Capacity Planning in Virtualised Environments Using Model Driven Engineering

Rafidah Binti Pakir Mohamad

Doctor of Philosophy

University of York

Computer Science

October 2015

Abstract

Capacity planning is an important activity in computing for optimising resource usage while avoiding performance degradation. The demand for computing resources is triggered by application workloads running on virtual or physical machines. With today's technology, resource scalability can be achieved through server virtualisation, by having scalable *virtual machines* running on a physical server. However, these scalable *virtual resources* run on limited *physical resources*, especially in small to medium scale data centres. The management of virtual and physical resources impacts upon application performance and introduces a cost for all parties. There is a need to measure the virtual and physical resource requirements in facilitating cost-effective capacity planning. This research identifies three main management phases for a capacity planning process for a data centre implementing server virtualisation: capturing application workloads, managing virtual resources and managing physical resources. This research proposes an approach that leverages domain specific modelling and model transformation to estimate resource requirements based on predicted application workloads for certain time periods. Model-Driven Engineering (MDE) was utilised to automate the identified process. A transparent, automated and repeatable MDE process for generating predictions for resource usage from workload models and sets of Domain Specific Modelling Languages (DSMLs) that allow resource and workloads logs as well as predicted workloads to be precisely captured using models were designed, implemented and evaluated with case studies. The MDE process exploits model transformation, comparison and merging, is modularised so that it can be configured for different kinds of capacity planning applications and technical infrastructures.

Contents

List of Tables

List of Figures

Listings

List of Algorithms

Acknowledgements

Firstly, I thank All Mighty God for making this thesis possible. I am sincerely thankful to my both supervisors, Dr. Dimitris Kolovos and Prof. Richard Paige for their invaluable encouragement, guidance, solid supervisions and continuous motivations throughout my doctoral research studies.

I would also like to thank my internal examiner, Dr. Fiona Polack for her comments and feedback from the initial assessment until the current stage.

I thank all the technical staff in the Department of Computer Science for their constant technical support internally and externally. Non-stop support by Alex Cooper is really appreciated. I also thank Dr. Derek Wang for his technical support in initial infrastructure setting.

I would also like to thank my friends and colleagues who have supported me along the way by offering me valuable advice, by sharing expertise and also experience.

I would like to express my warmest appreciation to my husband, Noorul Ameen Mohamed Eliyas for his patience, support and understanding. Last, but by no means least, I also express my gratitude to my four loving children who enrich my life and my PhD studies in colourful ways.

Declaration

I declare that this thesis is a presentation of original work and I am the sole author. This work has not previously been presented for an award at this, or any other, University. All sources are acknowledged as References.

List of Publications:

1. Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Resource Requirement Analysis for Web Applications Running in a Virtualised Environment. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 632–637, Dec 2014. doi: 10.1109/CloudCom.2014.134. URL `http://www.computer.org/csdl/proceedings/cloudcom/2014/4093/00/4093a632-abs.html`

2. Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Cloud Computing Workload and Capacity Management Using Domain Specific Modelling. In *CloudMDE Workshop, co-located with ECMFA 2012*. CEUR Proceedings, July 2012. URL `http://www2.imm.dtu.dk/conferences/ECMFA-2012/workshops/?page=CloudMDE`

3. Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Modeling Workloads, SLAs and their Violations in Cloud Computing. In Christopher M. Poskitt, editor, *Fourth York Doctoral Symposium on Computer Science*. Department of Computer Science, The University of York, UK, October 2011. URL `http://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/468/YCS-2011-468.pdf`

Chapter 1

Introduction

1.1 Background

A data centre is a central computing repository for storage, telecommunications and other associated components. Studies show that server utilisation in real world data centres is estimated between 5% and 20% [7, 8]. In addition, servers are rarely completely idle and only reach 10% to 50% of their maximum utilisation level [10]. These show that servers in real data centres are underutilised. However, underutilised servers still consume a great deal of electricity when idle [18, 75]. Furthermore, additional energy is needed for cooling the heat produced by the underutilised servers.

Running a data centre can be very expensive. Cost-effective data centre management by implementing virtualisation and server consolidation are possible solutions to maximise server utilisation [75]. Server virtualisation reduces the number of physical servers by having several virtual machines running on a single server. Intelligent resource management reduces the power consumption in data centres by having fewer servers and, as such, require less cooling.

Implementing server virtualisation separates the computing resources in a data centre into physical and virtual resources. Virtual resources are completely isolated server installations within a normal physical server. These work like real standalone servers, while in reality they still share the physical resources of the hosting server; such architectures make use of *virtual machines* (VMs). In addition to reducing cost by minimising the number of servers, virtualisation also has other advantages. The VMs can be migrated to another physical server and mirrored as often as needed for multiprocessing. Usually, VMs are also called *instances* or *guest machines*, and they are scalable. To optimize physical server utilisation, VMs are allocated to physical servers based on the workload pattern stated in the Service Level Agreement (SLAs) [35]. Even though VMs are scalable, they need to be managed properly to minimise failure on a physical machine [37].

Virtualisation can be used to create multiple virtual data centres within a physical data centre. Therefore, virtualised environments comprise two components. The first component is the *physical data centre*, which is typically operated by a data centre management team and provides a physical computing infrastructure. In this work, they are referred to as the *physical infrastructure provider* (PiP). The second component is the *virtual data centre*, managed by the virtual servers management team, which administers the virtual resources in the virtual environments provided by the PiP and they are referred as the *virtual infrastructure provider* (ViP). A ViP usually owns a number of virtual machines running at unknown locations in physical data centres. Managing these virtual machines is called virtual data centre management. Physical resources in physical data centres may be shared by a number of ViPs through the creation of VMs. Virtualisation as an enabling technology for resource sharing has been commonly adopted in many computing environments, such as in grid, cluster or cloud computing. Cloud computing is an approach which aims to minimise costs for both consumers and providers of the services; software, platform and infrastructure. It employs the concept of utility computing first proposed in 1965 [63].

The focus of this research is on capacity planning for the virtual and physical data centres which based on application workloads. Capacity planning is a process of resource management to ensure that sufficient resources are allocated to satisfy the required level of computing demand [37]. In virtualised environments, two types of capacity planning are conducted. Firstly, virtual data centre capacity planning defines the VM resource requirements to the PiP to host the application. This process is performed by the ViP and strongly depends on the fluctuation of application workloads [21, 37]. Secondly, the physical data centre capacity planning allocates physical resources to virtual machines, based on the VMs' resource requirements. This process is performed by the PiP in the physical data centre. It is essential that there are sufficient resources in virtual and physical data centres. This can be done by performing effective capacity planning, which will avoid over-provision of resources whilst at the same time maintaining the required level of performance.

Operating a physical data centre can be very costly: it requires a suitable

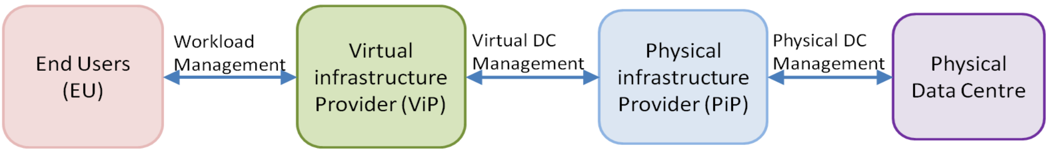


Figure 1.1: The relationship between parties involved for the management phases by implementing virtualised environments.

place to house computing equipment, which consumes a substantial amount of power both in functioning and for cooling. Sufficient physical resources need to be provided by PiP as required by ViP to create VMs. On the other hand, the ViP is still responsible for the capacity planning of their virtual data centre, to ensure the acceptable performance of applications running in the virtual environments. Techniques such as auto-scaling [27, 54, 86] and cloning [53, 79] enables VM capacity to be more elastic, which allows the VM to be resized on demand. These techniques need to be considered in virtual data centre capacity planning. The elastic virtual machines are created using limited physical resources. Therefore, there is a need to consolidate virtual machine demand into the available physical resources as a process in physical data centre capacity planning.

1.1.1 Capacity Planning in Virtualised Environments

Three management phases can be identified in performing capacity planning in a virtualised environment; i) application workload management by capturing application workloads and the usage of resources that they have triggered for *resource requirement analysis* and estimation of future workloads and resource requirements [9, 21, 25, 34, 65, 69], ii) virtual data centre management by estimating virtualised resource requirements [4, 21, 25, 28, 55, 69] and iii) physical data centre management by estimating physical resource requirements [27, 28, 46, 78]. Figure 1.1 shows the relationship between the identified phases and the related parties in performing capacity planning by implementing virtualised environments.

Workloads generated by end users utilise the virtual machine resources in virtual data centres. The ViP needs to provide sufficient resources with the correct specifications of virtual machines offered by the PiP. This is to assure the adequate performance of the hosted applications and minimise the cost by not over-provisioning the resources. Previous work focuses only on a particular application, while in practice multiple applications act as services managed by a ViP. Each application's workload is different depending on its behaviour, software design and the technologies it builds on. Moreover, resource requirements are initiated by application workloads. Capturing resource requirement based on the application workloads is required in capacity planning process.

Existing capacity planning approaches, focus either on estimating VM resource requirements based on single application workloads or on managing physical resources in physical data centres. Abrahao et al. [4], Delimitrou and Kozyrakis [25], Mark et al. [55], Roy et al. [69] proposed operational models for capacity planning in virtual data centres. On the other hand, Dougherty et al. [27], Ejarque et al. [28], Khazaei et al. [46], Sun et al. [78] proposed cost and operational models for physical data centres. To enable comprehensive capacity planning and end-to-end traceability of resource requirements, it is important to integrate these phases under a common framework. Integrating application workload, virtual capacity and physical capacity management in a unified framework can deliver significant benefits in terms of automation and traceability. A systematic and transparent process in performing capacity planning based on known application workloads assists the prediction of future resource requirement in virtual and physical data centres.

The main computing resources of interest in capacity planning are: CPU, memory, storage, and bandwidth (incoming and outgoing network) use. Certain works have combined all those resources as a unit [4, 69] while others study only a selected component or a selection of specific components. Ejarque et al. [28], Tan et al. [80] focus on CPU and memory usage and Sun et al. [78] include storage in their study. Additionally, Delimitrou and Kozyrakis [25] explore the combination of four components by extending their studies to include network resources.

1.1.2 Model Driven Engineering

MDE is a well known approach to software engineering which advocates constructing precise models using Domain Specific Modelling Languages (DSMLs). These models are managed using automated tools such as transformation, analysis and validation engines, and model-to-text generator [70]. MDE enables reusability of the DSMLs and model management techniques. The use of models to represent abstraction of the real world has been widely practised in many areas of knowledge [45]. To improve understanding of the concepts within the domain, abstraction is used to reduce their complexity.

In this work, Domain Specific Modelling (DSM) is utilised to facilitate rigorous specification and automated analysis in the context of capacity planning. A DSML-based approach to support capacity planning in virtualised environments is proposed in this thesis. The novelties are:

i. Domain Specific Modelling Languages (DSMLs) that allow workloads and resources to be precisely captured using models.

ii. A transparent, automated and repeatable MDE process for generating predictions for resource demand from estimated workload models.

iii. The MDE process, which exploits model transformation, comparison and merging, is modularised so that it can be configured for different kinds of capacity planning applications and technical infrastructures.

1.2 Motivation

Current capacity monitoring tools are able to collect, store and display resource metrics over time in order to help operators to identify resource usage patterns. However, this is not always sufficient for virtual environments; resource usage patterns need to be associated with the workload that is being processed at the time to enable better-informed capacity planning. Sets of DSMLs together with model management techniques are proposed to facilitate the capacity planning process. In the domain of interest of this study, the information required to populate the models can be gathered from logs recording the *workload* processed by an application running in a virtualised environment, and logs recording the *resource usage* of the VM that hosts the application. Log analysis is conducted in the initial phase to capture application workload with the resource usage and define the workloads estimation model. Later, capacity planning for virtual and physical data centres can be performed with a traceable approach based on the workload models and the relationship between workloads and the resource usage they have triggered.

1.3 Research Scope

This study focuses on the design of DSMLs and model management techniques to facilitate capacity planning in virtualised environments. The scope of this work is as below:

i. Capturing the application-specific relationship between application workload with the resource usage and to define workload estimation.

ii. Estimating virtual resource requirements for predicted workloads for a defined time period.

The research provides a fully integrated solution for capacity planning process in virtualised environments using DSM. A number of DSMLs are developed, implemented and evaluated in order to capture application workloads and express virtual machine resource requirements. The selection process of virtual machine packages offered by PiP and the integration of the virtual data centre management phase with the physical data centre management phase is briefly discussed and suggested as future work.

1.4 Research Hypothesis

The hypothesis of this work is that MDE and DSML techniques can be used to support *modular* and *reusable* capacity planning in virtualised environments. Capacity planning is a process to ensure that resources in a data centre are sufficient to support its computational needs. One criteria of capacity planning is to ensure the infrastructure resources are sufficient with the demand of workloads. Capacity planning involves estimation of infrastructure resources such as storage, processors, memory and bandwidth over a future period. In this work, capacity planning is restricted to the estimation of CPU, memory, incoming network and outgoing network in virtualised environments that run a single application. A Virtualised environment in a data centre is implemented to optimised computing resource utilisation by sharing computing resources. Chapter 2 provides detailed discussion on capacity planning and virtualisation. In this context:

i. *modular* means that every step of the capacity planning process is self-contained and the structure of its expected inputs and outputs is specified in a rigorous manner;

ii. *reusable* means that steps/components can be shared between different capacity planning processes.

DSML enables a high level of abstraction by allowing the specifications being captured in a modular manner. The details of the specifications are captured with concrete models conforming to the DSML. The modularity in DSML enables reusability, where the models and its activities can be utilised to facilitate capacity planning of different applications. Model management techniques can be used to manipulate the details captured in the model. These features of MDE can be utilised to automate a transparent capacity planning process to estimate the resource requirements based on the predicted workloads of an application.

Domain specific modelling (DSM) has been adopted in many fields such as automotive [45], telecommunications [22], transportation [81] and many other safety-critical systems. There is possibility of utilising MDE in data centre and, in this work, DSM is utilised in capacity planning to facilitate rigorous specification and the automated analysis of workloads.

1.5 Research Objectives

In MDE, models are used as core artefacts that drive the entire software development process. The models represent abstractions of the domain of interest. These domain-specific models raise the level of abstraction by specifying the solution in a language that directly uses concepts and rules from the domain. MDE techniques such as domain-specific modelling, model-to-model, text-to-model and model-to-text transformation are the MDE features utilised in this work. The modularity of those techniques makes the process in the domain self-contained within models. This also promotes reusability, saving effort from repetitive tasks and reducing the semantic gap between the problem and the implementation.

The use of MDE in capacity planning in virtualised environments raises four research objectives. The objectives which drive the research are as follows:

i. To identify the capacity planning phases processes in virtualised environments based on a systematic literature review.

ii. To design systematic and model-based processes with a focus on the initial capacity planning phases.

iii. To design and implement DSMLs and model management techniques to support the identified processes.

iv. To evaluate the modularity and reusability of the proposed DSMLs and model management techniques.

Systematic literature reviews were conducted to explore the domain by identifying the phases involved in performing capacity planning in a virtualised environment. A high level framework of capacity planning which includes the related phases was designed based on the scope of the research. The literature review defines the concepts to include in a systematic process, current practices and the possibility of utilising MDE. Therefore, the literature review was conducted as the first objective. As the second objective, systematic processes were designed by further exploring the domain within the scope. The third objective is to design the DSMLs and model management techniques as MDE solutions to support the processes identified under the second objective. The proposed MDE solutions were implemented with selected tools in a virtualised environment as implementation. Case studies were used to apply the proposed MDE solutions. The fourth objective is to evaluate the modularity and reusability of the proposed MDE solutions with an additional case study. The plan for achieving the objectives of this research is presented as the research methodology in the following section.

1.6 Research Methodology

Experimental and exploratory research methods are applied in performing this research work. In order to achieve the research goals, nine main steps were performed. Figure 1.2 shows the research methodology and flow of the steps.

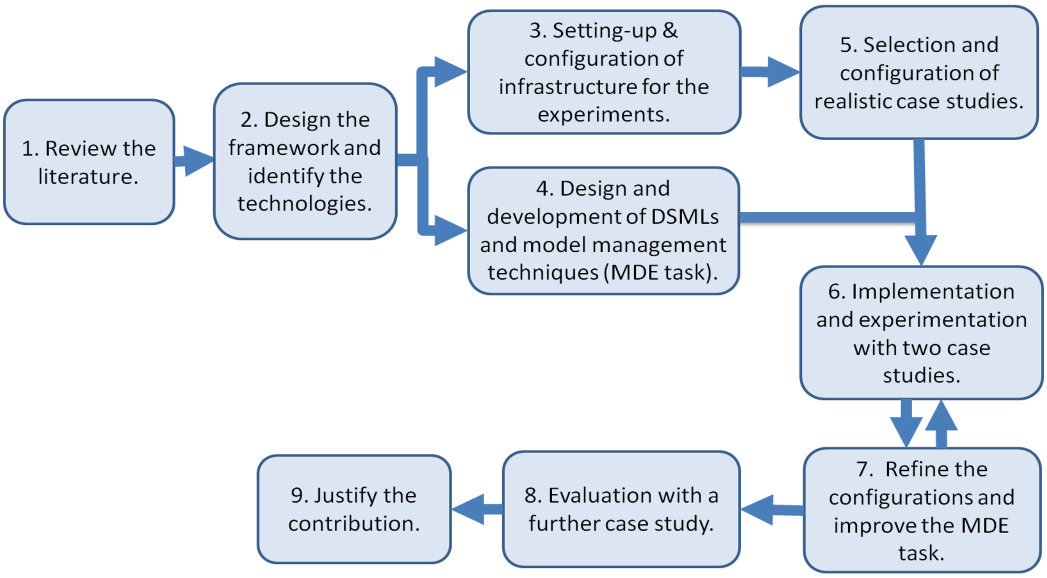

1. Review the literature.
2. Design the framework and identify the technologies.
3. Setting-up & configuration of infrastructure for the experiments.
4. Design and development of DSMLs and model management techniques (MDE task).
5. Selection and configuration of realistic case studies.
6. Implementation and experimentation with two case studies.
7. Refine the configurations and improve the MDE task.
8. Evaluation with a further case study.
9. Justify the contribution.

Figure 1.2: Research Methodology.

Detailed descriptions of the steps are as follows:

1. A literature review on the history of data centres, virtualisation, resource management and capacity planning. In addition, a literature review on model driven engineering with a focus on domain specific modelling and model management techniques were also carried out.

2. A framework for capacity planning process in virtualised environment was developed. At the same time, the hardware and software requirements used to facilitate the development, implementation and evaluation of the research work were identified.

3. The infrastructure to perform the experiments for implementation and evaluation was set-up. The configuration of the infrastructure to log information regarding the application workloads and resource usage as input data was performed.

4. In parallel to setting up the infrastructure, design and development of the DSMLs and the related model management techniques were conducted.

5. A selection of realistic case studies to be used for implementation and evaluation were identified.

6. Implementation of the developed DSMLs and model management techniques were performed by repeating the experiments with two case studies.

7. The developed DSMLs and model management techniques were refined. This process was repeated with several simulations with the selected case studies to improve the metamodels and the model management techniques. At the same time, the infrastructure used in this work was also improved to produce the required logging information. The applications from the case studies were reconfigured to provide accurate log information and relevant resource utilisation.

8. The experimental evaluation method was used by applying the proposed MDE solutions with a third case study. Step 7, stated above, is an initial phase of the experimental evaluation method and is called the exploratory phase. The system requirements and technical facilities were identified and developed in the exploratory phase. In the second phase, an evaluation was conducted to measure the reusability and modularity of the proposed MDE solutions with an additional third case study. For the evaluation case study, a resource-intensive virtualised web application was used, which required extending the core

capacity planning domain-specific languages and transformations. Additionally, the extensibility, completeness and prediction capability of the proposed MDE solution were evaluated.

9. The usefulness of the proposed solution in answering the research goals was assessed.

The steps in the research methodology are aligned to achieve the research objectives discussed in Section 1.5. Steps 1 and 2 were conducted to achieve the first objective. Step 4 was performed to obtain the second objective. Consequently, steps 3, 5, 6 and 7 were designed to achieve the third objective. Finally, steps 8 and 9 were implemented to achieve the fourth objective.

1.7 Research Outcomes

The proposed work is aimed at assisting capacity planning managers who are providing and/or using a virtualised environment to estimate future resource requirements in physical and virtual data centres. This is achieved by integrating three management phases in performing capacity planning in virtualised environments. Sets of DSMLs and model management techniques are proposed as a traceable and modular solution. It is anticipated that using DSML models to specify workloads will render capacity management more flexible, precise and effective. The practicality of using the suggested approach is measured by using two cases studies during implementation and evaluated with a third case study.

1.8 Thesis Structure

The remainder of this thesis consists of eight chapters. Chapters 2 and 3 draw upon the scientific research literature to provide background and motivation for the research proposal presented in Chapter 4. Chapter 5 discusses the design of the DSMLs and model management techniques as MDE solutions to automate the identified capacity planning processes. The application of MDE solutions was carried out on two case studies and were demonstrated in Chapter 6. To further evaluate the practicality and reusability of the MDE solutions, a third case study was conducted. Chapter 8 concludes this report and provides directions for future work. A brief description of each chapter follows:

1.8.1 Chapter 2: Literature Review

This chapter provides a comprehensive review of related literature. It covers the evolution of data centres up to the current paradigm called "cloud computing". Server virtualisation as a key enabler technology in resource sharing are discussed. Also, server utilisation and capacity planning in virtualised environments are presented. The research gap is identified and a possible solution is outlined.

1.8.2 Chapter 3: Model Driven Engineering

This chapter provides an overview of Model Driven Engineering (MDE) and DSM. The use of DSMLs and tools related to modelling is discussed. Finally, the overall review of the chapter is presented with a possible solution for the research challenges identified in Chapter 2, by utilising model driven technologies and techniques.

1.8.3 Chapter 4: Domain Analysis

The overall aim of the proposed research is outlined along with the research plan. The proposed framework to facilitate capacity planning in virtualised environments is elaborated upon.

1.8.4 Chapter 5: Design of MDE Solutions

The detailed design of the DSMLs and its associate model management techniques are discussed. Two processes, ReRA and ViRR were automated with the proposed MDE solutions.

1.8.5 Chapter 6: Applications of MDE Solutions

This chapter discusses the application of the proposed MDE solutions with two cases studies in a virtualised environment. The design of the infrastructure to perform the experiments is also presented. The outcome of applying the MDE solutions for each case studies by reusing and extending the proposed MDE solutions are discussed.

1.8.6 Chapter 7: Evaluation

The evaluation to demonstrate the validity of the proposed solution was conducted with a third case study. The assessment process and the results are presented in this chapter.

1.8.7 Chapter 8: Conclusion

The conclusion and future work are discussed in this final chapter. The contributions and limitations of the proposed solution are discussed and directions for future work are identified.

Chapter 2

Literature Review

2.1 Introduction

This chapter provides an overview of resource management in the field of computing generally, and in virtualised environments specifically. The chapter begins with a discussion on data centres; we argue that the techniques used for resource management processes are related to the evolution of data centres. It discusses the concept and the key terms of virtualised environments which later inspired the popularity of cloud computing as utility computing. This chapter generally focuses on resource management activities in a data centre where capacity planning is a main concern. The chapter concludes with a review of possible solutions for capacity planning in virtualised environments.

2.2 Data Centres

A data centre is a component which consists of a number of servers and facilities for computing. Data centres are also referred to as server rooms and server farms. As per Telecommunications Infrastructure Standard for Data Centres (TIA-942)[1], the evolution of data centres started with the creation of the first computer UNIVAC in the 1940s. Later in the 1950s, IBM introduced its mainframe computer [44]. The first distributed computing system was used in an airline reservation system which went into full operation with two

[1] `http://www.adckrone.com/eu/en/webcontent/support/pdfs/enterprise/generic/102264be.pdf`

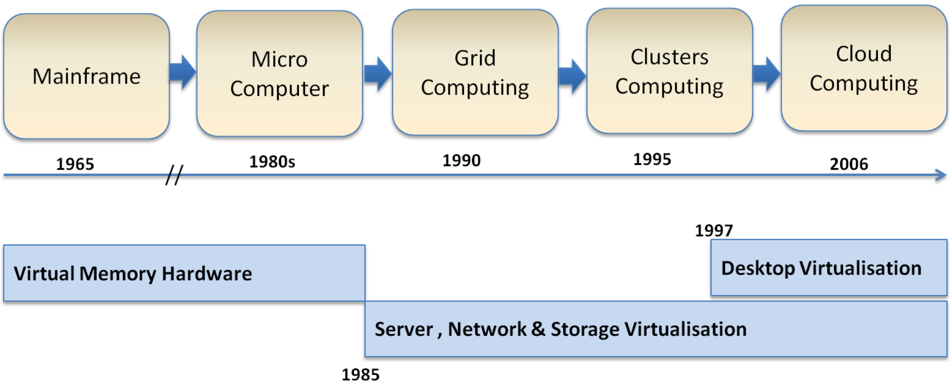


Figure 2.1: An Evolution of Data Centres

powerful IBM mainframes, sixteen data storage devices and 1,000 terminals in 1965 [19].

