \end{aligned}$$

where $!(rest, r_{rest})$ denotes the *Leader* taking a rest spontaneously at the rate of r_{rest}, and when it does this, it remains in its current position. $!(n, r_{mv}p_n)$ denotes the *Leader* moving a step north by doing an spontaneous action n at the rate of $r_{mv} \times p_n$.

Furthermore, we assume the *Follower* can only move passively when the *Leader* informs it to do so. Thus, we define the *Follower* agent as:

$$\begin{aligned} F(x,y) ::= &?(n, p_l).F(x,y+1) + ?(s, p_l).F(x,y-1) + \\ &?(w, p_l).F(x-1,y) + ?(e, p_l).F(x+1,y) \end{aligned}$$

where the *Follower* agent can move a step in a direction via an induced action, and p_l encodes the probability for the *Follower* to respond to the *Leader*'s movement action which means that the *Follower* may not obey the command from the *Leader* with probability $1 - p_l$.

The perception functions for actions n, s, w and e are simply defined as:

$$\begin{aligned} u(n, \ell, X, \ell', X') &= 1 \\ u(s, \ell, X, \ell', X') &= 1 \\ u(w, \ell, X, \ell', X') &= 1 \\ u(e, \ell, X, \ell', X') &= 1 \end{aligned}$$

which means that the *Follower* will definitely perceive the command from the *Leader*.

2.3 Scenario 2: Active Followers

In this scenario, we allow the *Follower* to be a little bit smarter. More specifically, we introduce an internal *Clock* agent which allows the *Follower* to move actively instead of just listening to the *Leader*'s command. The *Clock* agent is simply defined as:

$$\begin{aligned} Clock(x,y) ::= &!(cn, r_c).Clock(x,y) + !(cs, r_c).Clock(x,y) + \\ &!(cw, r_c).Clock(x,y) + !(ce, r_c).Clock(x,y) \end{aligned}$$

which means that the *Clock* agent will perform each self-jump action cn, cs, cw and ce spontaneously at the rate of r_c.

Then, the *Follower* agent becomes:

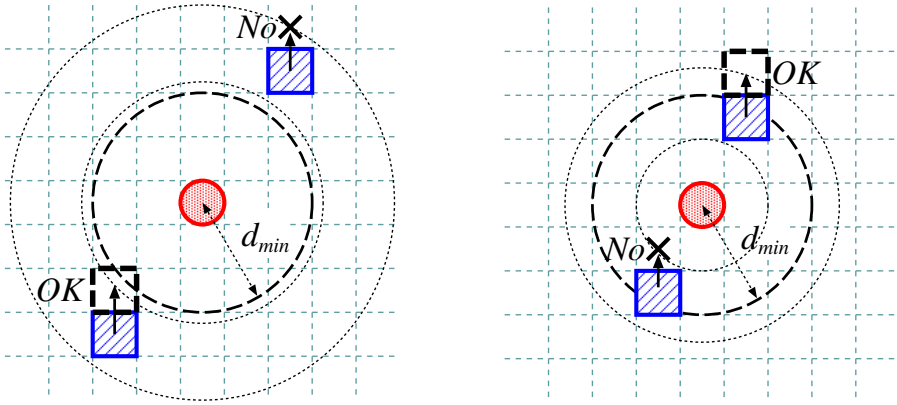
$$\begin{aligned} F(x,y) ::= & ?(n,p_l).F(x,y+1) + ?(s,p_l).F(x,y-1) + \\ & ?(w,p_l).F(x-1,y) + ?(e,p_l).F(x+1,y) + \\ & ?(cn,p_c).F(x,y+1) + ?(cs,p_c).F(x,y-1) + \\ & ?(cw,p_c).F(x-1,y) + ?(ce,p_c).F(x+1,y) \end{aligned}$$

where p_c encodes the probability for the *Follower* to *listen* to a clock instruction.

Then, we define the associated perception function for actions n and cn as:

$$u(n \mid cn, \ell, X, \ell', X') = \begin{cases} 1 & \text{if } ((dist(\ell.x, \ell.y+1, L.x, L.y) < dist(\ell.x, \ell.y, L.x, L.y) \\ & \quad \wedge dist(\ell.x, \ell.y+1, L.x, L.y) > d_{min}) \\ & \quad \vee ((dist(\ell.x, \ell.y+1, L.x, L.y) > dist(\ell.x, \ell.y, L.x, L.y) \\ & \quad \wedge dist(\ell.x, \ell.y, L.x, L.y) < d_{min})) \\ 0 & \text{otherwise} \end{cases}$$

which can be interpreted in the following way: the *Follower* will only perceive the n or cn action from the leader or the internal clock if a step north will let the *Follower* become closer to the *Leader* and the distance to the *Leader* is still larger than d_{min} (Figure 2a), or it will be farther from the *Leader* but the current distance to the *Leader* is less than d_{min} (Figure 2b). To save space, we will not show the perception functions for other actions as they are defined in a similar way.



a) b)

Fig. 2. Behaviour of the follower: a) $dist(\ell.x, \ell.y+1, L.x, L.y) > d_{min}$, b) $dist(\ell.x, \ell.y+1, L.x, L.y) < d_{min}$.

2.4 Scenario 3: Multiple Followers

Here, we put multiple *Follower*s in the system in order to observe some interesting collective behaviour. We assume that *Follower*s always try to avoid bumping into each other. Thus, we add a simple protocol to *Follower*s by modifying the perception functions. For example, the perception functions for n and cn are modified as

follows:

$$u(n \mid cn, \ell, X, \ell', X') == \begin{cases} 1 & \text{if } (|F(\ell.x, \ell.y+1)| = 0 \wedge \\ & \quad ((dist(\ell.x, \ell.y+1, L.x, L.y) < dist(\ell.x, \ell.y, L.x, L.y) \\ & \quad \wedge\, dist(\ell.x, \ell.y+1, L.x, L.y) > d_{min}) \\ & \quad \vee ((dist(\ell.x, \ell.y+1, L.x, L.y) > dist(\ell.x, \ell.y, L.x, L.y) \\ & \quad \wedge\, dist(\ell.x, \ell.y, L.x, L.y) < d_{min}))) \\ 0 & \text{otherwise} \end{cases}$$

where $|F(\ell.x, \ell.y+1)|$ denotes the number of *Follower*s in location $(\ell.x, \ell.y+1)$. This means that a *Follower* will only take a step north when there are no other *Follower* agents in that location. Again, we will not show the modified perception functions for other actions as they are changed in a similar way.

3 Performance Measures

Traditionally, performance measures derived from probability distributions can be broadly divided into three categories:

State-based: an expectation over the states of the system. In its simplest form this is the probability that a certain property holds (Boolean values attributed to states). Utilisation is an example of this type. But such measures can also be based on more meaningful values for states, such as queue length where the value for each state is the number of customers in a queue. When the probability distribution is the steady state distribution the derived values will the average values, where at other times they will be transient, based on the transient probability. When spatial information is also represented in the system, the states of interest may be those in which certain spatial conditions are satisfied. Thus we might think of a form of spatial utilisation, the percentage of time that a particular location or set of locations are occupied.

Rate-based: an expectation over the rates of the system. Typical examples are throughput, loss probabilities, collision probability etc. Essentially these are also calculated as expectations over the states but the rewards associated with the states are now the rate at which events occur within the given state. Again either the transient or the steady state probability distribution may be used in the calculation of the expectation. Here again spatial conditions may be used to identify the states of interest. For example, a collision relies on the state condition that two agents are in the same location at the same time.

Time-based: an average time, or a probability distribution with respect to time with respect to some behaviour. The classic example is perhaps response time which, via Little's Law can be expressed in terms of throughput (a rate-based measure) and average number (a state-based measure). For non steady state measures, a passage time calculation will usually be required.

It is reasonable to expect that in spatio-temporal systems we will also be able to define space-based measures, analogous to time-based ones, which are derived from state and rate-based ones.

3.1 Performance measures of the Leader-Follower Scenarios

When we come to measuring our leader-follower system there are multiple different dimensions to consider and we may choose to abstract one of more either through projection or by averaging.

- The first dimension is *state*. This is the fundamental record of the behaviour of the system. We assume that the behaviour of the agents is characterised by random variables which range over the state space. In this simple example the agents do not have any logical state beyond their current position. But in general we can imagine that agents are also fulfilling some other role in addition to their motion and so they may have other characterisations of state, orthogonal to their location.
- The second dimension is *time*. In the simplest performance analysis we consider the behaviour of a single system or agent with respect to time. This may be transient or elapsed time, or abstract time, in the sense that consideration of steady state behaviour essentially removes the time dimension by assuming stationarity. In this dimension it makes sense to consider the rate at which events occur, the probability of an event occurring within a time bound, the cumulative probability of events, etc.
- The third dimension for our systems is *space*. Here we do not have an abstraction equivalent to steady state, but we do have the possibility to take average values over all space; for example, this is often done in ecological models. If there are different types of agents competing over space (e.g. predators and prey) the system may be characterised at a certain time by the total number of each type present disregarding spatial placement, even though interaction is location aware. This form of spatial abstraction, does not seem appealing from a performance perspective, but is often carried out as a mathematical expediency as more detailed representation is computationally expensive or intractable.
- The fourth dimension, for a collective system, is the *population* of individuals. Here we again typically make an abstraction by shifting to a count or proportion of individuals exhibiting certain characteristics rather than retaining full information about each individual. Thus we may wish to record the number of agents at a given location at a given time, or calculate averages either with respect to time or with respect to location. Or we may consider the behaviour only at steady state when time is abstracted.
- Finally, when we analyse our system through discrete event simulation we have a fifth dimension which is the instances or *trajectories* of our system on which measurements are based.

3.2 Basic measures

Here we are particularly interested in spatio-temporal properties that incorporate both the second and third dimension. The simplest way of doing this is to consider a measure over one dimension at all points in the other. At the state level we can define for any agent A, $loc(A,t)$ be the the location of A at time t as the projection of the spatial dimension onto the time dimension. Conversely, we can define $visit(A,l)$ to be the set of time instants in which agent A was at location l. Measures loc and $visit$ are complementary in the sense that:

$$t \in visit(A,l) \quad \Longrightarrow \quad loc(A,t) = l$$

and

$$loc(A,t) = l \quad \Longrightarrow \quad t \in visit(A,l)$$

Note that there is a big difference in the codomain of the two measures, since loc returns a single point in space, while $visit$ is a relation that returns a set of locations. The difficulty lies in the fact that while an agent can only have one location at a time, it can be at that location multiple times. To define a simpler function we can for example restrict to the last visit or the first visit, and define $\mathit{firstVisit}(A,l) = \min\{visit(A,l)\}$ and $\mathit{lastVisit}(A,l) = \max\{visit(A,l)\}$. This however might not be a proper function, since location l might not be visited by agent A in the considered scenario. An alternative would be define functions over the relation. For example, we can express the *age* of an agent with respect to a location,

$$age(A,l) = |visit(A,l)|$$

where for a relation $\mathcal{R}$, $|\mathcal{R}|$ is the size of the relation. In effect, this counts the number of times that agent A has visited location l.

We can use simulation to derive a large number K of trajectories and use it to estimate the probability that any of the previous properties hold. We can regard this as projecting the measure onto the fifth dimension, the space of trajectories. For example, we can express the probability that an agent A was at location l at time t as:

$$\mathbb{P}(A \text{ is in } l@t) = \frac{\sum_{i=1}^{K} \mathbf{1}(loc(A,t) = l \in traj_i)}{K}$$

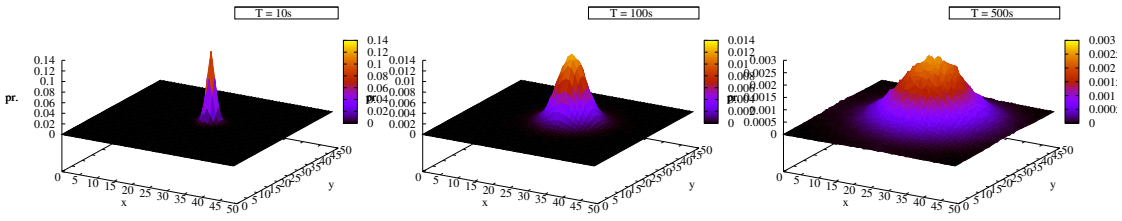
and the probability that a location l is visited by agent A exactly n times as:

$$\mathbb{P}(A \text{ visits } l \text{ exactly } n \text{ times}) = \frac{\sum_{i=1}^{K} \mathbf{1}(age(A,l) = n \in traj_i)}{K}$$

where $\mathbf{1}(predicate)$ is an indicator function which has the value 1 when the predicate is true and the value 0 otherwise.

Figure 3 shows for every location l the probability of the leader being there at times $t = 10, 100, 500s$. In this case the leader performs a step in one of the four directions on average every $4s$ ($r_{mv} = 0.25$) and never rests ($r_{rest} = 0$). The four directions are all equally probable ($p_n = p_e = p_s = p_w = 1/4$). The grid is 50×50 and the leader starts in position $(27, 27)$. The number of simulation runs used is $K = 200,000$. Note that the distribution tends to a bivariate normal distribution

centered in the initial position of the leader, with the variance that increases with time. This is natural since the leader moves according to a pure random walk in the four directions.



$T = 10s$ $T = 100s$ $T = 500s$

Fig. 3. Cell occupancy probability for the leader at different time instants.

Figure 4 shows the distribution of the *age* for four different locations $l_1 = (26, 26)$, $l_2 = (28, 28)$, $l_3 = (25, 25)$ and $l_4 = (24, 24)$. Since the leader starts at $(27, 27)$, locations l_1 and l_2 are at the same distance. Since the movement of the leader is not biased in any direction, they are identical. Locations l_3 and l_4 are at increasing distance: as can be clearly seen, the probability of not entering that location $n = 0$ increases whilst the probability of entering it a larger number of times decreases with the distance from the initial position.

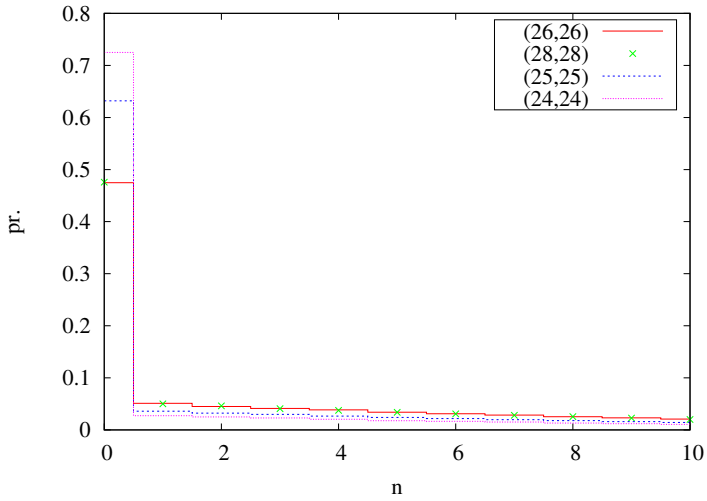


Fig. 4. Distribution of *age*, the number of passages at different positions, for the leader starting at (27,27).

Let us now focus on the probability that a given location l is first/last visited by agent A at time t as:

$$\mathbb{P}(A \text{ first/last visits } l@t) = \frac{\sum_{i=1}^{K} \mathbf{1}([first/last]Visit(A, l) \leq t \in traj_i)}{K}$$

This measure allows us to analyze another peculiarity: temporal distributions, when computed via simulation, are always affected by the finite duration of the considered

trace. In particular, $visit(A, l)$ will always correspond to the set of time instants in which location l was visited during the time horizon spanned by the simulation. If agent A will visit location l after the end of the simulation cycle, this will not be included into $visit(A, l)$. Figure 5 shows the probability distribution of the time at which a fixed location $l = (26, 26)$ is either first visited or last visited. Two different temporal horizon lengths are considered for the simulations. It is interesting to see that the first and last visit to a location tend to coincide at the end of the simulation period: this occurs because if the agent passes over the target cell only once during the considered time horizon, we have $firstVisit(A, l) = lastVisit(A, l)$. It can also be clearly seen that the distribution of the last passage time is a measure that clearly depends on the simulation interval, while the first passage time is less influenced, as long as the time horizon is large enough to allow the agent to reach the considered location.

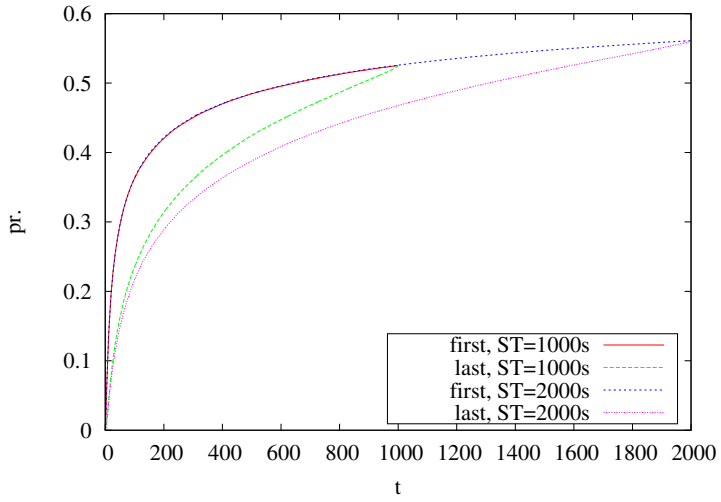


Fig. 5. Distribution of the time of the first and the last visit to location $(26, 26)$ for two different simulation horizons $ST = 1000s$ and $ST = 2000s$.

3.3 Derived measures

From these basic measures we can construct more interesting ones such as distance, $dist$:

$$dist(A_1, A_2, t) = \|loc(A_1, t) - loc(A_2, t)\|$$

e.g. $dist(F, L, t)$ is the distance between agent F and agent L at time t. Thus when we have a single follower we can plot $dist(F, L, t)$ with respect to time to see how the distance between the follower and the leader evolves over time. Again, in a simulation study, the distance can be averaged across all the trajectories to have a global idea of the system behaviour. This is reported in Figure 6 for different behaviours of the leader and of the follower. In particular, both a random movement (Figure 6a) and a fixed route (Figure 6b) for the leader are considered, for different internal clock speed. In all cases, the follower always perceives the clock message, even if this is performed at a different rate ($p_c = 1$). A very noisy channel is considered, with the probability of missing the direction message sent by the leader equal to 95% ($p_l = 0.05$). The target minimum distance between the leader and

the follower is 2.1. For the random walk case, the distance from the leader becomes too high only when the follower does not perform any action to catch the leader ($r_c = 0$). When the clock is considered, the follower can always maintain a good distance from the leader for the two considered speeds of $r_c = 1/16$ and $r_c = 5/32$. This is because the random motion confines the leader to an area that is always relatively close to its initial position. For the follower it is enough to perform some infrequent check to catch up with the leader. If instead a fixed route is followed, the messages sent by the leader become of paramount importance. In this case, the follower is not able to catch up with the leader, unless it performs clock actions at a rate much larger than the speed at which the leader is moving.

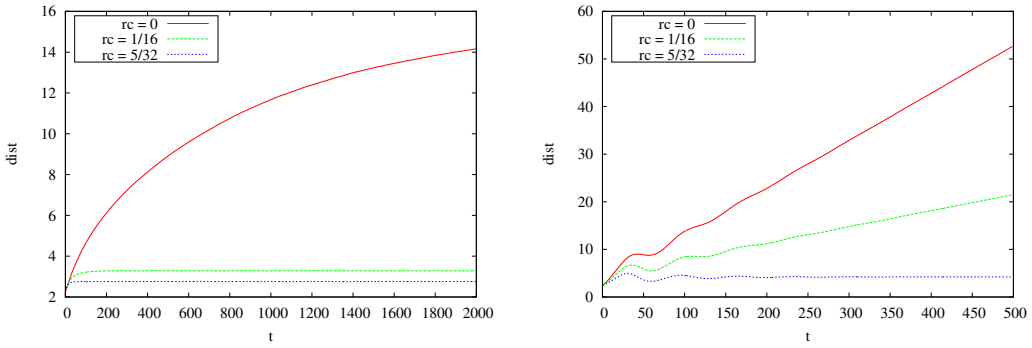


Fig. 6. Average distance between the leader and the follower for different clock rates: a) random walk, b) leader with a predetermined route.

We also assume that we can detect collision. Two agents A_1 and A_2 are said to collide if there exists t such that $loc(A_1, t) = loc(A_2, t)$. In the simple scenarios, 1 and 2, which have only a single follower, we are then interested in the cases where $loc(F, t) = loc(L, t)$. Again, we can use simulation to calculate the probability of a collision at a particular time. We can estimate the probability of collision at time t as

$$\mathbb{P}(collision@t) = \frac{\sum_{i=1}^{K} \mathbf{1}(loc(F,t) = loc(L,t) \in traj_i)}{K}$$

This is shown in Figure 7 for the same cases considered before. For the random movement, the case in which the follower does not perform any recovery action has a large hit probability at the beginning, and rapidly reduces with time. This however is caused by the fact that the leader tends to go far away from the follower thus reducing the hit probability. This is even more visible in the route based movement case, where the hit probability is zero due to the fact that the follower starts losing the leader from the beginning. In the other cases, it is clear that the hit probability converges to a limit value, that is a characteristic of the follower's behaviour, and depends on his parameters. For the route based movement, this is observed only for the case in which the follower reacts at a very high speed, since that is the only one where it is able not to lose the leader.

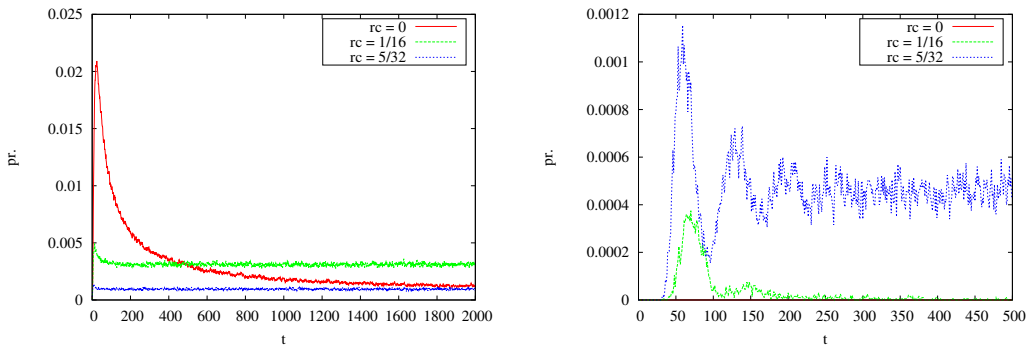


Fig. 7. Hit probability between the leader and the follower for different clock rates: a) random walk, b) leader with a route.

Collision may be generalised to being within the minimal distance d_{min}:

$$\mathbb{P}(too\ close@t) = \frac{\sum_{i=1}^{K} \mathbf{1}(dist(F, L, t) < d_{min} \in traj_i)}{K}$$

Figure 8 shows such probabilities for different values of d_{min} for the case with $r_c = 5/32$. As can be seen, while the hit probability can be very low, by enlarging the distance, the probability of the follower being at the considered contour becomes more tangible.

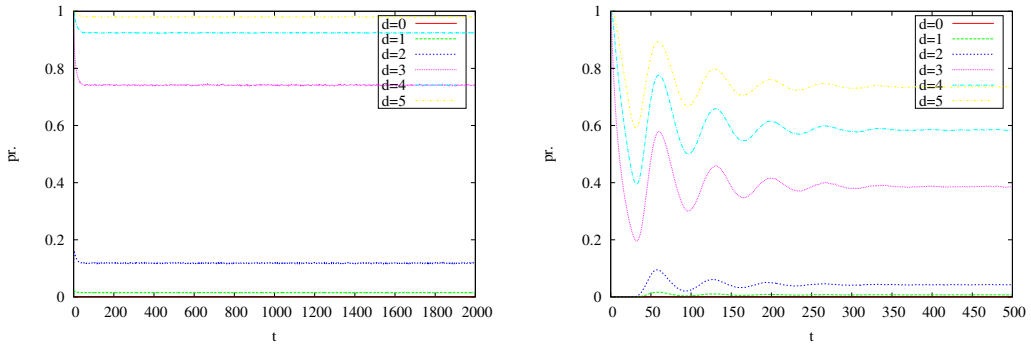


Fig. 8. Probability of the follower being within a given distance from the leader for: a) random walk, b) leader with a route.

3.4 Temporal distance measures

We could also calculate a temporal distance *tdist* between the agents at a specific location l, comparing their first visit times:

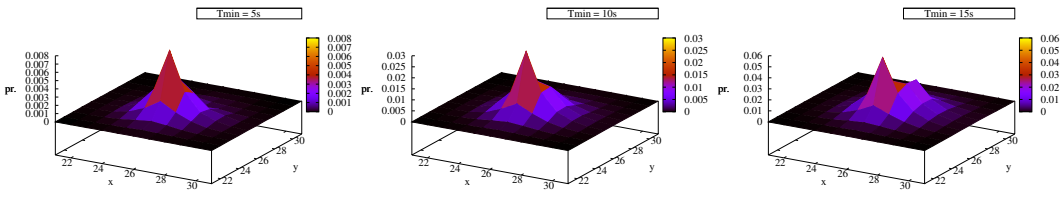
$$tdist(A_1, A_2, l) = \begin{cases} |firstVisit(A_1, l) - firstVisit(A_2, l)| & \text{if } age(A_1, l) > 0 \\ & \wedge\ age(A_2, l) > 0 \\ +\infty & \text{otherwise} \end{cases}$$

Note that considered temporal distance is meaningful only if both agents have visited location l. We have set the temporal distance to $+\infty$ if either of the agents has not passed through the considered location.

When we consider trajectories, we can use the previous definition to compute characteristics with respect to space rather than time. For example, we can define the probability that the passage of two agents in a location l is too close in time as:

$$\mathbb{P}(too\ close@l) = \frac{\sum_{i=1}^{K} \mathbf{1}(tdist(F, L, l) < t_{min} \in traj_i)}{K}$$

where t_{min} is some minimal separation in time that we seek to enforce. Note that by definition of $tdist$, this probability will be zero if the both agents have not passed through l. Figure 9 shows the probability that the temporal distance is less than $t_{min} = 5s$, $t_{min} = 10s$ and $t_{min} = 15s$, for the case with $r_c = 5/32$ and random movement for all the locations $l = (x, y) \in [21, 31] \times [21, 31]$. Although the maximum is in all cases for the initial location of the leader $l = (27, 27)$, the shapes tend to be less symmetric as the threshold increases. This reflects the fact that the protocol tends to keep the initial displacement between the leader and the follower.



$t_{min} = 5s$ $t_{min} = 10s$ $t_{min} = 15s$

Fig. 9. Probability that the temporal distance for the cells in the range $[21, 31] \times [21, 31]$ is less than the considered thresholds t_{min}.

3.5 Multiple followers

When we have multiple followers we can think of defining the same measures for each of the followers, but what we would really like is some measure that reflects the collective behaviour.

Thus thinking of the distance from the leader, when there are N followers we can define $\widehat{dist}$ as the average distance at time t as follows:

$$\widehat{dist}(L, t) = \frac{1}{N} \sum_{i=1}^{N} dist(F_i, L, t)$$

The problem with this is that it can have the same value for very different distributions of followers over space.

Instead we might think of some form of *contours* recording the number, or proportion of agents that are within increasing distances from a given location at a given time. Thus at time t, if there are N agents, the contour $C(l, 1, t)$ would be

defined as

$$C(l,1,t) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\|loc(A_i,t) - l\| \leq 1)$$

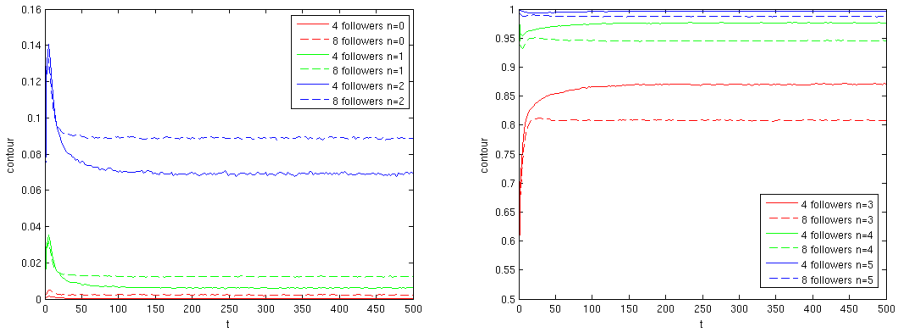
and in general

$$C(l,n,t) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\|loc(A_i,t) - l\| \leq n)$$

In the specific case of the followers and the leader we can define

$$C(loc(L,t),n,t) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\|loc(F_i,t) - loc(L,t)\| \leq n)$$

Figure 10 shows the contour values with 4 and 8 followers. As can be seen from the figure, in the simulation with 4 followers, followers are less likely to break the minimum distance but more likely to keep a good distance from the leader than in the simulation with 8 followers. This is because followers always try to avoid bumping into each other. Thus with more followers, the probability of not perceiving the movement action is also higher (see the perception function in Section 2.4).



(a) $n < d_{min}$ (b) $n > d_{min}$

Fig. 10. The contour $C(loc(L,t),n,t)$ with different number of followers, where $d_{min} = 2.1$

Analogously we can think of time contours,

$$C(t,\delta,l) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=0}^{\delta} \mathbf{1}(loc(A_i,t+j) = l)$$

or

$$C(t,\delta,l) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(loc(A_i,t+\delta) = l)$$

depending on whether we consider the time contours cumulatively or not.

4 Conclusions and future work

In this paper we have made an initial study of the types of measures that it can be interesting to study in systems in which there are both temporal and spatial aspects

of behaviour. These are important characteristics in many collective adaptive systems (CAS), which are geographically distributed systems comprised of interacting but autonomous agents. We have illustrated the ideas with a simple leader-follower system studied in a number of different scenarios via simulation.

Whilst our example system is simple, it is sufficient to highlight the rich forms of information that can be derived from models of CAS, and it is easy to see how the measures we investigated could be adapted to real-life systems. For example, smart transportation systems are examples of CAS [12], where regulatory requirements impose spatial-temporal conditions. Bus operators are subject to the *headway requirement* on frequent routes, which can be regarded as a special case of our leader-follower scenario. Here the timetable would play the role of the leader whilst buses providing the service are the followers. The headway requirement imposes conditions on the spatial and temporal separation of the followers in order to ensure that there is a regular service for the users. In this case the behaviour of the leader is deterministic but the behaviour of the followers, the buses, is subject to stochastic factors, such as traffic and weather conditions as well as human interaction.

In future work we will investigate our identified measures further to see how well they match to the user and operator performance requirements for CAS. In the current work we have worked from first principles, assessing the data available from our simple scenario and the spatio-temporal measures that can be built. An alternative approach would be to work with a spatio-temporal logic to define properties of interest. The use of temporal logic in the context of Markovian-based performance models is well-established [2] and supported by tools such as PRISM [8]. Spatial Logics have also been studied for many years [1] but to the best of our knowledge has yet to be applied in the quantitative context of CTMCs, although recent applications include data verification for CAS [4]. There is little formal treatment of the combination of spatial and temporal logic although it has been considered in a informal way in the analysis of video sequences [3]. Here spatial until formulae were interleaved with temporal until formulae to express conditions on the relative positions of objects in an image as time progressed. In quantified spatio-temporal logic we would seek to attribute a value to such properties, just as probabilities are associated to temporal properties expressed in CSL.

References

[1] Aiello, M., I. Pratt-Hartmann and J. van Benthem, editors, "Handbook of Spatial Logics," Springer, 2007.

[2] Baier, C., B. R. Haverkort, H. Hermanns and J.-P. Katoen, *Model-checking algorithms for continuous-time markov chains*, IEEE Trans. Software Eng. **29** (2003), pp. 524–541.

[3] Bimbo, A. D., E. Vicario and D. Zingoni, *Symbolic description and visual querying of image sequences using spatio-temporal logic*, IEEE Trans. Knowl. Data Eng. **7** (1995), pp. 609–622.

[4] Ciancia, V., S. Gilmore, D. Latella, M. Loreti and M. Massink, *Data verification for collective adaptive systems: spatial model-checking of vehicle location data*, in: *Proceedings of the 2nd FoCAS Workshop on Fundamentals of Collective Adaptive Systems*, London, England, 2014, to appear.

[5] Feng, C. and J. Hillston, *PALOMA: A process algebra for located markovian agents*, 11th International Conference on the Quantitative Evaluation of Systems (2014), to appear.

[6] FET Proactive FOCAS Project 600708, *QUANTICOL: A quantitative approach to management and design of collective and adaptive behaviours*, www.quanticol.eu.

[7] Hillston, J., "A Compositional Approach to Performance Modelling," CUP, 2005.