The evolution of data centres from the era of mainframes until today's cloud computing is illustrated in Figure 2.1. Mainframes were very powerful machines and were also very expensive to operate and maintain. The mainframe provided the full computing facilities and the users accessed the mainframe through terminals. Terminals are input and output devices communicating with the mainframe without computer processing. The next generation of computing involved microcomputers, which were initially only able to run office usage applications. The main applications still ran in the mainframe and microcomputers replaced the terminals. Then the technology gradually moved to grid computing, commodity clusters, virtualised clusters, and more recently to cloud computing [32]. Virtualisation was introduced in 1965 with virtual memory hardware virtualisation [43, 44]. Since that, the subsequent technology in the evolution of data centres has enhanced the virtualisation concept as a backbone for resource optimisation and cost reduction.

Grid computing, commodity clusters, virtualised clusters, and cloud computing are categories of distributed systems [32]. In distributed computing, end users access the applications hosted in the computing machines located in the data centres. Figure 2.2 shows an overview of distributed computing which include cluster, grid, supercomputer, Web 2.0 and cloud computing [32]. Cloud and Web 2.0 are more towards service oriented and highly scalable. However cluster and supercomputer are application oriented but supercomputer is more scalable compare to cluster. Cluster computing focuses on traditional application-oriented systems. Where else, grid computing is used in both application-oriented and service-oriented systems. Resource sharing

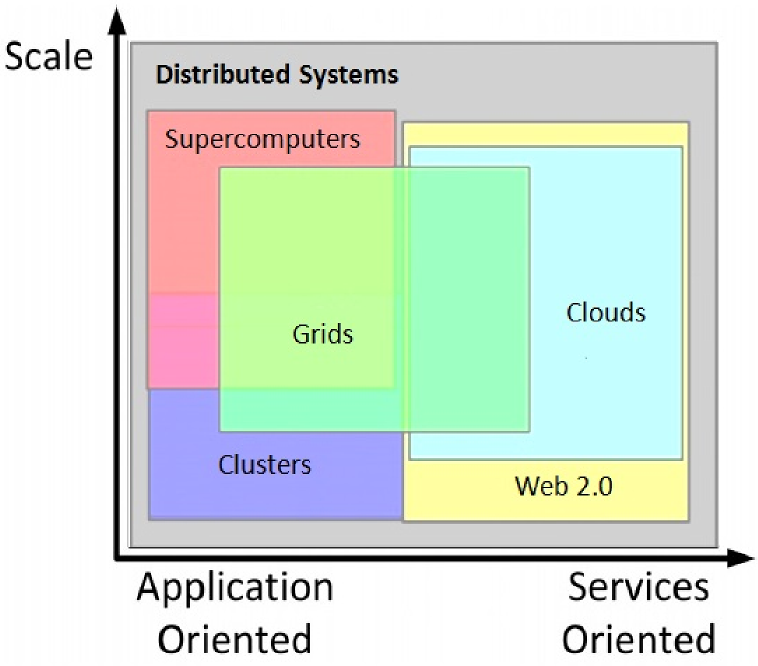


Figure 2.2: Overview of Distributed Computing [32].

features of grid computing are utilised by both cloud and cluster computing. On the other hand, there are differences between cloud and grid computing in terms of the business model, architecture, resource management, programming model, application model and security model [32]. Virtual and psychical resource provisioning are two key steps in executing application requests under grid resources in cloud computing [67].

This section discusses of cluster computing, grid computing and cloud computing according to Foster et al. [32]. Cluster computing makes use of multiple servers which are interconnected. These form a single, highly available system, whereas grid computing enables federated dynamic resource sharing. This can also be considered as a combination of several clusters. Resources of grid computing are accessible through a network by several institutions. These resources are normally used for a common use by the institutions. Cloud is mainly connected through the internet and supports various types of services such as Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS).

The subjects of implementing virtualisation in distributed computing and optimised computing resource utilisation by sharing computing resources are discussed in the following sections.

2.2.1 Virtualisation

Virtualisation has existed since the beginning of the mainframe era as illustrated in Figure 2.1. In 1965, mainframes with virtual memory hardware technology were used in industry [43]. Later after 1985, server virtualisation, network virtualisation and storage virtualisation emerged to optimise resource utilisation. Desktop virtualisation was introduced in 1997 where multiple operating systems were able to operate in a desktop. Section 2.2.2 discusses these virtualisation types in detail.

> *Virtualisation is a framework or methodology for dividing the computing resources of a computer into multiple execution environments. Hardware and software partitioning, time-sharing, partial or complete machine simulation, emulation, quality of service, and many others are the concepts or technologies related to virtualisation [43].*

2.2.2 Type of Virtualisation

Four common types of virtualisation are listed below [43]:

i. **Server Virtualisation**
Server virtualisation facilitates resource optimisation where a number of virtual machines run independently on a single server. This reduces the requirement to have many physical servers which directly reduces the cost of purchasing and operating the physical servers. Furthermore, electric power consumption also decreases with fewer physical servers running with optimised resource utilisation.

ii. **Virtual Networks**
Create an illusion that the end users are connected directly to an organisation network and resources, without a direct physical connection to the internet. Virtual networks are also called virtual private networks (VPNs).

iii. **Virtual Storage**
Storage virtualisation enables end users and applications to access scalable and redundant physical storage.

iv. **Desktop Virtualisation**
Multiple operating systems are run on a single computer and allows a user to switch between them.

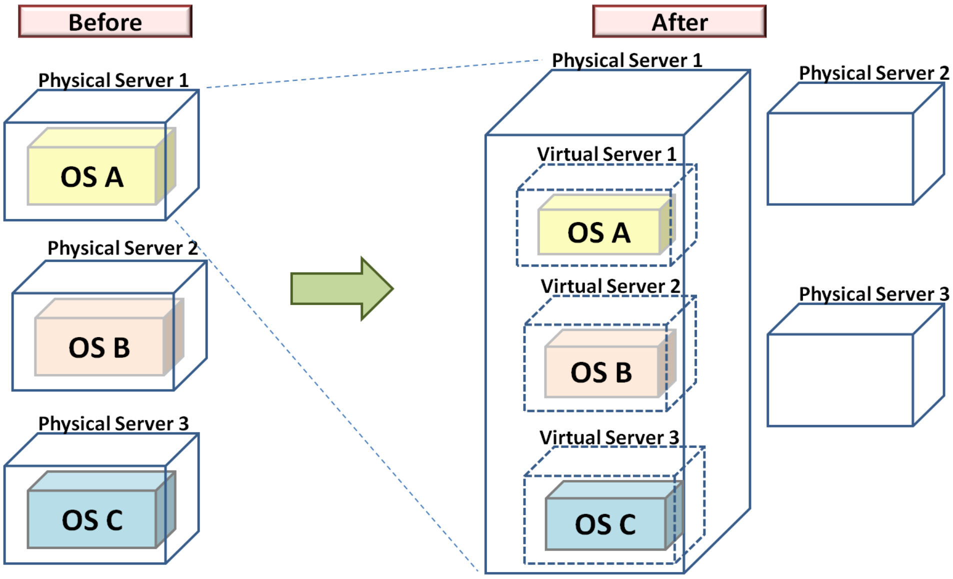


Figure 2.3: Before and After Implementing Server Virtualisation.

2.2.3 Server Virtualisation

In server virtualisation, a single physical server appears as many virtual servers [43]. Each virtual server runs the same or different operation systems individually. This provides greater computing resources utilisation with less computing equipment, lower power consumption and also supports multiple operating systems. The operating systems and the applications running in it, are concurrently accessible with a selected network configuration. Figure 2.3 illustrates the differences before and after implementing server virtualisation. Fewer physical servers are required to operate an equal number of operating systems. At the same time, this maximises physical server resource utilisation and minimises the number of physical servers.

2.2.3.1 Components of Server Virtualisation

Server virtualisation, mainly consists of three components; host, guest and hypervisor [66]. These components are illustrated in Figure 2.4. The host is the physical server or *physical machine* (PM) used to set the virtualisation environment. The guest is the VM that runs within the host. The guest shares the physical resources of the host with other virtual machines running in the same host and with the host itself. However, it behaves as an individual server. A hypervisor is also called a virtual machine monitor (VMM) and it

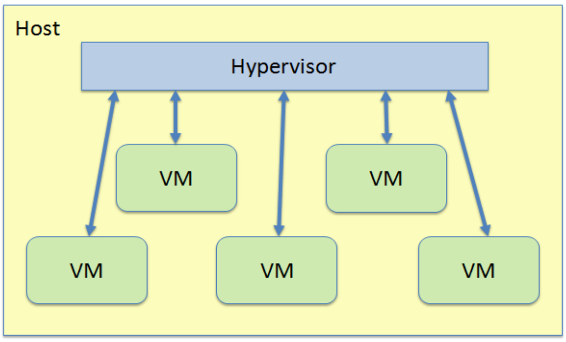


Figure 2.4: Server Virtualisation Components.

is installed on the host to manage the virtualised environment. It is a piece of computer software, firmware or hardware that is installed on the host, which creates and runs virtual machines. There are mainly two types of hypervisor; *Type1* and *Type 2*. Figure 2.5 illustrates the differences between these types.

A Type1 hypervisor is installed onto bare metal or directly on the hardware platform (i.e. on hardware). The hypervisor runs directly on the base operating system of the host's hardware. It controls itself and also monitors the guest's operating systems. The hypervisor supports two types of *full virtualisation*; *share kernel virtualisation* and *kernel level virtualisation*. Shared kernel virtualisation takes advantages of the Linux and UNIX operating systems. The system kernel is shared between the guests and the host. Examples of Type1 hypervisors supporting shared kernel virtualisation are Linux VServer, Solaris Zones and Containers, FreeVPS and OpenVZ. Kernel level virtualisation leverages the latest generations of CPUs from Intel and AMD (x86 processor). In kernel level virtualisation, the hypervisor itself manages the host hardware and guest operating systems. Xen, VMware ESX Server and Microsoft's Hyper-V are some hypervisors that provide kernel level virtualisation.

A Type2 hypervisor is installed onto an existing operating system of the host and supports *paravirtualisation*. Under paravirtualisation, the kernel of the guest operating system is modified specifically to run on the hypervisor. Therefore, the hypervisor performs the task on behalf of the guest kernel. VMware Workstation, Microsoft Virtual PC, and Oracle VirtualBox are examples of Type2 hypervisor.

In terms of performance, A Type1 hypervisor provides superior performance since there compared to Type2 [20]. However, studies conducted by Adams and Agesen [5] for x86 virtualisation show that Type2 is better in terms of

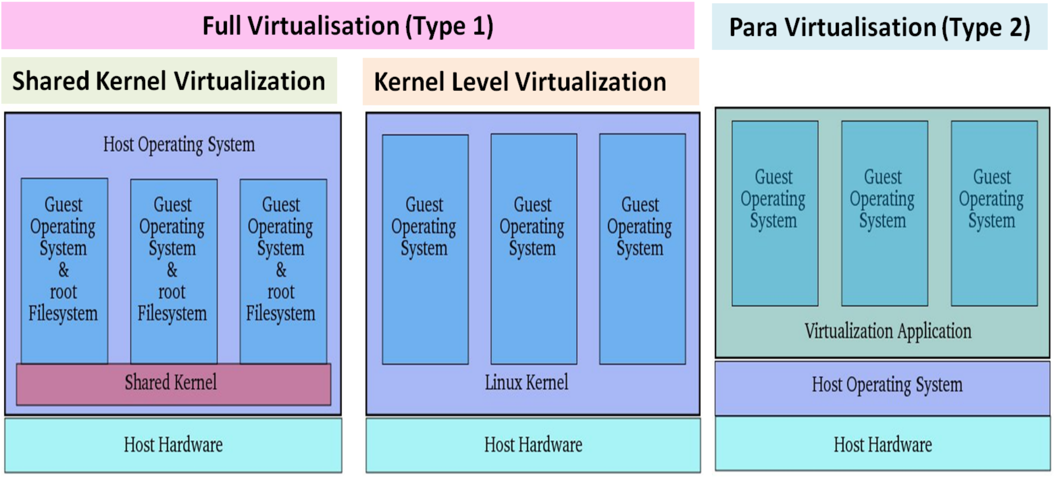

Full Virtualisation (Type 1)
Para Virtualisation (Type 2)
Shared Kernel Virtualization
Kernel Level Virtualization
Host Operating System
Guest Operating System & root Filesystem
Guest Operating System & root Filesystem
Guest Operating System & root Filesystem
Shared Kernel
Host Hardware
Guest Operating System
Guest Operating System
Guest Operating System
Linux Kernel
Host Hardware
Guest Operating System
Guest Operating System
Guest Operating System
Virtualization Application
Host Operating System
Host Hardware

Figure 2.5: Type of Hypervisor. Adapted from [2].

performance for memory management. Selection of an appropriate hypervisor as a tool to facilitate virtualisation is determined by many factors such as cost and performance.

2.2.4 Cost-Efficiency in Data Centres

Managing a large data centre is very expensive [50] and the PiP needs to take all necessary actions to improve cost-efficiency. Virtualisation and server consolidation are solutions for maximising server utilisation [75]. Server virtualisation optimises physical server's resource utilisation by having several virtual machines running on a single server, thus reducing the number of switched on physical servers in data centres. Intelligent resource management reduces power consumption in data centres by having fewer switched on servers and as such requiring less cooling.

Figure 2.6 shows the average cost in data centres based on percentages of cost components discussed by Greenberg et al. [38] and the total cost of data centres in the world in 2007 [50]. *Servers* include hardware such as CPUs, memory and storage disks. *Infrastructure* refers to power distribution units (PDU) or electric wiring and cooling facilities in data centres. Also, *Power Consumption* represents electricity costs and *Network* refers to links, bandwidth and equipment for networking purposes.

According to Quiroz et al. [67], *VM Provisioning* consists of creating VM instances to host each application request, whilst matching the specific char-

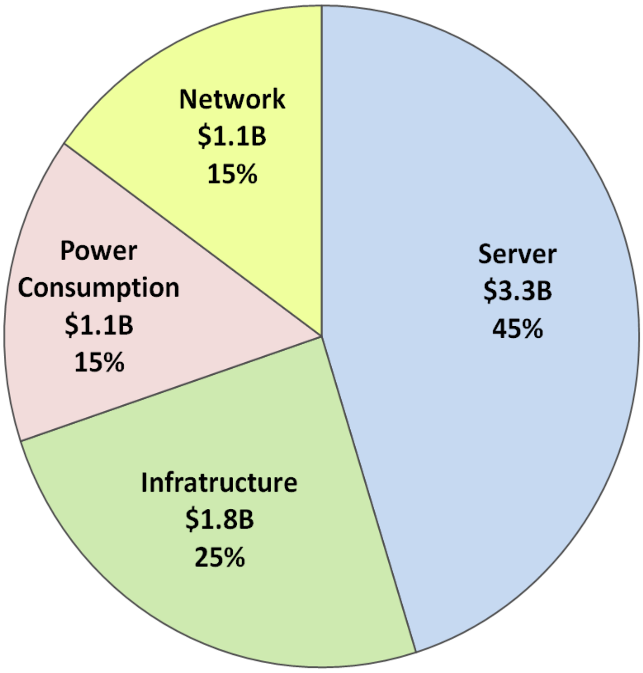


Figure 2.6: Average Cost in Data Centres.

acteristics and requirements of a request. Mapping and scheduling these requests onto distributed PMs is called *Resource Provisioning*. Figure 2.7 illustrates virtual machine and physical machine resource provisioning in data centres [67].

To optimize PM utilisation, VMs can be allocated to PMs based on the workload patterns stated in Service Level Agreements (SLAs); this technique is known as SLA-oriented resource allocation [17]. The ViP and PiP need to

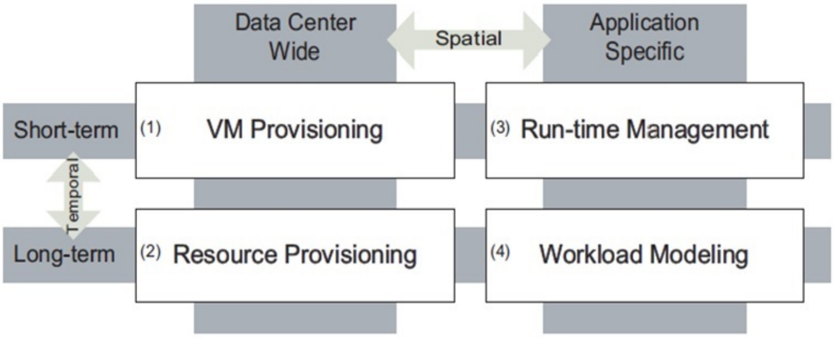


Figure 2.7: Data Centres Virtual and Physical Resource Provisioning [67].

estimate virtual and physical resource requirements accordingly. Resource demand is commonly estimated based by recording and extrapolating on observed (real) workloads [65].

2.2.5 Server Utilisation

Server utilisation is the extent to which resources provided by a server are actively used. Commonly, the percentage of processor being used is measured to indicate the usage level. Disk, memory and network are also important metrics in measuring server utilisation. Studies show that server utilisation in real world data centres is estimated between 5% and 20% [7, 8]. Furthermore, an observation for 5000 servers for 6 months showed that the servers are rarely completely idle and these servers operate between 10% to 50% of their maximum utilisation level [10]. This shows that servers in real data centres are typically underutilised. However, these underutilised servers still consume a lot of electricity to stay on [18, 75]. Additional energy is also needed for cooling the heat produced by underutilised servers. Server virtualisation is a widely adopted approach to maximise server utilisation [75].

2.3 Resource Management

Resources in a data centre are separated into physical and virtual [67]. VMs as virtual resources are completely isolated server installations within a PM (host). A VM operates like a real standalone server, while in reality it shares physical resources of the host. The main advantage of a VM is that it can be migrated to another physical server and mirrored as much as needed for multiprocessing. In order to conduct resource management, capacity planning is the initial activity. Since a virtualised environment has two types of data centres (physical and virtual), there is a requirement to conduct capacity planning by the respective parties in these data centres. Physical data centre capacity planning is determined by estimating the resource requirements for running various specifications of VMs. This demand for VMs is triggered by end users' workload and virtual data centre capacity planning allows the VM's resource requirements to be estimated.

2.3.1 Capacity Planning

Capacity planning (in general) is a process to ensure that resources in a data centre are sufficient to support its computational needs. In this process, available system resources are observed, performance is measured, and resource usage patterns are extracted to forecast the resources that need to be allocated to serve future workloads [74]. Capacity planning is an important activity to estimate the needs for computing resources and their cost. Traditionally, purchasing computing resources was a solution to overcome resource limitations in a data centre. Sometimes, leasing computing resources is also appropriate if the need is only for a temporary period and leasing is more cost-effective than buying. Mechanisms are implemented to estimate the future resource requirements as a capacity planning process. This procedure is not transparent, this is discussed further in the work of Allspaw [6] wherein he shares his experience in performing capacity planning.

Capacity planning in virtual environments aims to ensure that allocated virtual computing resources such as CPU, memory, storage and network bandwidth will be sufficient to support future computational needs. In this process, available system resources are observed and performance is measured [6, 74]. Also, resource usage patterns are determined to forecast the resources that need to be allocated to serve future workloads in compliance with the service's Quality of Service (QoS) requirements [65]. To achieve this, it is necessary to identify incoming workloads, to monitor resource usage, and to associate resource usage with the workloads that triggered it.

2.3.1.1 Resource Estimation and Prediction Techniques

Several approaches have been proposed in order to estimate resource requirements for different types of applications. A synthetic workload generator tool proposed in [9] evaluates the performance of VMs by performing synthetic requests on multi-tier web applications. In [21], a benchmark model is proposed to estimate the number of VMs required for hosting media stream applications based on their memory and disk requirements. Roy et al. [69] propose a resource allocation algorithm that estimates the number of required VMs based on statistical predictive techniques by considering the challenges of auto-scaling. Microscopic and macroscopic approaches to predict resource consumption for data centres by statistically characterising resource usage patterns are proposed in [80]. The microscopic approach focuses on resource usage prediction for a specific node; it demonstrates that using both CPU and memory usage data can improve the forecasting performance compared to a baseline method of calibrating CPU usage.

Web Application Type	**Workloads Generator**	**Workload Estimation**	**Parameter Access from Log File.**
Multi-user	Workload Specification Language (WSL) [9]. SPECweb99, SURGE, SWAT and httperf.	Probabilistic Finite State Machine and Maximum Likelihood Estimation [9].	Input, output, states, transitions and probability of a transition.
	Jean 2 model	Semantic descriptions [28].	*not applicable*
	KOOZA [25].	Markov Chain Models for storage, processor and memory. Simple queuing for network.	Storage: block size, type, randomness and inter-arrival times. Processor: CPU utilization. Network: arrival-rate.
	not applicable	Autoregressive moving average method (ARMA) [69].	Number of visits to a single page from the total number of customers, number of machines providing the service demand and the think time for clients.
Data Intensive	*not applicable*	Kernel Canonical Correlation Analysis [34].	Map time, reduce time, total execution time, map output bytes, HDFS bytes written, and locally written bytes.
Media streaming	Medisyn [21]	Mathematical model in a tool called MediaProf.	Time, file name, duration, file size, available users' bandwidth and elapse end time.

Table 2.1: Techniques and Tools for Workload Generation and Resource Estimation.

Various parameters are accessed from the application logs for each category of workload, since the nature of each category is different. Furthermore, it is difficult to establish generic workload prediction mechanisms. This is because the behaviour of the users of each application as well as its architecture and implementation style make each workload pattern unique.

Previous workload patterns and their associated system parameters are essential to estimate future workload. For this purpose, parameters required for each category of applications have been compiled in Table 2.1. Mainly statistical approaches have been used to estimate future workloads such as KCCA [34], PCA [80], regression analysis [69], and Maximum Likelihood Estimation [9]. Table 2.1 summarises the techniques and tools used for workload generation as well as the workload estimation methods and parameters used in these works. Besides these tools, JMeter[1] is an open source load generation tool used to simulate workload for multi category of applications and is widely used for performance testing.

Overall, statistical prediction techniques were used to estimate resource requirements as demonstrated by Islam et al. [42], Weisberg [84]. The ideal method depends on the relationship between resource usage patterns and the currently active workload.

2.3.1.2 Resource Usage Monitoring Tools

Capacity planning requires analysts to monitor resource usage over a number of dimensions: in terms of load (e.g., requests), over time, over outputs, etc. Such analyses rely on the tools available for monitoring resource usage. These include Ganglia[2], VBoxManage[3], CloudWatch[4] and Solarwinds[5]. Such tools are able to collect information related to the usage of resources such as CPU, memory, storage and network by a virtual machine and at predefined intervals (e.g. every 1 second). Furthermore, the operating systems of the VMs are also able to record resource utilisation through utility programs (e.g. the UNIX “top” program) [6]. This information can be stored for further processing and visualisation.

[1] http://jmeter.apache.org/

[2] http://ganglia.sourceforge.net/

[3] https://www.virtualbox.org/

[4] http://aws.amazon.com/cloudwatch/

[5] http://www.solarwinds.com/uk/

Capacity Planning	**Cost Model**	**Operational Models**
Virtual Data Centre	Reward and penalty based on respond time [4].	Normal and surge operations model [4]. Markov Chain Models; for storage, processor and memory and simple queue model for network [25]. Decision retrieving media file from memory or disk [21]. Minimum resource requirement [28]. Gradually increase the number of machines to identify the right number of resources required by calling the Mean Value Analysis [69]. Kalman filter, double exponential smoothing, and Markov prediction [55].
Physical Data Centre	Customer Priority model [28]. Cost with energy consumption and cost of VM creation [27].	Surplus resource distribution [28]. Prebooted and preconfigured VM instances with common feature [27]. Markov chain technique [46]. Multi-dimensional Resource Integrated Scheduling algorithm [78].

Table 2.2: Physical and Virtual Data Centre Capacity Planning.

2.3.2 Physical and Virtual Data Centres Capacity Planning

To be cost-effective, a sufficient amount of resources need to be allocated for the installation of VMs. This is to ensure an acceptable level of performance for the hosted applications and to minimise the over-provisioning or under-provisioning of resources. A PiP needs to run a minimum number of physical servers with optimum utilisation to fulfil a ViP's VM demands with the agreed response time for VM creation. The methods implemented in the environments need to be considered in capacity planning.

Several methods for performing capacity planning from the ViP and PiP perspective are outlined in Table 2.2. The methods are divided into cost and operational models for respective capacity planning. Capacity planning in virtual data centres is performed by ViP whereas the computing resources are virtually managed based on application workloads. On the other hand, PiP performs capacity planning in physical data centres based on VM demand triggered by application workload.

The basic computing resource components in capacity planning are CPU, memory, storage, disk and network use [6]. Certain works have combined all those resources as a unit, such as Abrahao et al. [4], Roy et al. [69]. On the other hand, the following studies only selected a component or a selection of specific components. Work presented in Ejarque et al. [28], Tan et al. [80] focuses on CPU and memory usage, Sun et al. [78] also include storage in their study, while Delimitrou and Kozyrakis [25] explore the combination of four components by extending their studies to include network resources. Ganapathi et al. [34] explores the execution time of data intensive workloads for scheduling and resource allocation.

2.4 Software Performance Engineering

Software performance engineering (SPE) is a broad research area that is mainly concerned with to the study of methods and techniques for improving the performance of software systems. SPE is an adjacent research area to capacity planning, however, since this thesis focuses on predicting resource requirements for existing applications - rather than improving the performance of these applications - this section does not provide a broad review of SPE literature. Of particular interest however, is the work on frameworks such as the Palladio Component Model (PCM) and Descartes.

PCM is a mature meta-model for component-based software architectures (CBSE) which contains resource and workload models as SPE components [11]. Resource metrics in resource models are defined individually within the resource container (server) and workload is classified as open and closed. A general metamodel of software workloads is presented in PCM. The Descarters meta-model (DMM) is an architecture-level modelling language for Quality-of-Service and resource management of IT systems, infrastructures and services [51]. PCM and a subset of DMM have been used to predict software performance of online applications [15, 16].

In SPE, three categories of performance model parameters are defined and those are operational profile (probability software used by user), workload and resource demand [11, 82]. The workload is divided into open and closed workload based on user classification. An open workload is triggered by users that are not known in advance such as users of the World Wide Web while a closed workload is triggered by known users such as users of a specific software in a local area network. The resource demands represent the amount of resources that a software requires to complete its task.

In this thesis, workload is defined as a task that a software needs to process. The processing of this workload triggers the resource demand. The type of workload is categories according to the software. For example, in web applications, workloads are initiated through browser requests. Other software, such as database management systems or operating systems has their own workload types. In the approach proposed later on in this thesis, each application has its own workload DSML to capture the predicted workload, as opposed to a generic metamodel provided by methods such as PMC. A dedicated metamodel enables precision and conciseness while a generic metamodel eliminates the upfront metamodelling effort. In this work, the former were prioritised over the latter.

2.5 Chapter Summary

As the aim of this research is to provide a solution for capacity planning in virtualised environments which comprise virtual and physical data centres, a review of the evolution of data centres was presented. The literature review continued with a field survey on virtualisation. Resource management is closely related to the operational cost of data centres and capacity planning is required to estimate realistic resource requirement. Therefore, resource management and costing in data centres were discussed. A literature survey on capacity planning related to resource estimation techniques for web applications and tools to monitor resource usage was also conducted and approaches for capacity planning in virtual and physical data centres were reviewed.

The next chapter proceeds with a review of Model-Driven Engineering and Domain-Specific Modelling principles and techniques, which are used in the context of the capacity planning framework developed in this work.

Chapter 3

Model Driven Engineering

3.1 Introduction

This project investigates the potential of Model Driven Engineering (MDE) for supporting capacity planning. MDE is an engineering approach that treats models as first-class artefacts and advocates constructing rigorous models using Domain Specific Modelling Languages (DSMLs). There are many standards compliant with MDE principles, some of which are illustrated in Figure 3.1 are Model Drivel Architecture, Model Integrated Computing and Software Factories. In MDE, models are managed using automated tools such as transformation, merging and comparison engines, model-to-text and text-to-model generators. Figure 3.1 also unillustrated respective tools for the identified standards. In this thesis, Domain Specific Modelling (DSM) and model transformation are utilised for the purposes of capacity planning. This chapter presents an overview of the main concepts, tools and processes related to MDE.

3.2 Models in MDE

The main principle of MDE is "*Everything is a Model*" [12]. Therefore, it is useful to explore the definition of *model* before further exploring MDE. The Oxford Dictionary defines model as[1]:

[1] `http://oxforddictionaries.com/definition/model`

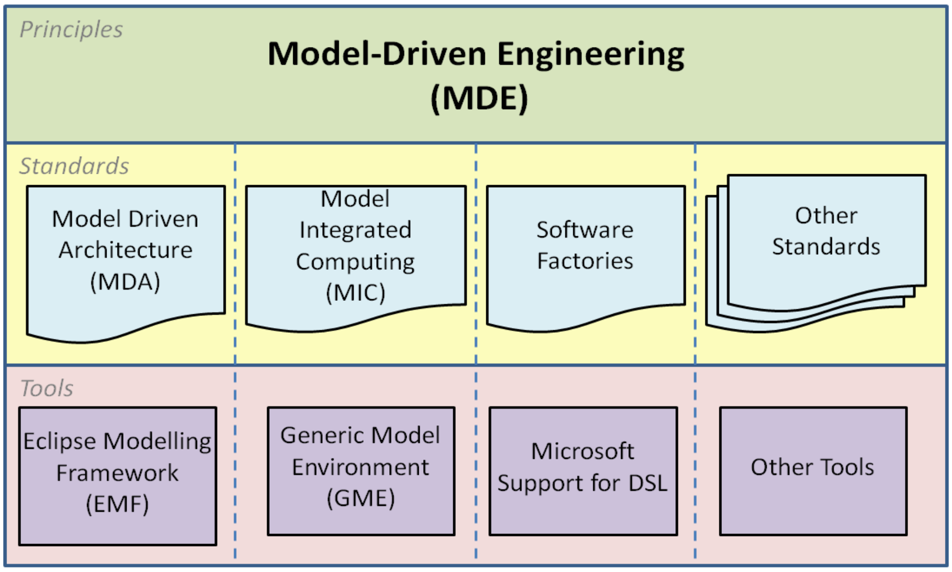

Principles
Model-Driven Engineering (MDE)
Standards
Model Driven Architecture (MDA)
Model Integrated Computing (MIC)
Software Factories
Other Standards
Tools
Eclipse Modelling Framework (EMF)
Generic Model Environment (GME)
Microsoft Support for DSL
Other Tools

Figure 3.1: MDE Standards and Tools. Adapted from [52].

> *[noun]*
> **1** a three-dimensional representation of a person or thing or of a proposed structure, typically on a smaller scale than the original.
> **2** a thing used as an example to follow or imitate.
> **3** a simplified description, especially a mathematical one, of a system or process, to assist calculations and predictions.
> **4** a person employed to display clothes by wearing them.
> **5** a particular design or version of a product.
> *[verb]* (models, modelling, modelled; US models, modelling, modelled)
> **6** fashion or shape (a three-dimensional figure or object) in a malleable material such as clay or wax.
> **7** (model something on/after) use (a system, procedure, etc.) as an example to follow or imitate.
> **8** display (clothes) by wearing them.

In the context of models in MDE, definition of number 5 and 7 are most suitable to define a model. Traditionally in Software Engineering, models have been used as initial design sketches mainly aimed for communicating ideas among software engineers. MDE promotes models as core artifacts that drive the entire software development process. The use of models to represent abstractions of the real world has been widely practised in many areas of knowledge [33]. Schmidt [70] and Favre [31] state that a model in MDE represents an abstraction of a domain of interest. Domain-specific models raise the level of abstraction by specifying the solution in a language that directly uses concepts and rules from the domain of interest [45].

In MDE, models are interactive artefacts that are manipulated by model management operations [48]. Models can be used to generate a software system through a series of automatic transformations [24, 73]. Software development focusing on models, moves the software development task from programmer to domain expert, this arguably increases productivity, whilst reducing development time [71, 72, 73]. Hutchinson et al. [40, 41] point out that the benefits of using MDE in practice are the abilities to: i) quickly respond to changes in requirements, ii) improve communication with stakeholders, and iii) increase productivity, maintainability, and portability.

3.2.1 Domain Specific Modelling

Domain Specific Modelling (DSM) is the practice of creating models for a specific domain with Domain Specific Modelling Languages (DSMLs) suitable for that domain [76]. The rationale behind DSM is that each application domain is characterised by its own set of abstractions which are represented precisely and effectively using tailored modelling languages instead of generic languages.

3.2.1.1 Domain Specific Modelling Language

In Computer Science, there are two main types of languages [13, 26]: *Domain Specific Languages (DSLs)* and *General Purpose Languages (GPLs).* DSML is a branch under DSL; Figure 3.2 shows the relationship between them [13].

A DSML consists of five fundamental components [23]: i)*concrete syntax* as human-centric representation, ii) *abstract syntax* as computer-centric presentation, iii) *semantic domain* which is defined as a separate model, iv) *display mapping* which links the abstract syntax to the concrete syntax, and v) *semantic mapping* which links the abstract syntax to the semantic domain. Figure 3.4 illustrates the relationship between these components.

A metamodel is a model that specifies the language, concepts, and constraints of other models [45]. The word 'metamodel' is a synonym of 'DSML'. A model is said to *conform to its metamodel* if it uses only the concepts defined in the metamodel and does not violate the defined constraints [12, 62]. Figure 3.3 shows the relationship between the domain, model and metamodel.

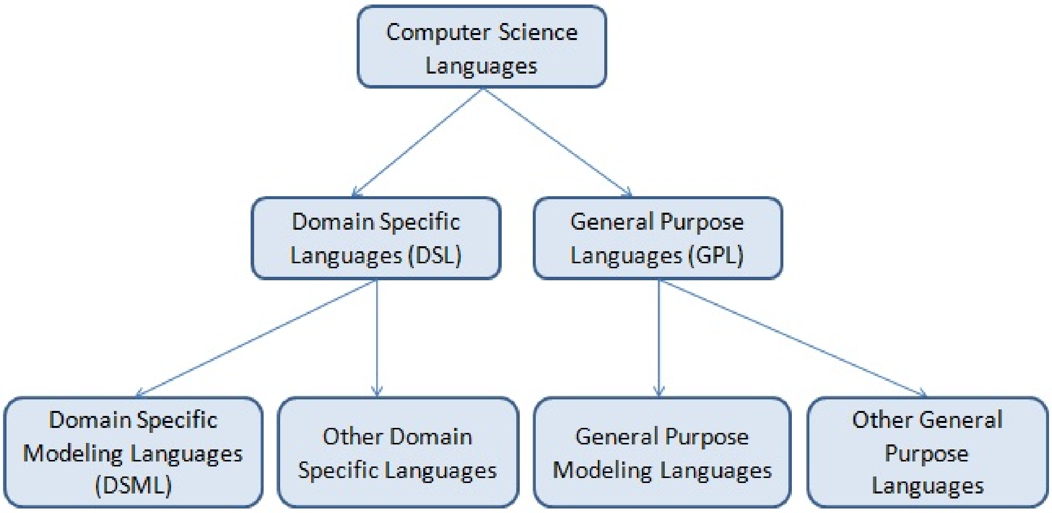

Computer Science Languages
Domain Specific Languages (DSL)
General Purpose Languages (GPL)
Domain Specific Modeling Languages (DSML)
Other Domain Specific Languages
General Purpose Modeling Languages
Other General Purpose Languages

Figure 3.2: Types of Languages in Computer Science [13]

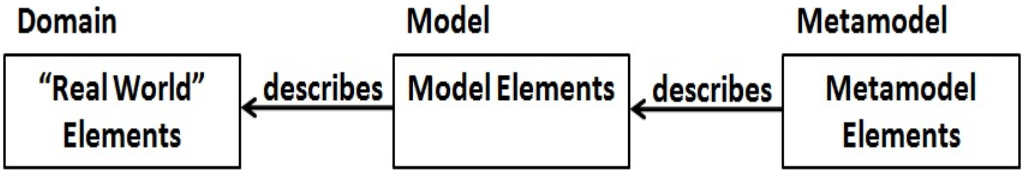


Figure 3.3: Relationship between the real world, model and metamodel [76].

Domain analysis, designing an abstract syntax and mapping the syntax to semantics are the processes involved in designing a DSML [23, 64]. Metamodels do not always need to be designed from scratch; they often reuse and build atop existing metamodels [23, 30]. Reuse has been shown to increase productivity and reduces the development time [71, 72, 73]. Emerson and Sztipanovits [30] identified the following benefits of metamodel reuse:

i. The avoidance of duplication of effort.

ii. The emergence of high-quality reusable metamodel fragments.

iii. The recognition of key metamodelling patterns and best practices.

iv. A significant reduction in the time-to-market for new DSMLs.

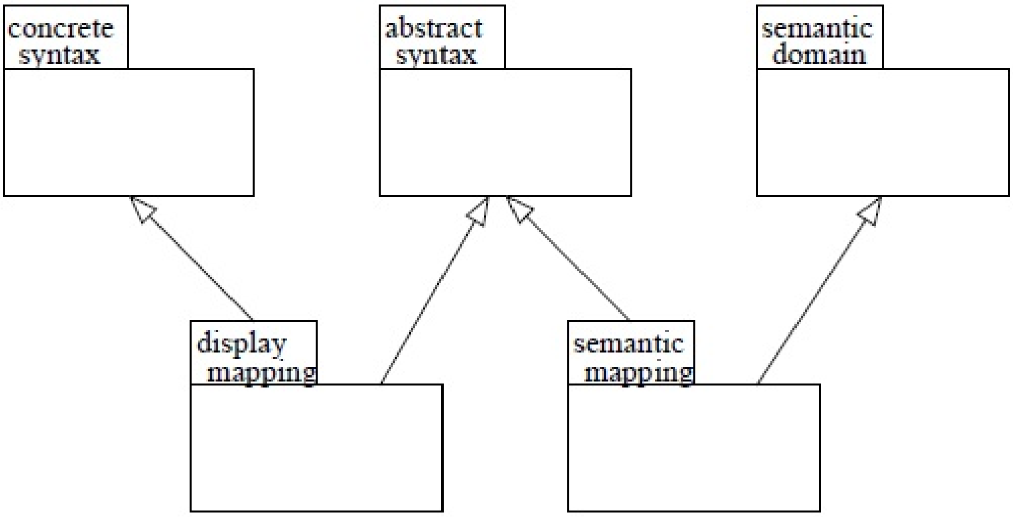


Figure 3.4: Components of DSML [23]

3.2.1.2 Model Transformation

Transformation is a fundamental concept in computer science, especially in software engineering [73]. This concept has been around since the appearance of second generation languages (2GLs) where programs written in assembly language were transformed to machine code by a compiler. The compiler behaves as a transformation engine, where the written computer programs are converted into machine code. This is called program transformation [24, 83]. The same concept is applied in MDE using code generation where source code is generated by a model transformation [73]. Czarnecki and Helsen [24] discussed that program transformation and model transformation are closely related but they are not identical. Czarnecki and Helsen [24] point out that:

> ...program transformation systems are typically based on mathematically oriented concepts such as term rewriting, attribute grammars, and functional programming, model transformation systems usually adopt an object-oriented approach for representing and manipulating their subject models.

According to Czarnecki and Helsen [24], model transformation is a process where a set of transformation rules are applied to transform one or more source models and produce one or more target models as output. Figure 3.5 shows the basic concepts of model transformation with a single source model and target model (in the general case a transformation can consume/produce more than one models).

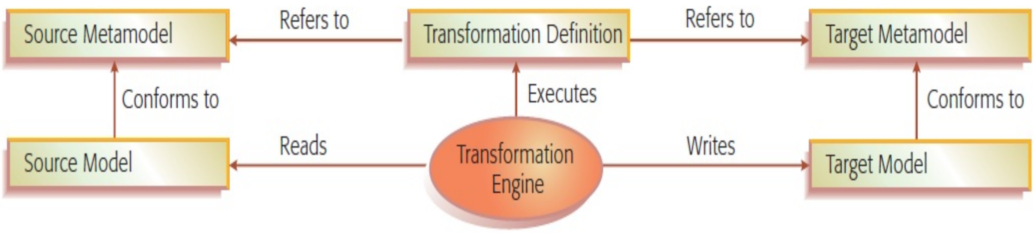


Figure 3.5: Basic concepts of Model Transformation [24].

Model transformation can be classified into model-to-model, model-to-text and text-to-model transformation [13]. In model-to-model transformation, input models are transformed into other models which can be expressed in a different DSML. Models can be also transformed into textual artefacts such as code, documentation and human-readable reports. This is known as model-to-text transformation. The source of model-to-model and model-to-text is a model or several models. However, in text-to-model transformation, the model is the target and the textual artefacts are the source from which the model is constructed.

3.3 MDE Technologies

MDE technologies largely provide automated model management facilities such as transformation engines and generators to analyse and synthesise various types of artifacts [70]. MDE standards such as MDA, MIC and Software Factories are available with their respective modelling tools as technology for MDE [52]. Models are abstract representations of real world and modelling is the process of creating the models conforming to their metamodel. Therefore, before the creation of the models, their metamodel need to be established. The process of designing the metamodel is called as metamodelling [60]. On top of these process, meta-metamodelling is required to enable the creation of metamodel. Meta of metamodel is needed to describe the metamodel [31]. It is referred to as a meta-metamodel which are implemented using available MDE tools. Paige and Rose [61] highlight that the criteria to support MDE are as following:

> MDE can be supported by any modelling language that (a) has a metamodel/grammar/well-defined structure; and (b) has automated tools that allow the construction and manipulation of models.

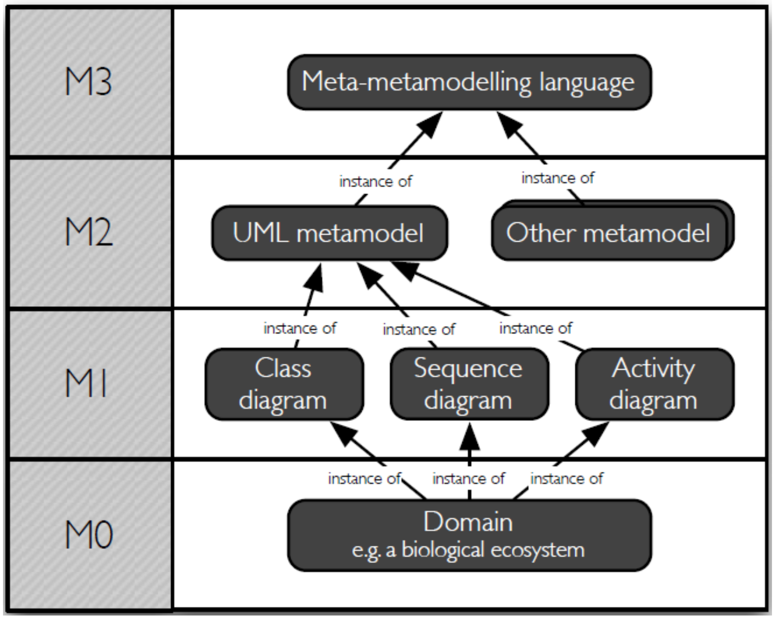


Figure 3.6: The layers in MOF architecture [85].

For example, Ecore is meta-metamodel in Meta Object Facility (MOF) where metamodels are describe using the Ecore structures. MOF[1] is an architecture to define meta-metamodels in Model Driven Architecture (MDA) standards. MDA is an approach to using models in software development and created by the Object Management Group (OMG) consortium[2] [56].

There are four layers in MOF architecture [3, 14, 85]: i) *M3* is a top layer providing a metamodelling language, ii) *M2* is a second layer to specify metamodels using M3, iii) *M1* is the third layer which contain models conform to the metamodels in M2 and iv)*M0* is the object layer representing real-world domain being modelled. Figure 3.6 shows the layers in MDA [85].

Other metamodelling technologies such as GOPRR[3], MetaDepth[4], or pure XML can be used to construct, manipulate and manage the model [61]. The following sections discus existing technologies that support MDE. Discussion is focused on *Eclipse Modelling Framework*[5] and *Epsilon*[6] as these technologies are used in this work.

[1] http://www.omg.org/mof/
[2] http://www.omg.org/mof/
[3] http://www.metacase.com/support/45/manuals/graphi%20metamodeling.pdf
[4] http://astreo.ii.uam.es/~jlara/metaDepth/
[5] http://www.eclipse.org/modeling/emf/
[6] http://www.eclipse.org/gmt/epsilon/

3.3.1 Eclipse Modelling Framework

The Eclipse Modelling Framework (EMF) is an open source modelling framework that has been developed as a pragmatic implementation of the MOF metamodelling architecture [13]. It provides stable and well maintained tool support for modelling activities, such as a graphical editor for defining metamodels and tools for automatically generating model editors from a metamodel. EMF is well established and widely used. It unifies Java, XML, and UML where metamodels can be defined using a UML modelling tool, an XML schema, or Java code [77]. In this thesis, EMF is used to implement the MDE approach. The data types in EMF Ecore Diagrams have prefix 'E'. For example, string data type is referred to as EString and applies to others. However, this is not applied in EMF codes.

Listing 3.1: EMF Codes to Create *School* Metamodel.

```
@namespace(uri="School", prefix="")
package School;
class School{
  attr String name;
  attr String address;
  val Room[1..*] rooms;
  val CentralClock[1] CC;
}

class Room{
   attr String name;
   val Buzzer[1] buzzer;
}

class Buzzer{
   attr String Id;
   ref CentralClock[1] CC;
}

class CentralClock{attr String name;}
```

Ecore is the metamodelling language of EMF [77]. Metamodels for example in Figure 3.7 shows the Ecore diagram for a contrived School metamodel. The same metamodel can be defined in a textual form as illustrated in Listing 3.1.

On top of EMF, Epsilon [47] is a model management framework that interacts with EMF models to perform common MDE activities (e.g. model-to-model and model-to-text transformation, model validation, model comparison and merging). The following section discusses in detail the structure and the facilities provided by Epsilon.

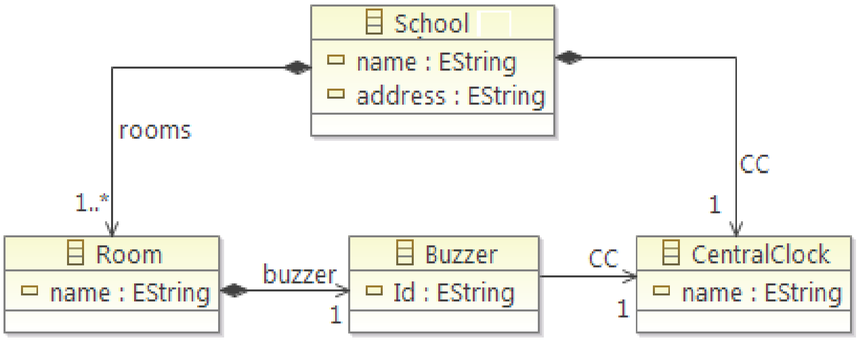


Figure 3.7: Sample of Ecore Diagram for *MySchool* Metamodel.

3.3.2 Epsilon

The name Epsilon stands for Extensible Platform of Integrated Languages for Model Management. It is a platform offering consistent and interoperable task-specific languages for model management tasks such as model transformation, code generation, model comparison, merging, refactoring and validation [47]. The core language of Epsilon is the Epsilon Object Language (EOL). A combination of the procedural style of JavaScript and the powerful model querying capabilities of OCL (Object Constraint Language)[1] is used in this interpreted model-oriented language [47].

Epsilon provides several task-specific languages for model management and utilities for modelling [47]. It provides Eclipse-based development tools and an interpreter that executes programs written in its languages. Epsilon is a component of the Eclipse Modelling project and provides strong support for EMF. However, it is not bound to EMF and the support for EMF is implemented as a driver of the Epsilon Model Connectivity (EMC) layer. Figure 3.8 illustrates the structure, the languages and the drivers in Epsilon.

In this work, three languages of Epsilon are used to develop model management programs to facilitate the capacity planning process. These languages are Epsilon Object Language (EOL) [48], Epsilon Transformation Language (ETL) [49] and Epsilon Generation Language (EGL) [68]. The following section discusses them in detail.

[1] http://www.omg.org/spec/OCL/

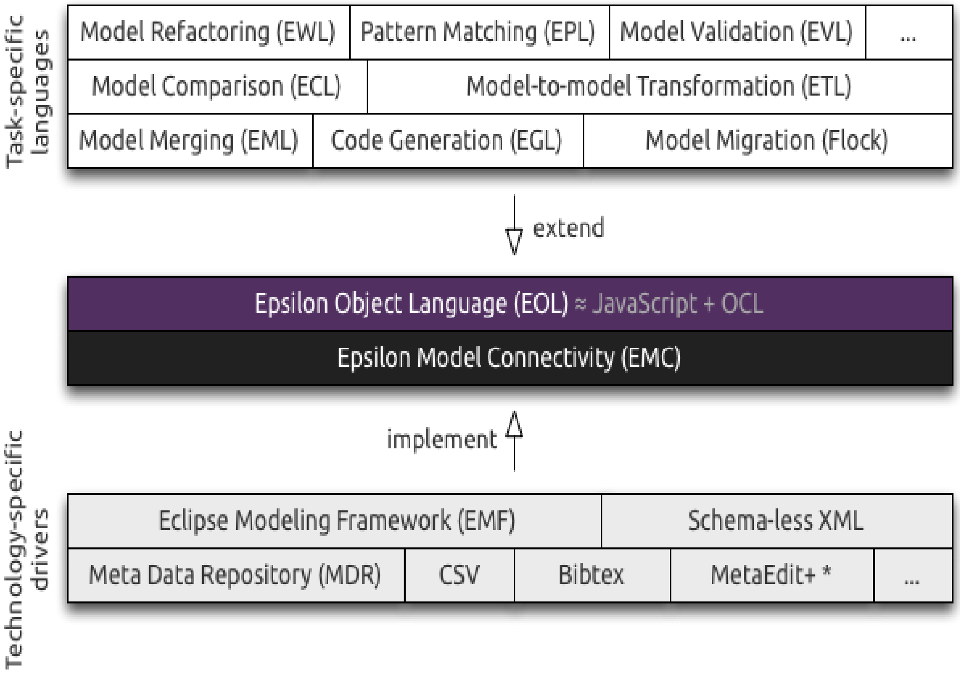


Figure 3.8: The Structure and Languages in Epsilon [1].