[8] Kwiatkowska, M. Z., G. Norman and D. Parker, *PRISM 4.0: Verification of probabilistic real-time systems*, in: *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, Lecture Notes in Computer Science **6806** (2011), pp. 585–591.

[9] Latella, D., M. Loreti, M. Massink and V. Senni, *Stochastically timed predicate-based communication primitives for autonomic computing*, Technical report, QUANTICOL project (2014).

[10] Milner, R., "The Space and Motion of Communicating Agents," Cambridge University Press, 2009.

[11] Prasad, K., *A calculus of broadcasting systems*, Science of Computer Programming **25** (1995), pp. 285–327.

[12] Reijsbergen, D. and S. Gilmore, *Formal punctuality analysis of frequent bus services using headway data*, in: *Proceedings of the 11th European Performance Engineering Workshop*, Florence, Italy, 2014, to appear.

[13] Weiser, M., *The computer for the 21st century*, Scientific American **265** (1991), pp. 94–104.

Development of a Smart Grid Simulation Environment [1]

J. Delamare , B. Bitachon, Z. Peng, Y. Wang
B. R. Haverkort, M.R. Jongerden [2]

DACS
University of Twente
Enschede, The Netherlands

Abstract

With the increased integration of renewable energy sources the interaction between energy producers and consumers has become a bi-directional exchange. Therefore, the electrical grid must be adapted into a *smart grid* which effectively regulates this two-way interaction. With the aid of simulation, stakeholders can obtain information on how to properly develop and control the smart grid. In this paper, we present the development of an integrated smart grid simulation model, using the Anylogic simulation environment. Among the elements which are included in the simulation model are houses connected to a renewable energy source, and batteries as storage devices. With the use of the these elements a neighbourhood model can be constructed and simulated under multiple scenarios and configurations. The developed simulation environment provides users better insight into the effects of running different configurations in their houses as well as allow developers to study the inter-exchange of energy between elements in a smart city on multiple levels.

Keywords: smart grid, simulation, renewable energy

1 Introduction

Over the last decades, there has been an increase in the availability and affordability of renewable energy sources in households, such as wind mills and, primarily, solar panels. In addition, the capacity of electricity storage devices has risen considerably. As a consequence, renewable energy production will potentially make up a large portion of the energy in the electrical grid system. The traditional electrical grid needs to be upgraded into a smart grid system, which is able to effectively and

[1] This work was realized as part of the e-balance project that has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement n° [609132].

[2] Email: j.s.r.delamare@student.utwente.nl, b.bitachon@student.utwente.nl, z.peng-1@student.utwente.nl, y.wang-6@student.utwente.nl, b.r.h.m.haverkort@utwente.nl, m.r.jongerden@utwente.nl

efficiently regulate the bi-directional energy flow and the interaction between the various devices on the grid. The electrical energy can flow to, or from the main grid, depending on the time of the day, weather condition and electricity prices. The algorithms which control different functions in a smart grid are complex and need to be studied through simulation before actual implementation to ensure proper operation in all scenarios.

In this paper we present a simulation model which describes the various elements within a smart grid such as user-demand, centralized and decentralized energy production, storage elements, etc. The final overall model simulates a smart grid at neighbourhood level involving several houses with a set of typical smart grid scenarios. Each house can be fitted with solar panels, a battery and its individual demand profile. The model gives insight in how the energy flows within a house, and between the houses in the neighbourhood, and thus can help in smart grid design. Furthermore, due to the modular set-up of the model, the simulation model can be easily extended to incorporate other devices and scenarios.

The Anylogic software [8] is used for the design of the simulation model. The simulation methods supported are system dynamics, agent-based and discrete event. The programming and setup is done using graphical components along with code based on the JAVA language. The software used is flexible and allows for the implementation of the multiple simulation methods within the model. For the inner operation of the model system dynamics is used as it effectively describe the flow of energy through the various components. Each component is then represented as an agent which then allows them to interact and exchange data with each other, this is done using the agent-based approach within the program.

The rest of this paper is structured as follows. Section 2 describes the functionality of the different model components. In Section 3 some simulation results of a number of example scenarios are given. In Section 4 a short overview of related work is given. Finally, we conclude in Section 5.

2 Model components

A smart neighbourhood consists of a number of connected smart houses. Each such house will be built up out of several components which will have configurable parameters to make each house have its own characteristic behaviour. In Figure 1 the overall model of an individual residential unit can is displayed with all the designed components connected which are able to communicate together through the ports found on each. In the following, the functionality of each component will be presented.

2.1 House

The *House* model is the main element which allows interaction between the other component models. All other components are connected to the house via ports which enable communication between them. The way which the produced and imported power is used and stored is determined by the settings of the house. For example, one can set that the battery is only charged by locally generated energy.

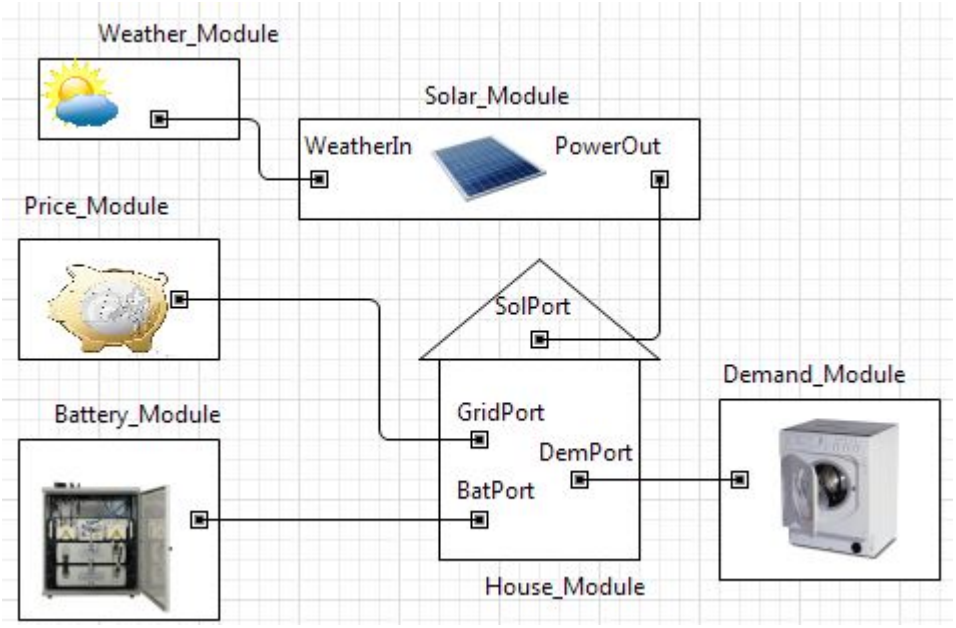


Fig. 1. Residential Unit

2.2 Demand

In the *Demand* module, the electricity demand profile of the house is defined. The profile gives the average power drawn over a given period of time, which may be dependent on the time of day and the day of year. Standardized demand profiles can be found on the internet, for example, EDSN [9] provides standard profiles for the Netherlands. With smart meters one could measure personalized demand profiles, in order to create fully customized simulations.

2.3 Weather

The *Weather* module provides the necessary weather data for computing the generated renewable energy. Since the model, for now, is limited to solar panels only irradiation data is used. However, this can easily be extended to include wind speed data for wind generation. For locations in the Netherlands, the weather data is readily available via the Royal Netherlands Meteorological Institute (KNMI) [12]. It provides Global Horizontal Irradiance (GHI) data, which can be used for the computation of the generated solar energy. The GHI consists of the Direct Normal Irradiance and Diffuse Normal Irradiance [10], and is measured over a horizontal plane, 0 degree tilting.

2.4 Solar Panel

The *Solar Panel* module converts part of the solar irradiance into electrical power [2]. The output power is highly dependent on the position of the panel with respect to the sun. However, to incorporate this fully it would require many input parameters from the user. So, in order to keep the model easily configurable and limit the involvement of the user, the relation between the GHI and the output power (P) is

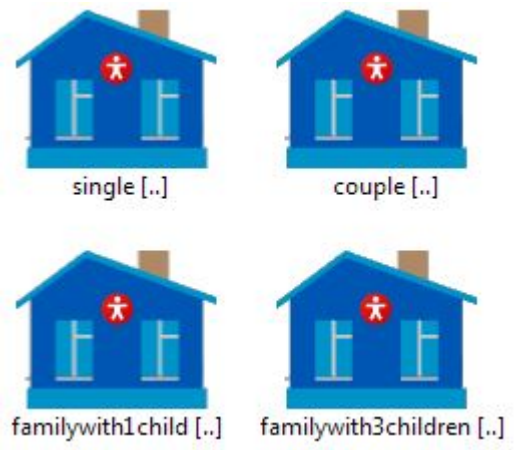


Fig. 2. Example of Neighbourhood model

given by the following approximate relation:

$$P = \epsilon \times A \times GHI,$$

where ϵ is the efficiency of the solar panels and A is the area of the panels.

2.5 Battery

The *Battery* is a component to be modelled as it is likely to be used as a storage element in a smart grid system. The battery model should be user-friendly, which means the battery specification can be adjusted by the users. Many different types of battery models have been developed for various applications over the years [3]. There are two crucial requirements for the battery model. The first is a small number of parameters which makes the model simple and easy to configure. The second is a high accuracy. In most cases, a trade-off between the number of parameters and accuracy are exist [4]. The battery is chosen to be modelled using the Kinetic Battery Model (KiBaM) [5]. Within this simulation the usage algorithm depends on the availability of power from the solar panel. The battery is used as a buffer when the panels are not able to deliver sufficient power.

2.6 Pricing Module

The *Pricing* module allows the user to keep track of the amount of money paid for the energy used as well as the amount credited when selling energy back to the grid. Different suppliers can be compared to show which packages are most convenient for the user's smart grid setup. Also, the financial data can be used to estimate the return of investment of a future setup.

2.7 Neighbourhood

By configuring multiple houses, a neighborhood can be created within the model, as visualized in Figure 2. For each house the different modules can be uniquely defined, to fit to the individual demand profiles and available battery capacity and solar panels. It is also possible to create multiple instances of identical houses to

create a larger population of houses. In this way, one can easily study the effect of an increase of the penetration of solar panels on the interaction of a neighborhood with the grid.

3 Simulation studies

3.1 Set-up

The simulation is run on two levels, the Neighbourhood and residential level, to observe the operation of the designed model. These simulation results will show how the requirements listed are met in the designed smart grid environment. At the neighbourhood level 14 residential units are simulated with the setup as shown in Table 1. The 14 residential units are build-up from 8 differently configured units, with different solar panel size and battery capacity. For five of these configurations multiple instances are used in the neighbourhood. The size of the household determines the demand profile of each house. The weather profile is the same for all houses. All of the following results are obtained by simulating one month in Summer.

Family Group	House A			House B		
	Units	Panel m^2	Battery Ah	Units	Panel m^2	Battery Ah
Single	2	7	60	3	5	50
Couple	1	9	70	1	13	65
Family, 1 Child	2	16	80	1	18	80
Family, 3 Children	2	25	90	2	21	94

Table 1
Smart Grid Residence Setup Scenario

3.2 Neigbourhood level

At the neighbourhood level, cumulative data of all the residential units can be gathered which allows for the study of the interaction between the neighbourhood and the portion of the grid it is connected to. In Figure 3, we display the energy trade figures for the given scenario. It shows how much energy is imported from the grid by the whole neighbourhood, and how much of the generated energy is exported to the grid. In the simulated period, approximately 4000 kWh has been generated by the solar panels. Nearly 2500 kWh has been used in the neighbourhood, either directly or by storing it in the batteries. The rest, approximately 1600 kWh, is delivered to the grid.

In Figure 4 the ratio of the energy supplied by the sources and storage to the demand can be seen. With the implementation of the above configuration nearly 50% of the energy required for the neighbourhood can be supported directly by the solar panels. Another 18% of the demand is supplied in directly by solar energy, through the batteries.

The user can study relations such as the effect of an increase of the number of solar panels on the amount of energy imported from and exported to the grid by the neighbourhood. Thus, one can investigate to what extend the neighbourhood could operate autonomously from the central grid.

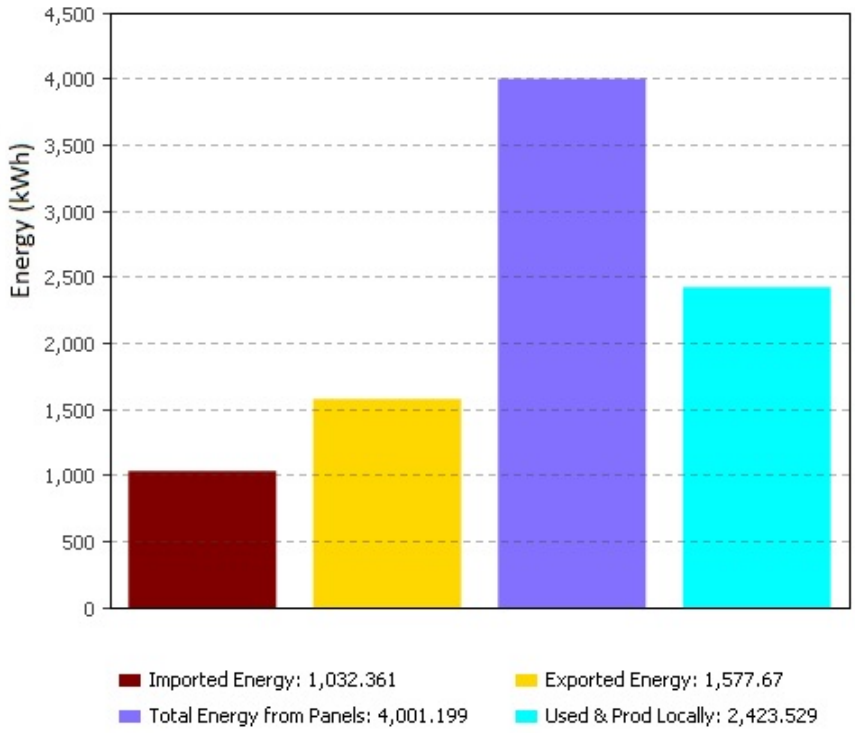


Fig. 3. Distribution of Neighbourhood Energy

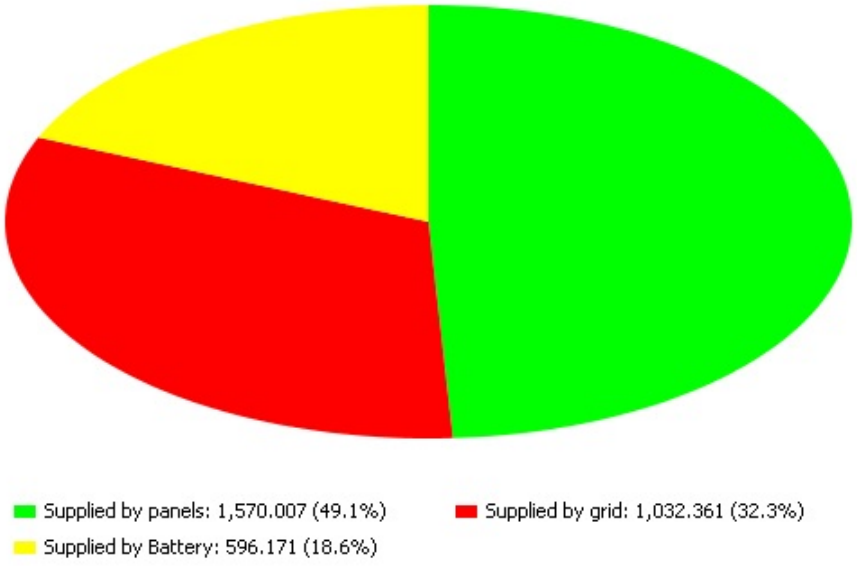


Fig. 4. Energy Source Distribution

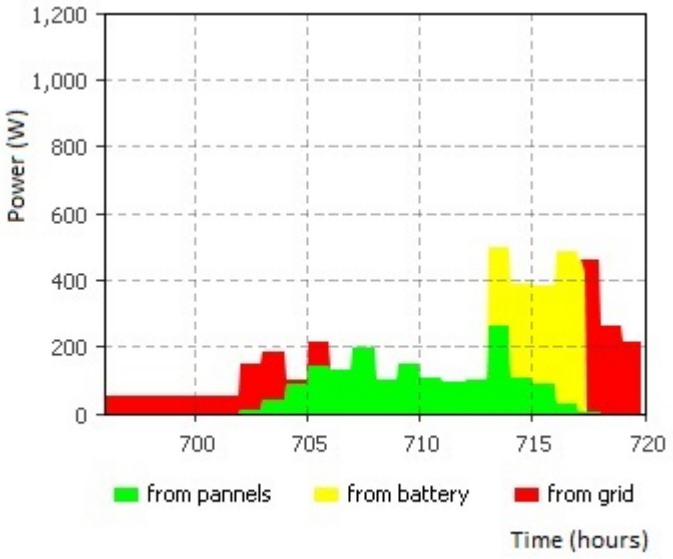


Fig. 5. Power Supplied by Sources and Power Usage

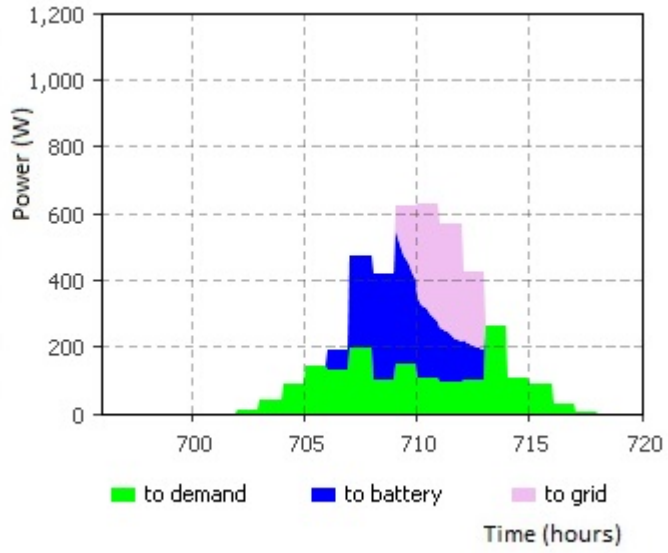


Fig. 6. Power distribution from panel production

3.3 Residential level

Within the same simulation run the effects for the individual houses can be studied as well. The shown results are from a household of a family with 3 children, with house type A. All the shown results can easily be obtained for the other household types as well.

At the residential, level data similar to that on the top level can be viewed, such as the distribution of the sources that are used to meet the demand. In addition to that data can be gathered and displayed for the power usage profile as can be seen in Figure 5. The usage profiles shown is for a period of 24 hours (hours 695 to 720 in

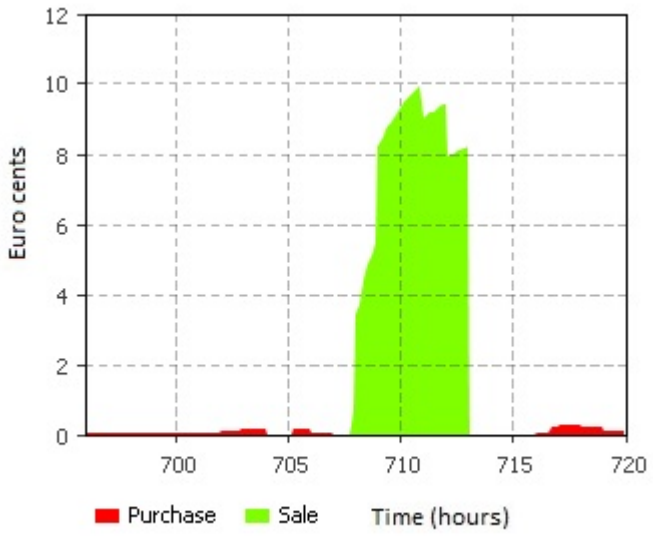


Fig. 7. Hourly Credit and Debit

our simulation) from the different sources. From such graphs the switching between energy sources can be carefully studied, especially when implementing complex control algorithms. Furthermore, in Figure 6 one can see the distribution on the total generated power from the solar panels to the various elements in the system.

Within the Pricing Module the user can keep track of the debit and credit, due to reselling excess back to the utility companies. Figure 7 shows the hourly sales for the chosen configuration, while in Figure 8 the total exchange over a period is shown, in this case one month. With these results users can get estimates about the net cost effects when using particular energy companies. Other algorithms could be implemented which could regulate energy use based on pricing models from the utility companies.

4 Related Work

The importance of smart energy distribution is stressed by organisations such as the Joint Research Centre at the Institute for Energy and Transport [11], which advices the European Commission on energy policy. The organisation promotes and encourages research in various areas of smart grid systems.

In [1] the authors develop a model of a gas station micro grid which is connected to PV panels as well as storage devices. The model is implemented in Anylogic and follows a similar methodology as that used in this paper. The energy flow within the system is determined using system dynamics and the elements of the gas station are built as components which communicate with each other. The gas station includes models for the car traffic which would visit a comparable station. Different configurations could be modelled to find the optimal energy balance for the system.

The Smart Grid Simulator [7] focuses on lowering the energy bill for the customers. It uses data regarding hourly energy consumption, energy prices, as well

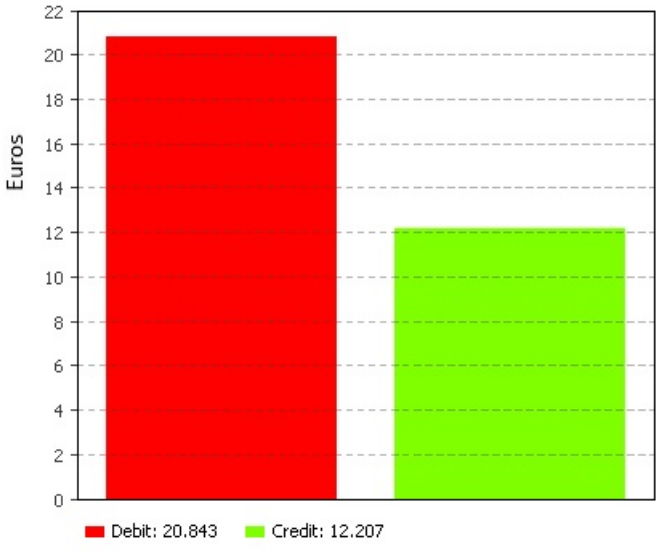


Fig. 8. Total money exchanged in a one month period

self-produced energy circumstances, in order to predict the amount of energy bought for the next day, lower the cost of electricity, optimize the utilization of the electricity, compare assortment of prices, run a brokering algorithm that dictates when energy is being bought, stored, or sold and predicting the renewable energy production.

In [6], an overview is given of the features and the capability of a smart energy city, one of which is lower energy consumption. The advantages of a smart city are demonstrated using Anylogic. The model is based on Demand Response (DR). The author argues that the DR based system will reduce the overall energy consumption of a city because the peak demand level can be lowered. In addition to DR the model also employs load control to decrease the energy consumption even further. There are two levels in the model. The first level is a smart building, modelled as an agent, the second level is the environment for the smart building agent. The first level of the model uses a price signal as the input, the price signal will then affect the decision making process inside the building, for instance to determine whether to use the energy from the battery, or whether some appliances need to be turned off. The main level of the model acts as the utilities company. The model assumes that the occupants of the building signed a contract that allows the utilities company to control a predefined load. This is done by using an algorithm that allows the utilities company to reduce and restore energy consumption level at any given time. The model, which has four buildings, was tested using a sequence of scenarios. The results show that the model does indeed lower the overall energy consumption level of the houses. The results also show that demand response with an addition of load control results in a further reduction of the energy consumption.

5 Conclusion

A smart grid simulation tool has been developed which can model residential units which have solar panels and batteries which are able to store excess energy. Differ-

ent living situations are available for the residential unit to allow for a more diverse data set when scaling the model to neighbourhood level. The model is able to use statistical data from local meteorological sources to represent the weather patterns and influence the solar panel output power. With the combination of the various elements a vast amount of data can be gathered which shows the interactions between the elements. Data can be gathered on both the financial level and the energy level which can be useful for both consumers and researchers.

The program was developed using the Anylogic simulation software which allowed for a flexible environment to implement the desired models. System dynamics and agent-based modelling were the methods primarily used in making the models. The agent-based approach allowed the residential units to be scalable and allow for multiple instances which adds diversity to the neighbourhood simulation.

The current model can be further expanded upon by incorporating other elements such as stationary electric vehicles as additional storage units, as well as other renewable sources. Algorithms can be implemented to study the behaviour in different scenarios, such as purchase/sell of power according to market price. Other algorithms could control the usage of the connected battery.

References

[1] Bazan, P. and R. German, *Hybrid simulation of renewable energy generation and storage grids*, in: *Simulation Conference (WSC), Proceedings of the 2012 Winter*, 2012, pp. 1–12.

[2] Duffie, J. A. and W. A. Beckman, "Solar engineering of thermal processes," John Wiley & Sons, 2006 p. 747.

[3] Jongerden, M. and B. Haverkort, *Which battery model to use?*, Software, IET **3** (2009), pp. 445–457.

[4] Jongerden, M. R., "Model-based energy analysis of battery powered systems," Ph.D. thesis, University of Twente (2010).

[5] Manwell, J. F. and J. G. McGowan, *Lead acid battery storage model for hybrid energy systems*, Solar Energy **50** (1993), pp. 399 – 405.

[6] Morvaj, B., L. Lugaric and S. Krajcar, *Demonstrating smart buildings and smart grid features in a smart energy city*, in: *Energetics (IYCE), Proceedings of the 2011 3rd International Youth Conference on*, IEEE, 2011, pp. 1–8.

[7] Ursachi, A. and D. Bordeasu, *Smart grid simulator*, International Journal of Civil, Architectural, Structural and Construction Engineering **8** (2014), pp. 519–522.

[8] *Anylogic*, 5 September 2014.
URL http://www.anylogic.com/

[9] *EDSN demand profiles*, 5 September 2014.
URL http://www.edsn.nl/verbruiksprofielen/

[10] *Global horizontal irradiance*, 5 September 2014.
URL http://pvpmc.org/modeling-steps/irradiance-and-weather-2/irradiance-and-insolation/global-horizontal-irradiance/

[11] *Joint research centre at the institute for energy and transport*, 5 September 2014.
URL http://iet.jrc.ec.europa.eu/

[12] *Uurgegevens van het weer in nederland*, 5 September 2014.
URL http://www.knmi.nl/klimatologie/uurgegevens/

Validation of automatic vehicle location data in public transport systems

Stephen Gilmore and Daniël Reijsbergen

Laboratory for Foundations of Computer Science
University of Edinburgh
Edinburgh, Scotland

Abstract

Performance metrics for public transport systems can be calculated from automatic vehicle location (AVL) data but data collection subsystems can introduce errors into the data which would invalidate these calculations, giving rise to misleading conclusions. In this paper we present a range of methods for visualising and validating AVL data before performance metrics are computed. We illustrate our presentation with the specific example of the Lothian Buses Airlink bus, a frequent service connecting Edinburgh city centre and Edinburgh airport. Performance metrics for frequent services are based on *headways*, the separation in space and time between subsequent buses serving a route. This paper provides a practical experience report of working with genuine vehicle location data and illustrates where care and attention is needed in cleaning data before results are computed from the data which could incorrectly reflect the true level of service provided.

Keywords: Public transport measurement and modelling, data cleaning, headway computation

1 Introduction

Modern engineered systems are reflexive. Through instrumentation and sensors, they collect data on their function and performance which is used to assess their progress and safe operation. Transport systems work in this way: a modern bus fleet has richly-instrumented vehicles which report their latitude and longitude, speed and heading. This data is streamed back over a data connection to an automatic vehicle location tracking system which feeds other systems such as real-time arrival prediction for bus passengers.

Judging by recent advances in the field of adaptive systems, it would seem that the future offers us a vision of self-organising, self-healing systems regulated and kept in check by their data-collection subsystems. Unfortunately, these data-collection subsystems are themselves often complex systems, with their own faults and problems, and intrinsic limitations to their engineering. It is not until one starts working with such subsystems that some of these problems begin to become evident. These problems increase in significance when regulators begin to calculate performance metrics for public transport services from historical Automatic Vehicle Location (AVL) data traces.

Determining service performance has until recently been done by human observers in place by the side of the road recording vehicle departures and applying intelligence and experience to interpret and record events. This approach has the benefit of ensuring that data is scrutinised before performance measures are calculated. In contrast, in the context where human intelligence is not applied (as in automated processing of historical AVL data traces), errors of interpretation can occur, and it is these errors which are our concern here.

In this paper, we present an experience report on the use of AVL data for obtaining headway and frequency measurements. The AVL data is provided to us by the Lothian Buses company, based in Scotland and operating an extensive bus network in Edinburgh. We consider the specific example here of the Airlink bus service, connecting Edinburgh city centre and Edinburgh airport. An undesirable feature of a frequent service is *clumping*, where two or more buses remain close to each other for an extended period. For this reason *headway*, the separation between successive buses, is an important metric for regulations and service operators.

In particular, we discuss the computation of headway measurements to evaluate the performance of bus routes in terms of specific measures of punctuality. We use a range of methods to visually represent both the data and the computed headways, including a visualisation tool that uses the Google Maps API and which was developed at the University of Edinburgh [1].

The AVL data which is made available to us records the position of each bus in the fleet in terms of Ordnance Survey of Great Britain eastings and northings measurements, which can be easily converted to more familiar latitude and longitude coordinates. The AVL data is specific to a particular bus, as determined by a unique bus identifier called a *fleet number*. The assignment of buses to routes is captured in a schedule which is drawn up before the bus service begins for the day, but may change without notice during the day in response to operational problems. This uncertainty about which buses are in service and which are not gives rise to part of the problems of interpreting the AVL data before metrics are computed.

The remainder of this paper is structured as follows. We first discuss the visualisation of AVL data in Section 2, before moving on to the isolation and removal of data errors in Section 3. We discuss the visualisation of headway data in Section 4 and the use of headway measurements in service level agreements in Section 5. We discuss related work that uses the same data or tools in Section 6, and conclude the paper in Section 7.

2 The value of data visualisation

One vitally important sub-task in undertaking a modelling exercise which is strongly rooted in data is to make all efforts possible to understand the data, its scope, and its limitations. In our work with the Lothian buses data we have created a visualisation tool to allow us to literally view the data in geographical context, against a map of the city of Edinburgh. This visualisation tool, shown in Figures 1, 2 and 3 allows us to revisit historical trace data on bus movement and to play or single-step through the data, visualising only those bus services which are of interest.

This tool has no predictive power, it can only render measurement data. Neither

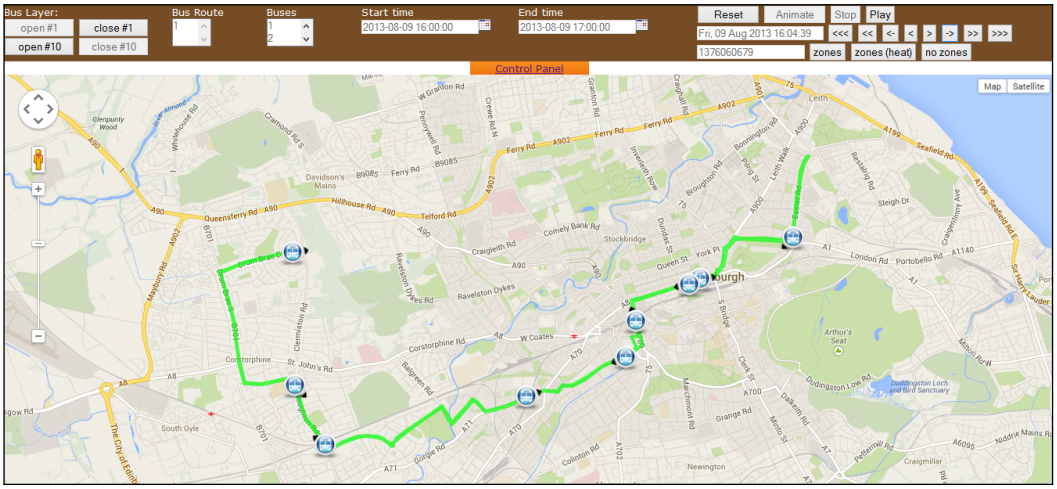


Fig. 1. The user interface allows the user to select bus routes of interest and dates and times of interest and step through the data to see events which occurred in the selected part of the city of Edinburgh.

has it any logical, inferential, deductive or verification capacity. Nonetheless, it was very valuable in allowing us to find some significant errors in the data, which we then set about removing in a systematic process of data cleaning.