3.3.2.1 Epsilon Object Language

EOL is an imperative programming language for creating, querying and modifying EMF models. The aim of EOL is to provide a reusable set of common model management facilities where task-specific languages can be implemented. Beside this, EOL can be used as a general-purpose standalone model management language to automate tasks that do not fall into the scope of task-specific languages such as ETL, EGL and others.

EOL programs are organized in modules where a body and a number of operations are defined. The body is a block of statements that are evaluated when the module is executed. Each operation defines the kind of objects on which it is applicable (context), a name, a set of parameters and optionally a return type.

Listing 3.2 shows an example of an of EOL program that creates an EMF model. Using this program, a model conforming to the *School* metamodel defined in Figure 3.7 can be created. The produced model will have a model element of type *School* with a *CentralClock* object and three *Room* objects. The creation of the model in the model must satisfy the structural constraints defined in the metamodel. For example the object *School* can only have one

ContralClock object and at least a *Room* object. Other model management activities such as querying and modifying existing EMF models can be performed with EOL programs. In this thesis, EOL is used to update, compare and merge models.

Listing 3.2: Sample EOL Codes to Create a Model conforming to School Metamodel.

```
//get School information
var sch : School := new School;
var CClock : CentralClock := new CentralClock;
sch.name:='SampleSchool' ;
sch.CC = CClock;

//get room and buzz information
for (i in Sequence{1..3}){
  var room : Room := new Room;
  room.name:='Room' + i;
  sch.rooms.add(room);

  var buzz : Buzzer := new Buzzer;
  buzz.Id :='Buzz' + i;
  buzz.CC := CClock ;
  room.buzzer := buzz;
}
```

3.3.2.2 Epsilon Transformation Language

ETL is a rule-based model-to-model transformation language that supports transforming many input to many output models, rule inheritance, lazy and greedy rules, and the ability to query and modify both input and output models. Source and target models are required with the rules to implement the transformation. Listing 3.3 shows the concrete syntax of ETL. The body of transformation rules is specified in EOL (e.g. Listing 3.3 in line 10 and 12, statement+). An example of implementing ETL were demonstrated with Listing 3.4 and Figure 3.9.

Listing 3.3: ETL Concrete Syntax[47].

```
(@abstract)?
(@lazy)?
(@primary)?
rule <name>
transform <sourceParameterName>:<sourceParameterType>
to (<rightParameterName>:<rightParameterType>
(, <rightParameterName>:<rightParameterType>)*
(extends (<ruleName>,)*<ruleName>)? {

(guard (:expression)|({statement+}))?

statement+
}
```

Figure 3.9 shows sample metamodels to explain ETL program in Listing 3.4. Since it is required to have source and target models in implementing ETL, two metamodels (*University* and *Student Record*) are used as an example. In Listing 3.4, "*S*" is a university model created conforming to *University* and "*T*" is student record model created conforming to *StudentRecord* metamodel. The ETL program presented in Listing 3.4 transform the S model to T model (the *University* model are transformed to *StudentRecord*) model according to the defined rules.

Listing 3.4: Sample ETL rules.

```
rule StudentToTranscript
      transform S:University!Student
      to T:StudentRecord!Transcript {
           T.student := S.name;
           T.items.addAll(S.grades.equivalent());
      }

rule ModuleGradeToTransItem
      transform G:University!Grade
      to TI:StudentRecord!TranscriptItem{
          TI.module:=G.module.name;
          TI.mark:=G.mark;
      }
```

For example, models to conform to the *University* and the *Student Record* metamodels in Figure 3.9 are used in ETL program in Listing 3.4. A *University* model has student information such as student name, modules taken by the student and also the grade obtained for those modules. A *StudentRecord* model captures transcripts information for all the student. The ETL program transforms a *University* model to a *StudentRecord* model according to

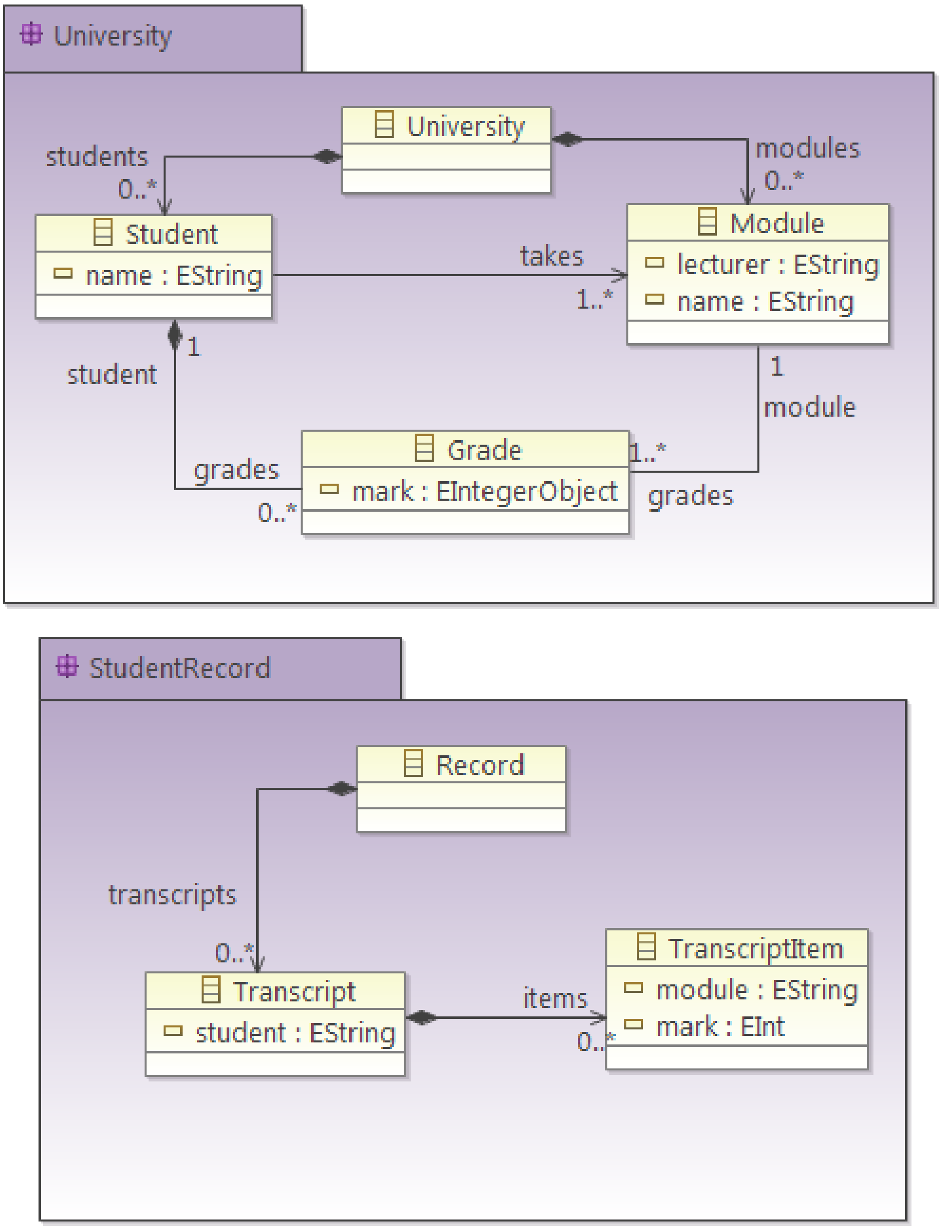


Figure 3.9: Sample DSMLs of University and Student Record.

the two defined rules (*StudentToTranscript* and *ModuleGradeToTransItem*). In *StudentToTranscript* rule, information (eg. student name and grades) of all the student captured in the *University* model (S) are transformed to an individual *Transcript* object of the *StudentRecord* model (T). The *ModuleGradeToTransItem* rule, transforms the grade of the modules taken by the student to *TranscriptItem* object of *StudentRecord* model. Line number 5 in Listing 3.4 invokes the *ModuleGradeToTransItem* rule.

3.3.2.3 Epsilon Generation Language

EGL is a template-based model-to-text language for generating code, documentation and other textual artefacts from models. EGL supports content-destination decoupling, protected regions for mixing generated with hand-written code and template coordination.

Listing 3.5: Sample Main EGL Program.

```
[\%
 var tran: Sequence;
 tran := S!Transcript.allInstances;

  for (t in tran){
    var genTxt : Template := TemplateFactory.load('GT.egl');
    genTxt.populate('tran1',t);
    genTxt.store(t.student +'.txt',true);
  }
\%]
```

The concrete syntax of EGL is similar to that of other template based text generation languages, such as PHP. The tag pair [% %] is used to define a dynamic section and text not enclosed in the tag pair is a static section. Listing 3.6 illustrates EGL program named *GT.elg* with dynamic and static sections. This program is called by another EGL template as presented in Listing 3.5.

Listing 3.6: GT.egl Program.

```
Student name : [\%=tran1.student \%]
          Subject                          Mark
         _________                        _________
    [\% for (j in tran1.items){ \%]
        [\%=j.module \%][\%="\t\t\t"\%]    [\%=j.mark \%]
    [\% } \%]
  ____________________________________________________________
```

EGL in this work is used to produce charts (using the Google Graph[1] library) to visualise the results of the various tasks of the capacity planning process.

[1] https://developers.google.com/chart/

3.4 Chapter Summary

Analysis of the literature suggests that MDE with DSM can be applied in various domains by constructing DSMLs for the domains of interest. In Chapter 2, the domain of interest was presented and the possibilities of utilising DSM and model analysis techniques to solve the identified problem were discussed. This chapter discussed concepts and technologies related to modelling, metamodelling and automated model management.

The following chapters discuss the utilisation of MDE in performing capacity planning specifically for virtualised environments.

Chapter 4

Domain Analysis

4.1 Introduction

Based on the literature review presented in Chapter 2, three main phases are identified in conducting capacity planning in virtualised environments. These phases are elaborated upon and discussed in detail in this chapter with a suggested framework for utilising MDE generally and DSM specifically. The benefits of using DSM in this research are also discussed. Along with an analysis of the literature, the scope, contribution and challenges of the research are considered.

4.2 Capacity Planning Phases

Three important phases are identified in running applications in virtualised environments. Firstly, managing fluctuating workloads initiated by end-users which consume the virtual resources of virtual machines used for hosting. Secondly, managing virtual machine resources, which are used to run the applications in a virtual data centre; and finally, managing physical resources in physical data centres. Previous research has treated these phases in isolation. This work proposes an integrated framework for capacity management from the end-user to the infrastructure service provider. A set of domain specific modelling languages (DSMLs) makes it possible to facilitate the integrations of these three phases.

Each application hosted on a virtual machine has its own unique character-

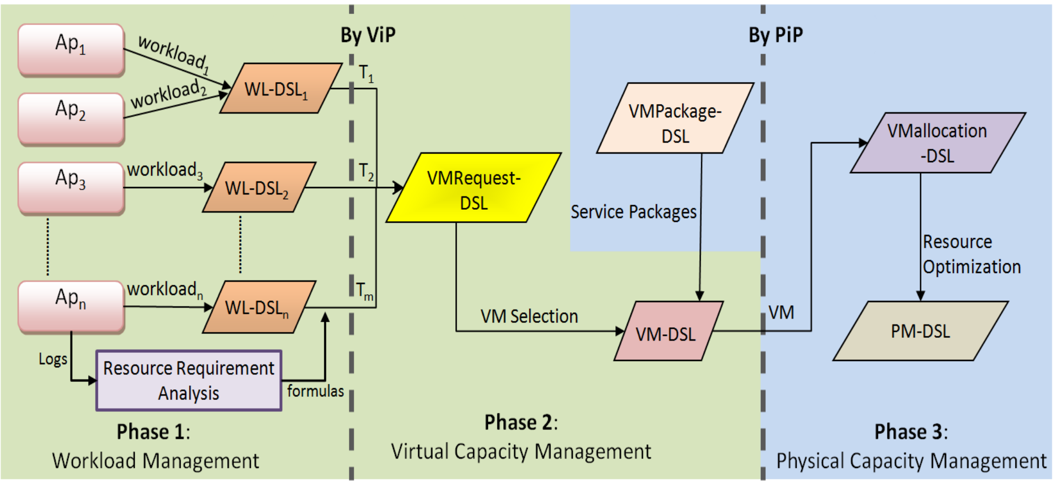


Figure 4.1: Capacity Planning Framework in Virtualised Environments.

istics (e.g. due to differences in requirements, architecture, implementation technologies) as such workload specification models are unique to each application. These can be expressed using a DSML tailored to the application. A DSML can precisely capture all the parameters needed to express actual and estimated user workloads, and the workload models can then be analysed and consolidated in order to estimate virtual and physical resource usage. While applications are unique and each of them requires its own DSML to specify its workload, it is anticipated that a single domain specific language will be sufficient to express virtual machine workload specifications by capturing requirements for CPU, memory and other resources of interest over time. Virtual machine workload models can be used to perform capacity management, in order to achieve an acceptable balance between performance and cost. Moreover, the information captured in the virtual machine demand model (VM-DSL) can be used to optimise physical resources in the PiP's data centre.

4.2.1 Capacity Planning Framework

In this work we propose an MDE framework for managing the different phases of capacity planning in an integrated way. The framework consists of Domain Specific Languages which are used to model the resource requirements estimates in each phase. The framework also uses model transformations to transform resource requirement models across the different phases of the process.

Figure 4.1 illustrates the flow of information through the different phases of capacity planning, the responsible stakeholders, and the MDE artefacts involved in each phase. Estimation of virtual resource requirements based on known application workloads is necessary for costing and performance management. The virtualisation technology (i.e. the virtualisation type and architecture), affect the performance of the VMs and also resource requirements. Resource requirement analysis needs to be performed in the selected virtualised environment to predict the virtual capacity requirements. Therefore, estimating virtual resource requirement is performed by ViP based on known workload information captured by the application owner.

Understanding the application workload patterns is beyond the scope of this research. Business analysis or long term analysis [65] of the workload demand are examples of ways to predict the application workload. Estimated workloads are captured with a unique WL-DSL for each application. Then, the transformation to virtual machine requirements is performed by ViP, while estimation of physical resource requirements is performed by PiP. The proposed framework with DSMLs for modelling virtual machine packages (VMpackages) and allocation of virtual machines to physical resources (VMallocation) is presented in Figure 4.1. Workload, virtual capacity and physical capacity management are clearly illustrated as three separate phases with their associated DSMLs represented as parallelograms. Transformations are used to map application workloads into unified VM requirements, to distribute unified VM requirements across a set of available VM packages, and finally to allocate VMs to physical machines. In the following sections, these three phases are discussed in detail and requirements are elicited for an integrated framework that can support end-to-end capacity planning.

4.2.1.1 Capturing Application Workloads

The application owner who hosts their services on ViP-managed resources needs to estimate the total resource requirements over different time periods in order to satisfy the QoS (Quality of Service) requirements of their services [36]. Resource requirements are based on the estimated application workload over specific time periods. Since each application is unique, an application-specific workload DSL (WL-DSL) is essential to capture its workload estimates. Application workload models conforming to their respective WL-DSL are constructed based on the expectations and previous experience of the application owner (e.g. number of users expected to interact with the application over a certain period of time, types of actions these users are expected to perform). While each application demonstrates its own characteristics, all WL-DSLs need to provide constructs for capturing information related to the time period in which each workload estimate refers to. These constructs can be pulled up into a core WL-DSL which application-specific WL-DSLs can inherit from.

4.2.1.2 Virtual Resource Requirements Estimation and Selection of VM Packages

The second phase involves estimating the virtual machine resource requirements based on the information captured in the application workload models. Resources such as CPU, memory, storage and bandwidth can be calculated from workload models using dedicated model-to-model transformations in the form of VM requirement models conforming to the VMRequest-DSL. Such model-to-model transformations are highly-specific to the application in question and need to be composed by the application owner as they are based predominately on domain knowledge and past experience. Later, the required virtual resources are mapped to a cost-efficient configuration of concrete VM packages offered by the PiP providers through another round of model-to-model transformations.

4.2.1.3 Consolidating VM Requirements to Physical Resource Requirements

In this last phase of the process, virtual machine requirements captured as VM-DSL models are consolidated and allocated to physical resources available in the PiP's data centre. PM-DSL can be used to model the physical resources available in the physical data centres. The VMallocation-DSL then captures possible allocations of VMs selected by the application owner under different optimization methods such as round-robin [46], greedy [28, 78], green [27] and surplus [28].

4.3 Analysis of Research Scope

This chapter provides an overview of full integrated solution for capacity planning from application workloads (phase 1) to virtual resources requirements (phase 2) and continue to physical resource requirements (phase 3).

This research focuses on integrating phase 1 and phase 2 by utilising Model Driven Engineering.

The performance of an application hosted in different virtualisation environments can differ, even though it is handling similar workloads with the same virtual resources specifications. The performance of VMs in virtualised environments is closely related to the virtualisation technology used. The virtualisation type and the architecture of virtualisation environments impact upon the performance of VM. This study focuses on the process of deriving the relationship between virtual resource requirement and workloads. This relationship is necessary when estimating resource requirement for specific virtualised environments in integrating phase 1 and phase 2 of the proposed framework.

Capacity planning (in general) involves predicting future computing resource requirements by monitoring a system's resource usage patterns, and comparing them with known or historical workload patterns. Capacity planning in virtual environments aims to ensure that allocated virtual computing resources such as CPU, memory, storage and network bandwidth will be sufficient to support future computational needs. In this process, available system resources are observed and performance is measured [74]. Also, resource usage patterns are determined to forecast the resources that need to be allocated to serve future workloads in compliance with the service's QoS requirements [65]. To achieve this, it is necessary to identify incoming workloads, to monitor resource usage, and to associate resource usage with the workloads that triggered it.

Therefore, two detailed modelling solutions are proposed with accompanying DSMLs. Firstly, Resource Requirement Analysis (ReRA) is performed to derive application specific formulas for general resource metrics such as CPU, memory, storage and bandwidth. Secondly, these formulas are used in the transformation engine to estimate resource requirements based on known workload patterns. This is called as Virtual Resource Requirement (ViRR).

4.3.1 Application Resource Requirement Analysis

Automating the estimation of resource requirements in the capacity planning process involves several steps. Resource Requirement Analysis (ReRA) of an application is the initial step in phase 1 of the framework and its position is illustrated in Figure 4.1. It involves observing the resource usage and correlating it with application workloads. DSMLs (and models) are used to

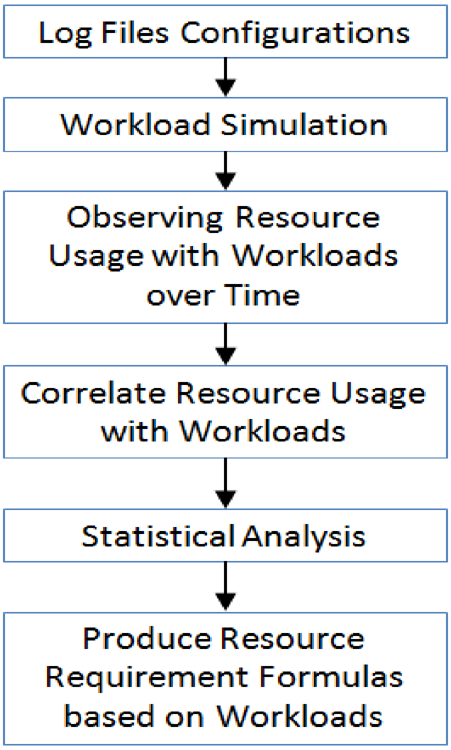

Log Files Configurations
Workload Simulation
Observing Resource Usage with Workloads over Time
Correlate Resource Usage with Workloads
Statistical Analysis
Produce Resource Requirement Formulas based on Workloads

Figure 4.2: Process of Resource Requirement Analysis.

precisely specify resource requirements and workloads and to facilitate the resource requirement analysis process, which then feeds into the remainder of the capacity planning process.

Information related to usage of selected resources can be extracted from log files generated by resource monitoring tools. Workloads being processed are also generally captured in the application log. Resource requirement formulas for the resource metrics can be obtained by monitoring and analysing the resource usage of the VMs involved with the workload being processed. The time granularity of resource usage measurements should be comparable to the workload log.

ReRA is a process used to analyse resource usage and produce resource requirement formulas based on an individual application workload. Real workloads typically comprise a combination of several workload types in a given application. Resource usage might differ according to workload type. As ReRA process analyses the resource usage of the identified workload type individually, simulated workload is used to analyse the resource usage of that workload type. The simulator provides control over the workload being analysed and enables additional input compares to the real workloads.

Figure 4.2 shows the process involved in ReRA. Information for the initial models is extracted from various logs files. These files should contain relevant information and this can be performed by setting the log files configuration. There is a requirement to simulate the application workload to observe the usage of resource metrics with the simulated workloads. Later, resource usage is correlated with the workload being processed. Statistical analysis is performed to estimate resource requirements for the workload and a statistical analysis tool is used to produce the resource requirement formulas for resource metrics. These formulas are used in a transformation of a respective application in the following section.

4.3.2 Estimating Virtual Resource Requirements

Virtual resource requirements are estimated based on predicted workload with attributes and formulas produced from ReRA. This step is implemented as a transformation to produce a *VMRequest-DSL* models based on respective WL-DSL models. Since applications are unique by themselves and have their own WL-DSL, the formulas retrieved for resource requirement estimation are also unique. Therefore, the transformation (T) used to transform *WL-DSL* to *VMRequest-DSL* is application specific. This process is abbreviated as ViRR (Virtual Resource Requirement). The long real-time log analysis of workload recording is required to identify the workload pattern [65]. This allows the capacity planning manager to estimate workloads based on previous log recordings. This estimated workload pattern acts as input to predict resource requirements for a given process.

4.4 Benefits of DSM

DSMLs provide precise and standardised ways to capture these common concepts and structures, that can be used by many different analysis tools. The details of the proposed DSMLs are presented in the following chapter, but first, the overall resource requirement analysis process that can facilitate capacity planning for web applications, is discussed.

A DSML-based approach is presented to support resource requirement analysis activities of capacity planning. The novelties are: DSMLs that allow resource and requests logs as well as workloads to be precisely captured as models, as well as a transparent, automated and repeatable Model-Driven Engineering (MDE) process for generating predictions for resource usage from workload models. The MDE process, which exploits model transformation, comparison and merging, is modularised so that it can be configured for different kinds of capacity planning applications and technical infrastructures.

An additional contribution is the ability to derive a set of application-specific resource requirement formulas for *resource metrics* (CPU, memory, network and storage). We demonstrate the modelling approach in a proof-of-concept web application examples in the following chapters.

4.5 Technical Challenges

There are several challenges in performing this research work. Below is a list of such challenges faced throughout the research:

i. **Establishing a virtualised environment.**
The proposed capacity planning solution is to be implemented in a virtualised environment. Therefore, a virtualised infrastructure is necessary to implement and evaluate the proposed modelling approach. If the infrastructure already exists, this research will be directed to focus more on modelling activities. Establishing a virtualised environment in itself is time consuming and requires specialised technical skills.

ii. **Time synchronisation.**
In the domain of interest in this thesis, the information required to populate models gathered from logs recording of the workload processed by an application running in a virtualised environment. And also, logs recording the resource usage of the VM that hosts the application. One of the elements used to link the log files is the time recorded in each file. Therefore, time synchronisation is an important and necessary action which needs to be taken to handle this issue.

iii. **Workload duration.**
The correlation is difficult to be performed for the applications' workloads which complete processing with a very small time period. For example in the case of web applications, the web server records the web requests as workloads after the completion of the requests. The workload simulator generates the following request once it has received the feedback from the web server of the completion of the previous request.