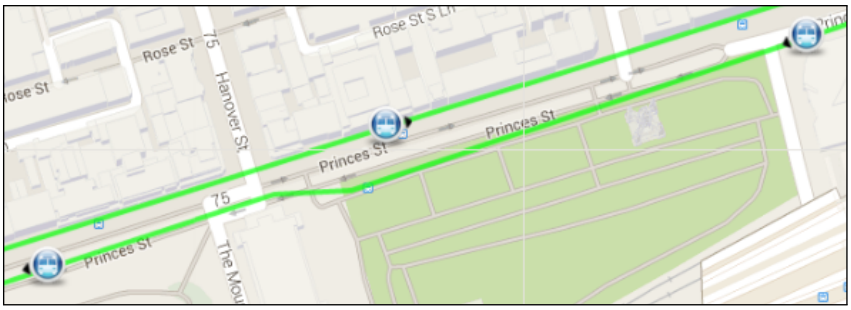


Fig. 2. The data can be accurate enough to confirm the direction of the bus by inspecting visually the side of the road on which it is driving.

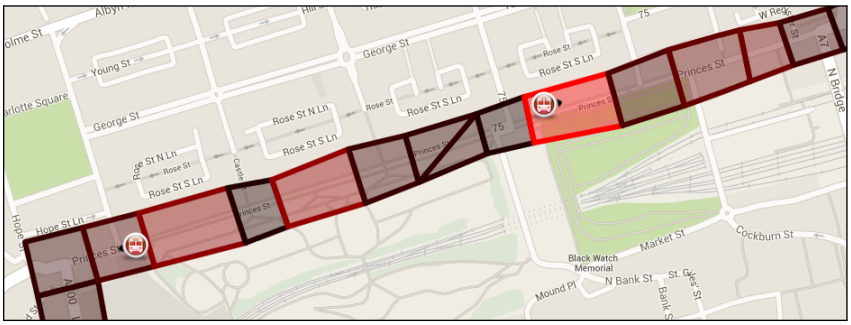


Fig. 3. A heat map representation of Edinburgh city centre showing the patches along Princes Street where buses have the longest sojourn time.

In working with data on bus movement from Lothian Buses we are fortunate to have useful domain knowledge about what buses can and cannot do. For example, we know that buses cannot teleport, so when we see that some Edinburgh buses appear to visit Wales (as in Figure 4) we know that this is only a phantom GPS result from the data-collection subsystem which we can discard. Similarly, Edinburgh

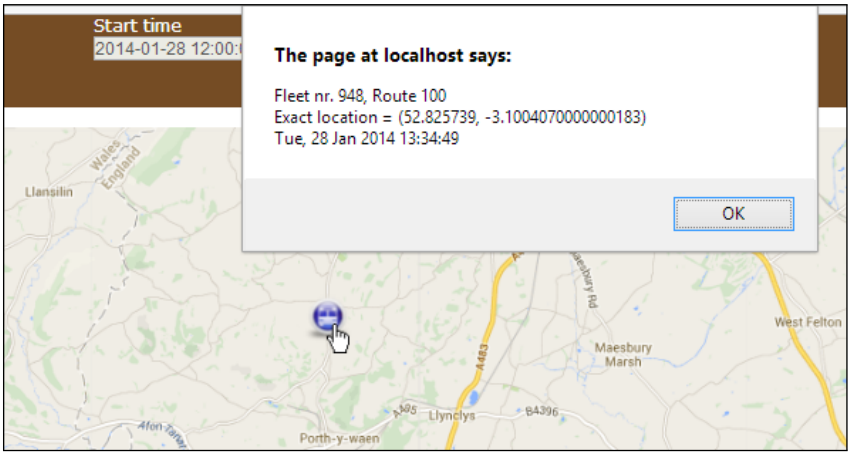


Fig. 4. The visualisation tool can be used to identify buses that appear in a peculiar place, such as a field near the Anglo-Welsh border.

buses are not amphibious, so measurement data which has them swimming about in the Firth of Forth is also to be discarded. Finally, Edinburgh buses cannot fly, so when we see data which when rendered on the map seems to show them flying over the rooftops like Ron Weasley's magical car, we know not to believe this. Our visualisation helps us to make sense of this kind of erroneous data by showing that it is a straight line between the final stop on a vehicle's last journey of the night and the bus depot where they are housed overnight.

These erroneous position reports come from vehicles which are not in service, or from measurement sensors which have not been powered down as completely as they should have been, or they are artefacts caused by interpolation in the system trying to fill in data points to compensate for the gap in the data caused by the location-tracking subsystem being switched off at the end of the day's use for a vehicle. However, the data does not record which buses are in service and which are not, so if using the data for purposes other than those for which it was being collected – as we are here – then we need to interpret with care and attention and clean the data to remove erroneous measurements such as these before calculating any measures of interest.

2.1 Visualisation of single bus trajectories

The validation of the service provision which we will conduct depends fully on the data, its quality and completeness, and our interpretation of the data. In order to provide ourselves with as thorough an understanding of the data as possible, we developed different views on the data, each of which had value in establishing some understanding of the data and bringing us insights which we found useful.

Our first visualisation, shown in Figure 5, rendered the data in a conventional map view. This was useful in helping us to see that this bus was providing the Airlink service on the day of interest, but it did not use the time content of the measurement trace.

Our second visualisation, shown in Figure 6, represented time in the abstract sense of *subsequency* in that we used different colours to represent phases of the journey which happen successively. From this visualisation we can see that the bus

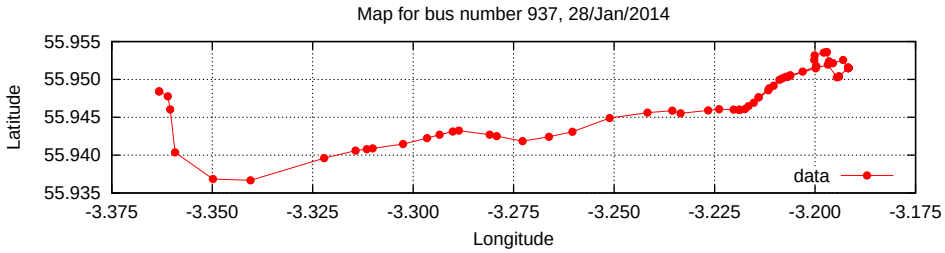


Fig. 5. Sample AVL data from one bus from the airport to the city centre, rendered in the conventional map view which plots latitude against longitude. This view abstracts from both the timing of events and their relative ordering in time.

is travelling from the airport in the west to the city centre in the east, but not at what time of the day (or night) this journey occurred.

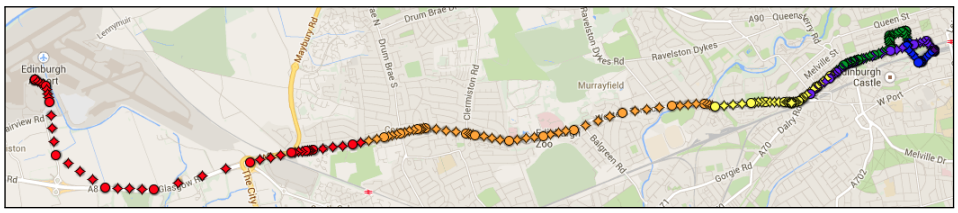


Fig. 6. A visualisation of the airport bus route showing buses on the route from Edinburgh airport in the west to Edinburgh city centre in the east. This visualisation represents sample AVL data from one bus from the airport to the city centre. The colouring imposed on points allows us to determine the direction of the journey.

To allow us to see the journey more clearly it is sometimes helpful to fill in the route in a little more detail by interpolating between the data points. Figure 7 presents such an interpolation. Depending on the use to which this interpolation is put, the measurement errors which are introduced by "cutting corners" as we see in Figure 7 might or might not be problematic.

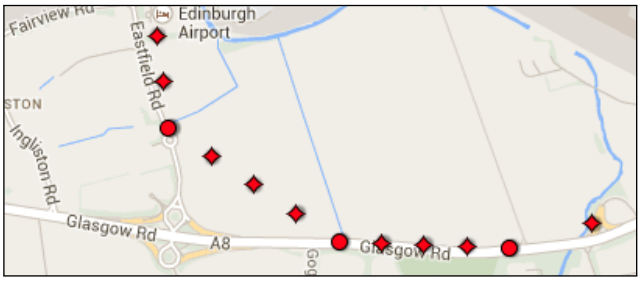


Fig. 7. Some simple interpolation is computed, but naively. The placement of these interpolation points suggests that the bus made the transition from Eastfield Road to Glasgow Road by driving across a field instead of navigating the roundabouts and joining via the slip road, as it of course did.

Finally, Figure 8 shows the AVL data as a time series. In this view, latitude and longitude are plotted separately against time. This has the advantage of allowing us to see when the bus journey happened and to identify positions where it is stationary for long periods of time, which was much more difficult to see in the map view. Against this, our intuitions about where in the city the bus is at any point in time are lost, because we have moved away from the map view.

Different views on the data have given us different insights but the identification of collective behaviour remains elusive. In a scenario where events depend not on

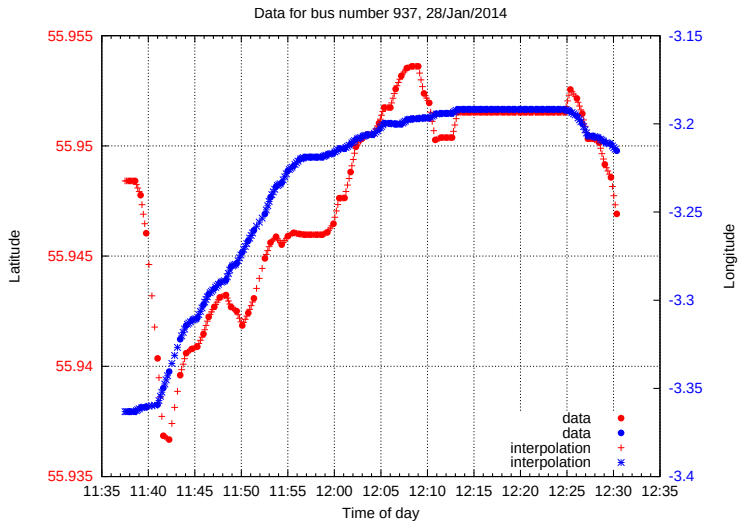


Fig. 8. Sample AVL data from one bus from the airport to the city centre, rendered as a time series. The bus is stationary when neither latitude nor longitude are changing as in this graph between 12:13 and 12:25. The stationary point at 55°57′05.5″N, 3°11′30.3″W is the Airlink bus stop on Waverley Bridge.

the behaviour of individuals, but on the behaviour over the long run, or a collection of observations, then representing a single individual trace is of little interest. More profound insights come from aggregating individual behaviours to look for trends and patterns.

2.2 Visualisation for collective systems

If we wished to learn the topology of the Airlink bus route in order to identify how and where it turns in order to execute the return journey then this collective view is much more helpful than the individual views which we have seen previously. If we simply plot all observations of a bus location which we are given, as we do in Figure 9, then this maps out the route without interpolation or approximation (up to the resolution of the GPS data available).

Deviations from the planned route can be seen in this view. In this view it is possible to determine that some roads are travelled relatively infrequently (in this case, once per day, the journey from the bus depot to the start of the route, and once per day the journey from the end of the route to the bus depot at night). Occasional diversions from the planned route would also show up in this view, provided that the deviation from the route is long enough that the position of the bus is recorded during the deviation.

Another long-run collective view of the data would be a *heat map*, allowing us to identify where in the city buses spend most of their time (as detected by noting more observations in this area than in others). To achieve this, we place a regular grid over the map of the city with a counter for each square in the grid. We increment the counter every time we see a GPS measurement placing a bus in this square, up to a ceiling of 100 observations per square. Mapping these numbers to a colour spectrum, we see that more-frequently-occupied squares will show up as being hotter than the less-frequently-occupied squares. This might confirm (or refute) our expectations about where delays occur along the route. Figure 10 shows

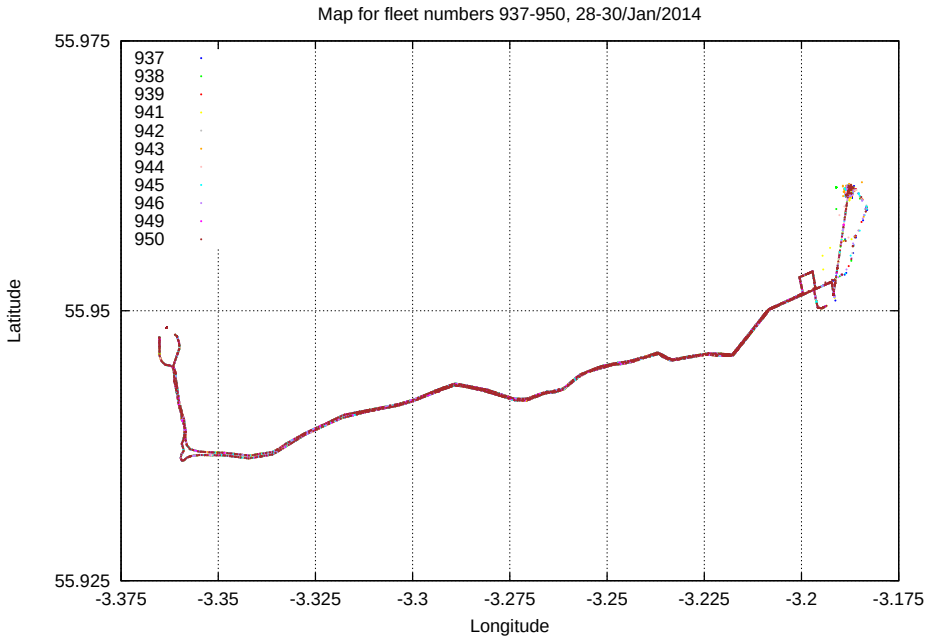


Fig. 9. Latitude and longitude data from eleven buses for two days

such a view for our data.

From this we can see that the least-frequently travelled part of the route is the journey to the depot in Annandale Street (in the top right-hand corner) because there are very few observations of buses in this region: only one or two observations for each bus over a two day period. We can also infer that the faster part of the route is on the Glasgow Road leading to the airport (in the bottom left-hand corner). There are more observations here than for the depot, and there is no possible branching off the route here so fewer observations in this region much come from the buses travelling faster here.

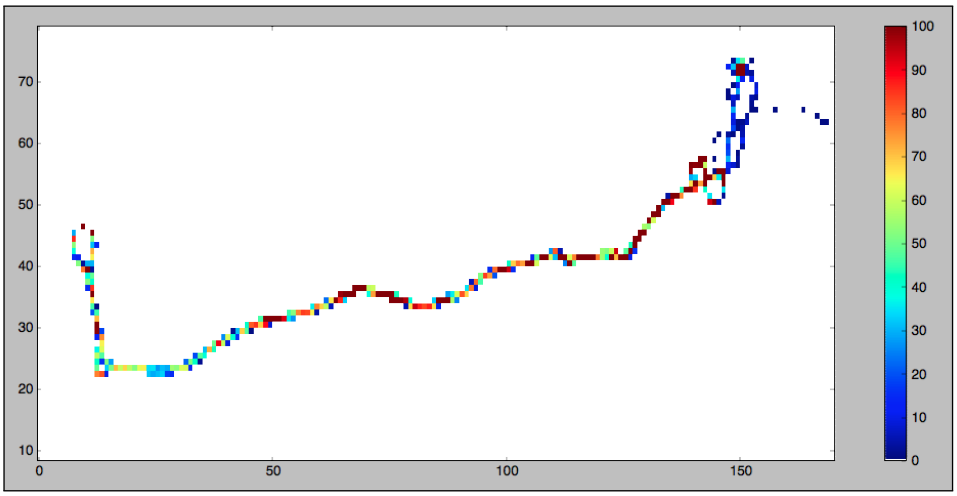


Fig. 10. Heatmap data from eleven buses for two days

3 Isolating errors in data

GPS data can contain both errors of omission and errors of inclusion. Figure 11 demonstrates both of these. The GPS data misses observations on Market Street because this street falls under the GPS shadow of tall buildings on the Mound, a steep hill climbing upwards from Princes Street. This is seen at the bottom of Figure 11 near the centre where there are no data points on Market Street until the roundabout with Cockburn Street and Waverley Bridge.

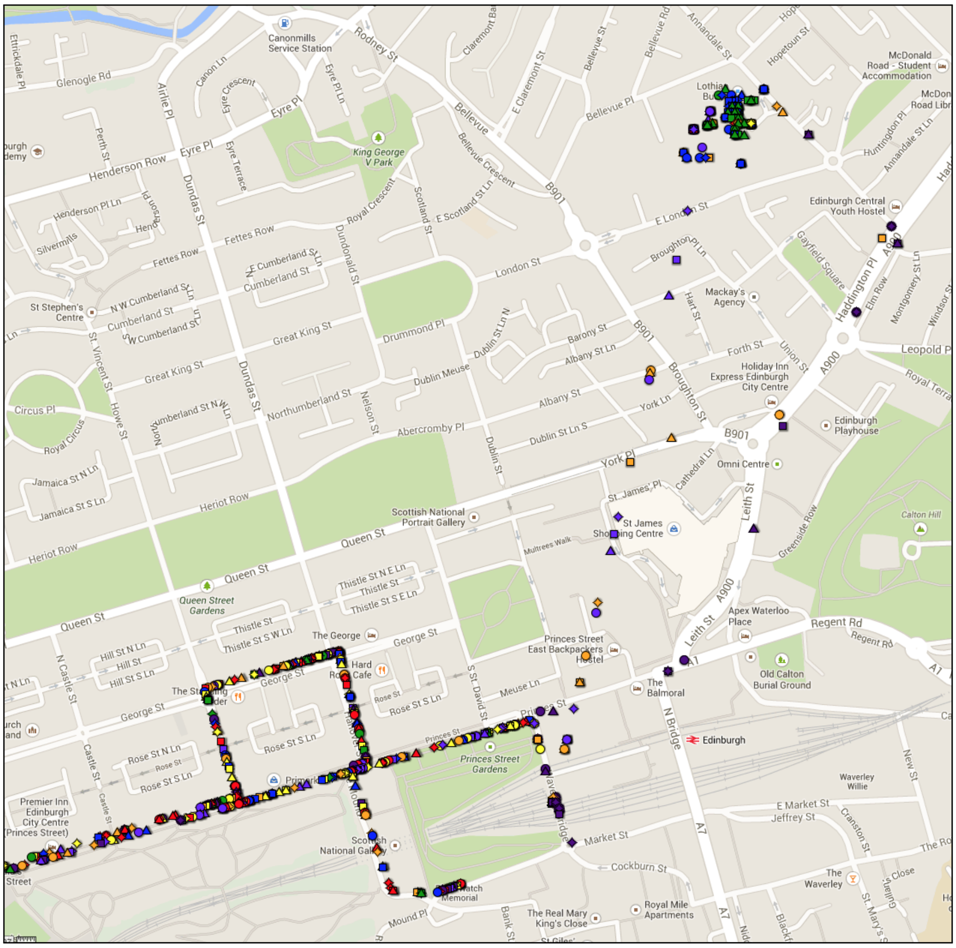


Fig. 11. GPS data for the Airlink bus showing Princes Street, Waverley Station and the Lothian Buses garage in Annandale Street.

Figure 11 also contains spurious data points which appear to have been generated by interpolation of observations between Waverley Bridge and the Lothian Buses garage on Annandale Street (note that these are reported data points given to us by the bus company, not like the interpolated data points which we introduced in Figure 7). These manifest themselves as a straight line on the map with interpolated points cutting across York Place and East London Street with no apparent regard for road layout.

The timestamps associated with these data points are either all from the early morning (04:30) when the service starts or last thing at night (12:00) when the

service ends. Because of this we believe that these data points are an artefact of cold starts or powering down of the GPS tracking hardware.

Once identified and isolated, erroneous GPS data can be conveniently removed using a GPS track editor such as GPSprune [2], as shown in Figure 12. The application shows derived statistics on the GPS track as well as showing the track in context in a map view, and relating positions on the route to their height.

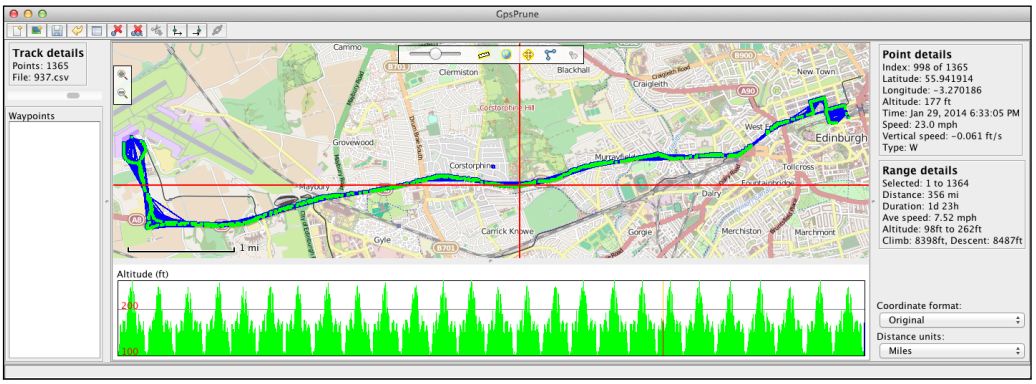


Fig. 12. GPS data being edited in the GPSprune application.

Using this tool we can conveniently eliminate the erroneous early-morning and late-night interpolated data points. This is a manual editing process, but it is made much more convenient because we can see the data points conveniently in context in a standard map view. We can define a region geometrically and then eliminate all of the points which fall within that region. Once this process is complete we are left with a clean data set where all of the erroneous data which we could identify has been eliminated, allowing us to progress on to considering the performance measure of interest (headway). As before, we use visualisation to help us to gain greater insights into the data.

4 Visualising headway

To allow us to approach the computation of headway-based metrics we can begin to look at the collection of eleven buses serving the Airlink bus route. Focusing in on a period of one hour between 11:30 and 12:30, and considering only longitude data as a proxy for progress along the route (because the journey from the city centre to the airport is mostly roughly east-to-west) we can obtain from Figure 13 a sense of headway as separation in time between successive buses.

Looking at the same eleven buses serving the Airlink route over a different granularity of two days, a different pattern emerges. We begin to obtain a sense from Figure 14 of days of service for this collection of buses in the fleet punctuated by overnight absences from service when the buses are stored in the garage.

Note that it is not obvious from published timetable information that the bus operation should follow a day-night pattern. The Airlink bus service runs 24 hours a day and in principle any of the buses from the fleet could be used at any time of day.

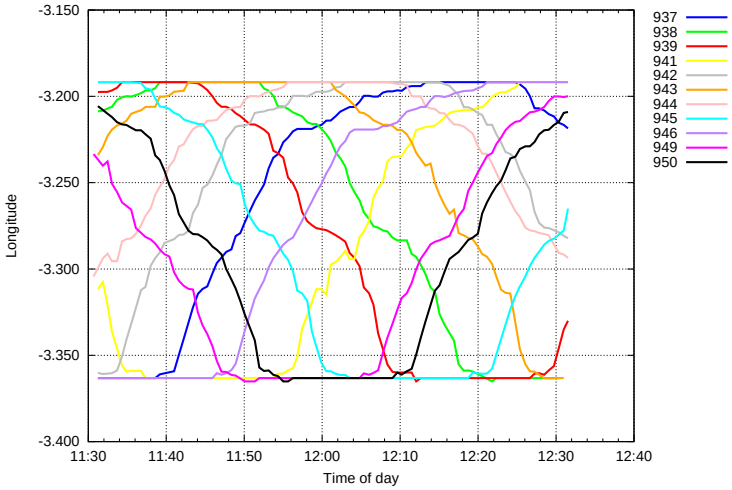


Fig. 13. Average longitude data from eleven buses for one hour

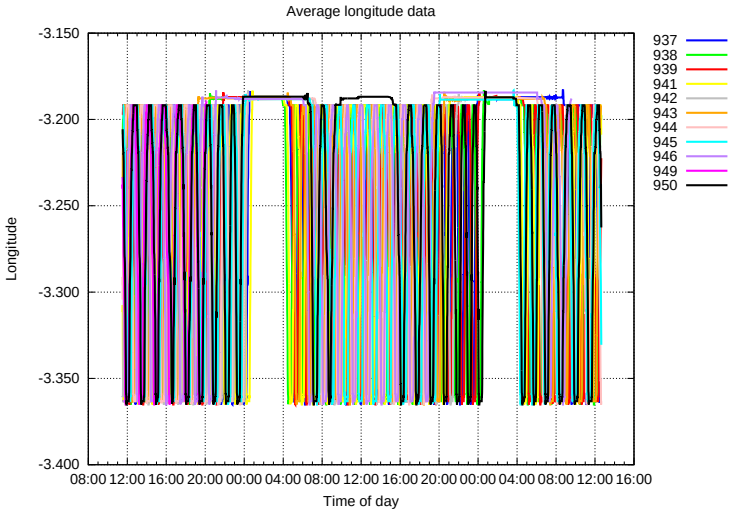


Fig. 14. Average longitude data from eleven buses for two days

Understanding whether or not a bus is in service is another important aspect of data cleaning when computing headways. Being widely-separated from a bus which is not in service is much less important than being widely-separated from a bus which is in service.

It is only when we appreciate the day-night pattern that we can notice buses which are not following the pattern. Isolating the data for bus number 950 in the fleet in Figure 15 we can see that it does not follow the established pattern because its second day of service is punctuated by an absence (from approximately 09:30 to 16:00) where it was not serving the Airlink bus route. (We can discover separately that the bus was taken to the garage for an unknown reason such as some kind of mechanical repair to the vehicle, or perhaps a routine service.)

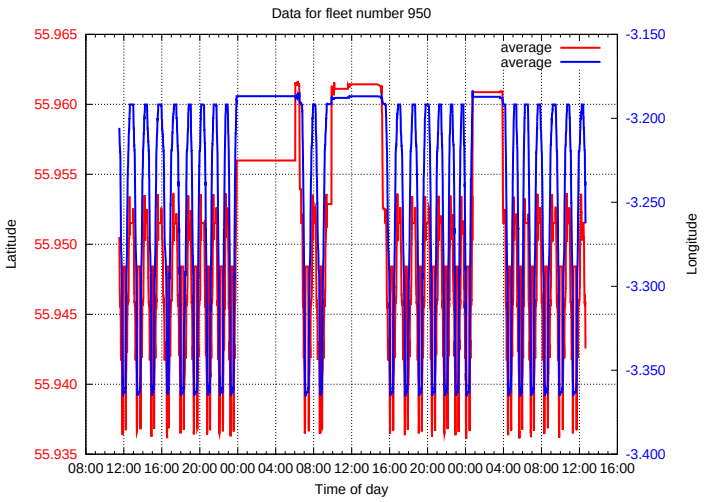


Fig. 15. Average latitude and longitude data from bus number 950 in the fleet over two days. During its absence from service in the middle of the second day this bus is in the Lothian Buses garage on Annandale Street.

4.1 Spatial separation of service instances

As an alternative to considering headway as separation in time, we could consider headway as separation on route, or at least separation in GPS position. We used the Haversine function to calculate the spatial distance of one bus from another, as determined by their latitude and longitude, giving their great-circle distances, as commonly used in navigation and spherical trigonometry.

In Figure 16 we present the Haversine distances in miles from fleet number 937, as a function of time across three days of observation data. This has the benefit of providing a succinct summary of relative bus movement. We can see for each of the three days of data presented that bus number 937 is not close to one of the other buses for an extended period. In each day it moves close to and away from other buses as they make their journeys to and from the airport. The bus which remains closest to fleet number 937 is fleet number 945 which is rarely more than three miles away from fleet number 937 on our second day of observation. Nonetheless, even in this case we can see that the buses are not 'clumping' as the relative distance between the two buses oscillates between zero and three miles throughout the day.

The overall result which we would hope to see for a well-running service is spatio-temporal separation where buses are not often close to each other for an extended period. Such a metric has some merits. It allows for reasonable adaptation to problems in service delivery (so that, for example, buses can on some occasions be close to others for an extended period, as can happen). However, it is not one of the metrics which has been defined by the regulators in this instance.

5 Headway-based service-level agreements

We now consider the service-level agreements which have been identified by the regulators for the service, as published by the Scottish Government. Firstly, there are two classes of bus service identified by regulators: *frequent* and *non-frequent*. Frequent buses depart at least every 10 minutes. Our concern in this paper is only

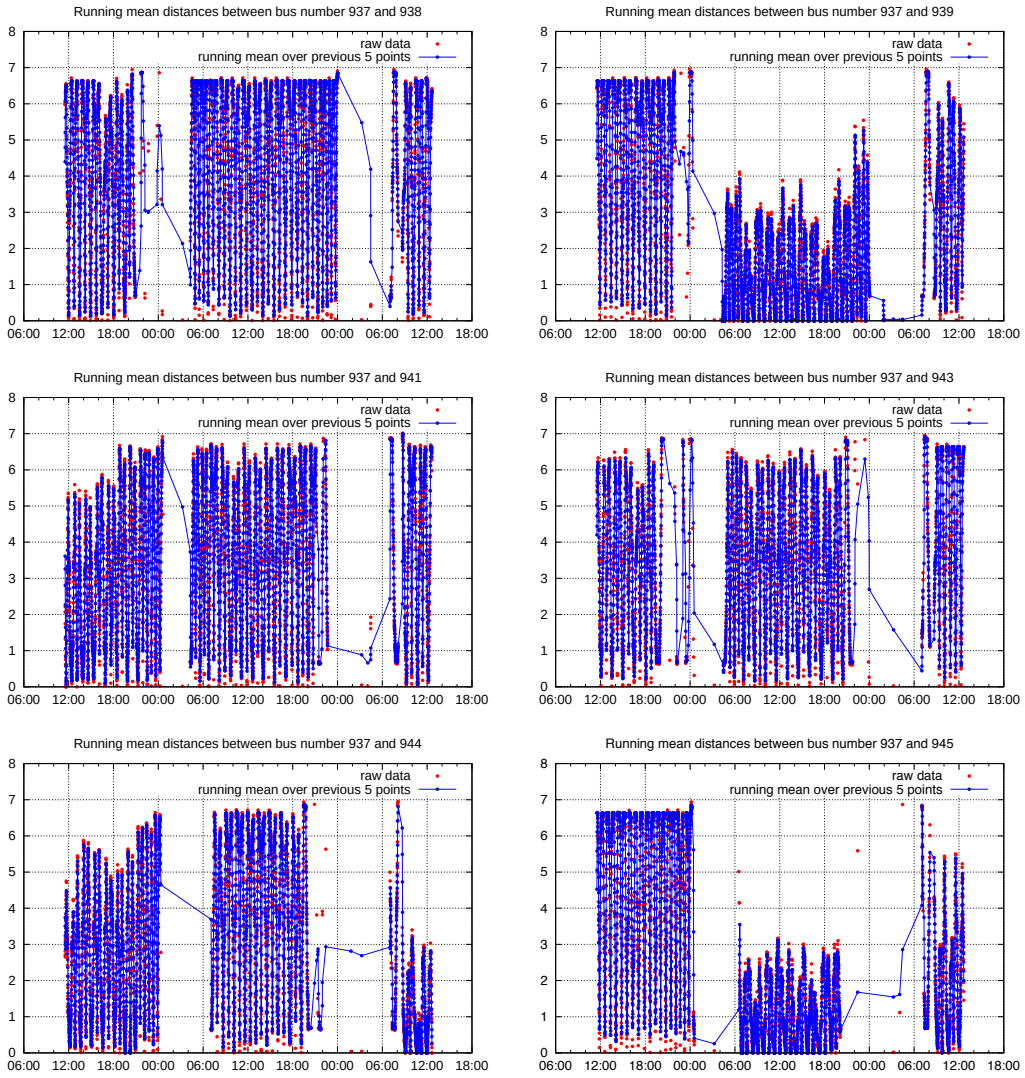


Fig. 16. Time series running mean distances in miles between fleet number 937 and fleet numbers 938, 939, 941, 943, 944, and 945.

with frequent services. The Airlink bus service is a frequent service departing at least every ten minutes between 04:00 and midnight.

A key characteristic of frequent services is that regulators are not primarily interested in timetable adherence, but rather in the amounts of time between bus arrivals — the *headways*. We focus on two of the three punctuality metrics for frequent services identified in the guidance document on Bus Punctuality Improvement Partnerships by the Scottish Government [3]; all three are related to headways.

(i) *Six or more buses* will depart from the starting point within any period of 60 minutes on 95% of occasions.

(ii) The interval between consecutive buses departing from the starting point will *not exceed 15 minutes* on 95% of occasions.

The first of these is a requirement on the *frequency* of departures, the second specifies the maximum allowable *headway* between buses.

These service-level agreements are themselves statements about *collective* behaviour. They do not specify that particular individual instances of the service must be correct, but that, viewed as a collection of observations, a large percentage of this collection (in this case, 95% of it) must be satisfactory according to the regulations.

Punctuality is important for regulators but it is of great value to passengers too. The importance of punctuality is such that it has been observed that the negative impact on passenger satisfaction of a decrease in punctuality can outweigh the positive effects of increasing the number of departures per day [4].