Therefore, there will be a small time interval in generating the following request to maintain certain number of request. Generally, the resource monitoring tools are able to capture the resource utilisation up to a second. If the request completes before the resource monitoring tool records the resource utilisation, the correlation is difficult to perform since there is possibility that the request log shows that there is no request recorded at that time. The suggestion to manage this challenge is discussed in Chapter 8.

iv. **Learning Curve.**
The work presented in this thesis involves many technical tools such as VirtualBox, JMeter, Epsilon, Apache2, PHP, Java and Matlab. Learning and adopting the relevant tools for constructing the technical solution is challenging as a beginner. Alongside this, for the novice, producing a simple solution with MDE is not easy. A basic understanding of the principle and terminology of MDE is necessary in addition to understanding the problem domain.

4.6 Research Contribution

A DSML-based approach to support capacity planning is presented in this thesis. The objectives, hypothesis and research questions were discussed in Chapter 1. The novelties of the research are sets of DSMLs that allow workloads to be precisely captured using models, as well as a transparent, automated and repeatable MDE process for generating predictions for resource usage from workload models. The MDE process, which exploits model transformation, comparison and merging, is modularised so that it can be configured for different kinds of capacity planning applications and technical infrastructures.

4.7 Chapter Summary

The analysis of the problem domain suggests that the integrated framework with three phases can be implemented by having DSMLs for the domains. The phases were explained with their respective DSMLs. Although a fully integrated solution for capacity planning was proposed with the framework, the focus of the research is in proposing a detailed modelling solution by integrating phase 1 and 2. The benefits of DSM, the research challenges and the contribution of the proposed solutions were also discussed. The following chapter explains the design of the proposed framework.

Chapter 5

Design of the MDE Solutions

5.1 Introduction

The design of the proposed DSMLs and model management activities to automate the ReRA and ViRR processes are presented and elaborated upon in this chapter. The DSMLs are considered together with their features, management operations, rules and the transformation relationships between them. The proposed framework was designed to integrate the first two phases of the capacity planning process as discussed in Chapter 4. The remaining phases are beyond the scope of this work. The set of DSMLs and the model management activities were designed to automate Resource Requirement Analysis process discussed in Section 4.3.1. In this and the following chapters, the name of the DSMLs and its artefacts including model managements techniques developed were presented in italics.

5.2 Resource Requirement Analysis

In Section 4.3.1, the general process involved in Resource Requirement Analysis (ReRA) was discussed. To automate this process, several DSMLs were designed together with supporting model management programs. Different types of applications can have different DSMLs to capture information about their workloads. The detailed discussion of the DSMLs is presented in Section 5.2.1, firstly, to give a better overview, the ReRA process is explained for an application running in a virtualised environment.

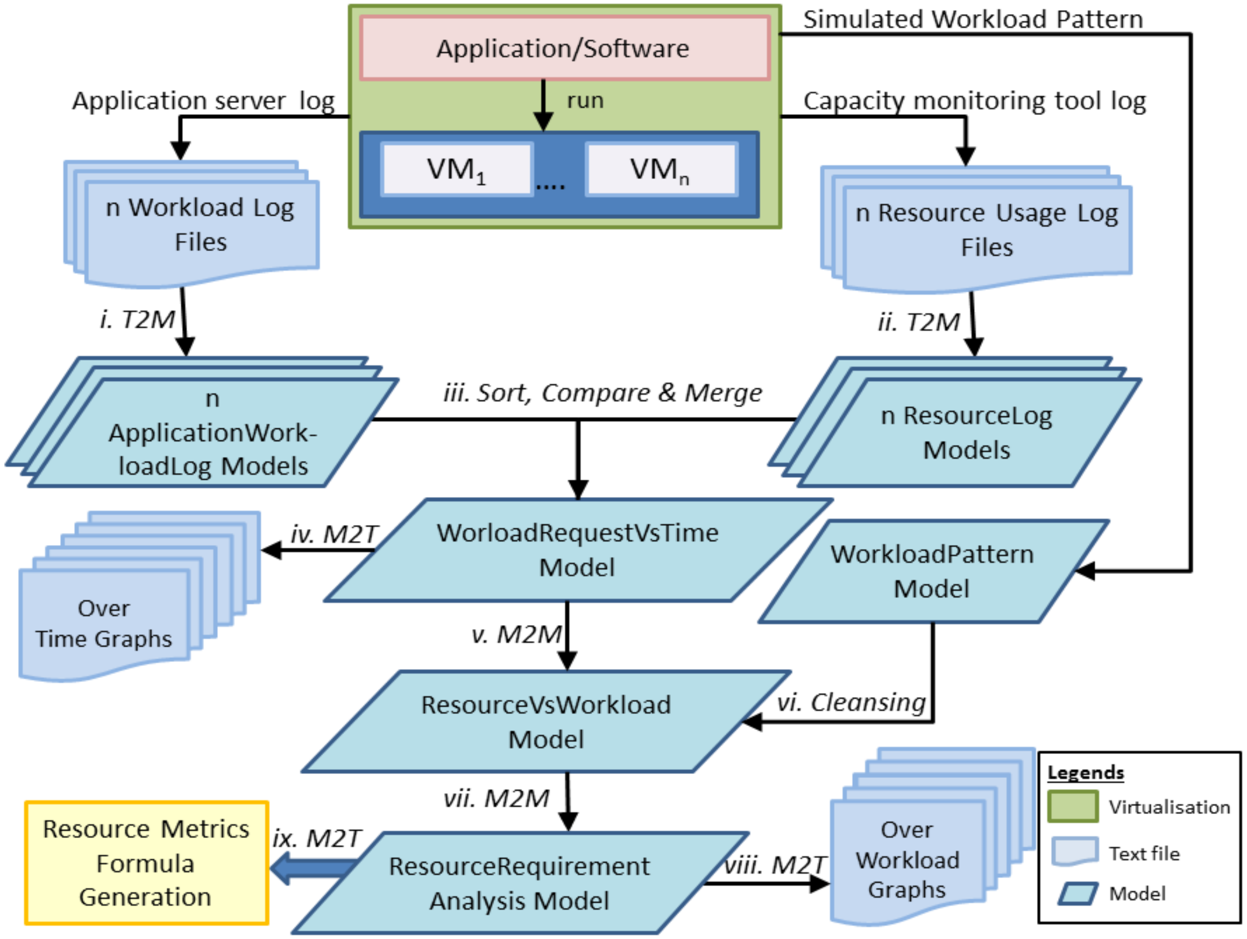


Figure 5.1: The General ReRA Process together with a Set of Models and Model Management Activities (indicated by Roman numerals and Italics) to Facilitate the Capacity Planning Process in a Virtualised Environment.

A graphical illustration of the general ReRA process is shown in Figure 5.1. An application can run on a single or on multiple (virtual or physical) machines. The incoming workloads of the application are captured in the application server log. The workload logs (Workload Log Files) are parsed to the *ApplicationWorkloadLog* model. The hosting virtual machine is monitored by resource monitoring tools which produce resource usage logs (Resource Usage Log Files). Parsing the logs into the *ApplicationWorkloadLog* and *ResourceLog* models shields the rest of the process from hard dependency on particular runtimes and VM monitoring tools. The log models are then compared and merged to produce a *WorkloadResourceVsTime* model. Then, transformations are used to produce a correlation (grouping) model (*ResourceVsWorkload*). To reconcile any measurement errors, *cleaning* is performed. Graphs are generated based upon the *WorkloadRequestVsTime* and *ResourceVsWorkload* models.

A suitable statistical analysis method can be applied to produce precise values. This task is open for future work to further improve the predictive capability. In this thesis, basic statistical method such as mean, median, maximum and minimum were calculated to feed into the *ResourceRequirementAnalysis* model. Formulas for the resource metrics to be used in the ViRR process were generated based on the *ResourceRequirementAnalysis* model. This equation extraction module was represented with a box labelled as Resource Metrics Formula Generation in Figure 5.1.

In the Resource Metrics Formula Generation module, an equation extraction tool can be used to produce respective formulas for each resource metrics. The input to this module is a text file generated with model-to-text transformation program of the *ResourceRequirementAnalysis* model. The generated text file can be any format supported by the equation extraction tool. The model-to-text transformation program might need to be customised to produce such format. The information captured in the *ResourceRequirementAnalysis* model is transformed as input data to the equation extraction tool with the generated file, such as in a comma-separated-values (csv) file. The output of the equation extraction tool consists of a set of formulas of the resource metrics produced based on the input file. Figure 5.2 illustrates the DSMLs involved and their interactions in automating the ReRA process. Related DSMLs, together with model management techniques presented in Figure 5.2, were designed to represent each activity involved in ReRA.

The ReRA process provides modular components for each activity. The modularity of DSMLS of ReRA process enables changes to a component without affecting other components. This enables equation extraction to be replaced with appropriate implementations such as applying machine learning. The machine learning method normally includes statistical analysis. Therefore, model-to-model transformation in step (vii) in Figure 5.2 can be replaced with model-to-text transformation to produce input file to the respective machine learning tool based on raw data captured in *ResourceVsWorkload* model. The requirement for using the *ResourceRequirementAnalysis* model depends on the statistical analytic capability of the tools used to produce the equations.

The input file format for different tools may vary. Therefore, the model-to-text transformation program need to be customised depending to the supported file format of the tool. The structure of *ResourceVsWorkload* and *ResourceRequirementAnalysis* DSMLs are not complex and changing to different tools only involve changes to the model-to-text program to produce supported file format. The effort required to replace the equation extraction tool depends on the complexity of the new tool, the format of the input it expects, the output it produces, and the familiarity of the engineer with the MDE technologies involved. Given the size of existing transformations (hundreds of lines of code), interfacing a new text-based tool with the system should not require more than a few hours for an MDE-literate engineer. The main characteristic of the tool is that it must be able to produce resource metrics equations with the data feed as input.

The ReRA process, used to retrieve formulas for resource metrics based on workloads for an application, is discussed above. The design of the DSMLs is described in the following section and the related model management operations are described in detail in Section 5.2.2.

5.2.1 DSMLs Design of ReRA

As discussed above, a set of DSMLs for automating the resource requirement analysis process was constructed. Figure 5.2 illustrates the set of DSMLs and the relationship between them. This figure shows the flow and details of the DSMLs presented in Figure 5.1. The Roman numerals in both figures are the model management activities discussed in Section 5.2.2. This section provides an overview of DSMLs organisation and semantics. The following are the DSMLs developed for capture the attributes involved in the ReRA process:

i. ***ApplicationWorkloadLog***
This DSML has abstract classes which can be inherited by the applic-

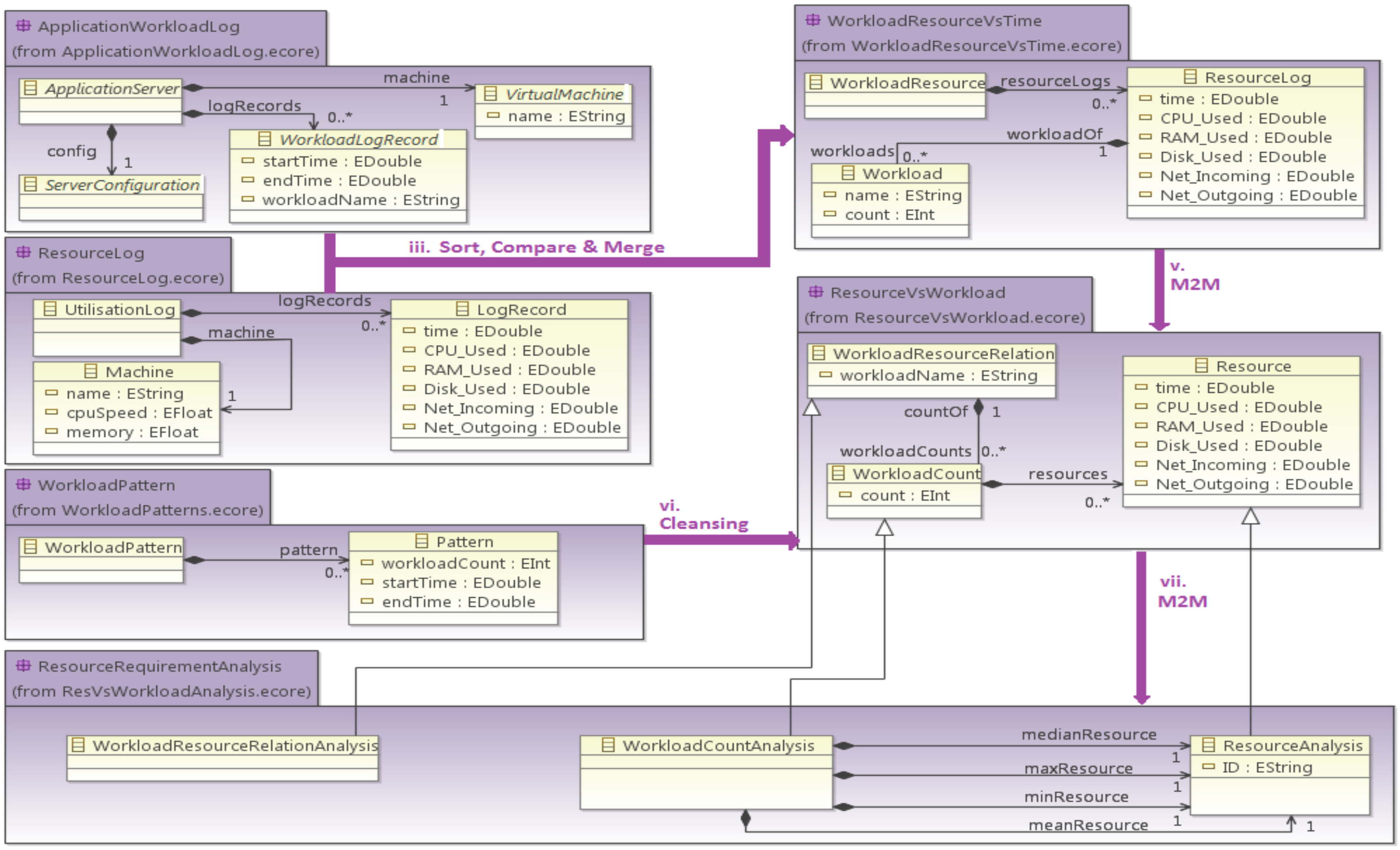


Figure 5.2: DSMLs of ReRA.

ation workload log DSML for particular application types. Application workload information such as start time, end time and workload name are extracted from the application server log. This information is sufficient to proceed with the ReRA process and the design is illustrated in Figure 5.3. The configuration of the server is recorded in *'Server-Configuration'*, while *WorkloadLogRecord* records the log recordings of the workload and *'VirtualMachine'* stores the information of the VM. In this thesis, the unit for time is standardised to seconds. Therefore, *'startTime'* and *'endTime'* are defined as EDoubel (double data type) to record floating-point numbers in seconds. Since EMF is used in this work, the data types presented in Ecore Diagrams have an 'E' prefix. The time captured in log recording is converted to seconds in the execution of the text-to-model transformation program.

ii. ***ResourceLog***

Information related to usage of selected resources is extracted from log files generated by the tools monitoring the resource usage of the machines involved. Figure 5.5 shows the design of *ResourceLog* DSML. The time granularity of resource usage measurements should be comparable to that of the workload log. The CPU and memory allocation of the machine is captured next to its unique identifier (*name*). The attribute *cpuSpeed* of class *Machine* represents the speed of a single processor in the machine, and *cpuUnit* represents the number of CPUs assigned to the machine. These values will be later used to convert the per-

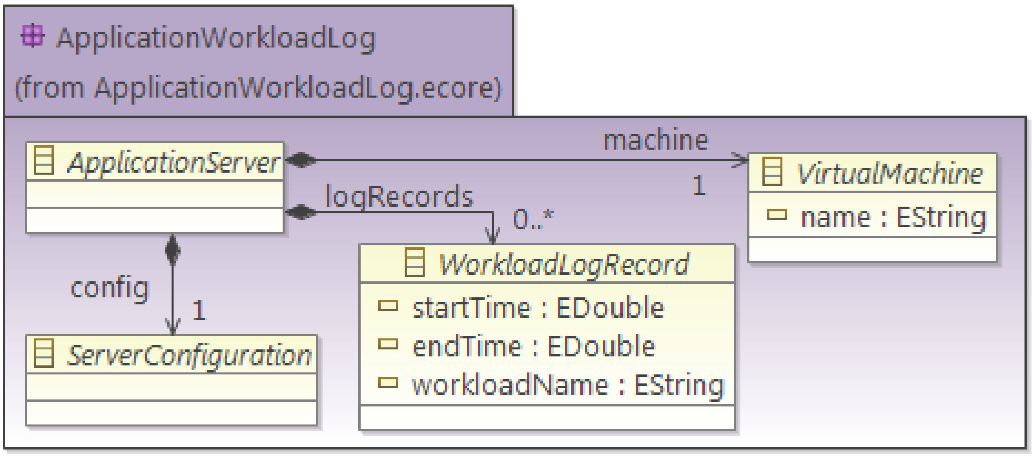


Figure 5.3: The Design of *ApplicationWorkloadLog* DSML.

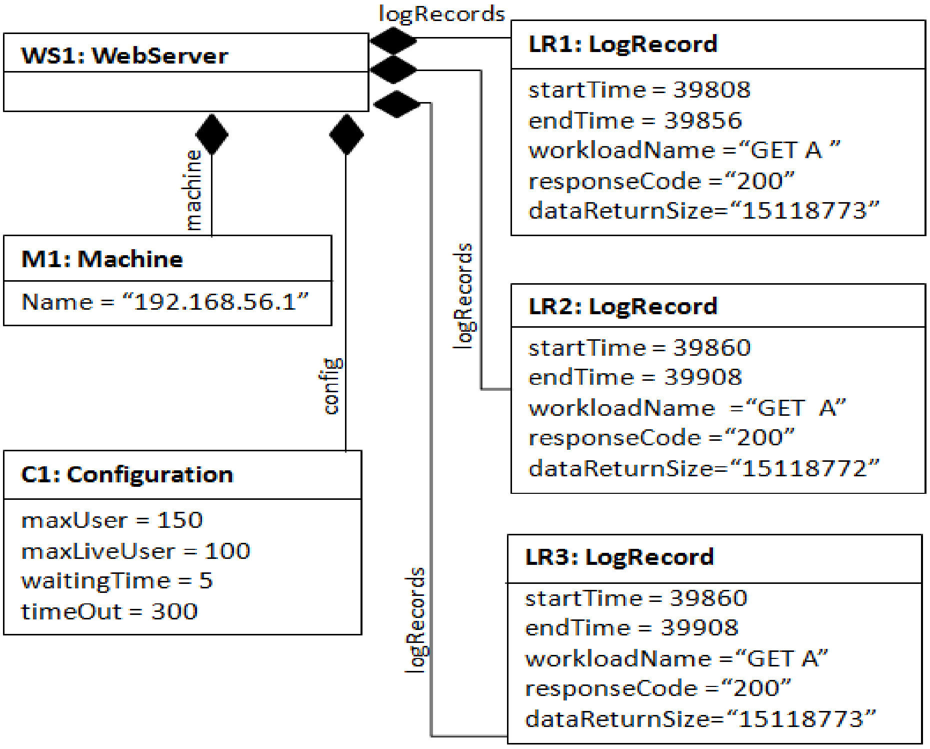


Figure 5.4: Example of extended *ApplicationWorkloadLog* instance for web application (this figure originates from the Media Stream case study which is presented in Chapter 6). For the DSML, refer Figure 5.3 and Figure 6.7.

centage of CPU usage recorded by the resource monitoring tool into an absolute figure. The same applies to memory – if required – but most resource monitoring tools supply the actual usage of memory rather than a percentage. Resource measurements such as CPU, memory, disk and bandwidth (incoming and outgoing network) are also recorded. In the respective models, the units used for CPU is Mega Hertz (MHz) and other resource metrics (memory, disk and bandwidth) were represented with Mega Bytes (MB) as units. The units used should be consistent to avoid conversions in automating the model management activities. Units such as Kilo and others also can be used to represents the resource metrics values, but it needs to be consistent.

The values recorded in resource usage log and the range supported by the defined resource metrics type influence the selection of units. The double data type is selected to capture resource utilisation since it accommodates 15 to 16 digits, with a range of approximately 1.7e308 to

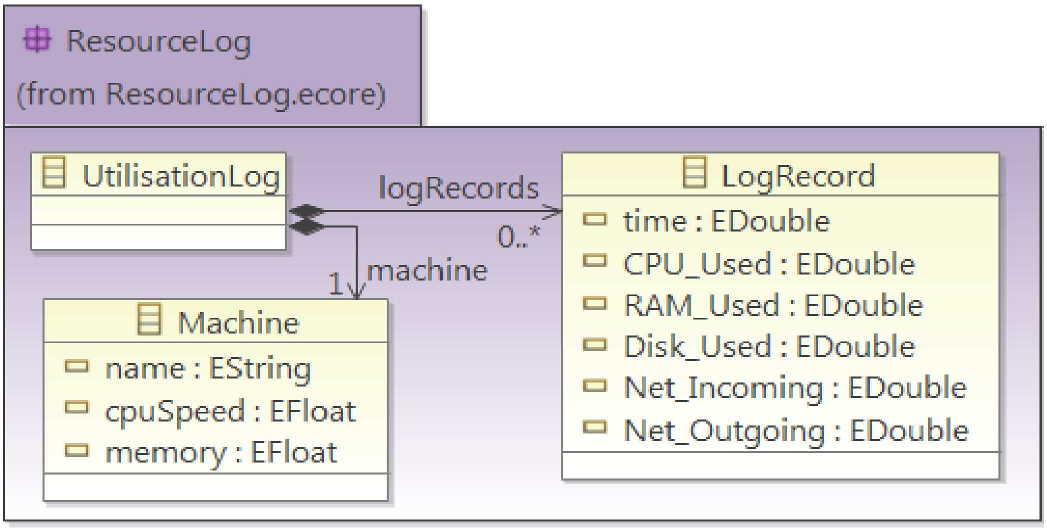


Figure 5.5: The Design of *ResourceLog* DSML.

1.7e+308. In this thesis, the unit for resource metrics was standardised to Mega and other units are converted to this unit.

Figure 5.6 demonstrates an instance of the *ResourceLog* DSML with data extracted from the resource usage log. In this example, the resource monitoring tool records resource utilisation for each second and the time is represented by the ordinal number of the second in the day. The resource usage log recordings for 3 seconds (39807, 39808 and 39809 seconds of the day) are transformed into individual *LogRecord* objects. The number of the log recordings is equivalent to number of *LogRecord* objects. The VM configuration with its unique name is captured in a *Machine* object.

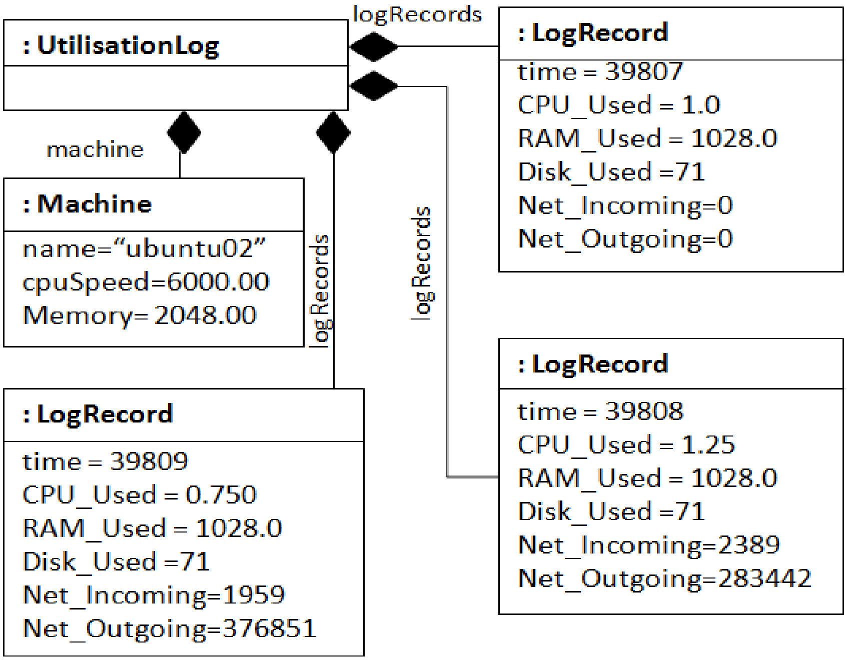


Figure 5.6: Example of *ResourceLog* Instance (for the DSML, see Figure 5.5).