5.1 Determining satisfaction of service-level agreements

Through visualisation we have been able to explore various aspects of the available data and investigate problems with the data which need to be resolved but the most important collective system metrics of frequency and headway have not yet been fully explored.

A modelling tool such as Traviando [5] allows us to process trace data and to compute measures of interest over the trace. The primary purpose of Traviando is to act as a post-mortem simulation trace debugger, diagnosing problems with simulation models through statistical, structural, invariant-based and model-checking analysis of output traces. However, because Traviando works with timed trace output, it is possible to invoke it on measurement data such as our time series of GPS observations of bus positions, even before a simulation model is constructed. Figure 17 shows headway observations which have been obtained in this way.

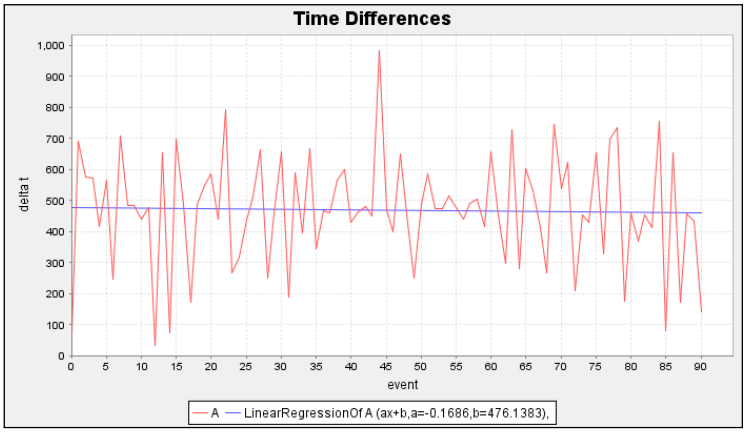


Fig. 17. Headway observations plotted using Traviando as time differences

The linear regression across this time series is centred on 476.13 secs, which is approximately 8 minutes, and comfortably less than the 15 minute interval between consecutive buses which is required by the regulator. Furthermore, we observe in Figure 17 that headways of over 15 minutes (900 seconds) are rarely observed, in this case only once in 90 observations.

We define a finite-state process to convert the departure data into a form where we can compute the *frequency* requirement that at least six buses should depart every hour. The process represents a forgetful observer, who counts departures, but forgets departures which happened more than one hour ago, as in Figure 18.

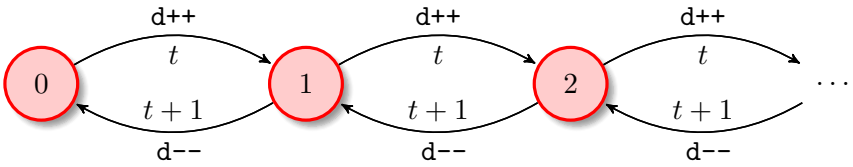


Fig. 18. Observers counting departures should note the times of departures and forget departures which are more than one hour old.

That is, the observer notes the occurrence of each departure of a bus from the start of the journey and records the time that this event occurred. An hour after any observed departure the departure event is discounted as being outside the relevant window as defined by the Traffic Commissioner regulations. This – not altogether straightforward – process is a reactive system which changes state in response to two types of events: bus departures and clock expiration.

This process allows us to track the frequency metric relating to bus departures, as seen in Figure 19 plotted as a counter value-event trace using Traviando. During the day the observations lie between 6 (the minimum allowable value) and 10. The low period on the second day corresponds to the time when bus number 950 in the fleet was in the garage. Time is abstracted away in this view although the relative ordering of events is maintained. This has determined that regulation (i) above has been satisfied across this observation period.

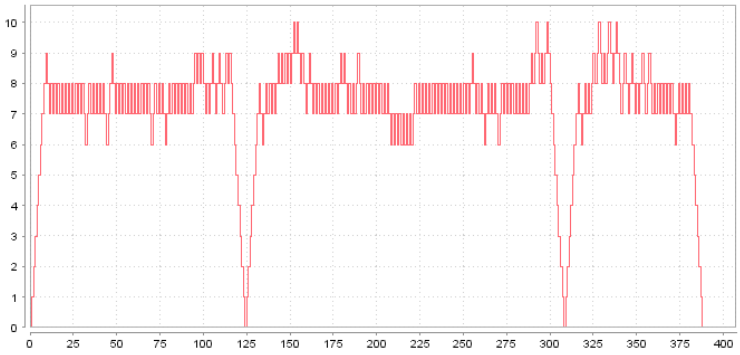


Fig. 19. The frequency metric of buses in the past hour.

6 Related work

The presented data analysis and visualisation methods have been used in several recent papers. In [6], the model checking tool Traviando was used to perform correctness checks on bus journey time data obtained by scraping the Edinburgh Bus Tracker website. In [7], the AVL dataset of this paper was used to obtain bus

sojourn time distributions of land patches in the Edinburgh city centre, which were then used to carry out 'what-if' analysis involving the introduction of trams. In [8], several statistical analysis techniques were used to evaluate the performance of several frequent services in Edinburgh (including the Airlink) in terms of the service level agreements discussed in Section 5, using the same AVL dataset.

7 Conclusions

In contrast to the results from a high-level model, measurement data has enormous authority. It is full of detail and quirks and seems to represent physical truth but as we have seen in examples above, it is not the whole truth, and it is not nothing but the truth either. In our experience so far in working with data on the QUANTICOL project we have always needed to use human intelligence to clean the data before any automated processing could begin. An outlier only becomes an outlier when an interpretation is placed on the other data points.

What we saw in this example was that our understanding was enhanced by processing the data in a range of ways before any reflection and consequential adaptation took place. Further, there were complex collective percentile-based performance metrics to satisfy which required some ingenuity for us even to compute.

In some respects, our smart transport case study is relatively easy to work with. Data is readily available, and latitude and longitude data is relatively easy to interpret and visualise, allowing us to see problems in the data and apply data cleaning. We have intuitions about buses and transport, and local knowledge of what happens in practice. Further, we have access to the personnel in the Lothian Buses company who operate the system in practice. We can ask them what are the problems which are of concern to them. We have the potential to have some influence on the practice of the company, even if only a slight influence. Based on our calculations and more detailed reasoning [8], our belief at this point is that Lothian Buses are meeting the Traffic Commissioner's regulatory instruments.

In the future, we hope to use heat maps similar to the one in Figure 10 to automatically learn the bus routes, using a variant of the algorithm described in [9]. Using a representation of the routes in the form of a graph, we would be able to detect and remove outliers by removing measurements that are too far away from the edges on their route. This would allow us to automate the data filtering process, which at the moment is largely done manually.

Acknowledgements

This work is supported by the EU project *QUANTICOL: A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours*, 600708. The authors thank Bill Johnston of Lothian Buses and Stuart Lowrie of the City of Edinburgh council for providing access to the data which was used for the case study. We would also like to thank Allan Clark for helpful comments on a draft version of this paper.

References

[1] Shao Yuan. Simulating Edinburgh buses. Master's thesis, The University of Edinburgh, 2013.

[2] Tim Pattinson. *Pruning GPS data with GPSprune*. Lulu, 2012.

[3] Smarter Scotland: Scottish Government. Bus Punctuality Improvement Partnerships (BPIP), March 2009.

[4] Margareta Friman. Implementing quality improvements in public transport. *Journal of Public Transportation*, 7(4), 2004.

[5] Peter Kemper and Carsten Tepper. Automated trace analysis of discrete-event system models. *IEEE Trans. Software Eng.*, 35(2):195–208, 2009.

[6] Ludovica Luisa Vissat, Allan Clark, and Stephen Gilmore. Finding optimal timetables for Edinburgh bus routes. In *Proceedings of the Seventh International Workshop on Practical Applications of Stochastic Modelling (PASM'14)*, 2014.

[7] Daniël Reijsbergen, Stephen Gilmore, and Jane Hillston. Patch-based modelling of city-centre bus movement with phase-type distributions. In *Proceedings of the Seventh International Workshop on Practical Applications of Stochastic Modelling (PASM'14)*, 2014.

[8] Daniël Reijsbergen and Stephen Gilmore. Formal punctuality analysis of frequent bus services using headway data. In *Proceedings of the 11th European Performance Engineering Workshop*, Florence, Italy, September 2014. To appear.

[9] Jonathan J. Davics, Alastair R. Beresford, and Andy Hopper. Scalable, distributed, real-time map generation. *Pervasive Computing, IEEE*, 5(4):47–54, 2006.

Dynamic Subtask Dispersion Reduction in Heterogeneous Parallel Queueing Systems

Tommi Pesu and William J. Knottenbelt[1,2]

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

Abstract

The Fork-Join and Split Merge queueing systems are mathematical abstractions of parallel task processing systems where entering tasks are split into N subtasks and each of the N subtasks has its own service time distribution. The original task is considered completed once all the subtasks associated with it have been serviced. Performance of Split Merge and Fork-Join systems are often quantitatively analysed with respect to two metrics: the dispersion of subtasks and response time of tasks. The technique presented in this paper reduces subtask dispersion by delaying tasks that have a short average service time and starting the service of subtasks with long average service time straight away. Dynamic in our context refers to the ability to change subtask delay at any point in time before service is begun. However once the subtask has entered service it will continue service uninterrupted until it has finished.
This paper presents a new technique for minimising subtask dispersion. For example when comparing the new technique against currently existing methods using the same examples as described in their publications it is able to get an improvement of around 66%. The improvements are a result of a correction to an earlier technique. The previous research did not take into account the dynamic nature of the system. Therefore expected subtask dispersion of the system diverged with actual subtask dispersion of the system.

Keywords: dynamic dispersion reduction, fork-join, split merge, queueing networks

1 Introduction

Due to an ever increasing demand for performance and speed in the modern world and the eventual exhaustion of possible optimisations to single process systems more and more of the world is turning towards parallel and distributed systems. This trend is especially apparent in the IT world where companies are building distributed storage facilities, multi-core processors, RAIDs [11,10] and huge distributed computing platforms. However IT is not the only area where such demand is needed. In finance equities are nowadays traded lightning fast on a growing amount of exchanges and high dispersion and response time lead to monetary losses, manufacturers are making complex products with ever growing supply chains and even hospitals in hospitals patient care is being improved with queueing models [1].

[1] Email: ttp09@imperial.ac.uk
[2] Email: wjk@doc.ic.ac.uk

Queueing network models are a mathematical tool to describe task flow in parallel networks. The completion of one big process is split into a number of subtasks which each must be completed for the whole task to be completed. This paper investigates how to reduce the Dispersion of subtasks and response time of tasks in fork-join and split merge systems. The results are then compared against existing research that has been done on subtask dispersion reduction in fork-join and split merge systems.

Subtask dispersion is the difference in time that it takes for the first and last subtask to finish. Task response time on the other hand is the time it takes from the point when a task enters the queue to be processed to the point it has been fully serviced. Due to split merge systems being synchronous they tend to have low dispersion, but high task response time. On the other hand fork-join systems are asynchronous and tend to have smaller response time but higher subtask dispersion.

This paper examines split merge and fork-join systems that use delays on the processing of subtasks to reduce subtask dispersion. This paper makes a distinction between two types of delay adjusting systems that in the past have not be clearly distinguished. In the first type of system once a delay is set, it cannot be changed. The processing of the subtask is begun once the delay has run out. In the second class of systems, it is possible to preemptively modify the delay of a subtask at any time before it has begun service. However, once service has begun it is not possible to change it. We assume the system is notified instantaneously when any subtask finishes service.

This paper then proceeds to define a new way to calculate dispersion for split merge and fork-join systems that is able to take into account the fact that subtasks are interrupted and delays are cut short if its sibling task finishes service. The paper explores 2-server exponential and deterministic split-merge systems to offer some intuition behind the dispersion reduction technique. It then proceeds to a 3-server test case to demonstrate that it is able to deliver substantial reduction in dispersion compared to existing methods.

2 Preliminaries

This section contains a brief introduction to terms that are fundamental to the understanding of this paper. The section includes an introduction to split-merge and fork-join systems. Both systems are queueing network models for describing the processing of a set of subtasks in parallel. In addition it describes related quantitative metrics, including response time, subtask dispersion and a trade-off metric. The trade-off metric can be used to make decisions when both subtask dispersion and task response time are regarded as important.

2.1 Parallel Queueing Systems

2.1.1 Split Merge

In the split merge system considered in this paper arriving tasks have an interarrival rate that is exponentially distributed with a rate of λ. The system structure is shown in Figure 1. If no task is currently in service an arriving task enters service straight

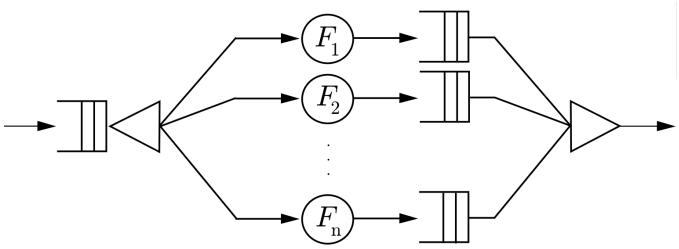


Fig. 1. A split merge system [12].

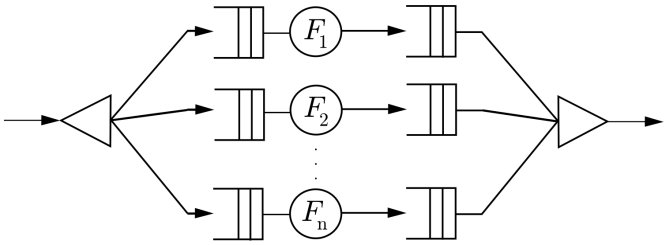


Fig. 2. A fork-join system [12].

away. Otherwise it enters a queue to wait for its turn. When a task completes service it leaves the system. Next the task first in the queue enters service, if there are no tasks in the queue the process is idle.

When a task enters service it is split into N subtasks. Each of these subtasks is then processed by its own server. The subtasks have their own unique service probability distribution times. The task is considered to be done with service once all N subtasks have been serviced by their respective servers.

2.1.2 Fork-Join

As shown in Figure 2, the fork-join system is quite similar in structure to the split merge system, but buffering of incoming tasks takes place at subtask-level instead of at task-level. The tasks again have an interarrival rate that is exponentially distributed with a rate of λ. Arriving tasks are immediately split into N subtasks. Each individual server serving the subtasks then has its own queue. The subtask servers independently process all subtasks waiting in their own queue. Once all the subtasks of a task have been serviced by their corresponding server the task is considered complete.

2.2 Performance Metrics

2.2.1 Task Response Time

Task Response time is the length of time it takes for a task to get processed. The clock is started when the task first enters the system. Once all the subtasks belonging to that task have been completely serviced the clock stops. Response time has been a very intensively researched topic in queueing systems over the last 50 or so years, see e.g. [2,8,9].

For split merge systems it is possible to calculate response time analytically with

the Pollaczek–Khinchine formula (1) that is defined for M/G/1 queues:

$$E[R_\lambda(t)] = \frac{\rho + \mu\lambda Var[X_{(N)}]}{2(\mu - \lambda)} + \mu^{-1} \tag{1}$$

Here μ is the service rate, λ is the arrival rate and $\rho = \lambda/\mu$ is the utilization of the server. The split merge system can be thought of as an M/G/1 queue where the probability distribution for task service time is defined by the probability distribution of last subtask's finishing time. With the theory of heterogeneous order statistics it is possible to calculate the probability distribution of all the tasks being finished at time t. The cumulative probability distribution is given by Equation (3). From the distribution it is then possible to calculate the mean and variance of the distribution, which are needed for the Pollaczek–Khinchine formula.

For fork-join systems there currently exists no analytical formula to calculate response time except for simple cases [4,5]. In the case of a fork-join system the response time needs to be calculated quantitatively by simulating the system.

2.2.2 Dispersion

Subtask dispersion is the time difference between the finishing times of the first and last subtasks. With N subtasks the expected times of first and last subtask finishing times can be calculated by using the theory of heterogeneous order statistics [3]. The cumulative distribution function for the first subtask to finish is given by Equation (2) and last by Equation (3). The research on subtask dispersion of parallel processing systems is quite recent and although some literature does exist [14,15,13].

$$F_1(t) = Pr\{X_{(1)} < t\} = 1 - \prod_{i=1}^{N}[1 - F_i(t)] \tag{2}$$

$$F_N(t) = Pr\{X_{(N)} < t\} = \prod_{i=1}^{N}[F_i(t)] \tag{3}$$

Heterogeneous order statistics be used to define subtask dispersion in the following way, which is shown in Equation (4) and (5).

$$E[D(t)] = \int_0^\infty F_1(t) - F_N(t)\mathrm{d}t \tag{4}$$

$$E[D(t)] = \int_0^\infty 1 - \prod_{i=1}^{N}(1 - F_i(t)) - \prod_{i=1}^{N} F_i(t)\mathrm{d}t \tag{5}$$

The way dispersion is calculated here has been successfully used to calculate subtask dispersion of a split-merge systems [15,13] and for analysing instantaneous configurations of fork-join systems [14]. In the case of the fork-join algorithm in [14] subtasks are set to start processing immediately after a sibling task finishes if they have not started already. However the equation 5 does not take this into account and therefore is unable to minimise the dispersion of a dynamic parallel system correctly. In Section 3.1 a new formula is derived that is able to take into account tasks being put into service immediately after a sibling task finishes service.

2.2.3 Trade-Off Metric

Sometimes both dispersion and response time of the system are equally important. In these cases it is possible to measure the effectiveness of the system with the trade-off metric. [15] The trade-off metric is defined as the product of dispersion and response time, i.e.

$$T(\lambda, t) = E[D(t)]E[R_\lambda(t)] \tag{6}$$

A similar metric has been explored in the context of the energy–response time product analysis of power policies for server farms [6,7]. For split merge systems the trade-off equation can be expressed as:

$$T(\lambda, t) = \left[\int_0^\infty 1 - \prod_{i=1}^{N}(1 - F_i(t)) - \prod_{i=1}^{N} F_i(t)\mathrm{d}t \right] \left[\frac{\rho + \mu\lambda Var[X_{(n)}]}{2(\mu - \lambda)} + \mu^{-1} \right] \tag{7}$$

For fork-join systems the trade-off metric has to be quantitatively measured through simulations, since – to the best of our knowledge – there are no closed form solutions for either subtask dispersion or task response time.

3 Method

This section introduces a way to calculate dispersion of a split merge system where a *start work* signal is sent to sibling subtasks once service of a subtask is completed. The model assumes that a delay applied to a subtask can be arbitrarily preempted at any point before service of the subtask has begun. However, once a subtask has begun service it will be serviced uninterrupted until it completes service. The *start work* signal is sent, because removing delays after the first sibling subtask finishes service reduces both subtask dispersion and task response time.

3.1 Calculating subtask dispersion in dynamic split merge system

It is explained here how the minimum subtask dispersion of a dynamic split merge system can be found by choosing an appropriate delay vector $\mathbf{d}$. Equation (8) displays the formula that is minimised in order to find the minimal delay vector $\mathbf{d}$. The formula is split into two parts: $T(i, t, \mathbf{d})$ and $E_r(i, t, \mathbf{d})$. The last two conditions on $\mathbf{d}$ guarantee that delays are non negative and that no unnecessary delays are added.

The first function $T(i, t, \mathbf{d})$ calculates what is the probability that server i is the first server that finishes servicing its subtask at the time t, with the given delays $\mathbf{d}$. This result is calculated by multiplying the probability density function of server i finishing at time t with the probability of all the other servers not finishing before time t. The mathematical formulation of this can be seen in Equation (9).

The second function $E_r(i, t, \mathbf{d})$ calculates how long the rest of the servers will take to complete their service, with the given delays $\mathbf{d}$. The average time for the rest of the servers to complete service is computed with the help of heterogeneous order statistics presented in [15].

$G_j(t, t', d_j)$ represents the probability distribution function of the jth server. If $t' < d_j$ then the jth server has not yet begun service of its subtask and servicing is

begun immediately. Otherwise the server has already started servicing its subtask. In this case the service time is renormalised to take into account that some service has already been performed and the service of the subtask has not been completed. The two part G-function is the key difference compared to previous work presented in [14,15,13]. We seek:

$$d_{\min} = \arg\min_{\mathbf{d}} \sum_{i=1}^{N} \int_0^\infty T(i,t,\mathbf{d}) E_r(i,t,\mathbf{d}) dt \tag{8}$$

where

$$T(i,t,\mathbf{d}) = f_i(t-d_i) \prod_{j \neq i} [1 - F_j(t-d_j)] \tag{9}$$

and

$$E_r(i,t',\mathbf{d}) = \int_0^\infty [1 - \prod_{j \neq i} G_j(t,t',d_j)] dt \tag{10}$$

with

$$G_j(t,t',d_j) = \begin{cases} F_j(t) & \text{if } t' < d_j \\ F_j(t-(t'-d_j)|t>0) & \text{otherwise} \end{cases} \tag{11}$$

subject to conditions

$$\prod_{i=1}^{N} d_i = 0 \tag{12}$$

and

$$\forall i \quad d_i \geq 0 \tag{13}$$

3.2 Deterministic 2 server example

The concept of a split merge system was explained in Section 2.1.1. In this section we will apply the dispersion reduction formula above to a split merge system with 2 servers. The service times of the servers are 1 and 2 as shown below:

$X_1 \sim \text{Det}(1)$
$X_2 \sim \text{Det}(2)$

Due to the deterministic function causing problems with integration our example will use uniform functions with a small range to approximate it as shown in Equation (14).

$$\text{Det}(n) \approx \text{Uni}(n - 0.001, n + 0.001) \tag{14}$$

The results of the experiment can be seen in Figure 3. The optimal delay is naturally $\mathbf{d} = (1, 0)$ and when set delays deviate from the optimal solution dispersion grows. Increasing server 1 delay past optimal delay causes an increase that caps at 1. The capping is due to the dynamic delay interruption. Server 2 behaves similarly with the delay capped at 2. Figure 4 demonstrates how the system works when the first queue is delayed by 1 time unit at the beginning of each processed task.

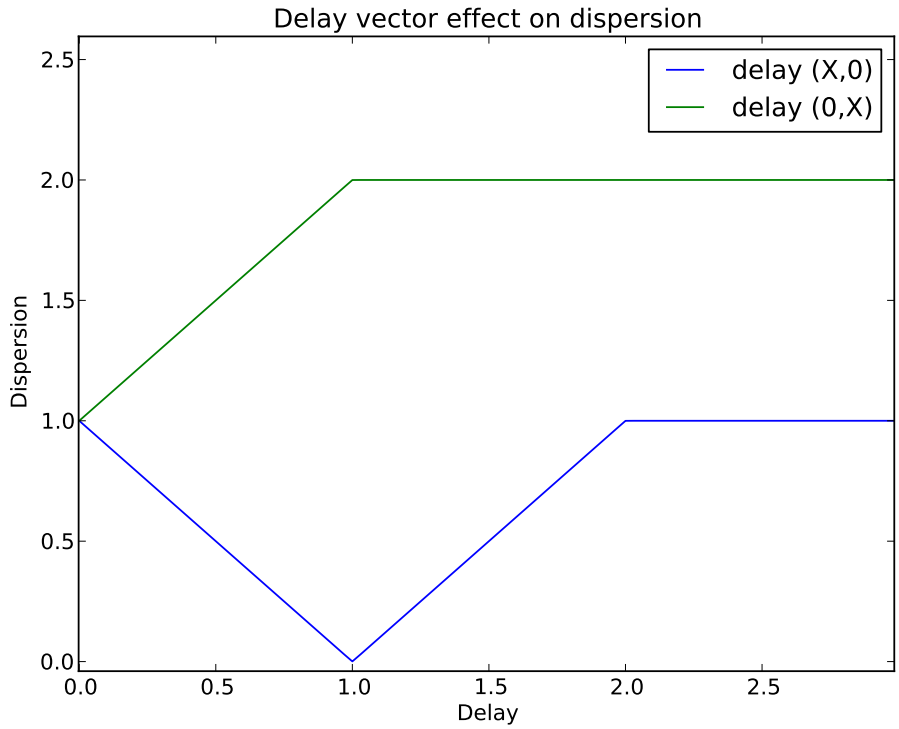


Fig. 3. Demonstration of how the delays affect dispersion in the two server deterministic example.

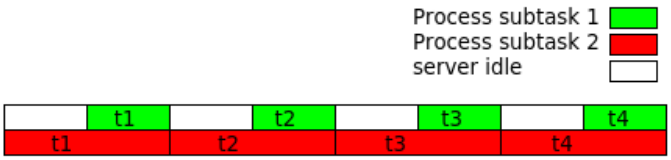


Fig. 4. Demonstration of how the deterministic two server case processes its tasks.

3.3 Exponential 2 server example

In this section a split merge system with exponentially distributed subtask service times are analysed. The exponential distribution has a parameter λ which is the inverse of average service time. For the first server $\lambda = 1$ and second server $\lambda = 2$. Therefore the average service times of the two servers are 1 and 0.5. The details of the servers are shown below:

$$X_1 \sim \text{Exp}(\lambda = 1)$$
$$X_2 \sim \text{Exp}(\lambda = 2)$$

The results of the experiment can be seen in Figure 5. Intuition says that when a delay is set on the server 1 the dispersion should increase towards 1.0. This is, because then the probability of server 2 finishing first increases towards 1. This is due to server 1 average completion time after server 2 completes is 1. A similar argumentation can be done for setting delays on server 2 instead. As the delay grows towards infinity chance of server 1 completing service first grows. The average service time of server 2 is 0.5. Therefore dispersion with infinite delay on

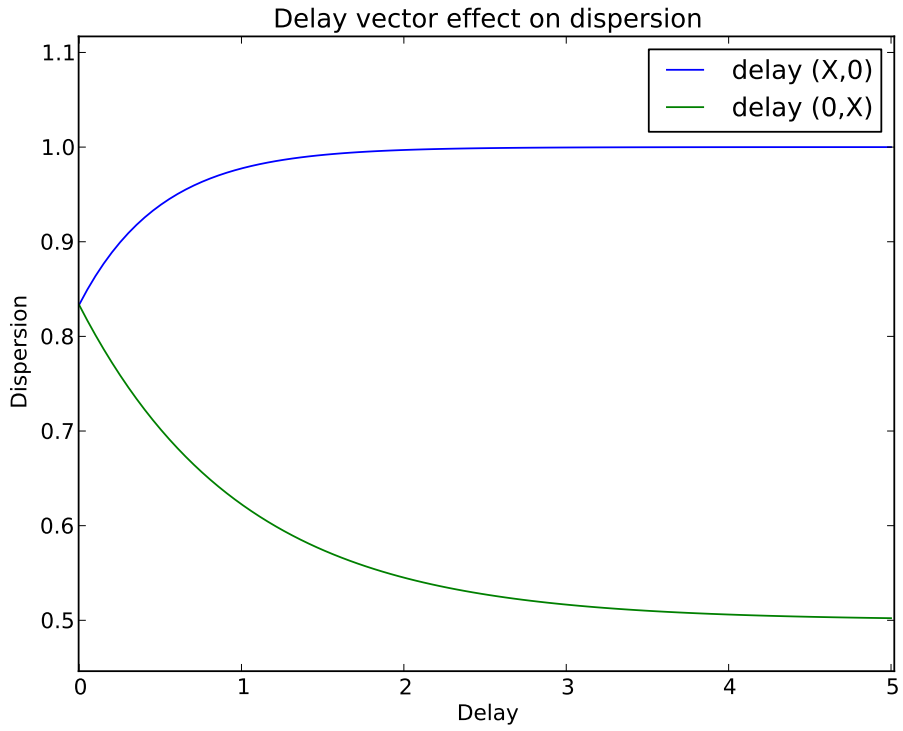


Fig. 5. Demonstration of how the delays affect dispersion in the two server exponential example.

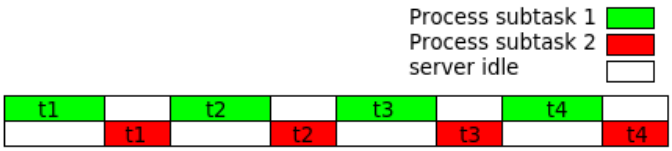


Fig. 6. Demonstration of how the exponential two server case processes its tasks.

server 2 is 0.5. The optimal delay that minimises dispersion is $\mathbf{d} = (0, \infty)$, as infinite delay on the server 2 guarantees that the server 1 will always finish first. Figure 6 demonstrates how the delays are applied in the two server exponential case. It can be seen that server 1 completes its service before service on server 2 is begun.

3.4 Fork-join systems

The throughput of a parallel system is maximised when at least one of the servers is performing work on a subtask throughout the time a task is being processed. If this is not the case as is in the exponential case in Figure 6 it is possible to decrease response time by removing idle time from the processing of each subtask. This however will increase subtask dispersion. The effect of removing idling time on subtasks servicing can be observed in Figure 7.

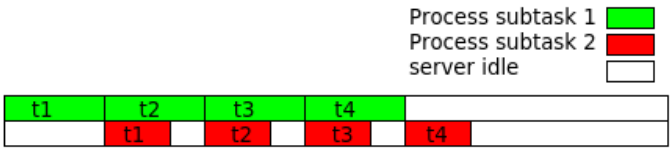


Fig. 7. An example of how delays can be removed from a split merge system when processing as a fork-join system

4 Results

In this section we present results of existing methods for subtask dispersion reduction in fork-join and split merge systems and compare them against the algorithms described in this paper. The example uses the same probability distributions as the paper [14]. Split merge results will be evaluated analytically. The fork-join systems will be simulated with a benchmark that will include 5 repetitions of 5 million tasks each. For each run the average task response time, subtask dispersion and trade-off penalty will be noted. The probability distribution setups of the servers are shown below. The interarrival time of new tasks entering the system is exponentially distributed with $\lambda = 0.78$ tasks per time unit.

$$X_1 \sim \text{Exp}(\lambda = 1)$$
$$X_2 \sim \text{Exp}(\lambda = 5)$$
$$X_3 \sim \text{Exp}(\lambda = 10)$$

4.1 Existing methods from previous publications

The five methods described here are the same as in the paper [14]. Methods 1-4 do not use interrupt to begin service immediately after a sibling task has finished while Method 5 uses it.

Method 1 is a vanilla split merge system where no delays are applied. The tasks are processed one at a time. The service of next task does not begin before all the previous task's subtasks have finished. Corresponding performance metrics are:

Task response time:	5.195 time units
Subtask dispersion:	0.976 time units
Trade-off:	5.069 (time units)^2

Method 2 [15] is a split merge system where dispersion is minimised with the formula described in Section 2.2.2. The resulting delays for subtasks are: $\mathbf{d} = (0, 0.524, 0.585)$. Corresponding performance metrics are:

Task response time:	33.638 time units
Subtask dispersion:	0.783 time units
Trade-off:	26.345 (time units)^2

Method 3 [13] is a split merge system where trade-off is minimised with the formula described in Section 2.2.3. The resulting delays for subtasks are: $\mathbf{d} = (0, \sim$

$0, 0.068)$. Corresponding performance metrics are:

Task response time:	5.286 time units
Subtask dispersion:	0.946 time units
Trade-off:	4.999 (time units)^2

Method 4 is a vanilla fork-join system with no delays applied between subtasks. Each task is split into 3 subtasks which then each individually queue for their respective servers. Corresponding performance metrics are:

Task response time:	4.555 time units
Subtask dispersion:	4.480 time units
Trade-off:	20.406 (time units)^2

Method 5 is a fork-join system with a dynamic subtask reduction algorithm [14]. This algorithm uses interruptions to start processing of sibling subtasks once a subtask finishes. The system uses definition of dispersion from Section 2.2.2. Corresponding performance metrics are:

Task response time:	4.675 time units
Subtask dispersion:	0.768 time units
Trade-off:	3.590 (time units)^2

4.2 New methods presented in this paper

The methods described here are described in Section 3 of this paper. They all use the interrupt to begin service immediately after a sibling task has finished. The results have been calculated with the same simulation setup as the fork-join systems in the previous subsection.

Method 6 is a split merge system that uses the new dispersion calculation in Section 3.1 to calculate delays. This algorithm uses interruptions to start processing of sibling tasks once a task finishes. The resulting delays for subtasks are: $\mathbf{d} = (0, \infty, \infty)$. Corresponding performance metrics are:

Task response time:	6.333 time units
Subtask dispersion:	0.257 time units
Trade-off:	1.628 (time units)^2

Method 7 is a fork-join system that uses Method 6 to establish a minimally dispersed split merge system. idling time is then squeezed according to the principles of Section 3.4. Corresponding performance metrics are:

Task response time:	4.818 time units
Subtask dispersion:	0.269 time units
Trade-off:	1.296 (time units)^2

5 Conclusions

Three main conclusions can be drawn from the results. First the *start work* interrupt that removes delays on other subtasks preemptively once the first sibling subtask finishes is able to reduce both task response time and subtask dispersion greatly. This can be seen when the task response times of Methods 1–3 and Method 6 are compared. In addition, Method 6 is able to greatly decrease subtask dispersion, with only a slight increase in the task response time of Methods 1 and 2. Against Method 3 it is able to produce improved results on both metrics.