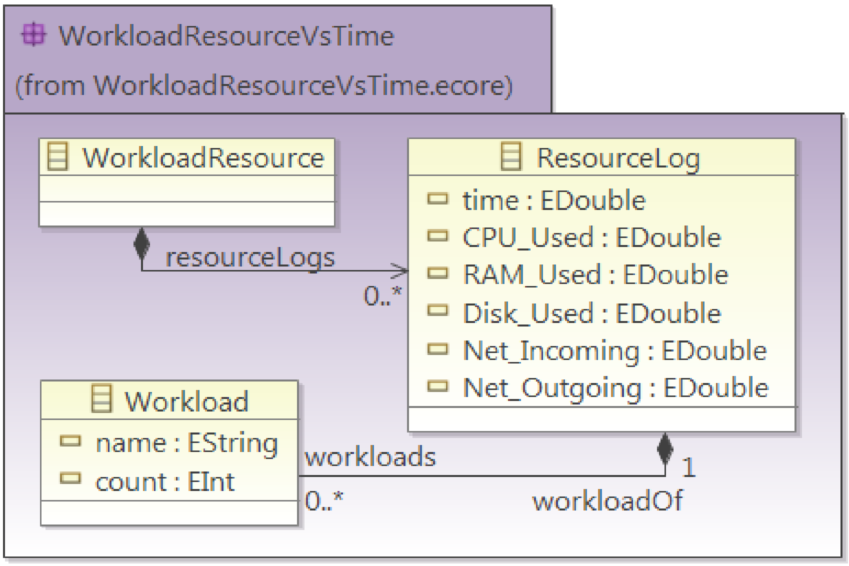


Figure 5.7: The Design of *WorkloadRequestVsTime* DSML.

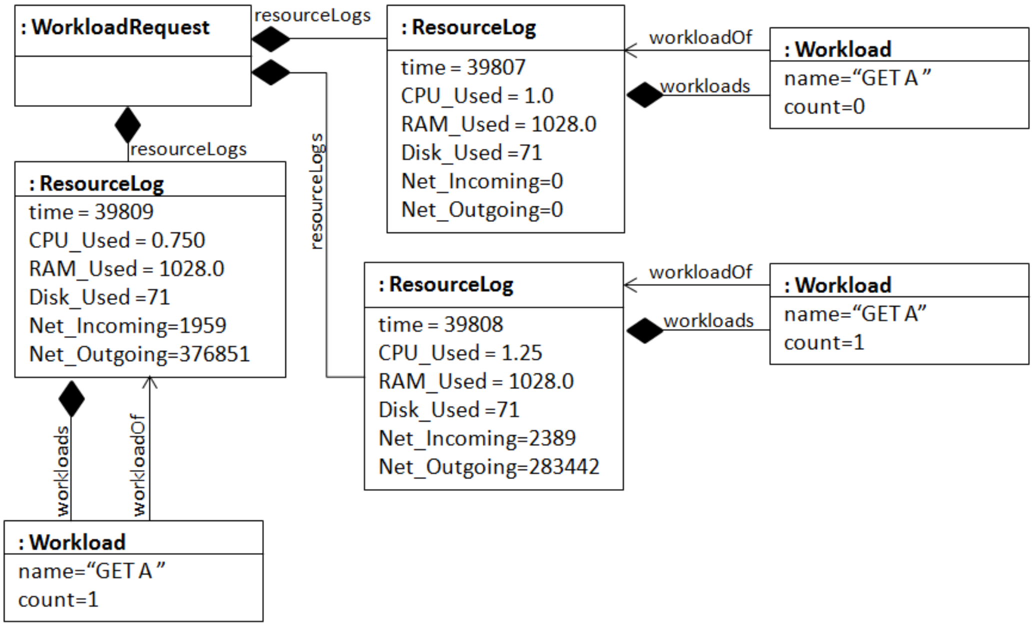


Figure 5.8: Example of WorkloadResourceVsTime Instance (for the DSML, see Figure 5.7).

iii. ***WorkloadResourceVsTime***

The *WorkloadResourceVsTime* DSML provides structures for correlating the occupancy of the system (number of active workloads) with the usage of each resource at that time. Figure 5.7 illustrates the design of this DSML. The number of active workloads is obtained by comparing and merging the *ApplicationWorkloadLog* and *ResourceLog* models. For example, comparing and merging the models presented in Figure 5.4 and Figure 5.6 produces the model presented in Figure 5.8.

iv. ***ResourceVsWorkload***

This DSML provides structures for grouping resource usage information by the number of concurrent workloads that the application was processing at the time of each measurement (i.e. how much CPU/memory etc. the machine consumed while processing 0, 1, 2 .. n concurrent requests). Figure 5.9 illustrates the design and Figure 5.10 demonstrates a model that conforms to this DSML.

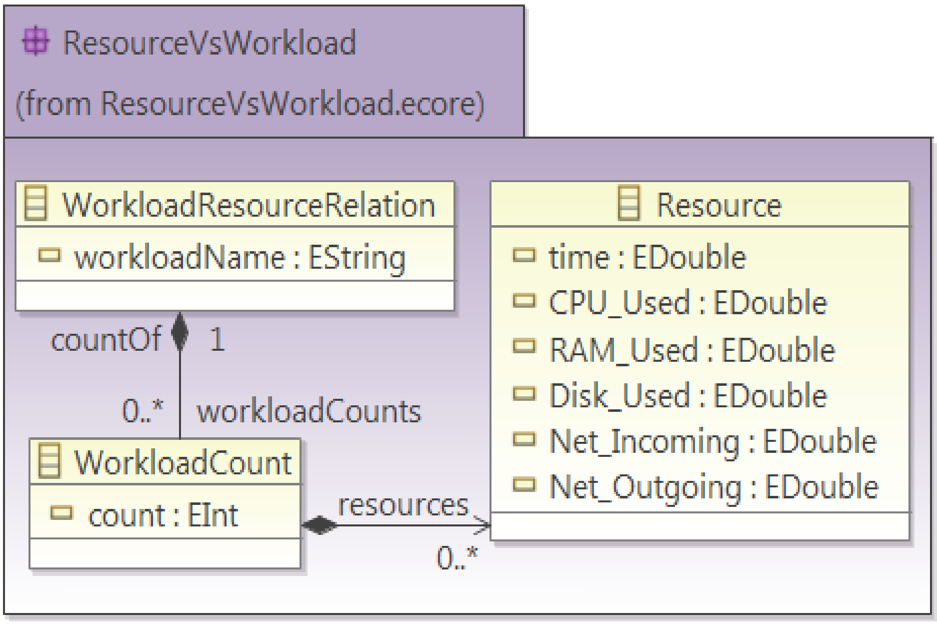


Figure 5.9: The Design of *ResourceVsWorkload* DSML.

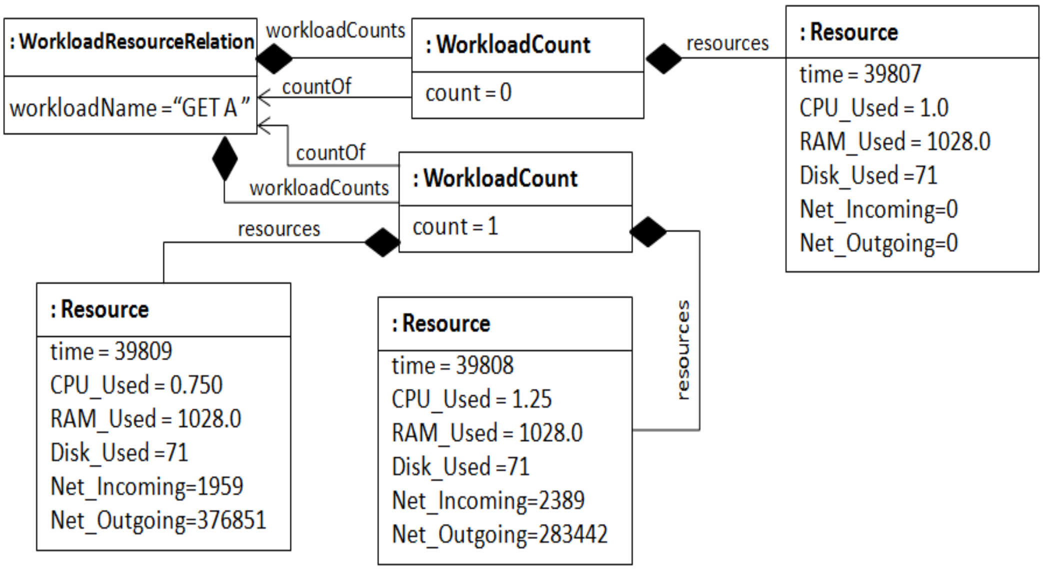


Figure 5.10: Example of ResourceVsWorkload Instance (for the DSML, see Figure 5.9).

v. ***WorkloadPattern***
The workloads are simulated and the observed arrival sequence is captured in a *WorkloadPattern* model. This model is used to adjust synchronisation errors in the *ResourceVsRequest* model caused by differences in the timestamps reported by the runtime and the VM monitoring tool. The design of this DSML is presented in Figure 5.11.

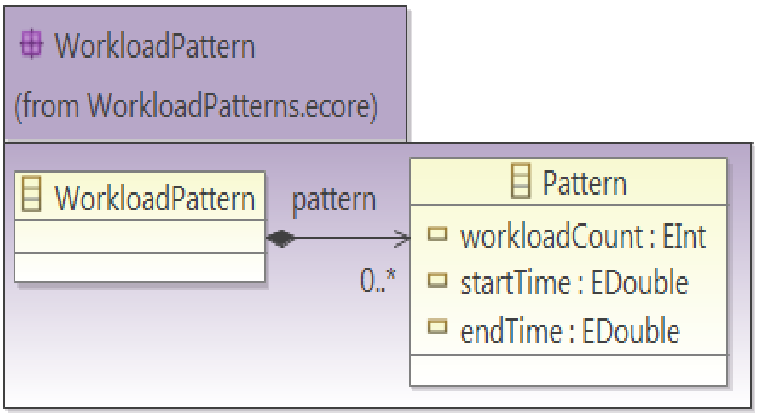


Figure 5.11: The Design of *WorkloadPattern* DSML.

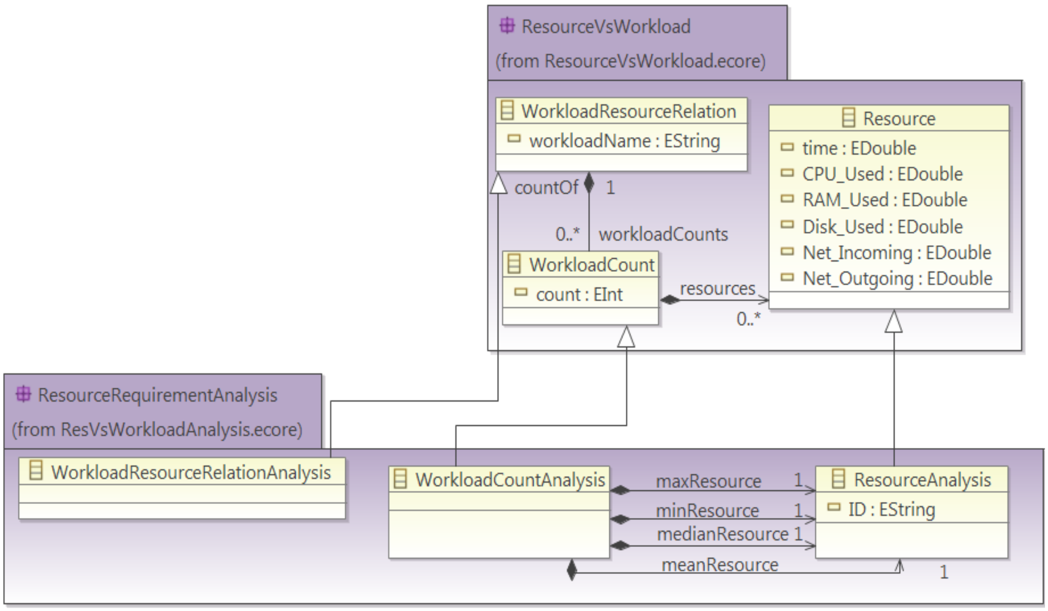


Figure 5.12: The Design of *ResourceRequirementAnalysis* DSML.

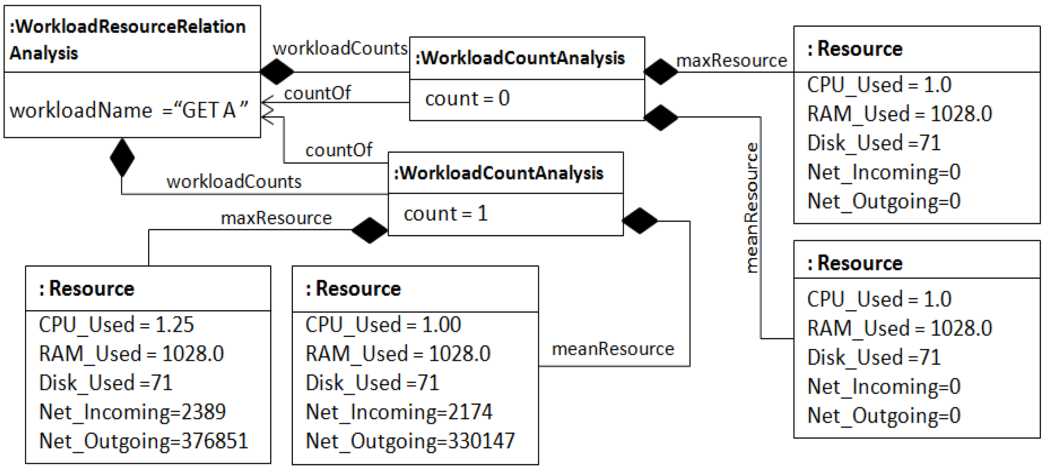


Figure 5.13: Example of ResourceRequirementAnalysis Instance (for the DSML, see Figure 5.12).

vi. ***ResourceRequirementAnalysis***
This DSML is an extension of *ResourceVsWorkload* and complements the information stored in models which conforms to the *ResourceVsWorkload* DSML. This contains additional information relating to particular analysis techniques. As illustrated in Figure 5.12, the *ResourceVsWorkload* DSML provides constructs for representing basic statistical analysis measures such as mean, median, maximum and minimum. The domain expert can select different analytical techniques to synthesise *ResourceVsWorkload* models and use them for capacity planning. The improvement on this DSML and its model management activities is related to applied equation extraction module. Figure 5.13 shows an instance of this DSML with the average (*meanResource*) and maximum (*maxResource*) resource usage measurements for each number of workloads.

5.2.2 Model Management Activities in ReRA

Several model management operations have been implemented for managing models conforming to the proposed DSMLs. The high level of this process was illustrated with a flowchart in Figure 4.2. The discussion of the model management techniques implemented in this thesis is referring to Figure 5.1 together with models conforming to the DSMLs. In addition, the abbreviations T2M, M2T and M2M stand for; Text-to-Model, Model-to-Text and Model-to-Model transformations respectively. The operations referred to in this section have been implemented with relevant components of the Epsilon framework. The functionality of each operation is explained below, in accordance to the numbering used in Figures 5.1 and 5.2:

i. **Workload Logs to *WorkloadLog* Models**
The workload information stored in the workload log is extracted through a T2M transformation into *ApplicationWorkloadLog* models. The number of *ApplicationWorkloadLog* models is equal to the number of machines running the application. Each machine has log recordings that are transformed to their respective *ApplicationWorkloadLog* models. The algorithm to produce an *ApplicationWorkloadLog* model with a name called *WorkloadLog* model is demonstrated in Algorithm 1. The input to this T2M transformation activity is the workloads log recording file for a virtual machine while the output is an *ApplicationWorkloadLog*

model which contains *ApplicationServer*, *VirtualMachine*, *ServerConfiguration* and *WorkloadLogRecord* objects. The first line creates the *ApplicationWorkloadLog* model as *workloadLog*. Lines 2 to 4 create *ApplicationServer*, *VirtualMachine* and *ServerConfiguration* objects named as *appSer*, *VM* and *serConf* respectively. The information for *VM* and *serConf* are entered by the user with interfaces provided by the program (lines 5 to 6). These objects are referred by *appSer* (lines 7 to 8). The *ApplicationServer* is the main class in *ApplicationWorkloadLog* DSML and therefore *appSer* is referred by *workloadLog* (line 9). The workload logs are recorded into a file in the event it happened. Therefore, transferring log recordings into *WorkloadLogRecords* and assigning them to *ApplicationServer* is called in a loop. Lines 10 to 14 demonstrates this algorithm to process the workload logs of a machine.

Algorithm 1 Workload Logs to a *WorkloadLog* Model

```
Input: workloadlogFile : File
Output: workloadLog : ApplicationWorkloadLog
 1: Create a new workloadLog : ApplicationWorkloadLog
 2: Create a new appSer : ApplicationServer
 3: Create a new VM : VirtualMachine
 4: Create a new serConf : ServerConfiguration
 5: serConf ← ...                    ▷ user inputs the values through interface
 6: VM ← ...                         ▷ user inputs the values through interface
 7: appSer.config ← serConf
 8: appSer.machine ← VM
 9: workloadLog.add(appSer)   ▷ adding ApplicationServer object (appSer)
    to WorkloadLog model (workloadLog)
10: while not workloadlogFile.EOF do
11:     Create a new logRecord : WorkloadLogRecord
12:     logRecord ←...               ▷ feed the data extracted from log records
13:     appSer.logRecords ← logRecord
14: end while
```

ii. **Resource Usage Logs to Resource Models**
Similarly, another T2M transformation extracts *ResourceLog Models* from *Resource Usage Log Files* captured using VM monitoring tools. The algorithm to perform this transformation is presented in Algorithm 2. The input is the resource usage log recording file and the output is *ResourceLog* model of that machine. The components of *ResourceLog* model are *UtilisationLog*, *Machine* and *LogRecord* objects. The first line

in Algorithm 2, creates a *ResourceLog* model with a name *resourceLog.* Lines 2 to 3 creates *utilLog* as *UtilisationLog* object and *machine* as *Machine* object. The machine information is entered by the user with interface provided by the program and *utilLog.machine* refers to that object (lines 4 to 5). The main component in *ResourceLog* model is *UtilisationLog* object, therefore *utilLog* is assigned to *resourceLog* in line 6. The resource usage log recordings are then transferred into a *LogRecord* object called *logRecord* and assigned to *utilLog* in a loop to capture the entire resource usage log records in that file.

Algorithm 2 Resource Usage Logs to a *ResourceLog* Model

```
Input: resourcelogFile : File
Output: resourceLog : ResourceLog
 1: Create a new resourceLog : ResourceLog
 2: Create a new utilLog : UtilisationLog
 3: Create a new machine : Machine
 4: machine ←...                    ▷ user inputs the values through interface
 5: utilLog.machine ← machine
 6: resourceLog.add(utilLog)    ▷ adding UtilisationLog object (utilLog) to
    ResourceLog model (resourceLog)
 7: while not resourcelogFile.EOF do
 8:     Create a new logRecord : LogRecord
 9:     logRecord ←...               ▷ feed the data extracted from log records
10:     utilLog.logRecords ← logRecord
11: end while
```

iii. **Sort, Compare & Merge Workload and Resource Models**
In this activity, M2M transformation with necessary adjustments, such as sorting and time conversion, are performed on the models before they can be compared and merged. The inputs of this operation are the *ApplicationWorkloadLog* and *ResourceLog* models. The output is a single *WorkloadResourceVsTime* model which combines *ApplicationWorkloadLog* and *ResourceLog* models. Algorithm 3 shows the logic in performing this activity. Line 1 creates a *WorkloadResourceVsTime* model as *workResVT* and line 2 creates *WorkloadResource* object as *workRes* which it is referred by *workResVT* at line 3. The *WorkloadLogRecord* objects in *workloadLog* model were sorted according to captured *startTime* at line 4. The observation with web applications shows that

the workload log is recorded once it complete. Therefore, the order of the recording is based on completion time (*endTime*) and need sorting according to start time to perform correlation with resource usage. The processor speed stored in *ResourceLog* model is assigned to a parameter named *processorSpeed* at line 5. This value is used to convert percentage of CPU usage recording into standardise unit (in this thesis is Mega Hz) as presented at line 10. The resource usage logs information stored in *ResourceLog* model were transformed to *ResourceLog* of *WorkloadResourceVsTime* model and associated workload being processed at that time are also captured. This operation is presented at lines 8 to 22 in a loop. Correlation based on time is performed at line 17 by counting the number of similar workloads being processed at the time the resource usage was recorded.

Algorithm 3 Sort, Compare & Merge Workload and Resource Models

```
Input: $workloadLog$ : WorkloadLog, $resourceLog$ : ResourceLog
Output: $workResVT$ : WorkloadResourceVsTime
Create a new $workResVT$ : WorkloadResourceVsTime
Create a new $workRes$ : WorkloadResource
$workResVT.add(workRes)$    ▷ adding WorkloadResource object (workRes) to WorkloadResourceVsTime model (workResVT)
Sort $workloadLog.LogRecord$ by $startTime$
$processorSpeed \leftarrow resourceLog.UtilisationLog.machine.cpuSpeed$

for $rl \in resourceLog.LogRecord$ do
    Create a new $resLog$ : $workRes.ResourecLog$
    $resLog.time \leftarrow rl.time$
    $resLog.CPU\_Used \leftarrow (rl.CPU\_Used * processorSpeed)/100$
    $resLog.RAM\_Used \leftarrow rl.RAM\_Used$
    $resLog.Disk\_Used \leftarrow rl.Disk\_Used$
    $resLog.Net\_Incoming \leftarrow rl.Net\_Incoming$
    $resLog.Net\_Outgoing \leftarrow rl.Net\_Outgoing$

    Create a new $workload$ : $workRes.Workload$
    $tmpCount \leftarrow$ count of $workloadLog.WorkloadLogRecord$ where it's
      $startTime \leq resLog.time$ and $endTime \geq resLog.time$
    $workload.count \leftarrow tmpCount$
    $workload.name \leftarrow workloadLog.WorkloadLogRecord.workloadName$
    $workload.workloadsOf \leftarrow resLog$
    $resLog.workloads \leftarrow workload$
    $workRes.resoureLogs \leftarrow resLog$
end for
```

iv. **Capacity Monitoring Graph Generation**
Resource usage and concurrent workloads count graphs are generated based on time. The outcome of this step is a set of graphs as illustrated in Figure 5.14 (these graphs originate from the Media Stream case study which is presented in Chapter 6). These graphs were generated based on information stored in a *WorkloadResourceVsTime* model by performing M2T transformation. For example, to produce graphs using Google graphs, the program needs to produce respective text files in the Google graphs format.

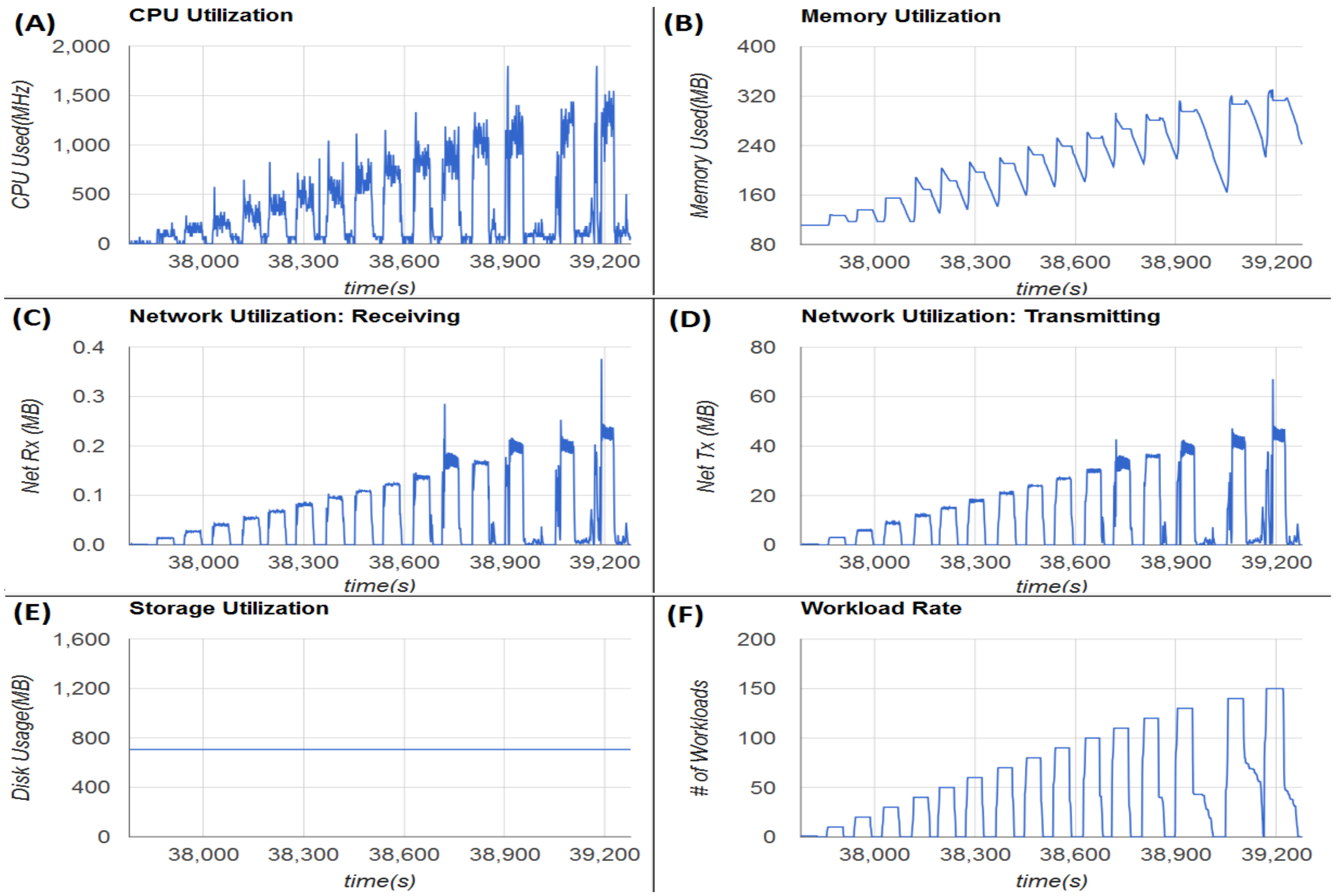


Figure 5.14: Sample of capacity monitoring graphs.

v. **Generation of *ResourceVsWorkload* Model**

The *WorkloadRequestVsTime* model is analysed and synthesised to produce a *ResourceVsWorkload* model. Multiple resource usage measurements are grouped by the number of concurrent workloads that the application was processing at the time they were recorded. The algorithm to transform a *WorkloadRequestVsTime* model into a *ResourceVsWorkload* model and is presented in Algorithm 4. The input for this activity is *WorkloadRequestVsTime* model and the output is *ResourceVsWorkload* model. In Algorithm 4, those models are named as workResVT and resVwork. The object of *WorkloadResourceRelation* class is the base for *ResourceVsWorkload* model, and it is created as *wvr* at line 3. The name of the workload is copied from *wls* (*Workload* object of *WorkloadRequestVsTime* model) at line 4. Individual ReRA process need to be performed to retrieve resource requirement formulas for a given workload type. Therefore, the workload name should be distinctive. At line 5, the set of unique counts of *WorkloadRequestVsTime* model's workloads (*wrt.Workload.count*) are captured in a collection variable named *workCountSet*. This count represents the number of concurrent workload being processed at a time. For each workload count in *workCountSet*, a WorkloadCount object of *WorkloadResourceRelation* model (*tmpWC*) is created. The count value from *ResourceVsWorkload* model is assigned to *tmpWC* and the resource usage recordings to process that concurrent workload are transferred into the *Resource* object of *ResourceVsWorkload* model (*tmpRes*). These operations are conducted in the nested loop demonstrated at lines 6 to 20.

Algorithm 4 Producing a *ResourceVsWorkload* Model

```
Input: $workResVT$ : WorkloadResourceVsTime
Output: $resVwork$ : ResourceVsWorkload
$wls \leftarrow$ all $workResVT.workloads$ and point on first
$wrt \leftarrow$ all $workResVT.WorkloadResource$
$createnewwvr : resVwork.WorkloadResourceRelation$        ▷ WorkloadResourceRelation object (wvr) is created and added to ResourceVsWorkload model
$wvr.workloadName \leftarrow wls.name$
$workCountSet \leftarrow ...$   ▷ a set of unique numbers in $wrt.Workload.count$
for $wCount \in$workCountSet do
    create new $tmpWC : resVwork.WorkloadCount$
    $tmpWC.count \leftarrow wCount$
    for $workload \in wls$ where its $count = wCount$ do
        create new $tmpRes$ : resVwork!Resource
        $tmpRes.time \leftarrow workload.workloadOf.time$
        $tmpRes.CPU_Used \leftarrow workload.workloadOf.CPU\_Used$
        $tmpRes.RAM_Used \leftarrow workload.workloadOf.RAM\_Used$
        $tmpRes.Disk_Used \leftarrow workload.workloadOf.Disk\_Used$
        $tmpRes.Net_Incoming \leftarrow workload.workloadOf.Net\_Incoming$
        $tmpRes.Net_Outgoing \leftarrow workload.workloadOf.Net\_Outgoing$
        $tmpWC.resources \leftarrow tmpRes$
    end for
    $wvr.workloadCounts \leftarrow tmpWC$
end for
```

vi. **Cleansing**
Performing step v (Generation of *ResourceVsWorkload* Model) can produce noise that can affect correlation due to time synchronisation issues between the host and the VMs. *WorkloadPattern Model* is used to clean the *ResourceVsRequest Model.* This step is optional.

vii. **Produce Analysis Model**
Statistical analysis to get the minimum, maximum, mean and median value of resource usage measurements was applied to the *ResourceVs-Request* model and the results are stored in a *ResourceRequirementAnalysis* model. Algorithm 5 shows the transformation rules applied when transforming *ResourceVsWorkload* model (source) to *ResourceRequirementAnalysis* model (target). Lines 3 to 6, transform the relationship between workload and resource usage (*WorkloadResourceRelation* of *ResourceVsWorkload* model) to analytical values by capturing the number of workloads (*workloadsCounts*) and statistical values for the interested resource metrics along with the workload names (*workloadName*) into the *ResourceRequiremntAnalysis* model. At line 5, the second transformation was applied, where operations were called upon to calculate the minimum, average, maximum and median resource requirement for the number of concurrent workloads. These functions were represented as *getMin*, *getAvr*, *getMax* and *getMedian* accordingly at lines 10 to 13 in Algorithm 5.

Algorithm 5 Producing Analysis Model

```
Input: $source$ : ResourceVsWorkload
Output: $target$ : ResourceRequirementAnalysis
create new $target.WorkloadResourceRelationAnalysis \leftarrow source.WorkloadResourceRelation$    ▷ transformation1 is called

rule transformation1 transform source.WorkloadResourceRelation to target.WorkloadResourceRelationAnalysis
    $target.workloadName \leftarrow source.workloadName$
    $target.workloadCounts \leftarrow source.workloadCounts$    ▷ rule transformation2 is called
end rule

rule transformation2 transform source.WorkloadCount to target.WorkloadCountAnalysis
    $target.count \leftarrow source.count$
    $target.minResource \leftarrow source.getMin()$
    $target.meanResource \leftarrow source.getAvr()$
    $target.maxResource \leftarrow source.getMax()$
    $target.medianResource \leftarrow source.getMedian()$
end rule
```

viii. **Resource Requirement Vs Request Count**
A set of graphs are generated to visualise resource metrics and workloads relationship obtained through the analysis conducted above. The graphs are plotted to illustrate the extracted equation (linear, quadratic and etc.) of resource consumption. For example, Figure 5.15 shows the example of CPU requirements based on number of workloads being processed. Basic statistic elements such as median, mean, minimum and maximum values were calculated and presented in this type of graph.

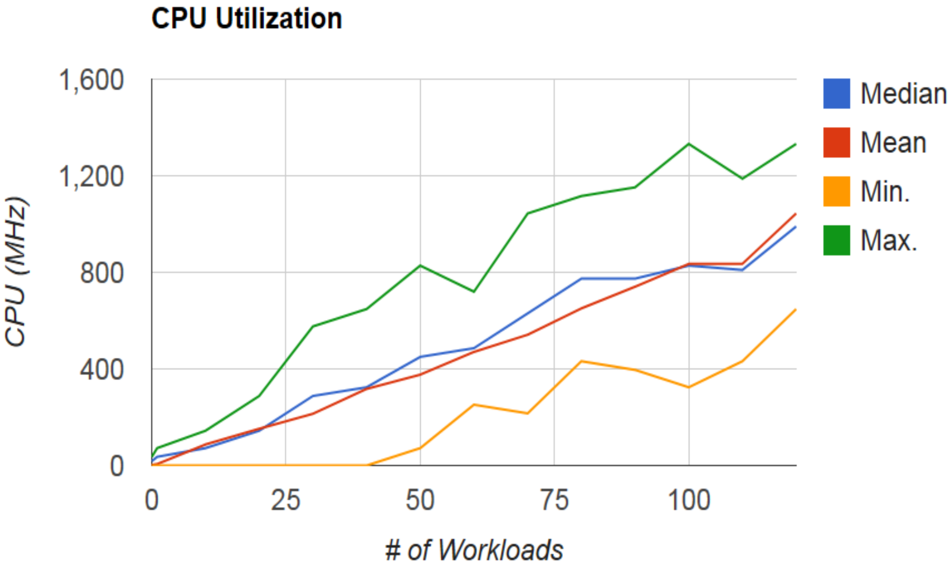


Figure 5.15: Example of resource utilisation graphs for CPU based on workloads.

ix. **Generating Formulas of Resource Metrics**
A CSV file is produced by extracting selected statistical data from *ResReqAnalysis* models. This file is analysed with data analysis tools to produce correlation formulas for resource metrics. In this research, Matlab was used to produce the formulas. However, any suitable tools can be used to produce the formulas.

The outcome in performing ReRA are: i) a set of resource requirement formulas for resource metrics and ii) graphical presentations of resource requirements based on workloads. The formulas are used in the transformation to estimate resource requirements based on workload patterns which are captured in the WL-DSL of the applications.

5.3 Virtual Resource Requirement

In this section, the MDE solutions of the proposed framework are discussed with an abstract DSMLs for the application workload since each application workload is unique. General rules in transforming application workload to resource requirements and a DSML for capturing resource requirement schedules are also presented.

5.3.1 DSMLs Design of ViRR

Mainly, two DSMLs are proposed to integrate first two phases of the capacity planning process in a virtualised environment. Since *WL-DSLs* is specific to an application, an abstract DSML named *ApplicationWorkload* which acts as a common core for individual WL-DSLs is proposed. This DSML is expandable to model application-specific DSML which capture workloads of a particular application. An Application workload model is transformed into a *ResourceRequestPlan* model using model-to-model transformations. Figure 5.16 illustrates the abstract syntaxes of the two DSMLs.

Application workloads are specified for hour-long time periods as *TimeSlotRequests*. Each application has a unique identifier and the capacity planning period is captured using a *startDate* and an *endDate*. Each period is captured as a *TimeSlotRequest* with a start time (*from*) and an end time (*to*). The *ResourceRequestPlan* DSML acts as a resource request calendar for the capacity planning period. It shows the resource requirements for a particular application on a daily basis with time slots described in the related WL-DSL.

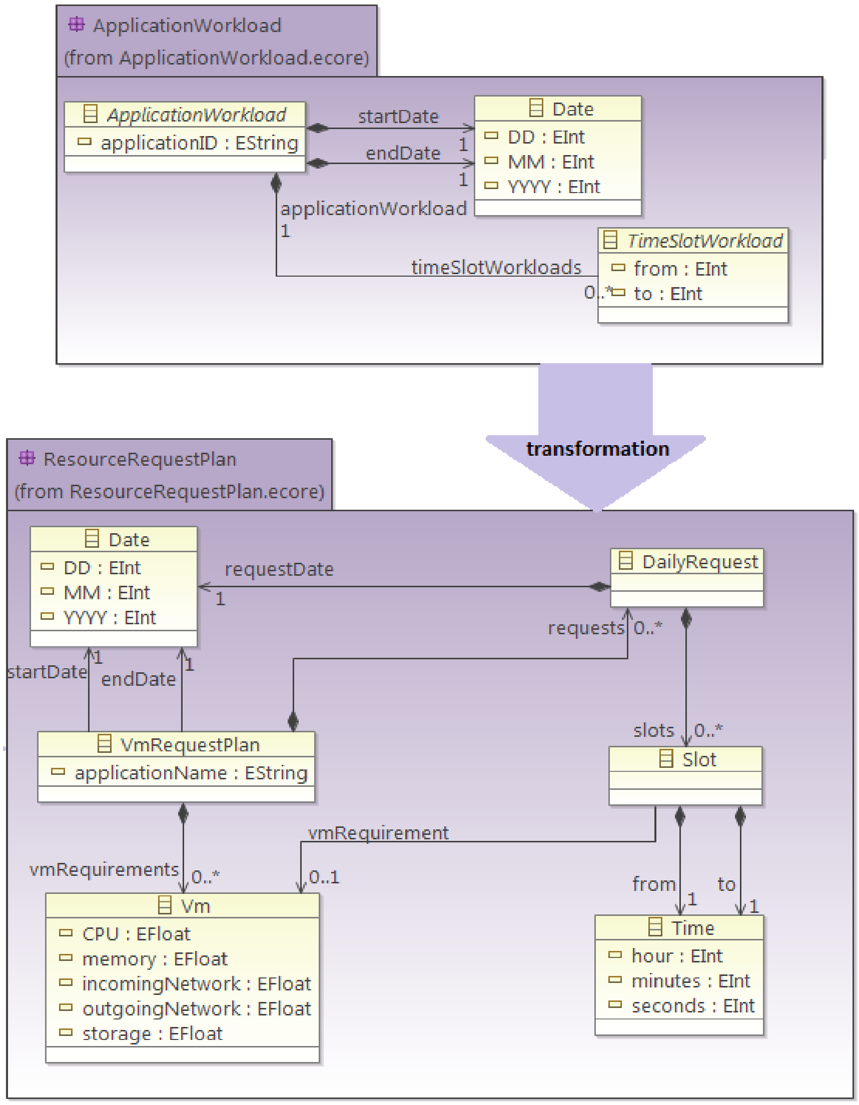


Figure 5.16: DSMLs of ViRR for an application.

A daily resource requirement timetable is generated for a specified capacity planning period (*startDate* and *endDate* of *VmRequestPlan*).

At this time, Epsilon does not natively support date and time data types. Therefore, the *Date* is recorded with three attributes (*dd* as day of the month, *mm* as month of the year and *yyyy* as the year). These attributes are integers and the respective model management activity manipulates the values. This *Date* format has been applied throughout the thesis. The same was applied to *Time*, with *hour*, *minutes* and *seconds* as attributes. The time granularity in *ResourceRequestPlan* DSML was increased up to seconds, although the application workloads are specified for hour-long time periods. This is to enable the usability of the design in the event of application of resource allocation policy.

5.3.2 Model Management Technique

A model-to-model transformation comprising two transformation rules is used to map application workloads to virtual machine requirements schedule. The formulas for resource metrics are retrieved through the model management techniques of ReRA process discussed in Section 5.2.2. These formulas are used in the transformation when estimating resource requirements for the time slot defined in the application workload model (line 18). Algorithm 6 shows the structure and the rules applied in transforming application workloads to resource requirement plans. The input for this activity is an *ApplicationWorkload* model (*source*) and the output is a *ResourceRequestPlan* model (*target*). The nested loop in lines 6 to 13 generates a calendar-based resource requirements plan.

Algorithm 6 Transforming Application's Workloads to VmRequestPlan

```
Input: $source$ : ApplicationWorkload
Output: $target$ : ResourceRequestPlan
create new $target.VmRequestPlan \leftarrow source.ApplicationWorkload$    ▷ rule1 is called

rule rule1 transform source.ApplicationWorkload to target.VmRequestPlan
    $currentDate \leftarrow source.startDate$
    $noOfDays \leftarrow source.endDate - source.startDate$
    for $i = 0$ to $noOfDays$ do
        create new $requestDate$ : Date
        for $timeSlotWorkload \in source.timeSlotWorkloads$ do
            create new $target.VM \leftarrow timeSlotWorkload$ ▷ rule2 is called
            ...                              ▷ details of target.VM.slot are captured
        end for
        $currentDate \leftarrow currentDate + 1$      ▷ currentDate to next date
    end for
    ...                   ▷ startdate and enddate of VmRequestPlan are assign
end rule

rule rule2 transform source.TimeSlotWorkload to target.VM
    ...            ▷ resource metrics of target.Vm are assign based on ReRA formulas
end rule
```

5.4 Chapter Summary

The design of the DSMLs and model management activities for ReRA and ViRR were presented in this chapter. The DSMLs were discussed with their structure and semantics. The respective model management activities were presented as algorithms in pseudo-code. The following chapter demonstrates the implementation of the discussed processes on two case studies.

Chapter 6

Applications of MDE Solutions

6.1 Introduction

In this chapter, the implementation plan to apply the proposed MDE solutions presented in Chapter 5 and the requirement to use case studies were explained. The focus of this chapter is related to implementing the proposed MDE solutions as an exploratory phase. The system level requirements and the application of the proposed MDE solutions with two case studies are discussed in this chapter. The general design of DSMLs together with the model management operations were discussed in the previous chapter. In this thesis, the DSMLs and respective model management programs were implemented using EMF (Ecore) and Epsilon. It should be noted that the core of the proposed approach is not bound to these technologies and that any other 3-level metamodelling architecture and compatible model management platform could be used instead for implementation purposes.

Two web applications running in a virtualised environment (VirtualBox) are used as case studies (Media Stream and Part of Speech Tagging) in this chapter to demonstrate and refine the proposed approach. An additional web application was used for evaluation of the proposed MDE solutions and is elaborated in the following chapter. The architecture of the virtualisation environment is presented first, before discussing additional DSMLs developed specifically for each case study. The outcome of each case study, the improvements to the process and the MDE solutions specific to this visualisation environment are discussed.

6.2 Implementation Plan

As discussed in Section 1.6, an experimental research method was employed, under which method, exploratory is defined as the first phase, which is followed by an evaluation phase [29]. In this thesis, an exploratory phase was conducted as implementation where the proposed MDE solution discussed in Chapter 5 was applied using EMF(Ecore) and Epsilon with two case studies running in a virtualised environment. Improvements were made while applying the proposed MDE solutions in the exploratory phase to provide sufficient facilities before proceeding to the evaluation phase.

In the exploratory phase, the system requirements and its components were identified. These were discussed in detail in Section 6.3. Mainly, the technical tasks were conducted in this phase as preparation for the evaluation. The implementation of the proposed MDE solutions was reviewed and improved using two case studies running in a virtualised environment. The ideal case studies for this approach are applications that receive concurrent and homogeneous requests. The improvements were added to provide sufficient facilities for evaluation.

In the evaluation phase, a third case study was used to evaluate the reusability and modularity of the proposed MDE solutions. The findings from the

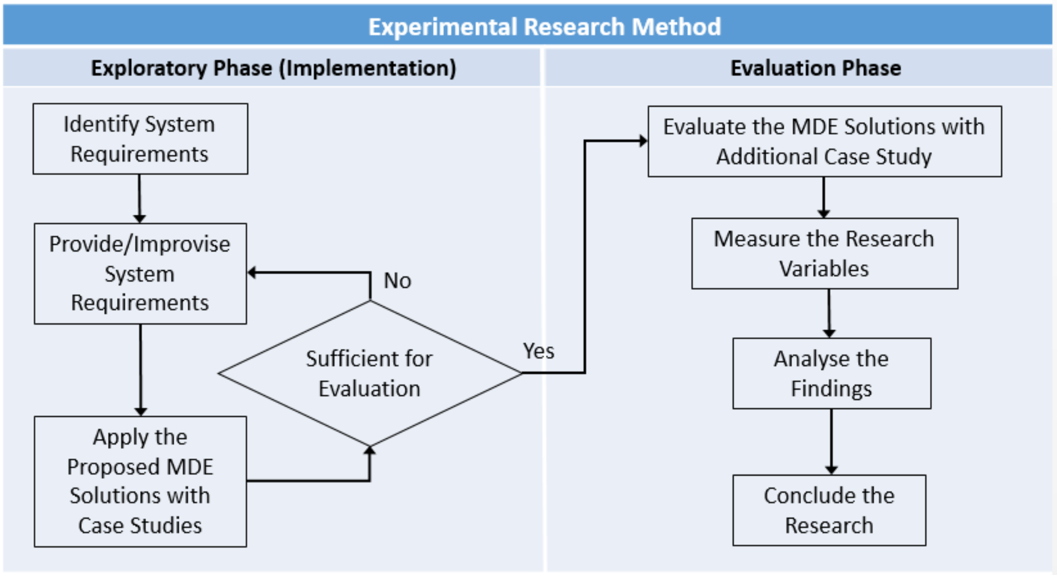

Experimental Research Method
Exploratory Phase (Implementation)
Evaluation Phase
Identify System Requirements
Provide/Improvise System Requirements
No
Sufficient for Evaluation
Yes
Apply the Proposed MDE Solutions with Case Studies
Evaluate the MDE Solutions with Additional Case Study
Measure the Research Variables
Analyse the Findings
Conclude the Research

Figure 6.1: Implementation (Exploratory Phase) and Evaluation Plan of the Research.

evaluation were analysed in Chapter 7 and the conclusions were discussed in Chapter 8.

The processes involved in both phases are illustrated in Figure 6.1. A detail discussion of the first phase (implementation) is presented in the following sections of this chapter. The discussion of the second phase (evaluation) is presented in the following chapter.

6.3 System Requirements

The general design of DSMLs and model management operations for the ReRA and ViRR processes were discussed in Chapter 5. The requirements for the system that implements these processes are listed below:

i. The system shall provide accurate predictions of capacity;

ii. The system shall support multiple statistical analysis techniques;

iii. The system shall make use of rigorously-defined models for all configuration and data collection activities; and

iv. The system shall make use of model management techniques for all validation and transformation activities.

For the processes to be applicable, the following assumptions must hold:

i. The resource utilisation and application workload log recordings for the application under capacity analysis must be available; and

ii. The virtualisation technology must enable reliable correlation of resource utilisation and application workload log recordings.

The target user of the proposed solution is a capacity planning manager who needs to estimate resource requirements for predicted workloads. Target users are expected to be familiar with the technologies related to the following components of the system:

i. *Resource monitoring tool/s* to capture the resource utilisation of the virtual machine with timestamps.

ii. *Workload simulator* to generate the workload of the application.

iii. *Workload log record* which captures the application's workload being processed with timestamps.

iv. *Model management tool* to apply the proposed DSMLs and the model management operations.

v. *Virtualised environment* (server virtualisation) as a platform to host an application. The hypervisor of the selected virtualised environment may be either full or para virtualisation.

vi. *Applications* hosted in the virtualised environment as the case studies. Ideal case studies for this approach are applications that receive concurrent and homogeneous requests.

vii. *Statistical method* to analyse the correlation between workloads and resource utilisation.

viii. *Equation extraction tool* to produce correlation formulas between workload and resource utilisation.

Table 6.1 summarises the tools/methods used to demonstrate the application of the proposed MDE solutions in this thesis. It is also possible to apply other related tools/methods which accommodate the discussed components in order to implement the proposed MDE solutions.

6.4 Virtualised Environment

A virtualised environment is necessary in order to conduct the two case studies presented in this chapter and it refers to server virtualisation as discussed

No.	**Components**	**Tools/Methods**
i.	Model management tool/s	EMF (ecore) and Epsilon
ii.	Virtualised environment	VirtualBox
iii.	Resource monitoring tool/s	VBoxManage and disk file utility (df)
iv.	Workload simulator	JMeter
v.	Workload log record	Apache log (access.log file)
vi.	Application	Web applications
vii.	Statistical method	Basic statistic (mean, maximum, minimum, median)
viii.	Equation extraction tool	Matlab

Table 6.1: System Components that Satisfy the System Level Requirements to Apply the Proposed MDE Solutions.

in Section 2.2.3. The main components of server virtualisation are the hypervisor, host and virtual machine. Both Type 1 (full virtualisation) and Type 2 (para virtualisation) hypervisors have different influences on the performance of the virtual machine [5, 20]. Therefore, the resource requirements for operating an application in different virtualisation environments might vary. To facilitate the capacity planning manager, who hosts an application in a virtualised environment, ReRA provides modular and reusable steps/components to produce relevant resource requirement formulas that are specific to the selected virtualised environment. The formulas are then used in ViRR to estimate the resource requirements based on the predicted workload of the application running in that virtualised environment.

In this thesis, the virtualised environment is set-up using VirtualBox[1], a stable and freely available virtualisation product by Oracle. VirtualBox supports several types of operating systems for host and VMs, such as Linux, Windows and OS X. The virtualised environment presented in this chapter uses Linux in both host and VMs, since it is freely available. VirtualBox comes with a resource monitoring tool called VBoxManage[2]. The following is a detailed design of this virtualised environment.

Listing 6.1: VBoxManage script to collect VM resource utilization for every second.

```
VBoxManage metrics collect --period 1 --samples 1 ServerName[3]
    CPU/Load/User ,CPU/Load/Kernel ,RAM/Usage/Used , Disk/Usage/
    Used , Net/Rate/Rx, Net/Rate/Tx, Guest/CPU/Load/User , Guest/CPU
    /Load/Kernel , Guest/CPU/Load/Idle , Guest/RAM/Usage/Total ,
    Guest/RAM/Usage/Free , Guest/RAM/Usage/Balloon , Guest/RAM/
    Usage/Shared , Guest/RAM/Usage/Cache , Guest/Pagefile/Usage/
    Total > FileName.vbm
```

A server with 2 dual-core 2GHz CPUs, 8GB memory and 300GB disk space was used to host the virtualised environment. Para virtualisation (virtualisation Type 2) is implemented in VirtualBox. Therefore, an operating system (OS) needed to be installed in the host, so Linux OS Ubuntu 12.04 was used. The resource utilisation of the host and VM can be monitored using VBoxManage, where textual resource utilisation information is captured. This information can be recorded in a text file and the data were extracted to create a *ResourceLog* model.