The second observation regarding the results is that the traditional method for calculating expected subtask dispersion is not valid for systems where subtask delay preemption is applied. In our case studies, the new method was able to reduce subtask dispersion by a factor of 3, which is a major improvement.

Finally a more subtle conclusion is that even though response time of the Method 7 is slightly worse compared to Method 5 its throughput is just as much or slightly better. As tasks 2 and 3 are serviced after task 1 finishes the response time grows. However only server 1 is a bottleneck of the process. Therefore it is able to process just as many tasks. A demonstration of this can be seen in Figure 7. By the time the response time timer for the task finishes, the bottleneck server has been servicing the bottleneck subtask of the next task for some time already.

References

[1] S. W. M. Au-Yeung, P. G. Harrison, and W. J. Knottenbelt. Approximate queueing network analysis of patient treatment times. In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools*, ValueTools '07, pages 45:1–45:12, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[2] François Baccelli, Armand M. Makowski, and Adam Shwartz. The fork-join queue and related systems with synchronization constraints: Stochastic ordering and computable bounds. *Advances in Applied Probability*, 21(3):pp. 629–660, 1989.

[3] Herbert A. David and H. N. Nagaraja. *Wiley Series in Probability and Statistics*. John Wiley & Sons, 1980.

[4] L. Flatto and S. Hahn. Erratum: Two parallel queues created by arrivals with two demands I. *SIAM Journal on Applied Mathematics*, 45(1):168–168, 1985.

[5] Leopold Flatto. Two parallel queues created by arrivals with two demands II. *SIAM Journal on Applied Mathematics*, 45(5):pp. 861–878, 1985.

[6] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155 – 1171, 2010.

[7] Anshul Gandhi, Mor Harchol-Balter, and Ivo Adan. Server farms with setup costs. *Performance Evaluation*, 67(11):1123 – 1138, 2010.

[8] Peter Harrison and Soraya Zertal. Queueing models of {RAID} systems with maxima of waiting times. *Performance Evaluation*, 64(78):664–689, 2007.

[9] P. Heidelberger and K.S. Trivedi. Analytic queueing models for programs with internal concurrency. *IEEE Transactions on Computers*, C-32(1):73–82, Jan 1983.

[10] Abigail Lebrecht, Nicholas J. Dingle, and William J. Knottenbelt. Modelling Zoned RAID Systems using Fork-Join Queueing Simulation. In *6th European Performance Engineering Workshop (EPEW 2009)*, volume 5652 of *Lecture Notes in Computer Science*, pages 16–29, July 2009.

[11] Abigail S. Lebrecht, Nicholas J. Dingle, and William J. Knottenbelt. Analytical and simulation modelling of zoned raid systems. *Comput. J.*, 54(5):691–707, May 2011.

[12] Iryna Tsimashenka. *Reducing Subtask Dispersion in Parallel Queueing Systems*. PhD thesis, Imperial College London, 2014.

[13] Iryna Tsimashenka, William Knottenbelt, and Peter Harrison. Controlling variability in split-merge systems. In Khalid Al-Begain, Dieter Fiems, and Jean-Marc Vincent, editors, *Analytical and Stochastic Modeling Techniques and Applications*, volume 7314 of *Lecture Notes in Computer Science*, pages 165–177. Springer Berlin Heidelberg, 2012.

[14] Iryna Tsimashenka and William J. Knottenbelt. Reduction of subtask dispersion in fork-join systems. In Simonetta Balsamo, William J. Knottenbelt, and Andrea Marin, editors, *Computer Performance Engineering*, volume 8168 of *Lecture Notes in Computer Science*, pages 325–336. Springer Berlin Heidelberg, 2013.

[15] Iryna Tsimashenka and William J. Knottenbelt. Trading off subtask dispersion and response time in split-merge systems. In Alexander Dudin and Koen De Turck, editors, *Analytical and Stochastic Modeling Techniques and Applications*, volume 7984 of *Lecture Notes in Computer Science*, pages 431–442. Springer Berlin Heidelberg, 2013.

A Hybrid Simulation Framework for the Analysis of Petri Nets and Queueing Networks

Esmaeil Habibzadeh, Demetres D. Kouvatsos (University of Bradford, UK),
Guzlan M.A. Miskeen (University of Sebha, Libya)

Abstract The dynamic behaviour and properties of discrete flow systems could be captured by employing suitable stochastic modelling tools, such as Petri Nets (PNs) and Queueing Networks (QNs), incorporating flexible and powerful features; such tools, however, suffer from some inherent limitations. Specifically, the applicability of PNs may be hindered due to their state space explosion problem as the number of tokens or size of the system increases whilst QNs are not generally suited to effectively model system dynamic features as concurrency, synchronization, mutual exclusion and/or conflict. To overcome some of these constraints and exploit the best attributes of both PNs and QNs modelling paradigms, this work reports the current state of progress into design and development of the HPQNS (Hybrid Petri Queueing Networks Simulation) framework. This computational tool has shown to improve the credibility and efficiency of performance related security modelling and evaluation studies based on discrete event simulations.

Keywords Performance, security, simulation, stochastic models, queueing networks (QNs), Petri nets (PNs)

Summary

1. Introduction

Petri nets (PNs) and queueing networks (QNs) have long been used as powerful modelling paradigms for performance analysis of a wide range of discrete flow systems such as computer systems and communication networks (c.f., [1-14]). Nevertheless, these formalisms have shown to suffer from some inherent limitations. PNs, for instance, are associated with the state-space explosion problem (or, 'largeness' problem), which arises when either the number of tokens or the size of the model grows is increasing progressively whilst that they have no scheduling strategies to maintain the order of arrival tokens in places (e.g., [1,4,6]). QNs, on the other hand, are suitable tools for capturing the security processing (e.g., encryption/decryption times) and transmission end-to-end delays of multiple classes of packets. However, they are unable to effectively model some important system dynamics such as concurrency, synchronization, mutual exclusion and/or conflict (e.g., [3-6]). To overcome or alleviate some of these constraints, a hybrid modelling framework needs to be developed by

combining the expressiveness and modelling power of QNs and extensions of PNs, where sub-models of PNs and QNs are allowed to coexist and interact with one another.

This work reports the progress that has been made so far towards the design and development of the Hybrid Petri Queueing Networks Simulation (HPQNS), a new software framework for the performance related security of high speed networks that has a number of advantages over above-mentioned and combinations (especially in the field of computer science) of the aforementioned conventional modelling techniques. In addition to the adoption of hardware contention and scheduling disciplines, the HPQNS framework facilitates the modelling and evaluation of software contention, synchronization, asynchronous processing and blocking, just to name a few. In other words, it enables the integration of both hardware and software aspects of system behaviour into the same model.

The explicit modelling of software aspects of system behaviour alone has demonstrated significant improvement in accuracy and efficiency of the performance models. Another major advantage of HPQNS framework is that, due to its object-oriented design and development, it helps render tractably the simulation of models of larger and more complex systems.

2. The HPQNS Framework

The present version of the HPQNS tool was designed and developed using Java programming language and as with all other Java applications, it is platform independent. The framework facilitates the implementation and configuration of a variety of system models; it enables us to run the simulations for as many times as required for the results to fit within certain confidence intervals. For data presentation purposes, the HPQNS tool provides suitable graphical curves so that brief summaries of simulation outcomes can easily be reviewed and analysed. At this stage, the statistics being collected as well as jobs/events tracking information are all stored in the system's main memory (RAM) throughout the simulation process; this is done to speed up and keep simulation runtimes reasonably low.

Unlike the development process of a software package, which usually involves a bottom-up approach to implementation (from smaller objects to bigger ones), it is much more straightforward to take a top-down approach (from the bigger components to smaller ones) to build the model – the class hierarchical structure of HPQNS can be seen in Fig. 1. As such, in its highest level, the HPQNS framework needs an empty virtual network object (environment) to be created first, which should include the individual nodes, the inter-arrival processes, the random number generators and the routing information matrix.

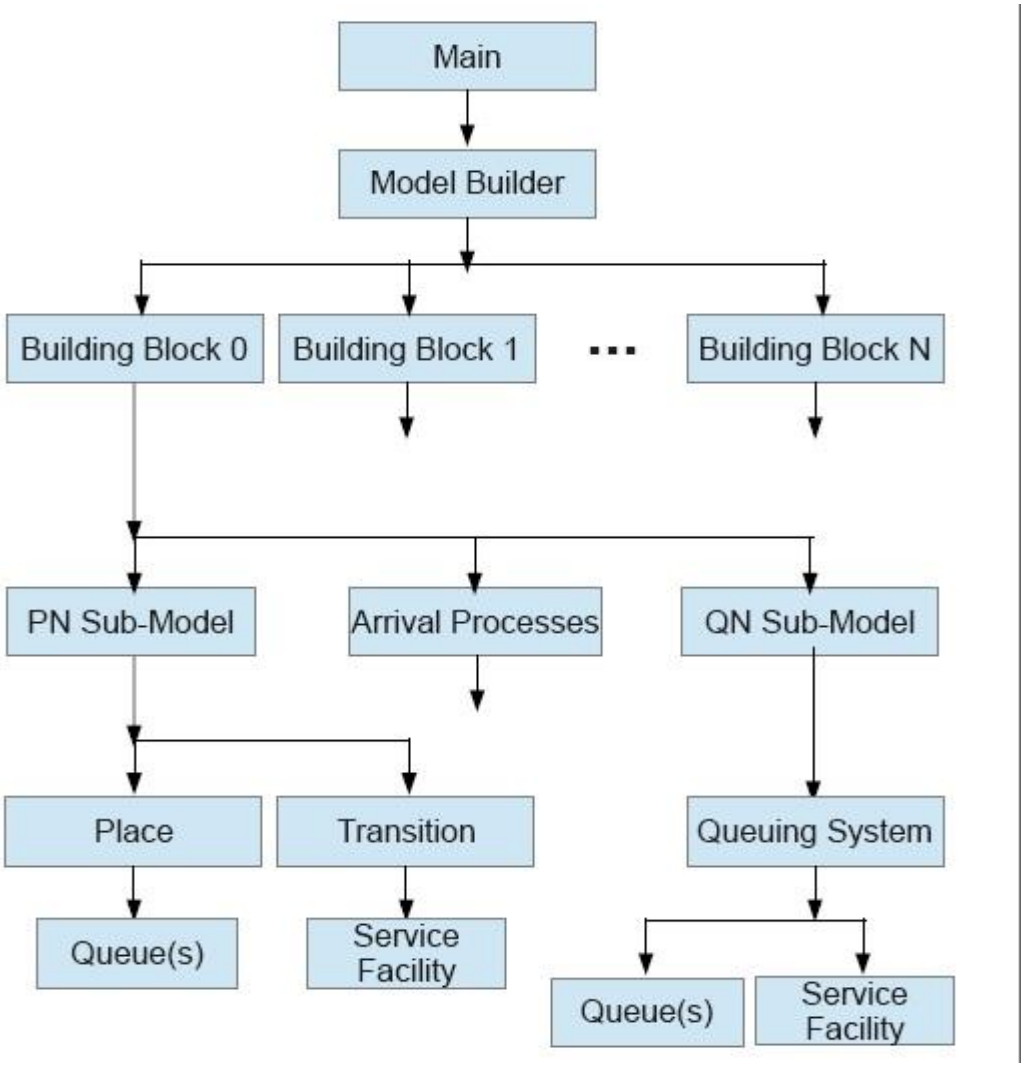


Fig. 1 The Class Hierarchy of HPQNS

3. Building Blocks

The individual nodes, also known as building blocks, are assumed to be homogeneous and virtually based inside the network object and are associated with a number of modelling levels. At the first level, the arrival traffic process should be specified and declared. This process is one of the most important role players in the model whether there is only one class of jobs or multiple job classes with different levels of priority. At the second level, the inter-arrival patterns should be clearly defined in terms of parameters of the selected probability distribution. At this level of model design, the HPQNS requires a mechanism by which the existing nodes will interact by exchanging group information. This interconnection is governed by a routing matrix called “InterNodeComm”; it is important that the entries of the routing matrix are appropriately set at this stage so jobs departing from nodes consult it

about the possible destination(s) to go to next. In addition, this matrix is responsible for ensuring that each node is receiving its own share of arrival traffic, if any.

The next level of modelling process takes place within the nodes by implementing a detailed model in either pure PN, pure QN or hybrid PN and QN to represent the dynamic behaviour of each node internally. At this level, the servers and their corresponding service times, queues and the scheduling mechanisms, etc. will clearly be defined and created. To manage all these features smoothly, the HPQNS heavily exploits the standard matrices commonly used in PNs and/or QNs, such as I, O, H, C matrices in PNs and NTM, CNTM ones in QNs. While these matrices are only responsible for routing jobs within their respective modelling paradigm, the HPQNS requires yet another matrix, which would manage the interactions between PN and QN sub-models; so-called "IntraNodeComm", the matrix determines the routing rules between PN and QN sub-models within each building block.

The applicability of the HPQNS tool is illustrated in Kouvatsos et al [15].

References

1. Kounev, S., & Buchmann, A. (2006). SimQPN- a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation, 63*(4), 364-394.

2. Kounev, S. (2006). Performance modeling and evaluation of distributed component-based systems using queueing petri nets. *Software Engineering, IEEE Transactions on, 32*(7), 486-502.

3. Kounev, S., & Buchmann, A. (2008). On the use of queueing petri nets for modeling and performance analysis of distributed systems. In V. Kordic (Ed.), *PetriNet, Theory and Application* (Vol. Petri Net, Theory and Applications, pp. 149-178). Vienna, Austria.

4. Balbo, G., Bruell, S. C., & Ghanta, S. (1988). Combining queueing networks and generalized stochastic petri nets for the solution of complex models of system behavior. *IEEE Transactions on Computers, 37*(10), 1251-1268.

5. Bause, F. Queueing petri nets-a formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of 5th International Workshop on Petri Nets and Performance Models, 1993* (pp. 14-23): IEEE. Balbo, G., & Chiola, G. Stochastic petri net simulation. In *1989 Winter Simulation Conference Proceedings, 1989* (pp. 266-276): ACM

6. Marsan, M. A., Bobio, A., & Donatell, S. (1998). petri nets in performance analysis: an introduction. In W. Reisig, & G. Rozenberg (Eds.), (Vol. LNCS1491, pp. 211–256). Berlin (Germany): Springer-Verlag.

7. Jensen, K. (1997). A brief introduction to coloured Petri Nets. In E. Brinksma (Ed.), *Tools and Algorithms for the Construction and Analysis of Systems* (Vol. 1217, pp. 203-208, Lecture Notes in Computer Science): Springer Berlin Heidelberg.

8. Bause, F. (1993). "QN+ PN= QPN"-combining queueing networks and petri nets. (pp. 1-15). Germany: University of Dortmund.

9. Kouvatsos, D., & Awan, I. (2003). Entropy maximisation and open queueing networks with priorities and blocking. *Performance Evaluation, 51*(2), 191-227.

10. Kouvatsos, D. D. (1994). Entropy maximisation and queueing network models. *Annals of Operations Research, 48*(1), 63-126.

11. Kouvatsos, D. D., Alanazi, J. S., & Smith, K. (2011). A Unified ME algorithm for arbitrary open QNMs with mixed blocking mechanisms. *Numerical Algebra, Control and Optimization (NACO)The American Institute of Mathematical Sciences (AIMS), 1* (4), 781 – 816, doi:doi:10.3934/naco.2011.1.781.

12. Becker, M., & Szczerbicka, H. (1999). PNiQ: Integration of queuing networks in generalised stochastic petri nets. *IEE Proceedings, Software, 146*(1), 27-32, doi:10.1049/ip-sen:19990153.

13. Becker, M., & Szczerbicka, H. Integration of multi-class queueing networks in generalized stochastic petri nets. In *IEEE International Conference on Systems, Man, and Cybernetics, 2001* (Vol. 2, pp. 1137-1142): IEEE. doi:10.1109/icsmc.2001.1309629.

14. Szczerbicka, H. (1992). A combined queueing network and stochastic Petri net approach for evaluating the performability of fault-tolerant computer systems. *Performance Evaluation, 14*, 217-226, doi:10.1016/0166-5316(92)90005-2.

15. Kouvatsos D. D., E. Habibzadeh & Miskeen G.M.A. Performance Modelling and Evaluation of Secure Dynamic Group Communication Systems in RANETs, Paper Summaries of UKPEW 2014 Proceedings, Sept. 2014.

A Case Study in Capacity Planning for PEPA Models with the PEPA Eclipse Plug-in

Christopher D. Williams and Allan Clark[1,2,3]

LFCS, School of Informatics
University of Edinburgh
Edinburgh, United Kingdom

Abstract

We report on the addition of *Capacity Planning* facilities to the PEPA Eclipse Plug-in, a software tool for analysing performance models written in the PEPA language. The PEPA language allows the compositional description of complex systems consisting of different kinds of processes. The capacity planning addition allows modellers to automatically search for the populations of processes that allows for an optimal trade-off between the performance of the system and the cost of acquiring or operating the components of the system under the modeller's control.

Keywords: PEPA, Capacity Planning, Optimisation, Performance Evaluation, Modelling

1 Introduction

In this paper we report on the capacity planning framework for Performance Evaluation Process Algebra (PEPA)[8] implemented in the PEPA Eclipse Plug-in[16]. PEPA is a language in which modellers can compositionally describe complex systems. Generally modellers define several different kinds of processes which interact with each other by sharing activities. Once the model is defined, it can be numerically evaluated via a suite of techniques to obtain performance metrics. If the model is accurate enough then these translate to, and provide insight to, the real system under investigation.

Typically a process is defined with several possible states. The performance metrics in the first instance are the long-term probabilities of a process being in each of its possible states. Where the model contains many copies of the same process, this is equivalent to asking the long-term populations of each state.

Generally, the populations, will provide the modeller with utilisation information. For example in a *Server-Client* model, a server process may have a set of

[1] Special thanks to Mirco Tribastone who contributed the case study
[2] Email: blasedefgmail.com
[3] Email: a.d.clark@ed.ac.uk

possible states, some of which correspond to a state in which the server is busy processing a particular kind of response and other states will correspond to a state in which the server is idle waiting for a client to make a request. Knowing the proportion of servers which are generally busy may tell the modeller if the service is over-provisioned, in that we have many servers which are idle because the number of servers is enough to satisfy the expected (and modelled) demand.

From the model and the long-term/steady-state population distributions we can derive performance measures which may more directly determine whether the service would be over or under provisioned. Two important measures are the throughput of particular actions or the time a particular component can expect to remain in a particular set of states. Again in a *Server-Client* setting, the throughput may tell us how many requests are responded to per unit of time, or it may tell us the rate at which requests must be dropped.

However the throughput of requests is commonly also dependent upon the rate at which requests are made, and hence may not be a reliable indicator of whether the system has enough performance for a given demand. For this we can calculate the expected time a given component is in a given set of states. Typically we would calculate the expected time a single client is in a state in which they have made their *request* and are waiting for that request to be responded to by the service. This gives us the response-time as observed by a typical consumer of the service.

Being able to predict the performance of a proposed service by modelling the service and calculating the response-time under a given client-load is clearly useful. However when designing the system we still have some flexibility around the number of components that we may deploy. In the simple case we may be able to deploy more or fewer servers. In a more complex environment there are different components that make up the service being offered. For example there may be web servers and database servers as well as an external authentication service.

When designing such a system, we would like to know what configuration of the system is optimal. So we wish to know what populations of server components meets some required performance standard. One can often meet a performance standard simply by deploying ever-increasing numbers of all server components. However, there is generally some cost associated with acquiring and/or operating each server and the costs may differ for different types of server. Hence we wish to find not only a configuration of the system that will satisfy the performance demands but also the cheapest such system configuration.

A modeller can always guess at a sufficiently low-cost configuration that might satisfy the performance demands and simply evaluate that configuration through their model. In previous work [13], an extension to the PEPA Eclipse Plug-in software is discussed. The extension implements an automatic search for the optimal configuration, ie. the extension implements automatic capacity planning for PEPA models.

In this paper we contribute a case-study demonstrating the value of the capacity planning extension to the PEPA Eclipse Plug-in. We begin in the following section with background information detailing PEPA, associated performance measures and capacity planning in general. This is then followed by a in-depth description of the case study scenario and the associated PEPA model. The results obtained from

running the software to obtain an optimal configuration of the server components in the case study are then discussed. We end with future work discussing how to make the software ever more general without sacrificing ease-of-use and the conclusion that the capacity planning extension is an important feature for modelling software.

2 Background

This section gives an overview of the PEPA modelling language and the necessity to provide capacity planning facilities. We first discuss PEPA, then describe how PEPA models can be evaluated to obtain basic quantitative information concerning how the model's component states evolve over time and their steady-state (or long-term) distributions. We then describe how more informative performance measures can be derived from this basic information. Crucially we are looking at the throughput of particular activities within the model as well as the expected time the model remains in a particular set of states. The latter is known as the average response-time in the specific case where the set of states represents a client waiting for service/response from a server.

2.1 PEPA

PEPA is a stochastically-timed process algebra where sequential components are defined using prefix and choice. PEPA models require these sequential components to cooperate on some activities, and hide others. A PEPA model typically consists of several sequential components, placed in cooperation. In the model:

$$P \underset{\mathcal{L}}{\bowtie} Q$$

The sequential components P and Q cooperate on the activities in the set $\mathcal{L}$. If activity α is in the set $\mathcal{L}$ then P and Q are required to cooperate on α. If activity β is not in $\mathcal{L}$ then either of P or Q, or both, may perform this activity independently. When $\mathcal{L}$ is empty we write $P \parallel Q$ instead of $P \underset{\emptyset}{\bowtie} Q$. We also allow the special cooperation $P \underset{*}{\bowtie} Q$ to be a synonym for $P \underset{\mathcal{L}}{\bowtie} Q$ where $\mathcal{L}$ is the set of activities performed by both P and Q.

Rates are associated with activities performed by each component. The symbol $\top$ is used to indicate that the component will passively cooperate with another on this activity. In this case the passive component may enable or restrict the activity from being performed by the cooperating component but the rate when enabled is determined by the actively cooperating component. The component $(a, r).P$ performs the activity a at rate r whenever it is not blocked by a cooperating component and becomes the process P. The component $(a, \top).Q$ passively synchronises on a and becomes process Q.

In PEPA models we often work with *arrays* of components. We use arrays to represent workload (such as a number of independent clients) or resources (such as a number of independent servers). We write $P[5]$ to denote five copies of the component P which do not cooperate and $P[5][\alpha]$ to denote five copies of the component P which cooperate on the activity α. That is, $P[5]$ is an abbreviation for

$P \parallel P \parallel P \parallel P \parallel P$ and $P[5][\mathcal{L}]$ is an abbreviation for

$$P \underset{\mathcal{L}}{\bowtie} P \underset{\mathcal{L}}{\bowtie} P \underset{\mathcal{L}}{\bowtie} P \underset{\mathcal{L}}{\bowtie} P.$$

The PEPA language is formally defined in [8]. Applications of the language are described in [7,11,10].

2.1.1 Evaluating the Model

PEPA is used to calculate performance measures. Traditionally this has been achieved by translating the PEPA model into its underlying continuous-time Markov chain. However for models with large numbers of components the state-space of the model is too large to traverse and other techniques have been utilised. We have used stochastic simulation and translation into Ordinary Differential Equations (ODEs) [9,17].

In this work we have utilised the translation into ODEs. The techniques described are generalisable to any method of obtaining results from a PEPA model, however if the search space is large we may evaluate many instantiations of the model, meaning that whichever method is used should be fast for all the candidate configurations of the model.

When the model is translated into a set of ODEs these can be numerically evaluated to provide a time series which describes the population levels of the states of each component kind over time. For some measures we are not interested in the evolution of the component state populations but rather the long-term proportion of the components in each state, known as the steady-state. To obtain these the ODEs can be numerically evaluated for increasing time until the populations are stable. This technique requires that your system does not exhibit oscillating behaviour.

2.1.2 Performance Measures

Once we have the long-term component populations we can calculate the expected performance of the system. The two most common performance measures that we are interested in are the throughput of particular actions or the average duration of a particular state, or set of states. For example when considering some kind of service we may wish to measure the throughput of responses made or the average response-time. The average response-time here would be the expected delay between a particular client making a request and that same client receiving a response to the earlier request.

Typically the average response-time is appropriate because it is not a measure that is penalised when the service is over-provisioned. When the service is over-provisioned the performance may be very high, but the throughput of responses can only be as high as the throughput of requests made by the users of the service. Response-time on the other hand can still be, and is likely to be, low, even when the rate of requests is low. In the case study presented in this work we focus on response-time.

Response-time can be calculated with an application of Little's Law[14]. Little's law states that the long-term average number of customers in a stable system L is equal to the long-term average arrival rate λ multiplied by the average time a

customer spends in the system W. Hence, $L = \lambda W$, re-arranging for W we have that $W = L/\lambda$. In our case W is the response-time we seek, and L is the long-term population of clients in states between their request and response activities whilst λ is long-term throughput of requests made.

When considering systems with many consumers we wish to evaluate the performance of the system as observed by a typical consumer. In other words we must be careful to measure the expected time between a request made by a particular client and the response received by that same client. Rather than the expected time between any request and the next response to any client. To achieve this we use a simple technique of *tagging* a single client, similar to the technique described in[5]. Tagging and specifying the states to be considered as part of the response-time, or the specific actions considered part of the measured throughput are discussed in [2] which introduces extended stochastic probes as a means for performance query specification. Automatically modifying the model to suit the performance query is discussed in [3].

Often a modeller must be careful to evaluate both response-time and throughput. Since a low throughput of requests may results in a low average-response time, but only because, for some reason, the demand is low. Similarly with no bound on the number of clients in a waiting state may mean that the throughput of responses is high, but that clients are waiting a long time for their requests to be satisfied.

2.2 Capacity Planning

Complex systems are commonly modelled to provide insight. Either this insight is used to aid the design of the system before it is built or it is used to understand a system already in operation. A complex system may have many components such as different kinds of servers. One aspect of the design of a system is reasoning about the most appropriate configuration, that is the numbers of different kinds of components. Modelling of a system can allow one to speculate about the most appropriate configuration and compare candidate configurations without physically implementing them.

Once efficient comparison of multiple candidate configurations is possible, it makes sense to perform a search for the best or most appropriate configuration. Several techniques exist for searching a space of candidate parameter configurations. Capacity planning has the unique property that usually the parameters in question are all integers since they represent a physical configuration of a system in terms of the populations of component types.

When considering configurations there are generally some trade-offs to consider. Usually some or all of the components over which the designer has control of their populations, have some cost associated with obtaining and running those components. For example a web-service must spend money to obtain and run the physical (or virtual) servers which host the web-service. However the designer will also wish to ensure that the service gives enough performance such that, for example, the response-time is sufficiently low.

We already know how to obtain performance measures from a PEPA model such as response-time as explained above. Suppose we have a model with only one kind

of server and a desired average response time from a given level of service demand or number of users. In this simple scenario it is straightforward to find the optimal configuration. We can simply evaluate the response-time when just a single server is allocated. If this response-time is low enough then this configuration can be reported as the most appropriate configuration knowing that all other configurations will cost more. If not then the model can be modified such that there are two servers and re-evaluate the response-time. In this way a simple brute-force search for the lowest number of servers which satisfies the response-time can be conducted.

In this simple case the first configuration found will be the most appropriate since we know that all other configurations we have tried do not satisfy the desired response-time and all configurations not yet tried will cost more. An obvious improvement would be a binary search. Both techniques would be performing a full search of the configuration space and guaranteed to find the best configuration.

However complex systems are often not as simple. There may be several kinds of servers each with a different cost to obtain and/or operate. Such a linear search for the best candidate solution can still be done if the candidate configurations can be ordered in terms of their operating cost. This may become prohibitively expensive to perform when the number of candidate configurations increases rapidly. Binary search may help in this scenario, but only if the candidate configurations are trivial to order and index in terms of cost.

Furthermore it may also be that the modeller does not have a strict performance threshold to achieve, but simply wants to trade-off good performance against the cost of operation. In this case the modeller might give a notional *cost* to each unit of response-time. Hence the *cost* of a candidate configuration is a combination of the cost of obtaining and operating the components *and* the predicted response-time achievable under that configuration.

In such a scenario we cannot do a linear search and stop at the first candidate configuration which satisfies the performance constraints because there are no strict performance constraints and a better trade-off may exist. One must search over the terrain of the entire search space. In this kind of search it may still be possible to perform a brute-force search and simply evaluate all possible configurations. This is only possible if the number of plausible configurations is low.

When brute-force search is not possible, search techniques exist which avoid evaluating all possible configurations. The work described here utilises such techniques specifically for PEPA models with associated performance measures. In essence then capacity planning is a *search* for the optimal configuration. Search heuristics make it possible to perform a search over a very large space of possible configurations without evaluating all possible configurations. This means that the absolute best configuration may not be found. However, even in such cases it is worth performing the search automatically. Such techniques often fall under the category of *evolutionary computing* of which there is a large literature base, for example [15,4,1].

2.2.1 Cost Functions

The evaluation of a particular configuration involves assessing how appropriately the configuration balances performance and operating cost. These two measures are combined into an overall *cost* of a candidate configuration. Hence, where, $cost_{pm}$

is the cost associated with the performance measure, $cost_{pop}$ is the cost associated with the candidate configuration populations and w_{pm} and w_p are weights, the *cost function* has the general form:

$cost = (w_{pm} \times cost_{pm}) + (w_p \times cost_{pop})$

However not all populations are weighted equally. One possible task is deciding how many of each different kind of server to deploy and it may well be that the different kinds of servers do not cost the same to acquire or operate. For N component kinds, P_i is the population of component kind i and C_i is the cost associated with a single component of kind i. Hence our populations component of the cost function becomes:

$cost_{pop} = \Sigma_1^N C_i \times P_i$

Recall that our performance measure may be associated with the throughput of an action or the average response-time. Additionally in either case we may desire either a high or low value. Hence the cost function must be capable of penalising both a high or a low value for a performance measure. The simple solution is to set a target value for the performance measure and calculate the difference from this value. The modeller sets the target and the direction, so for a performance measure which we wish to search for as low a value as possible we have:

$cost_{pm} = measured - target$

Similarly for a performance measure for where we are searching for as high a value as possible we have:

$cost_{pm} = target - measured$

Note that in both cases this value may become negative. That is perfectly acceptable and there is no reason to avoid negative costs. The search engine will simply search for the configuration which gives the lowest cost, whether that lowest value is negative or not.

However this simple measure assumes that the modeller would penalise performance linearly. Consider the case of measuring average response-time. The modeller may be interested in keeping the average below that which is noticeable by users and hence may be very keen to discard configurations which evaluate to a response-time of 1.0 time units or greater over those which evaluate to a response-time of less than 1.0 time units. However, the modeller may be less concerned about distinguishing between two configurations that evaluate to response-times of 0.1 and 0.2 time units. Preferring instead to distinguish those two configurations more according to the cost of the populations.

Providing non-linear cost functions adds significantly to the complexity of the user-interface provided for the modeller to specify their cost function. Hence we have approximated non-linear cost functions by utilising a penalty for missing the target. Hence our cost function for a performance measure for which we wish to search for the lowest possible value becomes:

$cost_{pm} = (target - measured) + (H(target - measured) \times penalty)$

Where H is the Heaviside function which returns zero when given a negative argument and one otherwise. This corresponds to a situation in which you may wish to adhere to a given service level agreement. For example the service level agreement may state that the average response-time observed by users is no more

than 0.5 time units. Hence we can heavily penalise all configurations which result in the model predicting an average response-time of more than 0.5 time units. This means that for configurations which result in a response-time better than 0.5 time units the search will still prefer better configurations, but that the weights on the performance and population costs can be set sensibly.

For configurations in which the average response-time computed is worse than the target then the penalty is applied. This allows the search to reject such configurations regardless of the population cost.

Recall that capacity planning is a search for the optimal configuration. Avoiding a brute-force search of all configurations is important because the search space of configurations is often infeasibly large. Search heuristics can avoid an exhaustive search by finding good areas of the search space. To enable this it is important that our cost function enables the search to be directed towards good areas in the search space. For this reason there is still a gradient on the performance cost when the target is not reached. That is, a configuration that misses the target response-time by a little, still evalutes to a lower (performance) cost than a configuration which misses the response-time target by a larger amount. This helps the search to move towards configurations in which the target will be met, rather than simply rejecting those that fail to meet the response-time target.

The weights of the overall cost function w_{pm} and w_{pop} allow the user to adjust the importance of reducing the cost associated with the performance measure against the cost associated with the populations of the components. The task that the user has is to set these weights such that neither component of the cost dominates the overall cost.

Unfortunately it is impossible for us to set a useful default here since we cannot know in advance how the populations are costed. In addition the unit used for the rates in the model is undefined. Hence determining how costly each unit of, for example, response-time is, is a task that is necessarily left for the user. To see this, consider that a model which used seconds as the unit can have all rates in the model multiplied by 1000. The steady-state probabilities will not chanage, but the any response-time measure we calculate will now be in the units of milliseconds rather than seconds. This new model is just as valid, but clearly there would be a much smaller real cost associated with a one time unit rise in average response-time. Hence the weights used in this model would need to reflect that.

Strictly speaking the weights are unnecessary since the user could always adjust the weights on the populations, but we include them as a convenience for the user. Typically the user will not run a single capacity planning search, but will run several searches modifying their search parameters accordingly.

We have given a brief description of the cost function considerations in this section. Cost functions can become very complicated. There is a natural tension between providing clear and usable software whilst also covering as many use cases as possible. Ultimately to be entirely generic would require that we allow the user to calculate their own cost function in some full evaluation environment such as a general purpose programming language. This currently remains future work but for now we hope to have provided enough flexibility for common use cases without adding significant bloat and complication to the software and its associated user

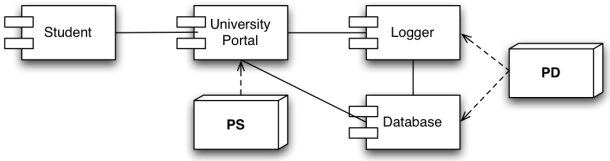


Fig. 1. Deployment diagram of the e-University case study. Solid connectors between components indicate request/reply communication. Dashed lines denote the deployment of services onto processors.

interface.

3 Case Study

Our example scenario concerns a previously studied [12] scenario which formed part of a case study of the SENSORIA project [6, Chapter 2]. It concerns a hypothetical European-wide virtual university in which students study remotely. The part of the case study considered in the above work and in this work concerns the course selection phase where students already matriculated to the university must enrol in specific courses. Although the students only enrol in a few courses per year they all do this at the same time, so it is important that sufficient provision is provided to maintain a responsive service.

The case study is comprised of a number of scenarios; here the scenario of interest is the Course Selection scenario, where students obtain information about the courses available at their education establishment and may enrol in those for which specific requirements are satisfied. Although the overall application is intended to be service-oriented, the scenario investigated here is such that the kinds of services available in the system do not to change over the time frame captured by this model. This reflects the fact that a university's course organisation is likely to be fixed before it is offered to students. Furthermore, minor changes are likely not to affect the system's behaviour significantly. The model will not consider other services which may be deployed in an actual application (e.g. authentication services) because their impact on performance is assumed to be negligible. The scenario also considers a constant population of students to capture a real-world situation where the university's matriculation process is likely to be completed before the application may be accessed.

3.1 The Model

The current authors are indebted to the authors of the above mentioned study [12] for allowing us to include their description of the model here.

The access point to the system is the University Portal, a front-end layer which presents the available services in a coherent way, for example by means of a web interface. There are four services in this model:

Course Browsing allows the user to navigate through the University's course offerings;

Course Selection allows the user to submit a tentative course plan which will be validated against the University's requirements and the student's curriculum;

Student Confirmation will force the student to check relevant personal details;

Course Registration will confirm the student's selection.

These components make use of an infrastructural *Database* service, which in turn maintains an event log through a separated *Logger* service.

The modelling paradigm adopted here captures the behaviour of a typical multi-threaded multi-processor environment used for the deployment and the execution of the application. The *University Portal* instantiates a pool of threads, each thread dealing with a request from a student for one of the services offered. During the processing of the request the thread cannot be acquired by further incoming requests, but when the request is fulfilled the thread clears its current state and becomes available to be acquired again. Analogous multi-threaded behaviour will be given to the *Database* and *Logger* components.

Performance issues may arise from the contention of a limited number of threads by a potentially large population of students. If at some time point all threads are busy, further requests must queue, provoking delays and capacity saturation. This model also proposes another level of contention by explicitly modelling the processors on which the threads execute. Here, delays may occur when many threads try to acquire a limited number of processors available. Furthermore, this may be worsened by running several multi-threaded services on the same multi-processor system, as will be the case in the deployment scenario considered in this model: *University Portal* will run exclusively on multi-processor *PS*, whereas *Logger* and *Database* will share multi-processor *PD* (see Figure 1).

3.1.1 General Modelling Patterns

Processing a request involves some computation on the processor on which the service is deployed. Such a computation in the PEPA model is associated with an activity *(type, rate)*, where *type* uniquely identifies the activity and *rate* denotes the average execution demand on the processor (i.e. $1/rate$ time units). A single processing unit may be modelled using a two-state sequential component. One state enables an *acq* activity to acquire exclusive access to the resource, while the other state enables all the activities deployed on the processor. Letting n be the number of distinct activities, the following pattern is used for a processor:

$$\begin{aligned} Processor_1 &= (acq, r_{acq}).Processor_2 \\ Processor_2 &= (type_1, r_1).Processor_1 \\ &\quad + (type_2, r_2).Processor_1 \\ &\quad + \ldots \\ &\quad + (type_n, r_n).Processor_1 \end{aligned} \tag{1}$$

Communication in this model is synchronous and is modelled by a sequence of two activities in the form $(req_{from,to}, rreq).(reply_{from,to}, rrep)$ where the subscript from denotes the service from which the request originates and to indicates the service required. A recurring situation is a form of blocking experienced by the service invoking an external request. Let A and B model two distinct interacting services, for example,

$$A = (req_{A,B}, r_{reqA}).(reply_{A,B}, r_{repA}).A$$
$$B = (req_{A,B}, r_{reqB}).(execute, r).(reply_{A,B}, r_{repB}).B$$

(2)

The communication between A and B will be expressed by means of the cooperation operator $A \underset{L}{\bowtie} B$ where, $L = \{req_{A,B}, reply_{A,B}\}$.

According to the operational semantics, A and B may initially progress by executing $req_{A,B}$, subsequently behaving as the process $(reply_{A,B}, r_{repA}).A \underset{L}{\bowtie} (execute, r).(reply_{A,B}, r_{repB}).B$.

Now, although the left-hand side of the cooperation enables $reply_{A,B}$, the activity is not offered by the right-hand side, thus making the left-hand side effectively blocked until *execute* terminates (i.e., after an average duration of $1/r$ time units). These basic modelling patterns will be used extensively in this case study, as discussed next.

3.1.2 University Portal

A single thread of execution for the application layer University *Portal* is implemented as a sequential component which initially accepts requests for any of the services provided:

$$\begin{aligned} Portal = {} & (req_{student,browse}, v).Browse \\ & + (req_{student,select}, v).Select \\ & + (req_{student,confirm}, v).Confirm \\ & + (req_{student,register}, v).Register \end{aligned}$$

(3)

The rate v will be used throughout this model in all the *request*/*reply* activities. In the following, the action type acq_{ps} is used to obtain exclusive access to processor PS .

Course Browsing is implemented as a service which maintains an internal cache. When a request is to be processed, the cache query takes $1/r_{cache}$ time units on average, and is successful with probability 0.95, after which the retrieved data is processed at rate r_{int}. Upon a cache miss, the information is retrieved by the *Database* service, and is subsequently processed at rate r_{ext}:

$$\begin{aligned} Browse = {} & (acq_{ps}, v).Cache \\ Cache = {} & (cache, 0.95 \times r_{cache}).Internal \\ & + (cache, 0.05 \times r_{cache}).External \\ Internal = {} & (acq_{ps}, v).(internal, r_{int}).BrowseRep \\ External = {} & (req_{external,read}, v).(reply_{external,read}, v). \\ & (acq_{ps}, v).(external, r_{ext}).BrowseRep \\ BrowseRep = {} & (reply_{student,browse}, v).Portal \end{aligned}$$

(4)

Course Selection comprises four basic activities. An initial set-up task initialises the necessary data required for further processing ($rater_{prep}$). Then, two activities are executed in parallel, and are concerned with validating the selection against

the university requirements ($rater_{uni}$) and the student's curriculum ($rater_{curr}$), respectively. Finally, the outcome of this validation is prepared to be shown to the student ($rater_{disp}$). The relative ordering of execution is maintained by considering three distinct sequential components. The first component prepares the data, then forks the two validating processes, waits for their completion, and finally displays the results:

$$\begin{aligned} Select &= (acq_{ps}, v).(prepare, r_{prep}).ForkPrepare \\ ForkPrepare &= (fork, v).JoinPrepare \\ JoinPrepare &= (join, v).Display \\ Display &= (acq_{ps}, v).(display, r_{disp}).SelectRep \\ SelectRep &= (reply_{student,select}, v).Portal \end{aligned} \tag{5}$$

The two validating processes are guarded by the fork/join barrier as follows:

$$\begin{aligned} ValUni &= (fork, v).(acq_{ps}, v).(validate_{uni}, r_{uni}).(join, v).ValUni \\ ValCur &= (fork, v).(acq_{ps}, v).(validate_{cur}, r_{cur}).(join, v).ValCur \end{aligned} \tag{6}$$

These components will be arranged as follows in order to obtain a three-way synchronisation:

$$Select \underset{fork,join}{\bowtie} ValUni \underset{fork,join}{\bowtie} ValCur \tag{7}$$

Student Confirmation is represented in the PEPA model as an activity performed at rate r_{con}. The service uses *Logger* to register the event:

$$\begin{aligned} Confirm &= (acq_{ps}, v).(confirm, r_con).LogStudent \\ LogStudent &= (req_{confirm,log}, v).(reply_{confirm,log}, v).ReplyConfirm \\ ReplyConfirm &= (reply_{student,confirm}, v).Portal \end{aligned} \tag{8}$$

Finally, *Course Registration* performs some local computation, at rate r_{reg}, and then contacts *Database* to store the information:

$$\begin{aligned} Register &= (acq_{ps}, v).(register, r_{reg}).Store \\ Store &= (req_{register,write}, v).(reply_{register,write}, v).ReplyRegister \\ ReplyRegister &= (reply_{student,register}, v).Portal \end{aligned} \tag{9}$$

The general pattern 1 is applied to processor PS as follows:

$$\begin{aligned} PS_1 &= (acq_{ps}, v).PS_2 \\ PS_2 &= (cache, r_cache).PS_1 + (internal, r_int).PS_1 \\ &\quad + (external, r_{ext}).PS_1 + (prepare, r_{prep}).PS_1 \\ &\quad + (display, r_{disp}).PS_1 + (validate_{uni}, r_{uni}).PS_1 \\ &\quad + (validate_{cur}, r_{cur}).PS_1 + (confirm, r_{con}).PS_1 \\ &\quad + (register, r_{reg}).PS_1 \end{aligned} \tag{10}$$

3.1.3 Database

This service exposes two functions for reading and writing data. Reading is a purely local computation, whereas writing additionally uses the Logger service. In this model, *Database* is only accessed by the university portal in states *External* and *Store* in equations 4 and 9, respectively. Let PD denote the processor on which *Database* is deployed, acquired through action acq_{pd} . Similarly to University *Portal*, a single thread of execution for *Database* is:

$$\begin{aligned} Database &= (req_{external,read}, v).Read + (req_{register,write}, v).Write \\ Read &= (acq_{pd}, v).(read, r_{read}).ReadReply \\ ReadReply &= (reply_{external,read}, v).Database \\ Write &= (acq_{pd}, v).(write, r_{write}).LogWrite \\ LogWrite &= (req_{database,log}, v).(reply_{database,log}, v).WriteReply \\ WriteReply &= (reply_{register,write}, v).Database \end{aligned} \tag{11}$$

3.1.4 Logger

This service accepts requests from *Student Confirmation* and *Database*, as described in equations 8 and 11, respectively. It is deployed on the same processor as *Database*, i.e., processor PD. Thus, one thread execution may be modelled as follows:

$$\begin{aligned} Logger &= (req_{confirm,log}, v).LogConfirm + (req_{database,log}, v).LogDatabase \\ LogConfirm &= (acq_{pd}, v).(log_{conf}, r_{lgc}).ReplyConfirm \\ ReplyConfirm &= (reply_{confirm,log}, v).Logger \\ LogDatabase &= (acq_{pd}, v).(log_{db}, r_{lgd}).ReplyDatabase \\ ReplyDatabase &= (reply_{database,log}, v).Logger \end{aligned} \tag{12}$$

Taking together 11 and 12 it is possible to write the sequential component that models the processor PD:

$$\begin{aligned} PD_1 &= (acq_{pd}, v).PD_2 \\ PD_2 &= (read, r_{read}).PD_1 + (write, r_{write}).PD_1 \\ &\quad + (log_{conf}, r_{lgc}).PD_1 + (log_{db}, r_{lgd}).PD_1 \end{aligned} \tag{13}$$

3.1.5 Student Workload

A student is modelled as a sequential component which interacts with the university portal and accesses all of the services available. The behaviour is cyclic and the student interposes some think time between successive requests. This results in a closed-workload type of behaviour which is typical of many performance studies:

$$\begin{aligned} StdThink &= (think, r_{think}).StdBrowse \\ StdBrowse &= (req_{student,browse}, v).(reply_{student,browse}, v).StdSelect \\ StdSelect &= (req_{student,select}, v).(reply_{student,select}, v).StdConfirm \\ StdConfirm &= (req_{student,confirm}, v).(reply_{student,confirm}, v).StdRegister \end{aligned}$$

$$StdRegister = (req_{student,register}, v).(reply_{student,register}, v).StdThink \tag{14}$$

3.1.6 System Equation

The multiplicity of threads and processors is captured in the system equation, in which all the sequential components illustrated above are composed with suitable cooperation operators to enforce synchronisation between shared actions. The complete system equation for this model is:

$$\begin{array}{c} StdThink[N_S] \\ \bowtie_{*} \\ \Big((Portal[N_P] \bowtie_{M_1} ValUni[N_P] \bowtie_{M1} ValCur[N_P]) \\ \bowtie_{M_2} \\ Database[N_D] \bowtie_{M_3} Logger[N_L]\Big) \\ \bowtie_{*} \\ (PS1[N_{PS}] \bowtie_{\emptyset} PD1[N_{PD}]) \end{array}$$

where:

$$M_1 = \{fork, join\}$$
$$M_2 = \{req_{external,read}, reply_{external,read}, req_{register,write}, reply_{register,write}\}$$
$$M_3 = \{req_{confirm,log}, reply_{confirm,log}, req_{database,log}, reply_{database,log}\}$$

It is worth pointing out that the separate validating threads ValUni and ValCur inherit the multiplicity levels of the thread Portal which spawns them.

3.2 Performance Measure

Given this model we wish to measure and optimise for the performance observed by a typical student. We are therefore interested in calculating the average response-time for student requests. From the definitions in 14 the students in the $StdThink$ state are not attempting to make use of the system. We therefore calculate the average time it takes from the moment a student actively uses the system by moving into the $StdBrowse$ state until the student returns to the $StdThink$ state.

For the default configuration of the model we obtain the results 22.95 time units for the average response-time. We now wish to optimise the configuration of the system to obtain satisfactory performance whilst spending as little as possible on the components.

3.3 Capacity Planning

The populations that a designer of the hypothetical e-university service may be able to control are those of the components: $Database$, $Logger$, PD, $Portal$, PS, $ValCur$ and $ValUni$, we do not expect the service to be able to control the average number of students accessing the service simultaneously. Therefore the modeller

assumes some fixed level of demand by fixing the initial population of *StdThink*, in this case to 600.

We are interesting in optimising for the response-time performance measure described above. In addition we would like to keep the cost of the system as low as possible.

We have limited data to allow us to obtain realistic rates for some of the activities in the model. Since the meaning of a unit of time in a PEPA model is unspecified we need only be concerned that the rates are of realistic magnitudes relative to each other.

In addition, as discussed in Section 2.2 we are able to place a threshold average response-time above which there is a heavy cost function penalty, to approximate a non-linear performance measure cost. In this study we somewhat arbitrarily specify this to be 15, but this is no more arbitrary than the unspecified unit of time used in the PEPA model itself. The 15 in question comes from the fact that the hand-optimised version of the model for 600 users reported in [12] (mentioned above as the source of our case study's PEPA model), is 15.248.

Without specialist knowledge it is difficult to weigh the importance of the reducing the average response-time against the importance of reducing the cost of the system. So we have also somewhat arbitrarily set the cost function weights w_{pm} (the weight of the performance measure component) and w_{pop} (the weight of the populations components).

Hence we have two arbitrary pieces of information included in our cost function, that is the penalty threshold for the performance measure and the ratio of weightings for the performance measure and population components of the cost function. However both such pieces of information would be available to a real-world modeller utilising our capacity planning extension.

3.4 Results

The previously mentioned work which introduced our case study [12] utilised the ODE response-time evaluation to find a configuration of the model with a low response-time for the case in which there are 600 students. Table 1 shows the populations for configurations found, using three search methods. The first is the hand optimised configuration reported in the above referenced work, the second is our heuristic-based search and the last is for a brute force search, in which the search space was limited to a small area around the optimal configuration found by the heuristic based search.

In the hand-optimised case for the original publication the authors held three of the server populations to be equal to each other but not fixed. So in their case $N_{VC} = N_{VU} = N_P$, this was not a restriction that we imposed on our capacity planning search. Such a restriction is indeed easily imposed, but we wished to allow the search as much flexibility as possible.

Our software has found a model that has a significantly lower average response-time, with a response-time of 9.430 compared to 15.248. This would not be impressive if the cost of the server configuration were not cheaper. The population of every server component kind cannot be lower than the hand-optimised version

Optimisation	N_D	N_L	N_{PD}	N_P	N_{PS}	N_{VC}	N_{VU}	Total	Avg Res
Hand	80	80	40	80	40	80	80	480	15.248
Search	28	21	28	81	51	17	43	269	9.430
Brute	26	19	27	77	49	15	36	249	10.242

Table 1
Optimal configurations found for three search techniques. The first is manual optimisation, the second is our heuristic-based search and the last is a brute-force search around in a limited region around the optimal configuration found by the heuristic-based search. Both the heuristic and brute-force search find configurations that have a lower total population *and* a lower average response-time. The heuristic-based search finds the lowest average response time of the three whilst the brute-force search finds a lower total population.

otherwise we would have at best the same average response-time. However, in our case we obtain a model that has a total population of server components that is significantly less than that of the hand-optimised version. A total server component population of 269 against 480 for the hand-optimised version.

There are only two server components for which the hand-optimised version has a lower population than the configuration found by our automatic search. These are, the number of *Portal* componets N_P and the number of PS components N_{PS}. In the case of the *Portal* component there is only 1 more of them. In the case of the multi-processors upon which the *Portal* threads run, the difference is more significant $50 > 41$ by more than 25%.

As we have stated it is difficult to provide realistic costs for each of the server components in such a hypothetical scenario. However our automatically optimised model is a significant improvement on the hand-optimised version unless *Portal* components or the multiprocessor systems that they run on are significantly more expensive than, for example, the multi-processors which execute the *Database* and *Logger* components.

We can assert that given our cost values for each of the components, the automatic search was able to find a configuration that had a significantly improved average response-time whilst simultaneously reducing the total cost of the server components.

The entire search took 4366 seconds, or just under 73 minutes. In doing so it solved 2426 models. We can say that each model therefore took approximately 1.8 seconds to solve on average. Each model may take a different time to solve because the rates affect how quickly the model is solved. In addition there is some time spent performing the search algorithm logic, but this will serve us as an approximation. All of the computations described here were performed on a standard desktop computer. In addition it is the relative, rather than absolute times that we are mostly concerned with.

3.5 Brute Force Comparison

As described above the alternative to performing a stochastic search over the space of potential configurations is to perform a brute-force search evaluating all possible configurations.

The time taken to solve the set of ODEs generated from the PEPA model de-

pends on the configuration, but is generally comparable across the configurations. As described above the capacity planning search is not instantaneous, but took around an hour. The naïve approach to a brute-force search would evaluate all possible configurations within our initial constraints. This would have meant solving 107374182400000000 distinct possible configurations and taken approximately 6399581450 years.

A modeller could of course be a little more clever about the ranges set on population levels to reduce the search space. Whenever one does this there is a trade-off as you are trading-off the possibility that a better solution exists outside your narrower ranges against the advantage of your search performing faster.

However, the best solution had a highest population of 81 and a lowest population of 17. Even if we set all ranges to be from this lowest value 17 to the highest value 81, which would require insight into the search space that the modeller does not have, then the search space still has $(81 - 17)^7 = 4398046511104$ possible candidate configurations. Hence searching the entire space with a brute-force approach will still take 2443359172835 seconds or approximately 77478 years.

However, one could use the driven search to provide a suitable search area in which to perform an exhaustive search. To perform our brute-force search in a reasonable amount of time we set the ranges for each configurable population to a range around the value that we have found from the capacity planning search. To further reduce this we held the number of *Database* components constant at 26 (this was in error it should have been 28). As a result our brute-force search had a more manageable number of configurations to solve: $125000 = (25 - 15) * (45 - 35) * (85 - 75) * (31 - 26) * (54 - 49) * (24 - 19)$. This too approximately 65 hours to solve. This best solution being shown in the table in Section 3.4.

So note, because of our error in setting the *Database* component population the brute-force search never found as good a configuration as the capacity planning search. This is the main problem with a brute-force search as opposed to a search heuristic, because there are so many configurations to solve we must reduce the available flexibility meaning that the modeller must already have significant insight into their model. Capacity planning can either be used on its own, or to find a good set of ranges in which to perform an exhaustive search.

3.5.1 Search Space

Figure 2 gives some idea of the search space of configurations. Each graph pair of graphs concerns one configurable component (in the interests of brevity we have included only two representative components, *Portal* and *Logger*). The left graph of each pair displays the results from the driven capacity planning search and the right displays the results from the brute-force search.

Each plotted dot represents a candidate configuration, the x-axis position determines the population of the candidate configuration for the particular server component kind depicted in that specific graph. The y-axis position determines the value of the cost function for that configuration. Recall that the cost function considers both the populations of all the configurable server components and the resulting average response-time.

The x-axis range on the brute-force search results are much narrower, because the

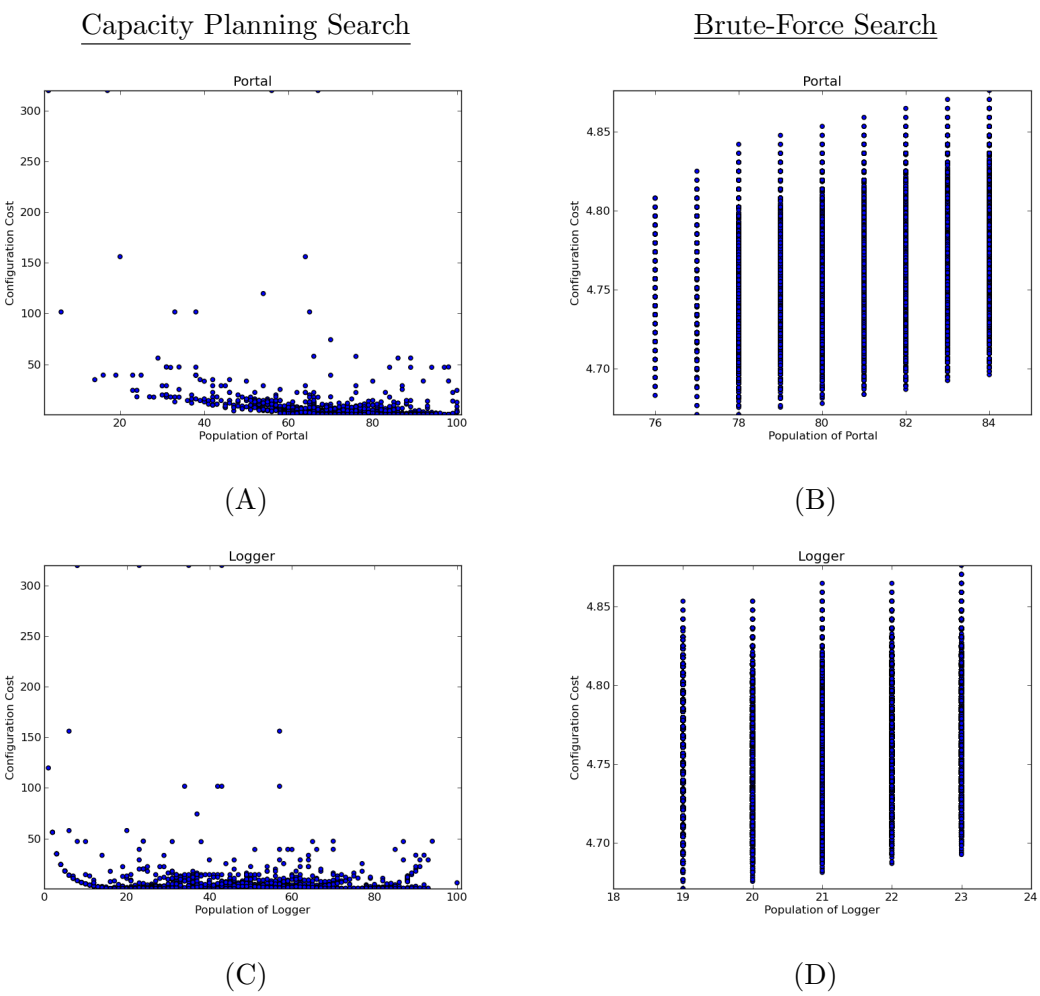


Fig. 2. Scatter plots showing the results for a selection of the configurable server components. The left-hand graphs depict the results for the capacity planning search whilst the right hand graphs depict the results for the brute-force search. The x-axis ranges are much smaller for the brute-force search since it is infeasible to evaluate all configurations when the range of possible values is large.

brute-force search was centred on a narrow range around each optimal configuration found by the capacity planning search. This is because it is infeasible to do an exhaustive search for larger ranges.

The capacity planning graphs exhibit a lower left corner slope. This indicates that for each of these components there is a lower-bound on the population such that populations below this result in too high an average response-time, regardless of the rest of the configuration.

Each graph additionally demonstrates that one cannot optimise for each component kind independently. For each population of each component kind a wide range of costs are possible. Hence one must optimise for *all* of the configurable component populations simultaneously, because the population of one affects both the sensitivity and optimal value of another.

4 Future Work

Although we think that the user has been given much flexibility in the configuration of their cost function we realise that there are surely scenarios which call for some cost function that cannot be expressed using our configuration interface. A more general solution would be to allow the modeller to express their own cost function in a general purpose programming language such as Java which is used in our implementation.

To provide this, some interface to the results and the model parameters would be required. This would also place something of a burden on the modeller so we would be keen to retain a simple gui-based configuration scheme that may be used as a first exploration of the configuration space, and/or by novice users.

Recall that our practice of having the user specify a target performance measure value is an approximation to a non-linear cost function. We think this is a good trade-off of complexity, easy of use and power of expression. However, we continue to investigate other possibilities.

Finally throughout this paper we have assumed that the modeller can either make a good guess to the level of expected demand or is prepared to over-estimate it. A further possibility is to perform multiple capacity planning searches assuming different levels of demand.

We could perform multiple capacity planning searches for different levels of demand automatically. Furthermore we may see adaptability to different levels of demand as a particularly good thing to have. For example some services can operate at different levels, in the most obvious case by simply turning servers on or off. Currently, whilst we may find a configuration which is particularly good for a particular level of demand it may not be very adaptable. Hence we continue to investigate ways in which we may reward configurations that are adaptable.

In the meantime we provide methods for the modeller to examine some of the configurations that have been found mid-search, but perhaps did not have the globally best cost, because adaptability is not accounted for in the cost function. This provides a further reason that it is particularly useful for the modeller to be able to examine elements of the search rather than simply the best configuration found.

5 Conclusion

When modelling service based systems such as the system modelled in our case study the modeller is unlikely to have great control over the level of demand. Therefore the system designer must be sure to provision enough service to satisfy a realistic level of demand.

Most realistic levels of demand can be satisified with enough service provision, but generally there is some significant cost to providing that level of service. If not then one need do little modelling but simply provision plenty of service component.

Assuming that there is some significant cost we would like to know how best to provide the required level of service. Even further we may not know the level of service we demand but we have some idea of how to trade-off the level of performance against the cost of the provision.

However, knowing this is not enough for many kinds of services. These are services in which there are more two kinds of components that need to be deployed to provide the whole service. In these kinds of scenarios there are simply too many plausible configurations of the service to try them all. Furthermore it is rarely obvious what the most efficient configuration is, or even how to improve on the current one.

Hence an automatic search through the configuration space can provide excellent insight for the modeller. We are of the opinion that not only the end result of such a search but many of the configurations and their associated costs found mid-search may be of interest to the modeller.

Performing such a search is a non-trivial task. A user-friendly GUI based tool which not only performs the search itself but guides the user through the configuration of the search is a significant help to the modeller. We have presented such a software tool in this paper.

The trade-off is that the developers of such a tool must consider the ways in which the modeller may wish to evaluate the efficacy of a particular model configuration. We think that so far we have a powerfully expressive method of configuration but we continue to investigate methods to be more expressive as well as more intuitive.

Finally we wish to claim that capacity planning, or more generally a heuristic search, is a useful addition to any modelling software. It is difficult to provide the correct interface, but this is ultimately worth the effort. The capacity planning extension to the PEPA Eclipse Plug-in project [16] will be available at the forthcoming release due in September 2014.

References

[1] Bäck, T. and H.-P. Schwefel, *An overview of evolutionary algorithms for parameter optimization*, Evol. Comput. **1** (1993), pp. 1–23.
URL http://dx.doi.org/10.1162/evco.1993.1.1.1

[2] Clark, A. and S. Gilmore, *State-aware performance analysis with eXtended Stochastic Probes*, in: N. Thomas and C. Juiz, editors, *Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008)*, LNCS **5261** (2008), pp. 125–140.

[3] Clark, A. and S. Gilmore, *Transformations in PEPA Models and Stochastic Probe Placement*, in: K. Djemame, editor, *Proceedings of the Twenty-Fifth UK Performance Engineering Workshop*, Leeds University, 2009, pp. 1–16.

[4] Deb, K. and D. Kalyanmoy, "Multi-Objective Optimization Using Evolutionary Algorithms," John Wiley & Sons, Inc., New York, NY, USA, 2001.

[5] Dingle, N. J. and W. J. Knottenbelt, *Automated Customer-Centric Performance Analysis of Generalised Stochastic Petri Nets Using Tagged Tokens*, in: *Third International Workshop on Practical Applications of Stochastic Modelling (PASM'08), Palma de Mallorca, Spain*, 2008.
URL http://pubs.doc.ic.ac.uk/pasm08-tagged/

[6] Elgner, J., S. Gnesi, N. Koch and P. Mayer, *Introduction to the sensoria case studies*, in: M. Wirsing and M. Hölzl, editors, *Rigorous Software Engineering for Service-oriented Systems*, Springer-Verlag, Berlin, Heidelberg, 2011 pp. 26–34.
URL http://dl.acm.org/citation.cfm?id=2043021.2043025

[7] Hillston, J., *The nature of synchronisation*, in: U. Herzog and M. Rettelbach, editors, *Proceedings of the Second International Workshop on Process Algebras and Performance Modelling*, Erlangen, 1994, pp. 51–70.

[8] Hillston, J., "A Compositional Approach to Performance Modelling," Cambridge University Press, 1996.

[9] Hillston, J., *Fluid flow approximation of PEPA models*, in: *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems* (2005), pp. 33–43.

[10] Hillston, J., *Process algebras for quantitative analysis*, in: *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)* (2005), pp. 239–248.

[11] Hillston, J., *Tuning systems: From composition to performance*, The Computer Journal **48** (2005), pp. 385–400, the Needham Lecture paper.

[12] Hillston, J., M. Tribastone and S. Gilmore, *Stochastic process algebras: From individuals to populations*, Comput. J. **55** (2012), pp. 866–881.

[13] Hillston, J. and C. D. Williams, *Capacity planning for PEPA models*, in: A. Horvath and K. Wolter, editors, *Proceedings of the 11th European Performance Engineering Workshop (EPEW 2014)*, LNCS **8721** (2014), to appear in.

[14] Little, J. D. C., *A proof of the queueing formula* $l = \lambda w$, Operations Research **9** (1961), pp. 380–387.

[15] Simon, D., "Evolutionary Optimization Algorithms," 2013.

[16] Tribastone, M., *The PEPA Plug-in Project*, in: M. Harchol-Balter, M. Kwiatkowska and M. Telek, editors, *Proceedings of the 4th International Conference on the Quantitative Evaluation of SysTems (QEST)* (2007), pp. 53–54.

[17] Tribastone, M., "Scalable Analysis of Stochastic Process Algebra Models," Ph.D. thesis, School of Informatics, The University of Edinburgh (2010).
URL https://dl.dropboxusercontent.com/u/13100903/papers/phd-thesis.pdf

Performance Modelling and Evaluation of Secure Dynamic Group Communication Systems in RANETs

Demetres D. Kouvatsos, Esmaeil Habibzadeh (University of Bradford, UK),
Guzlan M.A. Miskeen (University of Sebha, Libya)

Security mechanisms, such as encryption and authentication protocols, require extra computing resources and therefore, have an adverse effect upon the performance of communication networks. In this context, an investigation is undertaken, based on combined performance and security metrics (CPSMs), using hybrid stochastic Petri nets (SPNs) and queueing networks (QNs) towards the modelling and evaluation of performance vs. security trade-offs in robotic mobile wireless ad hoc networks (RANETs). Specifically, each robotic node is represented by open gated (G) coloured generalised stochastic PN (CGSPN) and QN (G-CGSPN-QN) hybrid models capturing, respectively, <mobility, encryption processing and security control> and <'intra' robot component-to-component communication and 'inter' robot-to-robot transmission>. The so called HPQNS [5] framework is employed as an effective tool to carry out numerical experiments in order to assess the credibility of the proposed G-CGSPN-QN model and associated CPSMs. Finally, future research directions at the level of a RANET are included.

Keywords Mobile Ad hoc NETworks (MANETs), Robotic mobile wireless Ad hoc NETworks (RANETs), Queueing Networks (QNs), Petri Nets (PNs) , CPSMs, HPQNS

Summary

1. Introduction

The performance of Robotic mobile wireless Ad hoc NETworks (RANETs) is of vital importance towards enhancing quality-of-service (QoS) in a wide range of applications [4]. However, it is known that RANETs have high security vulnerability due to the open medium, the dynamically changing network topology and the lack of infrastructure [3]. Thus, performance and security are mutually critical and should jointly be taken into consideration during the design, development and upgrading of RANETs.

On the other hand, security mechanisms such as en/decryption and security protocols require an extensive amount of computing power that would have an adverse impact on the performance of individual robots as well as the whole network. In this context, the applicability of Petri nets (PNs) and Queueing networks (QNs) hybrid modelling paradigms

for robotic nodes is explored and an open Gated Coloured Generalised Stochastic PN and QN (G-CGSPN-QN) model is proposed for each robotic node capturing intra/inter-robot communications. This model is comprised of modular interacting CGSPN and QN sub-models, which represent robots' mobility, and security processing &control and transmission delays, respectively.

Moreover, an investigation is undertaken involving security related performance metrics in terms of nodal throughput and utilisation of the channels ('servers') on each individual robotic node. In particular, the impact of security attacks and re-keying overheads on the performance metrics is examined and optimal trade-offs between performance and security with respect to various system security levels are identified.

2. Technical Contributions

Fig. 1 shows the proposed hybrid model for each robotic node, which comprises from modular hybrid CGSPN and QN sub-models. The gated mobility sub-model captures the robot's dynamic join/leave from the RANET where, as a result, it alternatively becomes available or goes out of reach. The en/decryption sub-model represents the amount of delay each packet would incur as the packets are now required to first go through en/decryption processes to be secured against outsider attacks. To deal with insider attacks, Intrusion Detection Systems (IDS) are usually developed, which are to detect compromised nodes and remove them from the network [3]. To reflect the behaviour of an IDS and its impact upon the system performance, a security control sub-model is introduced; this leads the robot to alternate between certain security states and effectively block the encryption process in case of security attacks. The latter is ensured by the use of an inhibitor link connecting P3 to T1 (Fig. 1). These sub-models are designed and implemented using an extended version of CGSPN, where places assume queues of finite capacity with Head-Of-the-Line (HOL) scheduling strategy. The final stage, however, includes the transmission sub-model, which is purely designed in QN of an arbitrary topology and finite capacity queues; this sub-model reflects any further process required to feed the packets back inside the robot or forward them to another robot in the RANET. These hybrid sub-models together with the routing matrices govern the intra-inter robotic communications.

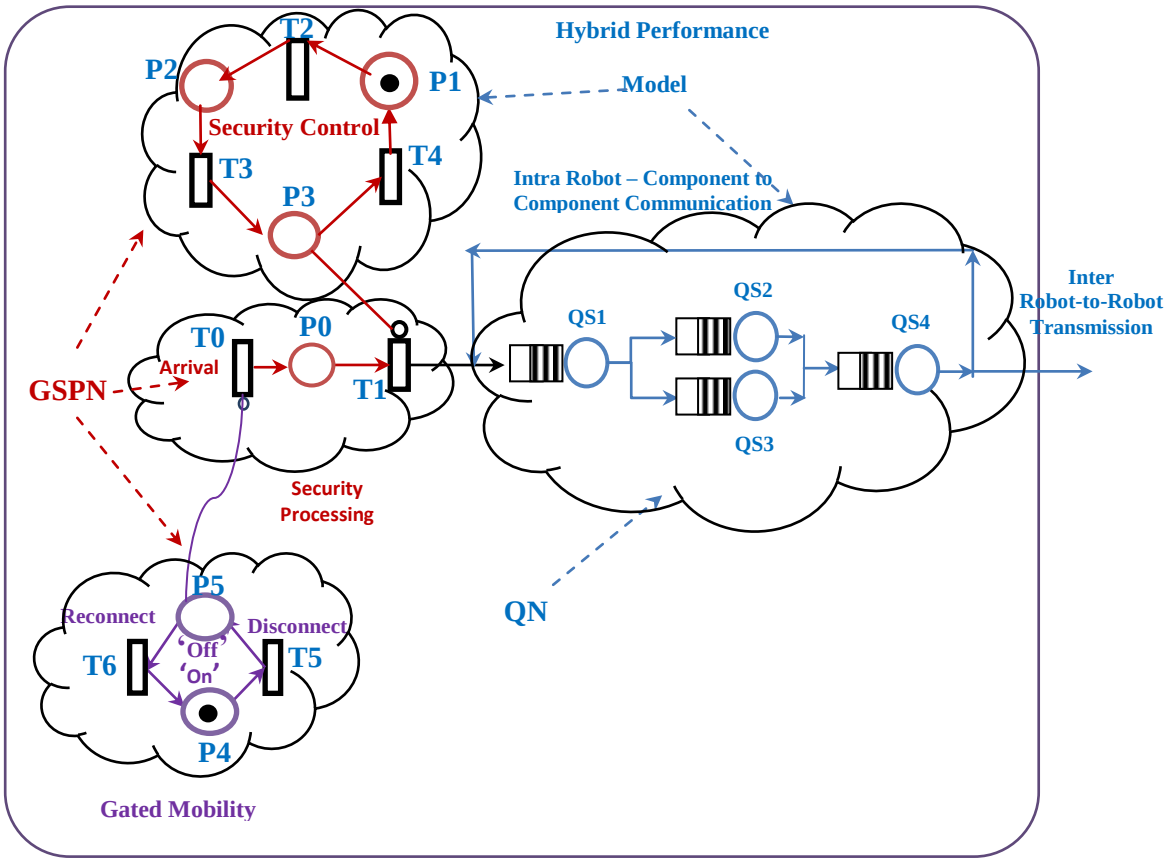


Fig. 1 An open G-CGSPN-QN hybrid model of a RANET node

To study performance vs security trade-offs, two different CPSMs are considered. The CPSM1 combines the performance metric of 'channel utilisation' associated with component QS4 (c.f., Fig. 1)) plus the security metric of 'probability of system being secure'. The CPSM2 is chosen to combine another set of performance and security metrics, namely the packet loss probability (PLP) at the encryption queue of Fig. 1 and the probability of system being insecure, respectively.

The HPQNS tool [5] is used to carry out the numerical experiments and assess the credibility of the proposed G-CGSPN-QN hybrid model for each robotic node. In particular, simulation results are generated for both CPSM1 and CPSM2, which are illustrated in Figs. 2 and 3 below. These figures demonstrate the effects of different security key lengths on performance and security of the robot. It is clear that CPSM1 and CPSM2 both give the same optimal encryption time. It is worth mentioning that the key length has a direct impact on the encryption time [2], which makes them interchangeable.

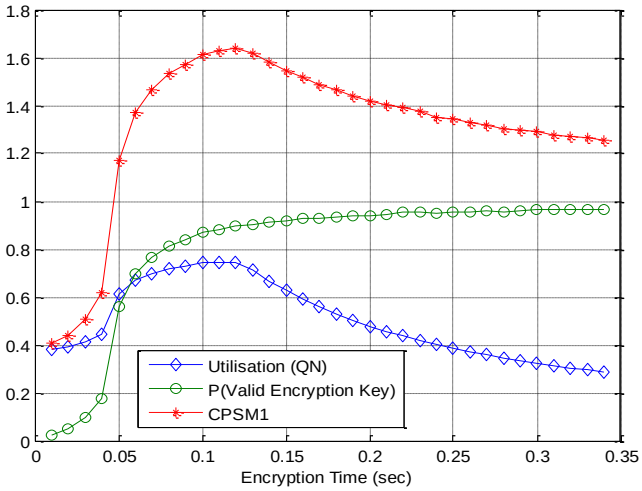


Fig. 2 CPSM1: Utilisation + P(Secure)

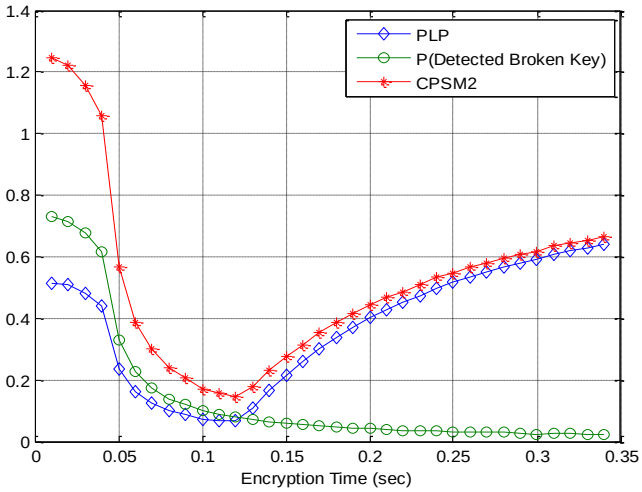


Fig. 3 CPSM2: PLP + P(Insecure)

3. Future Work

Further research work at the RANET level (c.f., Fig. 4)) may focus on

i) An extension to improve the mobility sub-model and provide an effective link with the security parts of the G-CGSPN-QNs hybrid model, where each join/leave process should lead to a renewal/regeneration of the security key.

ii) The characterisation of the IDSs with respect to two parameters, false negative probability (p_{fn}) and false positive probability (p_{fp}); the former may lead to information leakage and the latter may have further adverse impact on the performance.

iii) The energy consumption in RANETs, where the energy sub-model may capture either battery consumption or battery life (remaining energy levels); in this context, the trade-off

between energy (as a performance metric) and security could be examined in order to gain new insights into the efficient energy use in RANETs whilst maintaining the security in adequately high levels.

iv) Secure Dynamic Group Communication Systems (SDGCSs), which may be established by robots of extended G-CGSPN-QN hybrid models (Fig. 4), where IDSs are integrated with rekeying processes (RPs) to deal with insider and outsider attacks, respectively [3]. Investigation of trade-offs between performance and security across the SDGCS with respect to a number of CPSMs would be of significant value. In addition, the pros and cons of individual rekeying and batch rekeying as well as host-based and voting-based IDSs [3] could be analysed in greater details.

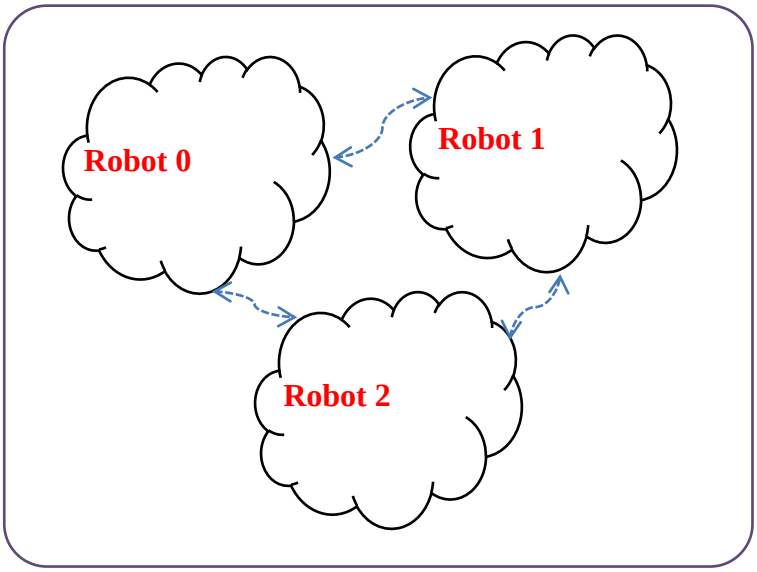


Fig. 4 A SDGCS with three robots exploiting G-CGSPN-QN hybrid models

v) Different probability distributions, where generalised exponential (GE) [1] would particularly be useful to find out how and why higher SCVs might influence the optimal points in performance vs security trade-offs.

vi) Additional CPSMs such as

- {P(Secure) + E(Throughput)} quite similar to {P(Secure) + E(Utilisation)}
- {P(Not Secure) + E(Response Time)}
- {P(Secure) + E(Remaining Energy level)} or {P(Not Secure) + E(Energy Consumption)}

References

1. Miskeen G., Kouvatsos D. D., Habibzadeh E. (2013). An Exposition of Performance–Security Tradeoffs in RANETs Based on Quantitative Network Models. Wireless Personal Communications, Springer. vol. 70, no. 3, pp 1121-1146

2. Wolter, K., & Reinecke, P. (2010). Performance and security tradeoff. In A. In. Aldini, Bernardo, M., Di Pierro, A. & Wiklicky, H. (Ed.), *Formal Methods for Quantitative Aspects of Programming Languages* (pp. 135-167). Heidelberg: Springer Berlin.

3. Cho, J. H., Chen, R., & Feng, P. G. Performance analysis of dynamic group communication systems with intrusion detection integrated with batch rekeying in mobile ad hoc networks. In *22nd International Conference on Advanced Information Networking and Applications-Workshops, AINAW 2008, Okinawa, 25-28 March 2008 2008* (pp. 644-649): IEEE

4. Zorkadis, V. (1994). Security versus performance requirements in data communication systems. *Third European Symposium on Research in Computer Security Proceedings, Computer Security - ESORICS*, 19-30.

5. Habibzadeh, E. & Kouvatsos, D. & Miskeen, G. (2014). A Hybrid Simulation Framework for the Analysis of Petri Nets and Queueing Networks. UK Performance Engineering Workshop, Newcastle University.

6. Balbo, G., Bruell, S. C., & Ghanta, S. (1988). Combining queueing networks and generalized stochastic petri nets for the solution of complex models of system behavior. *IEEE Transactions on Computers, 37*(10), 1251-1268.

7. Bause, F. Queueing petri nets-a formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of 5th International Workshop on Petri Nets and Performance Models, 1993* (pp. 14-23): IEEE

8. Becker, M., & Szczerbicka, H. (1999). PNiQ: integration of queuing networks in generalised stochastic petri nets. *IEE Proceedings, Software, 146*(1), 27-32, doi:10.1049/ip-sen:19990153.

9. Becker, M., & Szczerbicka, H. Integration of multi-class queueing networks in generalized stochastic petri nets. In *IEEE International Conference on Systems, Man, and Cybernetics, 2001* (Vol. 2, pp. 1137-1142): IEEE. doi:10.1109/icsmc.2001.1309629.

10. Szczerbicka, H. (1992). A combined queueing network and stochastic Petri net approach for evaluating the performability of fault-tolerant computer systems. *Performance Evaluation, 14*, 217-226, doi:10.1016/0166-5316(92)90005-2.

11. Bause, F. (1993). "QN+ PN= QPN"-combining queueing networks and petri nets. (pp. 1-15). Germany: University of Dortmund.

12. Miskeen, G., Kouvatsos, D. D., & Akhlaq, M. Performance and Security Trade-off for Routers in High Speed Networks In S. Hammond, S. Jarvis , & M. Leeke (Eds.), *UK Performance Engineering Workshop, University of Warwick, 2010* (pp. 119-128)

13. Jensen, K. (1998). An introduction to the practical use of coloured petri nets. *Lectures on Petri Nets II: Applications*, 237-292.

14. Bhatia, H., Lening, R., Srivastava, S., & Sunitha, V. (2007). Application of QNA to analyze the ‘queueing network mobility model of MANET. (pp. 1-6). Gandhinagar, India: Technical Report, Dhirubhai Ambani Insti tute of Information & Communication Technology (DAIICT).

Operating policies for energy efficient dynamic server allocation

Thai Ha Nguyen, Matthew Forshaw and Nigel Thomas

School of Computing Science, Newcastle University, UK

Abstract

Power inefficiency has become a major concern for large scale computing providers. In this paper, we consider the possibility of turning servers on and off to keep a balance between capacity and energy saving. While turning off servers could save power, it could also delay the response time of requests and therefore reduced the performance. Furthermore, as consistency is one of the most important factors for a system, we also analyse the level of consistency in the form of switching rate and fault occurrence. Several heuristic-based switching policies are introduced with a view to balance the cost between power saving, performance and consistency. Simulation results are presented and discussed with requests arriving according to a two-phase Poisson process.

Keywords: Energy efficiency, discrete event simulation, performance evaluation.

1 Introduction

The non-functional challenges facing large scale computing provision are generally well documented [1]. Amongst these the cost of energy has become of paramount concern. Energy costs now dominate IT infrastructure total cost of ownership (TCO), with data centre operators predicted to spend more on energy than hardware infrastructure in the next five years. The U.S. Environmental Protection Agency (EPA) attribute 1.5US electricity consumption to data centre computing [2], and Gartner estimate the ICT industry was responsible for 2With western european data centre power consumption estimated at 56 TWh/year in 2007 and projected to double by 2020 [3], the need to improve energy efficiency of IT operations is imperative.

Data centres, with their high density of power consumption and a steady growth in number, have become a major industrial energy consumer in the recent years. One of the most important factors that promoted their growth is that cloud computing has become a big trend in web services and information processing. The most significant advantage of the cloud is its flexibility. It offers the chance of shifting

[1] Corresponding author: nigel.thomas@ncl.ac.uk

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

capital expenditure to operational expenditure [4], which is ideal for starting a new service. Furthermore, since there is an increase in the quantities of data being collected for commercial, scientific or medical purpose, the big capacity of data centre is ideal to process such massive volume of data. As the cloud offers users an illusion of infinite computing resources on-demand [5], cloud computing is in fact essential to gather useful data from that enormous amount of information [6].

This paper is based on the work of Slegers *et al* [7,8] and is focussed on the notion that servers can be powered off and on according to demand in order to avoid the non-trivial energy requirements of idle servers. With perfect knowledge of arriving workload an optimal dynamic allocation of servers can be obtained which significantly reduces the overall energy demand of the system with no impact on performance, i.e. servers can be made available only when they are going to be used. Of course, we do not generally have a perfect knowledge of future workload and so an optimal dynamic solution is not practical. Instead we must investigate the trade-off between energy consumption and performance (e.g. response time) to determine the best practical method of reducing energy costs whilst not adversely affecting the quality of service. Two principle approaches to minimising energy consumption are apparent. In the first instance an optimal fixed provision of servers can be computed based on estimated workload. Depending on the variability in demand, this approach might lead to servers being idle for extended periods or to some tasks experiencing long waiting times during peak demand. The second approach is to compute a strategy to turn servers on and off based on the current (or past) state of the system. This approach minimises idle time by turning off servers, but potentially delays tasks which arrive in a burst as it takes time to turn servers back on. In addition, powering servers off and on may lead to faults which not only reduce the total available number of servers, but may also further delay an arriving task.

The remainder of this paper is organised as follows. In the next section we explain the context of this work in relation to other work on energy reduction. In Section 3 we describe the system model and introduce six heuristic strategies for controlling the number of servers powered on and off. This is followed in Section 4 by a brief description of the simulation environmen and we then a present and discuss the results of our experiments. Finally we present some conclusions and directions of further work.

2 Related Work

Although the issue of power efficiency for data centres had received significant concerns in the recent years, much less attentions had been paid to the option of dynamically turning servers on or off depended on the incoming requests. Some researches that were close to the idea had been introduced including [9] and [10]. However, they lacked the ability to dynamically turn servers on/off due to changes in the systems state. In [11], a dynamic server provisioning was examined using data traces from Windows Live Messenger. Since the workload of the Windows Live Messenger was periodic, i.e. it was predictable, this method was insufficient for the erratic and non-periodic workloads of a data centre. In [12], an M/M/c queueing system with removable servers and nontrivial setup times was examined.

The queueing model was relatively similar to this project, but servers could only be powered up and down one at a time. Similar to [12], the work of authors Gandhi *et al* [13] enabled multiple servers to be in powering mode at a time, while estimated the mean response time and the mean power consumption as key metrics.

On the other hand, this project was based on the previous works of authors Slegers *et al* [7,8], where the system consisted of a pool of homogeneous servers which could be in on, off or switching modes. The job requests arrived with high and low arrival rate and switching decisions were made by six different heuristics. The heuristics evaluated the number of jobs in queue along with the jobs arrival rate to optimize the system behaviour, given a trade-off between power consumption and response time.

With a similar system architecture to [7], the work of author Mitrani [14] focused on turning servers on and off in a block. The principle of this system was turning up a fixed number of reserves servers if the jobs queue exceeded a predefined threshold, while those reserves would be powered down in case of low workload. The two thresholds up (U) and down (D) determined the point of time when the system needed to turn on or off the reserves. In other words, m reserves servers would be powered up when the number of jobs exceed the threshold U, and those servers would be switched off when the jobs queue decreased to threshold D. Similar to [7], this system also used response time and power saving as key metrics. The switching question became how big the reserves should be, and how to choose the threshold U and D efficiently with the goal of balancing between performance and power consumption. Additionally, the paper also suggested the possibility of multiple server blocks with non-identical sizes. Server reserves could be powered up and down one after another as the job queue changed.

Another noticeable recent work was of the authors Yang *et al* [15]. It involved a resource management system which had the role of minimizing the number of active servers while keeping the overload probability to a standard threshold. This method did not measure the system performance by response time or energy consumption, but by an overload estimation model. The overload was calculated using the large deviation principle [16], while the decisions were made without any prior knowledge of the workload intensity.

The dynamic cluster reconfiguration model which consisted of a resource management system and a batch of server nodes. The servers could be in either active or sleep modes, and heterogeneous servers cluster was supported. In the resource management system, a server scheduling strategy including server management and job scheduling was introduced. The task scheduler had the role to allocate suitable resources for requested tasks, while the cluster reconfiguration controller could dynamically turn servers on or off to satisfy the workload demand. In other words, the goal of the dynamic cluster configuration was to turn off as many servers as possible and still be able to comply with the quality-of-service constraints. Interestingly, this research also proposed the possibility of independently applying of techniques like DVS and DPM in individual servers to achieve fine-grained energy consumption control.

Noticeably, none of the works above had mentioned the factor of consistency in a system. The closest measurement of consistency for such problem was found in [17]

in form of the rate of switching. That work focused on developing optimal policies for a single server system while taking the approximate response time, the energy consumption and the rate of switching into account. In this paper, we also consider consistency as the frequency of server powering on/off along with the occurrence rate of faults while switching. The measurement of consistency would be a beneficial metric for the policies to make switching decisions.

3 The Model

The system contained N homogeneous servers which can be in five states: powered up, powered down, powering up, powering down and fault state. The powered up servers could be working or staying idle, while there were only one mode each for the other states. The powered down mode was left ambiguous and could mean complete shutdown or hibernating, which consumed less or no power. During switching or fault modes, a server could not serve jobs but still consumed power. While faults could happen in almost any state, this paper only focused on the appearance of faults while switching, which was believed to have a high possibility of occurrence. Furthermore, the number of faults while switching should be an adequate index to measure the consistency of the system. Additionally, the modes were provided with specific costs, which were c_{up}, c_{idle}, c_{down}, c_{powUp}, $c_{powDown}$, c_{fault} respectively. While most of the costs denoted the relative energy cost for a server to run in that mode for a unit of time, c_{fault} reflected the relative loss for a server to remain in fault mode.

Furthermore, the system held an unbounded jobs queue which received job requests according to a two-phase Poisson process, i.e. a high and a low arrival period. During the periods, jobs were coming with the average mean of λ_{high} and λ_{low} for each unit of time correspondingly. The requests time was exponentially distributed with an average duration of μ, while the time of high and low periods were also exponentially distributed with mean ξ and η respectively. Similarly, other values were calculated using the exponential distribution, including the duration of fault t_{fault} and the switching time t_{up} and t_{down}. In addition from the energy costs and the fault cost, we also assigned the job holding cost c_{job}, which indicated the cost of holding a job in the system for one unit of time, in other words the need of processing jobs quickly. Moreover, there was also c_{pow} which reflected the energy saving benefit of keeping a server down for one unit of time. Again, those were only relative cost, i.e. $c_{pow} = 1$ and $c_{job} = 2$ simply meant that keeping a job in the system was double as expensive as the energy saving gained by powering down a server. To sum up, a data centres state could be described as follows:

$$S = \{j, \lambda, k_{up}, k_{down}, m_{up}, m_{down}, f\} \tag{1}$$

J denotes the number of jobs in queue, λ the arrival rate, while k_{up}, k_{down}, m_{up}, m_{down} and f are the number of servers which were currently up, down, powering up, powering down and at fault mode respectively. Furthermore, the sum of servers in all modes was always N (i.e there is no loss). A system state could move to the next state by the following possibilities:

- Staying the same without switching, the durations of modes and arrival periods

can be decreased by one unit of time.

- If the duration of a server mode reaches zero, the number of servers for that mode will decrease by one and the number of servers up/down will be increased by one accordingly.
- If the duration of an arrival period reaches zero, the system will move to the next period.
- Turning on x servers, which means m_{up} will be increased by x and k_{down} will be decreased by x respectively.
- Turning off x servers, which means m_{down} will be increased by x and k_{up} will be decreased by x respectively.
- The number of jobs j will be increased according to the current arrival rate, meanwhile j can also be decreased if the duration of any job reaches zero.
- The same procedure is also true for f. It can be increased with the rate m_f as long as there are servers being powered on or off. If the duration of fault mode for a server reaches zero, f will be decreased.

In fact, those possibilities do not happen individually, they were often combined with others to reach the next state. Finally, there were two metrics of calculating the energy usage of the system. The first metric was the energy consumption which was the total of energy costs for all servers in the system. This was a straightforward method which was understandable and easy to calculate. However, it could not fully determine the efficiency of a data centre. For example, a system could keep the number of powered up servers low to gain small energy consumption, but it was in fact non-profitable since the inadequate number of servers could not keep up with the incoming requests. Therefore, the energy efficiency metric was introduced with the view of taking power consumption together with other values into account. This metric was calculated as follows:

$$E_{eff} = k_{down}c_{pow} - jc_{job} - fc_{fault} - m_{up}c_{powUp} - m_{down}c_{powDown} \tag{2}$$

In other words, this metrics calculated the power saving benefit of having a server staying down, while took into account the costs of faults, of having too many jobs in the queue and of switching too many servers. It was in fact a trade-off between saving powers, consistency and performance.

4 Switching Policies

This section will introduce six switching heuristics of different characteristics with the view of balancing data centres power savings, consistency and performance. The heuristics have the role to inspect each state of the data centre and made decisions of powering servers on or off for the next state. While most methods depended on statistical theories to achieve solution, some others simply reacted based on the number of requests in queue.

4.1 Static Allocation Heuristic

This method employs the concept of making no changes in the number of active servers. In other words, the heuristic decided on the best possible number of servers to switch on, and made no additional switching after that. Unless there were faults occurred in the process of powering on, then the heuristic would decide to switch on more servers to compensate for the lost ones. First, the heuristic determined the average rate for both arrival periods by adding the jobs loads and then divided it to the mean duration of both periods:

$$\alpha = \frac{\xi\lambda_{high} + \eta\lambda_{low}}{\xi + \eta} \tag{3}$$

Therefore, the problem became a jobs queue with arrival rate α and a fixed number of servers n, while n must not exceed the total number of servers N. This is a well-known problem in queueing theory (see e.g. [18]) and hence we can calculate the probability of all n servers being occupied. Subsequently the mean response time for n servers was determined as followed:

$$t_n = \mu\left(1 + \frac{Ec(n,\mu)}{n-\mu}\right) \tag{4}$$

As μ is the mean load of the system, $n-\mu$ indicates that n should not be smaller than the mean load. Furthermore, with the use of Erlang-C formula $Ec(n,\mu)$, the equation $\frac{Ec(n,\mu)}{(n\mu)}$ describes the extra percentage of time that the system may spend to finish processing the job in comparison with the average duration μ. In other words, it is the extra percentage of time that jobs need to wait before getting processed. Subsequently, it is trivial to calculate the energy efficiency of the data centre for that response time:

$$E_{eff} = (N-n)c_{pow} - c_{job}t_n \tag{5}$$

Since the method keeps a fixed number of servers all the time, the cost of switching and faults can be excluded. Therefore, the energy efficiency is only dependent on the power savings and the job holding cost. Then the process can be easily repeated for all possible number of servers in order to determine the best efficiency value and therefore the optimal number of servers.

The idea of the method is to sacrifice the ability to respond to incoming jobs volume for the stability. Obviously, this is an especially stable method. Since the heuristic only requires a small ammount of initial switching, it contains almost no faults. However, this method does not have a good performance as we will observe. In fact, the number of servers that this method decides on is often more than enough to handle the low arrival rate, while being insufficient for the high arrival rate. Therefore, in the case that the high arrival rate lasts longer than expected, this heuristic is likely to perform badly because of the high job holding cost.

4.2 Semi-static Allocation Heuristic

In order to fix the disadvantage of the static allocation heuristic, the semi-static policies is introduced. This method works with the same principle as the static allocation, but it treats the two arrival periods separately. In other words, the semi-static heuristic uses the Erlang C formula to find out the best number of

servers for the high and low arrival periods separately. Therefore, those optimal numbers of serv-ers will be able to keep up with the arrival rates of both periods without the waste of turning on too many servers.

Unlike the static allocation method, the semi-static still needs to turn servers on or off between periods. However, since it uses the same mechanism as the static allocation, the semi-static does not take the switching time into account. Therefore, when the system is erratic, i.e. the durations of periods were short, or when the switching time is long, this method clearly shows a poor performance. In some extreme cases, the period might be over before the switching finished. Nevertheless, because switching times are often small in comparison to the periods duration and the length of jobs, the semi-static still has a good overall performance.

4.3 Idle Heuristic

This policy is the most straightforward method which depends entirely on the number of jobs in the queue. The idle heuristic has the strategy to turn off any idle server and turn on more servers if there are jobs waiting. To be more precise, the number of currently powering servers was also taken into account, i.e. more servers will be turned on if the number of jobs is larger than the total of active servers and powering on servers. This was clearly a very passive and unsophisticated method. While other heuristics try to predict the rate of incoming jobs and act correspondingly, the idle heuristic completely excludes the possibility of prediction. Understandably, this method requires a very high level of switching, which led to a big switching cost. Furthermore, even if the switching cost is small and the switching time is insignificant compared to the job duration, the idle heuristic would not necessarily be a good choice. If the job holding cost was significantly smaller than the power saving benefit, then it would be preferable to have some jobs waiting rather than turn on servers instantly.

Although the inefficiency and naivety of the idle heuristic is undeniable, its simplicity still possesses a strong point. While other methods would need a significant amount of computer resources to calculate statistical theories and run through various loops, the idle policy only requires a minimum of processing power.

4.4 Threshold Heuristic

This heuristic was proposes as an improvement from the idle heuristic. To be more precise, it is a generalisation version of the idle where a threshold j_{thres} is defined. If there were more than j_{thres} idle servers then the servers will be switched off. On the other hand, if the number of jobs is greater than the available servers and the threshold ($j > j_{thres} + k_{up} + m_{up}$) then more servers will be turned on. Clearly, the idle heuristic is equivalent to the case that $j_{thres} = 0$.

The idea of the threshold heuristic is to reduce the switching number of the idle heuristic by establishing j_{thres}. But the process of choosing a suitable threshold value is not trivial, especially when stability is also considered to measure the efficiency of a data centre. The threshold should significantly reduce the amount of switching, while not restricting the system too much to keep up with the incoming requests. Therefore, the task of selecting the threshold should take the job holding

cost along with the power saving and the number of switching into account.

4.5 High/Low Heuristic

The high/low heuristic also depends on the current arrival period of the system. But unlike the semi-static, this method takes the switching time into account. The high/low can be considered the most sophisticated heuristic since it analyses every factor of the system, including the arrival periods, switching time, processing time and the queue length. Furthermore, we assume that the system is very stable. In other words, jobs will arrive at a constant state of λ_{high} or λ_{low} while the job duration had the fixed value of μ.

Basically, the high/low heuristic is based on the job processing speed to estimate the average time when the job is finished. If the system did no switching, then it is trivial to estimate the time that the jobs were finished. On the other hand, if the system decided to switch on more servers, then after the switching time, the processing speed would increase and the time would be shortened. On the contrary, the processing speed would be slowed down and the time would increase if the system switched off servers.

Assuming there was no switching and the current number of working servers is k_{up}, then the jobs are processed with the speed of $\frac{k_{up}}{\mu}$. If there are j jobs in the system and the arrival rate is λ, then the average decreasing rate of the queue length should be $s = \frac{k_{up}}{\mu} - \lambda$. Therefore, the finished time for those jobs is $t = j/s$. Since the queue is assumed to be processed at a constant rate, the estimated efficiency can be calculated as follows:

$$E_{eff} = K_{down} c_{pow} t - \frac{j}{2} c_{job} t \tag{6}$$

With similar approach, we can also calculate the efficiency when the system decides to switch servers on or off. The switching also includes the servers that are currently in powering up or powering down mode. In this case, the process can be divided into two different phases: before and after the switching time $t\prime$. Assuming at time $t\prime$ the number of jobs was reduced to $j\prime$ and the numbers of servers up and down were $k\prime_{up}$ and $k\prime_{down}$ accordingly. Then $j\prime$ can be calculated as $j\prime = j - (s/j_{up})$ or $j\prime = j - (s/t_{down})$ depending on the type of the switching. After that equation (6) can be used again to estimate the after-switching efficiency $E\prime_{eff}$ with $j\prime$ jobs and $k\prime_{up}$ working servers. On the other hand, the before-switching phase is estimated using the initial values. Then the total efficiency of the whole process should be:

$$E_{eff} = k_{down} c_{pow} t\prime - \frac{j + j\prime}{2} c_{job} t\prime + E\prime_{eff} - C_{switch} \tag{7}$$

Where C_{switch} is the switching cost, calculated as $(k_{up} - k\prime_{up}) c_{powUp}$ for switching on and $(k\prime_{up} - k_{up}) c_{powDown}$ for switching off. Subsequently, the process is repeated for every possible switching to choose an optimal outcome.

The high/low is a sophisticated heuristic which measures the performance along with the stability of the system. However, due to the complexity of the calculation, this method may require much more processing resource in comparison to others. Furthermore, as the high/low heuristic makes decisions based on the arrival period, it faces the same problem as the semi-static when the arrival periods were short and erratic.

4.6 Average Flow Heuristic

The average flow heuristic uses the same method of calculating the energy efficiency as the high/low heuristic. The only difference between them is instead of two types of arrival periods, this heuristic averages out the high and low arrival rate into a single one:

$$\lambda = \frac{\xi\lambda_{high} + \eta\lambda_{low}}{\xi + \eta} \tag{8}$$

The rest of the analysis is similar to equations (6) and (7). By having a single arrival rate, the average flow heuristic resolves the weakness of the high/low heuristic when the durations of periods were very short. This means the average flow is more stable and required fewer switchings. However, it cannot keep up with a long high arrival period as well as the high/low heuristic.

5 Experimental results

A simulation for a data centre model was implemented using Java JDK. It used an additional library of JFreeChart [19] to display the real-time running of the system, along with the statistical results. From this experiments were undertaken to better understand the performance of the various heuristics introduced above. Each simulation run lasted 10000 units of (simulated) time, and the costs were calculated from an average of 50 runs.

First, the heuristics were compared in a system with different levels of high arrival rate. Second, we simulated a scenario in which the ratios between the job holding cost and the cost for servers to staying off were varied. The results also indicated a significant improvement of the threshold heuristic over the idle heuristic. In addition, an erratic system with short duration of both high and low arrival rates explained the necessity of policies that are independent on what the current arrival period is. Furthermore, a scenario in which the server powering time was changed from low to high was also investigated. Finally, a chart of total faults among policies was displayed to explain why a stable heuristic is always preferable.

Note that the average costs in the figures below were the energy efficiency indexes of the data centres and not the energy consumptions. Those are only relative numbers that rate the performances and compares between heuristics, which means a negative value does not denote that there is no improvement in the data centre.

5.1 Increased Bursts

This experiment is one of the most common situations in practice when the duration of the high arrival period is relatively small in comparison with the duration of the low period. The system contained $N = 90$ servers with the average request processing time $\mu = 3$. The arrival rate in the low period was $\lambda_{low} = 10$, while the high arrival rate increased from 10 to 30 which made the utilization 100% at the highest peak. Durations of the low and high periods had a mean value of 100 and 10 respectively. The benefit of a server staying down was 1 while the job holding cost was 2. These relative values indicated that having a job in the queue was twice as expensive as having a server powered up. There was also a 0.05% rate of faults

with the cost for each fault being 10, which slightly decreased the performance of the heuristics with high level of switching. Finally, the switching of servers up and down was considered with a cost of 3, while powering servers took 1 unit of time.

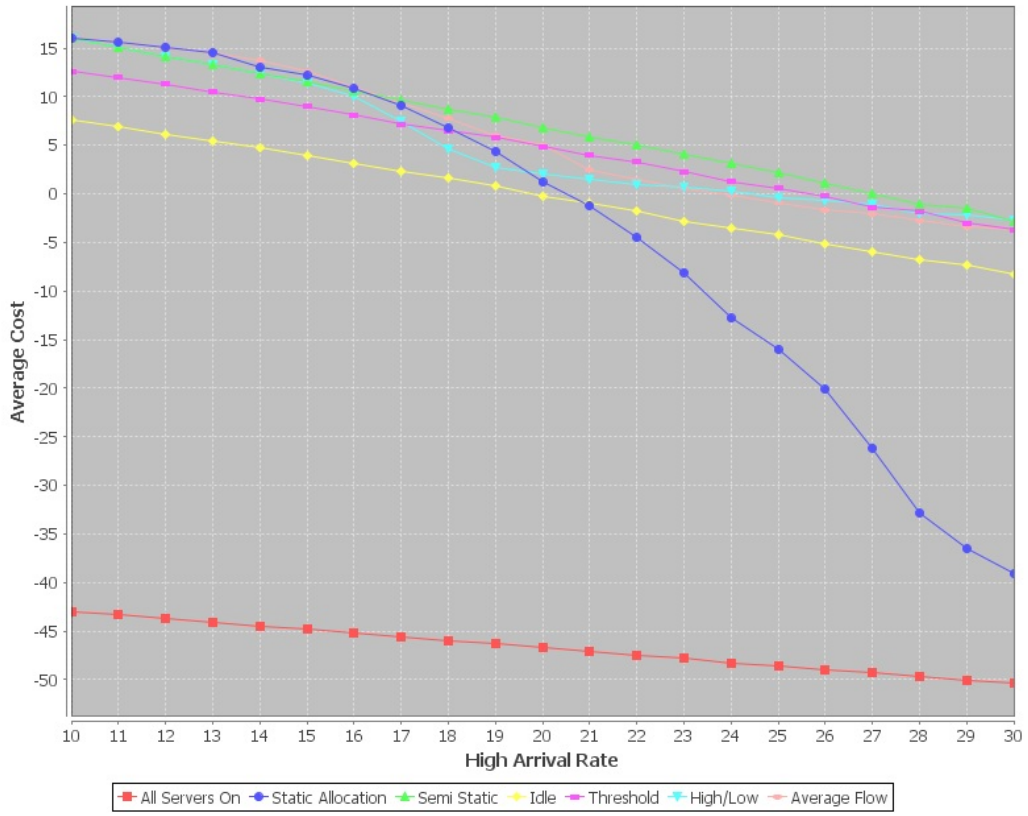


Fig. 1. The effect of increasing arrival intensity on heuristic performance

Figure 1 displays the performance of the heuristics with the setting of all-servers-on as a baseline cost, which clearly has the worst efficiency overall. However, as the high arrival rate kept increasing, the efficiency of the static allocation heuristic would eventually decrease to that of all-servers-on (since the static allocation will be all the servers). In certain occasions when the high period lasted longer than expected, the number of powered up servers would not be able to keep up with the increasing request, followed by an exceeding big jobs queue. On the other hand, the idle heuristic also performed badly in comparison with the rest policies. This was the most basic choice for a heuristic, but it involves too much switching which lleads to a high energy cost and higher occurrence of faults. As an improvement from the idle heuristic, the threshold heuristic with a threshold of 3, which was expected to have less switchings, showed a significantly better performance with its average costs always staying about 5 units beyond the cost of the former policy. On the other hand, the performance of the last three heuristics was quite remarkable with semi-static being the best policy because of its stability. It was also understandable that the semi-static and high/low heuristics, which made distinctive decisions for each arrival period, tended to perform better than other methods in the latter half of the chart when the gap between arrival rates of high and low periods was bigger.

This scenario was one the most common situation in practice, when high periods

only occurred at certain times of day. Out of the six heuristics, four performed quite well in this situation; however it was not enough to determine their performances and other characteristics would have to be taken into consideration in the later experiments.

5.2 Changing Cost Difference

While the last experiment focused on data centres whose priorities involved processing jobs quickly than having a server powered down to save energy, there are also systems which preferred to have their servers staying down until a certain level of jobs stacked up in the queue. Therefore, this section concentrates on the difference between the job holding cost and the benefit of servers powered off. Here the system contains 50 servers with mean request processing time $\mu = 3$. The other numbers were mostly the same as the last experiment, but the high and low arrival rates were 5 and 20, respectively. Additionally, the job holding cost had a value of 5, while the power saving of a server staying down was increased from 1 to 10, which indicated the priority of powering a server off in comparison with the need of quickly processing a job, from very low to very high priority.

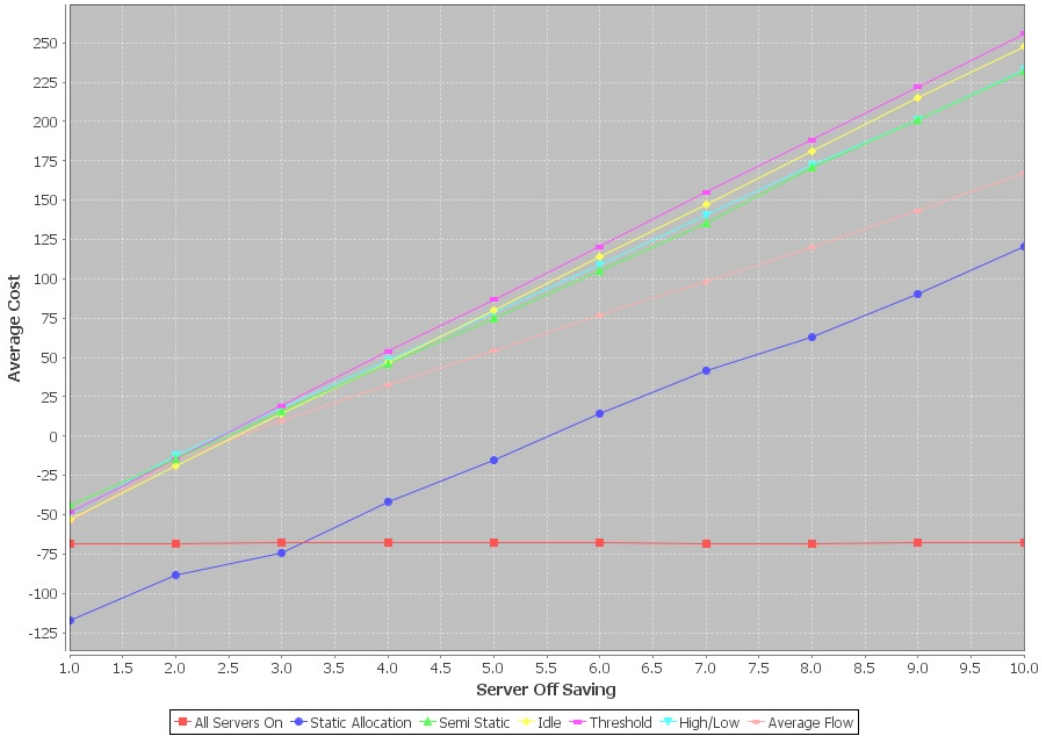


Fig. 2. The effect of increasing power cost on heuristic performance

As described in the previous section, the static allocation method performes worse when there is a big difference between high and low arrival rates. Except for the average flow policy, the other methods performed quite well with the threshold heuristic being slightly ahead of others. The idle heuristic also had a high result since its biggest disadvantage, the cost of powering servers on and off, is not significant enough in comparison with the job holding cost and the increased server

off saving. On the other hand, the threshold heuristic with a threshold of 3 performs exceedingly well as the server off benefit grew. This method is well balanced between making servers stay down with its threshold and still keep up with the increasing requests. Apart from that, the average flow policy, while averaging out the high and low period, had a noticeable lower performance than others. Its behaviour was somehow similar to the static allocation heuristic when the disadvantage was caused by prolonged high arrival periods.

5.3 The Threshold Heuristic

Since the last two experiments showed a significant improvement of the threshold heuristic over the idle heuristic, this section considered those two separately from other methods to have a more clearly view of the pros and cons of the threshold method.

In this case a system with 50 servers was measured with a high arrival rate of 30 jobs per unit time and a low arrival rate of 5 jobs per unit time. The server off saving was 8 while the job holding cost was 10, which indicated that processing jobs quickly was given slightly more priority than saving power. The cost of powering servers on and off was accumulating from zero to 10 to show the changes of the two heuristics in comparison with each other. Furthermore, the threshold was set to be 5 to make the result easier to distinguish.

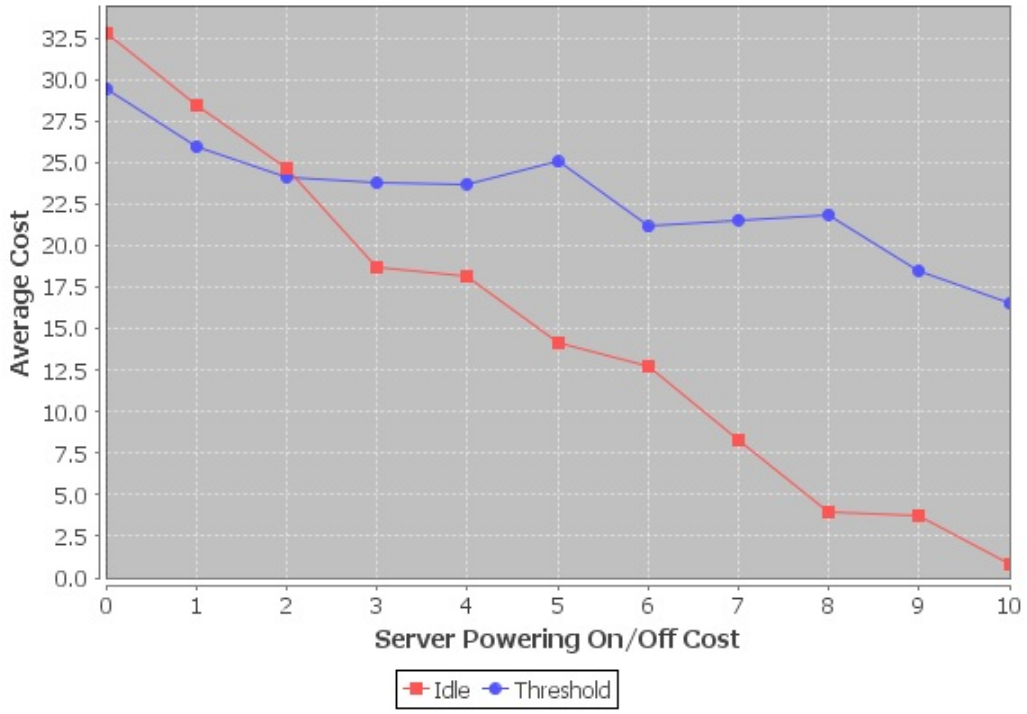


Fig. 3. The effect of increasing switching costs on threshold and idle heuristic performance

Not surprisingly, the idle heuristic had a better figure in the first part of the chart. Since the cost of turning servers on and off was notably lower than other costs, the advantage of threshold over idle heuristic was not significant enough. As

the power-ing cost grown bigger, the threshold also showed its strong point and clearly surpassed the idle heuristic. It was also clear that the threshold was not always a good choice over the idle policy. For data centres which had a slow arrival rate of request but long mean processing time, servers would prefer to be turned on intermediately instead of waiting for the jobs queue to pass the threshold, which was likely to take a long time.

5.4 Changing Period Duration

In the previous scenarios, the high/low and semi-static heuristics displayed an impressive performance. The main common point of those two methods is that they both handled the high and low arrival periods separately. However, there are also systems in which those methods do not work that well. In the case when the durations of periods are very short, the erratic system may move to the next period before the decision of this period took effect, which may lead to unnecessary switching. Therefore, policies which calculate the average requests arrival might be more suitable. This experiment was designed to display such a case. The system was set up just like the first scenario with 90 servers with a high arrival rate of 25. The high arrival period lasted 10 units of time while the low arrival period was varied from 10 to 100 units of time, which moved from a very erratic data centre with short period durations to a more stable one.

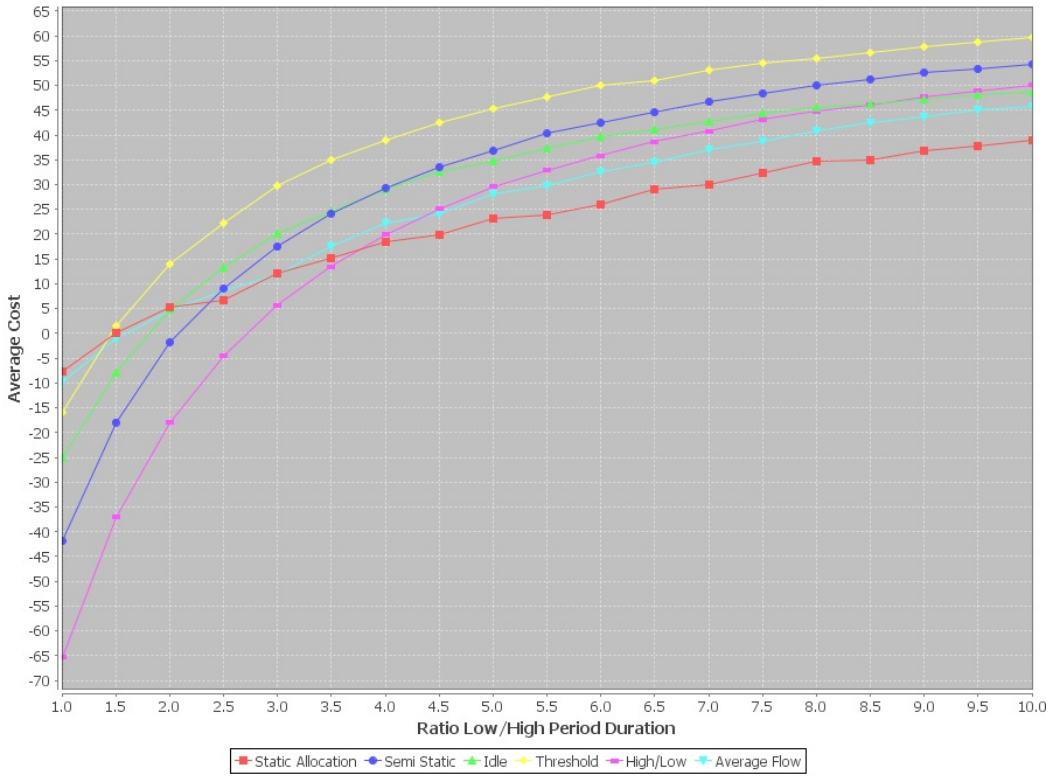


Fig. 4. The effect of increasing arrival duration on heuristic performance

The beginning of the chart displayed a considerable advance of the static allocation and average flow heuristics over the semi-static and high/low heuristics. As

the first two methods made decisions based on the average flow of requests, which is less dependent on what period the data centre is currently in, they gain a big advantage when the duration of low and high periods are not too much different. This situation changed in the latter half of the chart when the low period duration increased and the two period-distinct policies regain their value. In addition, the naive policy of the idle heuristic also worked quite well when the system was erratic, while the threshold continued its good performance by being the best heuristic most of the time.

5.5 Changing Switching Time

In this scenario, the case of systems with different server powering time was investigated. This experiment was designed to simulate situations when a data centre need to handle small requests which require only a short processing duration, while turning servers on would require a longer time. The system was based on the data centre in the first experiment with 90 servers. The average service duration take 5 units of time, while the time of powering servers varied from 1 to 10.

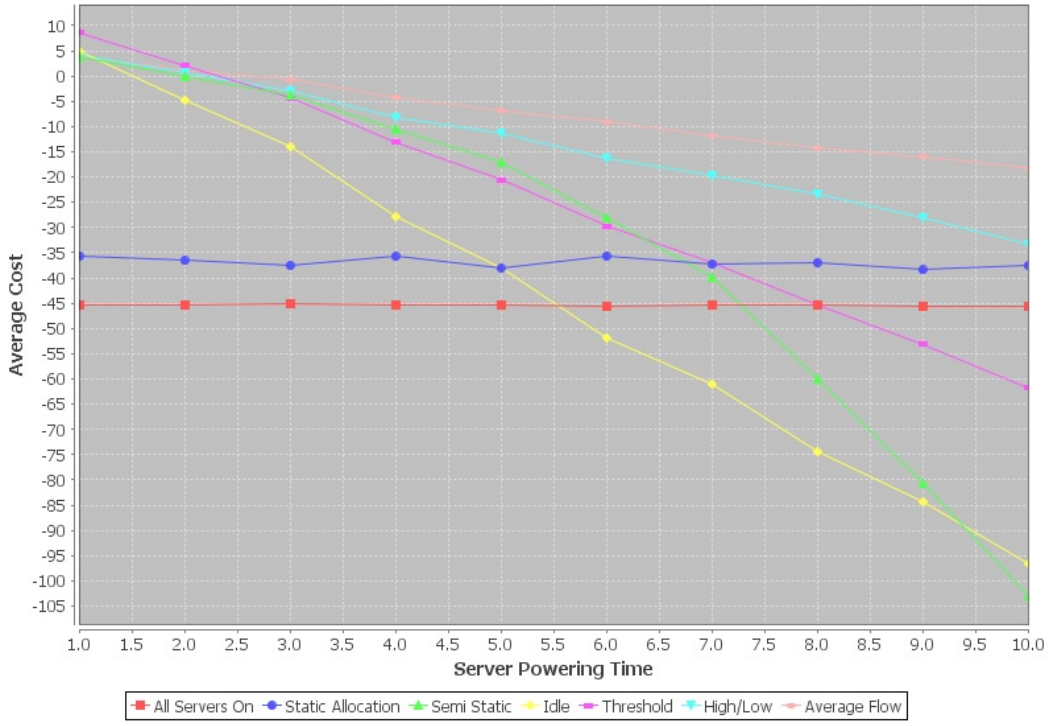


Fig. 5. The effect of increasing switching time on heuristic performance

As the server powering time increased, the performance from most heuristic gradually decreased, except for the static allocation method, since this heuristic did not need to turn on any more servers. The semi-static policy also made it decision without concerning about the powering time. Regardless of powering time, this method would behave the same. However, this heuristic still need to power servers on and off be-tween periods, which made semi-static the worst heuristic when the powering time passed 9 units. On the other hand, the idle and threshold policies decreasing patterns were quite similar with the threshold always performs

better than its predecessor. As the switching time grew longer, more requests would have to wait before being pro-cessed. On the contrary, the high/low and average flow heuristic did take the switch-ing time into consideration, which gave them the best figures above all. However, as the switching time kept increasing, they would eventually be surpassed by the stabil-ity of the static allocation policy.

This scenario may not have much use in practice, when the switching time is often insignificant compared to the duration of requests. However, it did point out the prob-lem within semi-static heuristic that this method did not calculate the switching time. As semi-static being one of the most well performed heuristic in the last experiments, an enhancement regarding this problem would be really useful for further practices.

5.6 The Fault Rate

One of the most important factors when considering the performance of a system is its consistency. It is also true with data centres. In this case, the consistency denoted the ability to avoid unnecessary switching, which tended to trigger faults. The system in Section 6.1 was measured again to get the average number of faults for each heuristic after a loop of 10000 units of time and 50 runs, while the fault rate of switching was calculated with a probability of 0.1%.

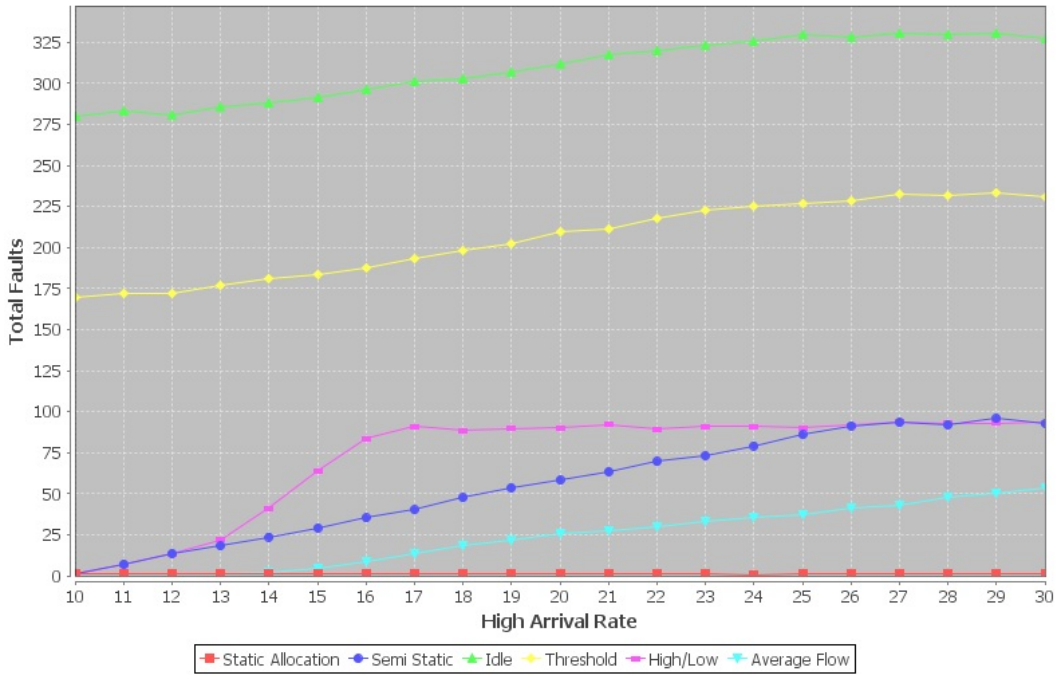


Fig. 6. The effect of increasing arrival intensity on fault rate

Obviously, the static allocation heuristic had the fewest faults above all, since this heuristic did not require any switching except for the initial powered on servers at the start of the system. On the other hand, it was understandable that the threshold and idle heuristics had the largest numbers of faults, as those two meth-ods kept turning servers on and off to keep up with the length of the jobs queue. Moreover, the semi-static and high/low policies had quite a low level of faults, while

the average flow method performed the most surprisingly with its number stayed as small as the static allocation method when the high arrival rate was low. The mean reason for the con-sistency of the average flow method was that this heuristic had a single average arrival rate, so its switching decision did not fluctuated between high and low periods like the last two heuristic.

As consistency being an indispensable factor for all systems, the number of switch-ing and faults will need to be taken into consideration when choosing a suitable heuristic for a data centre. The threshold policy, which had performed very well in the last experiments, may not be that impressive choice of a heuristic regarding its high number of switching.

5.7 Summary

From the experiments above, it is to be concluded that the idle heuristic is generally a poor choice of a policy. But it does not mean that this heuristic will perform badly in every situation. As an enhancement of the idle heuristic, the threshold heuristic showed some encouraging potential, however its high level of switching had become a great drawback. The static allocation method also did not perform well, since it preferred doing nothing over turning servers on, but that is also the reason why it has such high stability. The semi-static, high/low and average flow heuristics all have their own strengths and weaknesses which can be adapted for different situations. Especially in the case of the semi-static heuristic, if it could fix the problem of neglecting the switching time, this would be a really promising policy. Last but not least, as a stable system is always preferable, the consistency of the heuristics should be taken into account when considering their effectiveness.

There is clearly no best heuristic which suits every situation, as each heuris-tic works well in a specified situation and worse in others. There are many factors affecting the performances of the heuristic, including the differences between arrival rates and time, the length of switching time, the number of faults, etc. Since there is always a trade-off between performance and energy saving, it is the job of data cen-tres operators to find out the balance between them and to decide what policy is the most suitable for each specified data centre. Furthermore, there is even the possibility of having many policies for a single data centre, which can switch between different heuristics for different situation.

6 Conclusions

In this paper we have explored a model with multiple servers servicing an input stream of jobs. In order to limit the power consumption we allow servers to turn off and on according to demand. We have extended previous work in this area by considering the possibility that servers can fail whilst turning off or on. The costs of providing servers, holding jobs and failures have been incorporated into a new cost function which allows the performance of the system to be better understood.

We have proposed six heuristics for constructing a policy to manage the servers turning off and on and we have compared these numerically through a custom-built simulation. The simulation has been run with a number of scenarios to consider different operating conditions. The results of the simulation show that several of

the heuristics are capable of performing well under certain conditions, but there is no single heuristic that we can claim is always best. This suggests that one line of future work might be to consider an environment which is capable of employing multiple heuristics to obtain a better performance under more conditions.

Our approach here has a number of limitations. Firstly we have only considered delays which are negative exponentially distributed, whereas in reality this may not be the case. Given that we are simulating the model, there is no real reason why we could not consider general distributions to better understand the effects that different distributions might have on system performance. We have also assumed that all servers are identical, whereas in practice this may not be the case. Not only would different servers have different processor speeds, but they would conceivably have different energy consumptions. It would be feasible and interesting to model more than one type of server and to consider, for example, the impact of utilising N fast but energy inefficient servers or M slower but more efficient ones.

References

[1] Jarvis, A., Thomas, N., and van Moorsel, A., Open issues in grid performability. International Journal of Simulation, 5(5):312, 2004.

[2] Brown, R., Report to congress on server and data center energy efficiency: Public law 109-431. Lawrence Berkeley National Laboratory, 2008.

[3] Bertoldi, P., and Anatasiu, B., Electricity Consumption and Efficiency Trends in European Union Status Report 2009.

[4] Creeger, M. (2009). Cloud Computing: An Overview. ACM Queue, 7(5), 2.

[5] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, Y., Konwinski, A., and Zaharia, M. (2009). M.: Above the clouds: A Berkeley view of cloud computing.

[6] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., and Zaharia, M. (2010). A view of cloud computing. Communications of the ACM, 53(4), 50-58.

[7] Slegers,J., Thomas N., and Mitrani, I., Dynamic server allocation for power and performance. In Performance Evaluation: Metrics, Models and Benchmarks (pp. 247-261). Springer, 2008.

[8] Slegers,J., Thomas N., and Mitrani, I., Static and dynamic server allocation in systems with on/off sources. Annals of Operations Research, 170(1), 251-263, 2009.

[9] Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A. M., Doyle, R. P. (2001, Octo-ber). Managing energy and server resources in hosting centers. In ACM SIGOPS Operating Systems Review (Vol. 35, No. 5, pp. 103-116). ACM.

[10] Urgaonkar, B., Shenoy, P., Chandra, A., Goyal, P., and Wood, T. (2008). Agile dynamic provisioning of multi-tier internet applications. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 3(1), 1.

[11] Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., and Zhao, F. (2008, April). Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Ser-vices. In NSDI (Vol. 8, pp. 337-350).

[12] Artalejo, J. R., Economou, A., and Lopez-Herrero, M. J. (2005). Analysis of a multiserver queue with setup times. Queueing Systems, 51(1-2), 53-76.

[13] Gandhi, A., Harchol-Balter, M., and Adan, I. (2010). Server farms with setup costs. Perfor-mance Evaluation, 67(11), 1123-1138.

[14] Mitrani, I. (2013). Managing performance and power consumption in a server farm. An-nals of Operations Research, 202(1), 121-134.

[15] Yang, J., Zeng, K., Hu, H., and Xi, H. (2012). Dynamic cluster reconfiguration for energy conservation in computation intensive service. Computers, IEEE Transactions on, 61(10), 1401-1416.

[16] Dembo, A., and Zeitouni, O. (1998). Large deviations techniques and applications (Vol. 2). New York: Springer.

[17] Maccio, V. J., and Down, D. G. (2013, August). On optimal policies for energy-aware servers. In Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2013 IEEE 21st International Symposium on (pp. 31-39). IEEE.

[18] Kleinrock, L. (1975). Queueing systems, volume I: theory.

[19] http://www.jfree.org/jfreechart/