The script in Listing 6.1 shows the VBoxManage commands executed in

[1] https://www.virtualbox.org

[2] https://www.virtualbox.org/manual/ch08.html

[3] The name of the VM needs to be stated replacing ServerName

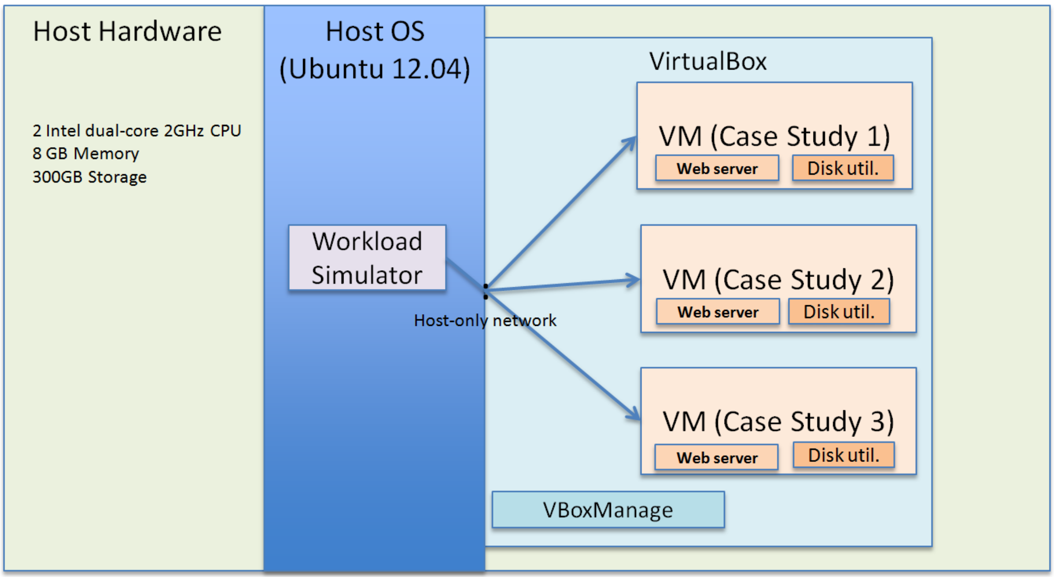


Figure 6.2: The architecture of the virtualisation environment.

the host used to record the resource utilisations of the VM. The resource utilisation log information file produced by VBoxManage was recorded with a *".vbm"* extension. A sample of this log file is presented in Listing E.3 and the list of resource parameters managed by VBoxManage is presented in Appendix B. Since this resource monitoring tool runs in the host, the tool only consumes computing resources of the host.

A limitation in the storage monitoring capabilities of VBoxManage was identified via observation of resource utilisation in the second case study. The reported storage usage metrics do not represent the dynamic changes of disk usage. Therefore, an additional disk monitoring tool in Ubuntu (*disk filesystem* utility) was used to record disk utilisation along with VBoxManage for this virtualised environment. A shell script was used to record disk utilisation reading into a *"log.dsk"* file. A sample of this log file is shown in Listing E.5. Although, this script runs in the VMs, it does not consume many resources, and as such the influence of monitoring resource usage on the application workload was ignored.

The architecture of the virtualised environment is presented in Figure 6.2. The VMs were configured to host the two selected web applications. Each web application was hosted in a different VM. The workload of web applications is recorded by the web server that hosts them. In this work, both applications were hosted in an Apache 2.4 web server and the workload was recorded in the *access.log* file of the Apache web server. Several configuration tasks may need to be performed to enable the required information to be recorded. For example, the default recording in the *access.log* file was configured to obtain detailed information about each *request*, such as request name, start time, end time, response code and size of data returned. Appendix A shows a list of configuration parameter names for the most commonly used web servers.

Listing 6.2 shows the configuration in the Apache web server (apache2.conf) to produce request log recordings (access.log). Web servers normally have a default configuration and need to be configured to produce required values. For example, in Listing 6.2, the default format of the access.log file in line 1 was disabled using the '#' symbol and replaced with the format in line 2. The meaning of the symbols can be referred at Appendix A. The updated format increases time granularity from seconds to milliseconds with the additional parameter '%{usec_frac}t' and also includes the duration it takes to complete the request with the additional parameter '%D'. Analysis of web servers presented in Appendix A shows a list of configuration parameter names for the most commonly used web servers. The web servers are configurable to produce request log recordings as input to produce the *RequestLog* model with text-to-model (T2M) transformation program. In this thesis, the T2M transformation program was written in Java. In EMF, models are can be presented in XML. Therefore, the T2M program extracts the information from the access.log file and feeds it into *RequestLog* model by producing XML file. Example of access.log file and *RequestLog* model in XML form are presented in Listings E.1 and E.2, respectively in Appendix E.

Listing 6.2: Configuration in apache2.conf file

```
1 #LogFormat "%h %l %u %t \"%r\" %>s %O \"%{Referer}i\" \" %{
      User-Agent}i\"" combined
2 LogFormat "%h %l %u %t %{usec_frac}t %D \"%r\" %>s %O \" %{
      Referer}i\ "\"%{User-Agent}i\"" combined
```

The network of the virtualisation environment needed to be configured to enable communication between the host and the VMs. For example, there are several ways to configure the network in VBoxManage and a host-only network was included to enable VM-to-VM and also host-to-VM communication.

A workload simulator generates application workloads and JMeter was used to simulate web requests of the applications as workloads. JMeter is an application to perform load tests on client/server software such as web applications. It also can be used to simulate a heavy load on a server, network or object to test its strength or to analyse overall performance under different load types [39]. Running the simulator within the VM where the application is hosted can consume additional resources. Therefore, the simulator needs to be installed in an external server. In this work, the simulator was installed in the host and generated workloads for the applications running in the VMs.

The following sections discuss the implementation of the proposed MDE solutions for the two case studies. Media Stream and Part of Speech Tagging web applications running in a VirtualBox virtualised environment were used as case studies in this chapter to demonstrate and refine the proposed MDE approach. Both case studies had a highly intensive resource utilisation and this enables workload correlation with resource utilisation. VirtualBox provided a sufficient virtualisation environment to conduct the research to meet the system requirements discussed in Section 6.3.

6.5 Extension of ReRA

The general ReRA process together with a set of high level DSMLs was discussed in Section 5.2. In this chapter, the designed DSMLs were extended and reused specific to web applications running in the VirtualBox virtualised environment.

The specific process used for web applications in the selected virtualised environment is presented in Figure 6.3. This figure is an extended version of Figure 5.1 presented in Chapter 5 in which the high level DSMLs described in Chapter 5 were used. The type and the name of the model management programs developed to automate the process are presented in the diagram and the code is available in Appendix G. Additional steps and models are required to produce *ResourceLog* models, since more than one resource monitoring tool are used. The use of two resource monitoring tools (VBoxManage and the Ubuntu disk utility) in the virtualisation environment has been discussed in Section 6.4. The log records produced by VBoxManage and the Ubuntu disk file system (df) utility were parsed into *VBoxVMMetrics* and *DiskUsageLog* models, respectively. These two models were merged and transformed into a *ResourceLog* model.

The overall design and the relationship of the DSMLs are presented in Figure 6.4. The *RequestLog* DSML is specific to web applications and was designed by extending *ApplicationWorkload*. Additional two DSMLs to capture

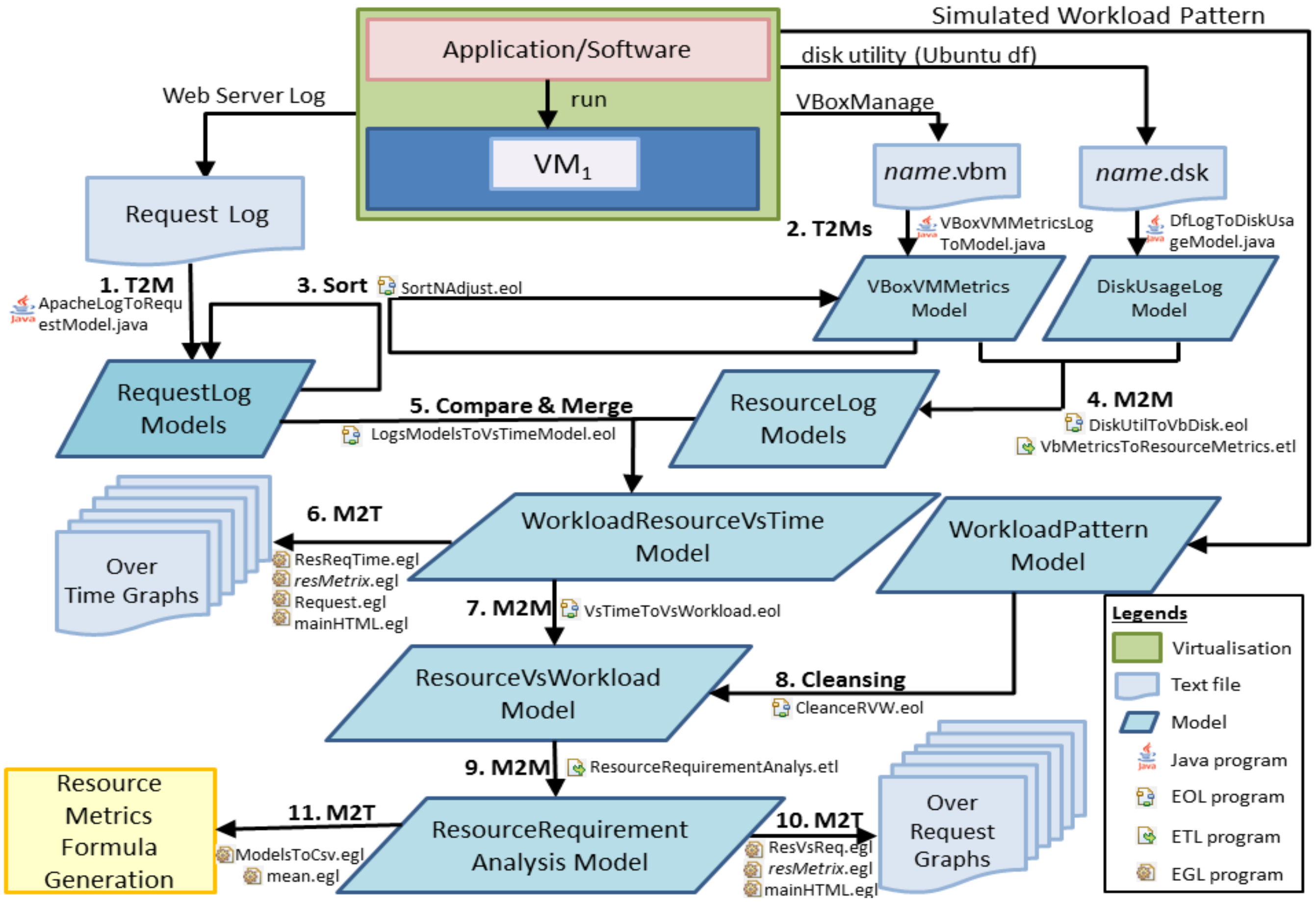


Figure 6.3: The ReRA Process with DSMLs and Model Management Techniques for Web Applications in the VBoxManage environment. (The numbers represent the flow of activities involved together with model management programs.)

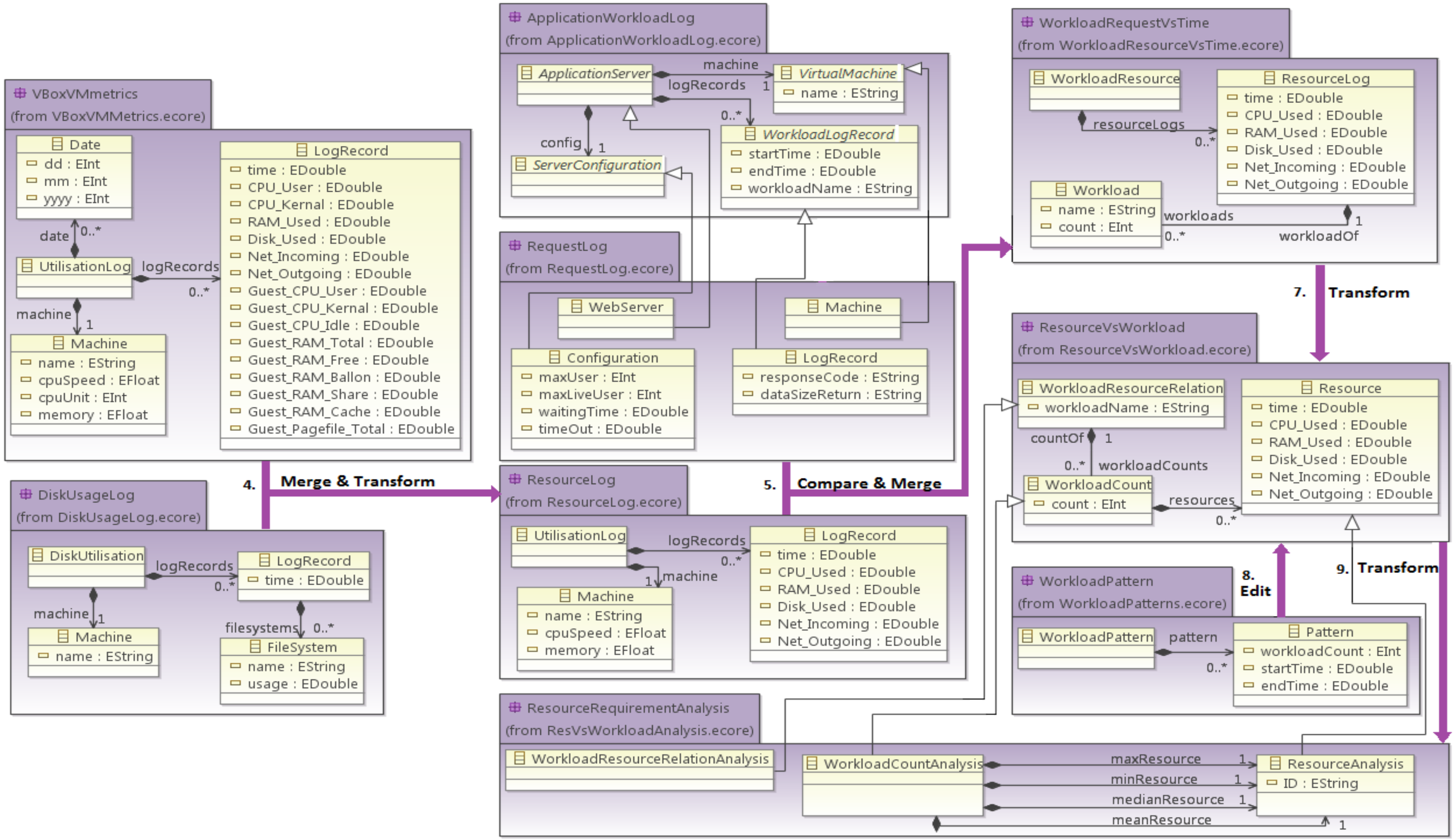


Figure 6.4: DSMLs Design for ReRA. *VBoxVMmetrics* and *DiskUsageLog* DSMLs are VirtualBox virtualisation environment specific. *RequestLog* DSML is specific for Web Applications and the others are general.

the log recordings from VBoxManage and the Ubuntu disk utility specific to the virtualised infrastructure were used. The other DSMLs and model management activities in Figure 6.4 are as presented in Chapter 5. Detailed discussion on these additional DSMLs and model management techniques follows.

6.5.1 Additional and Extended DSMLs of ReRA

Three additional DSMLs were specifically developed for this experiment. *DiskUsageLog* and *VBoxVMMetrics* are VirtualBox specific and *RequestLog* is specific for web applications. The following are the additional DSMLs:

i. ***DiskUsageLog***
Captures information related to disk utilisation. The storage usage (kilo bytes) of the file systems in the machine is captured together with the reading time. The storage usage unit is converted to megabytes with the model management activity discussed in Section 6.5.2 to ensure that the units are standardised. Figure 6.5 shows the design of this DSML. This DSML is designed specifically to capture log recording of the *disk filesystem utility*. An example of this log is presented in Listing E.5. The *time* of *LogRecord* in Figure 6.5 represents the second of the day to enable correlation based on time.

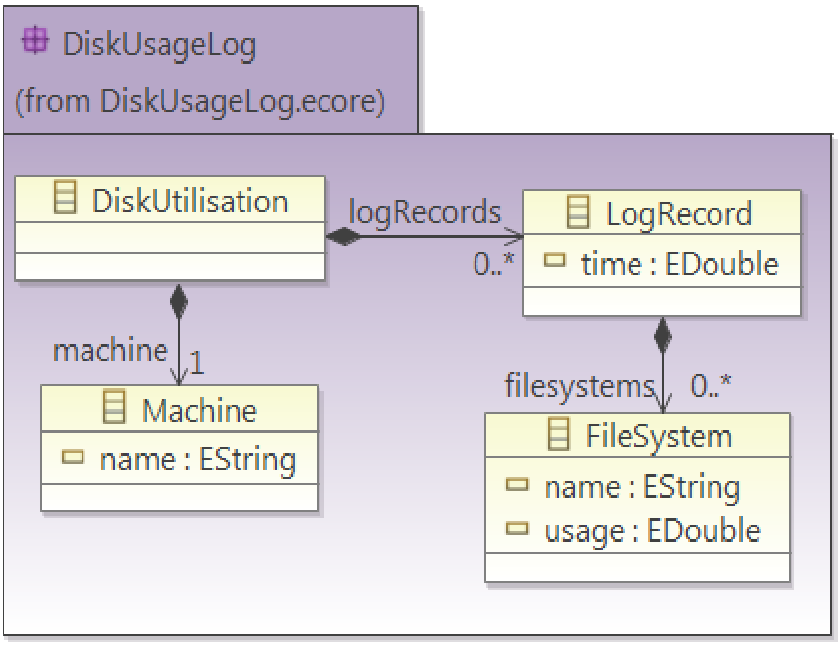


Figure 6.5: The design of *DiskUsageLog* DSML.

ii. ***VBoxVMMetrics***

Captures information related to the resource utilisation parameters of the VM which are produced by VBoxManage. Along with this, the VM specifications such as the CPU speed (MHz), memory (MBs) allocated and number of CPU units are captured. The abstract syntax of this DSML is illustrated in Figure 6.6. The attributes of *LogRecord* represent the resource metrics produced by VBoxManage. A sample of this log recording is presented in Listing E.3. The *time* in *LogRecord* stands for the seconds of the day. The log recording does not provide date and machine information. Therefore, the information of *Date* and *Machine* is input through the interface of the related T2M program. Date information is captured to allow a time adjustment to be performed if necessary. For example, in the virtualised environment, VBoxManage acquires time information from the BIOS which is not automatically adjusted for daylight savings. However, the OS is automatically updated with daylight savings. Therefore, an adjustment of the time recorded is required for the resource utilisation recorded within VBoxManage. This configuration information is later transferred to the *ResourceLog* model.

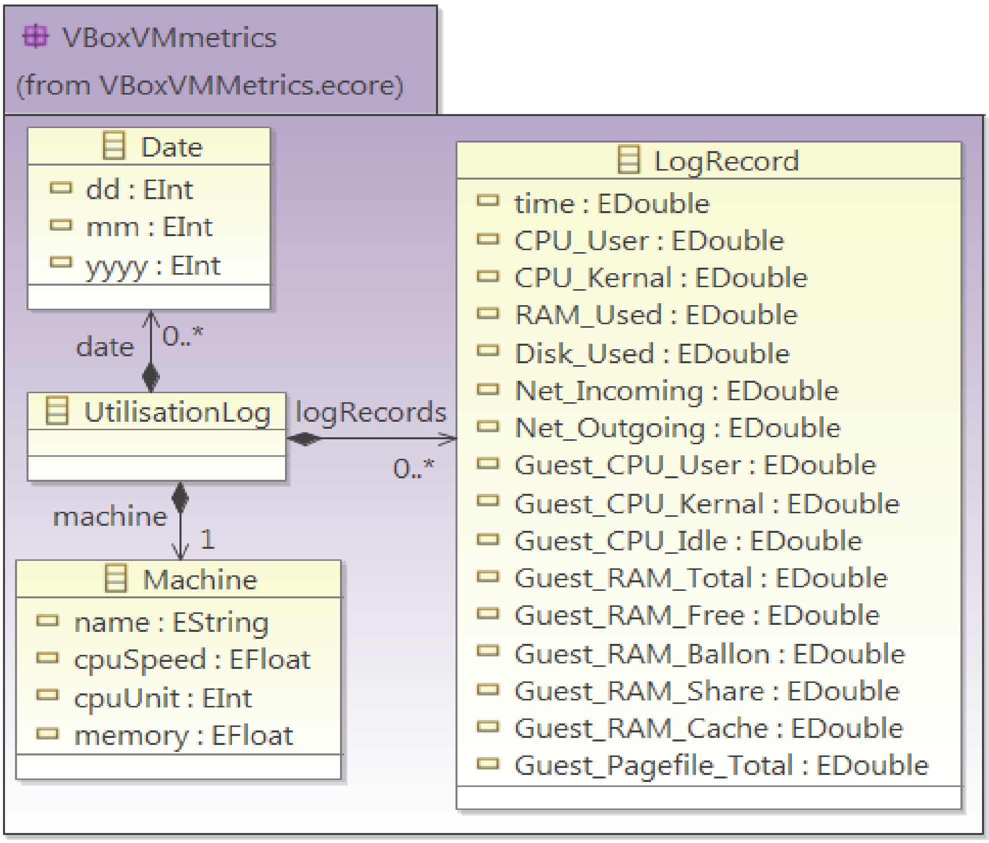


Figure 6.6: The design of *VBoxVMMetrics* DSML.

iii. ***RequestLog***
This application DSML is specific for web applications. It inherits classes from the *ApplicationWorkloadLog*, which is where information related to incoming requests is captured. These include; the start time, end time and the name of each workload (request) extracted from the web server's request log. This information is common to most web servers and the analysis is presented in Appendix A. The web server's configuration (maximum users, maximum live users, waiting time, and time out) is also captured as an instance of the *Configuration* class. Multiple models that conform to this DSML can be generated if the application is running on multiple machines.

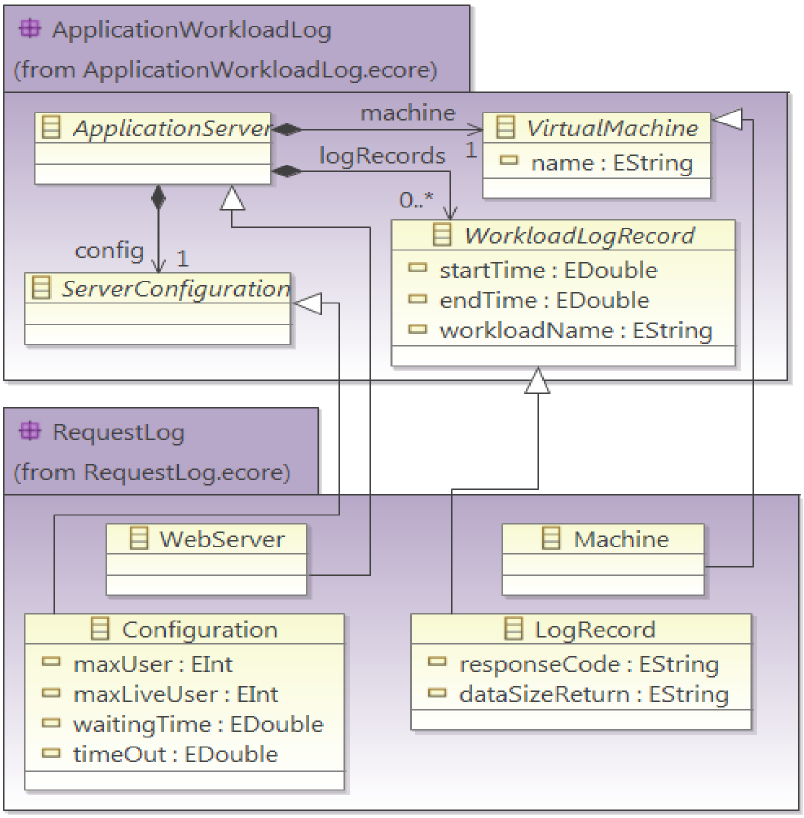


Figure 6.7: The design of *RequestLog* which extends the *ApplicationWorkloadLog* DSML specifically for web applications.

Figure 6.7 illustrates the design of the *RequestLog* and *ApplicationWorkloadLog* DSMLs and the relationship between them for web applications. Figure 6.8 demonstrates an instance of the *RequestLog* DSML capturing data extracted from a typical request log. A web server runs in a virtual machine with certain configuration. The requests processed in a web server are captured in the web server's log. Figure 6.8 shows an example of a *ReqestLog* model with three requests extracted from log recordings together with its virtual machine (machine) and web server configuration details.

6.5.2 Additional Model Management Activities of ReRA

Since resource utilisation is recorded with two resource monitoring tools, additional model management activities are required to manage the respective models. These activities are discussed below:

i. **Resource Usage Logs to Resource Models**
   Two programs were written to extract resource utilisation information through a text-to-model (T2M) transformation. The *VBoxVMMetricsLogToModel* program extracts the recording of resource utilisation from

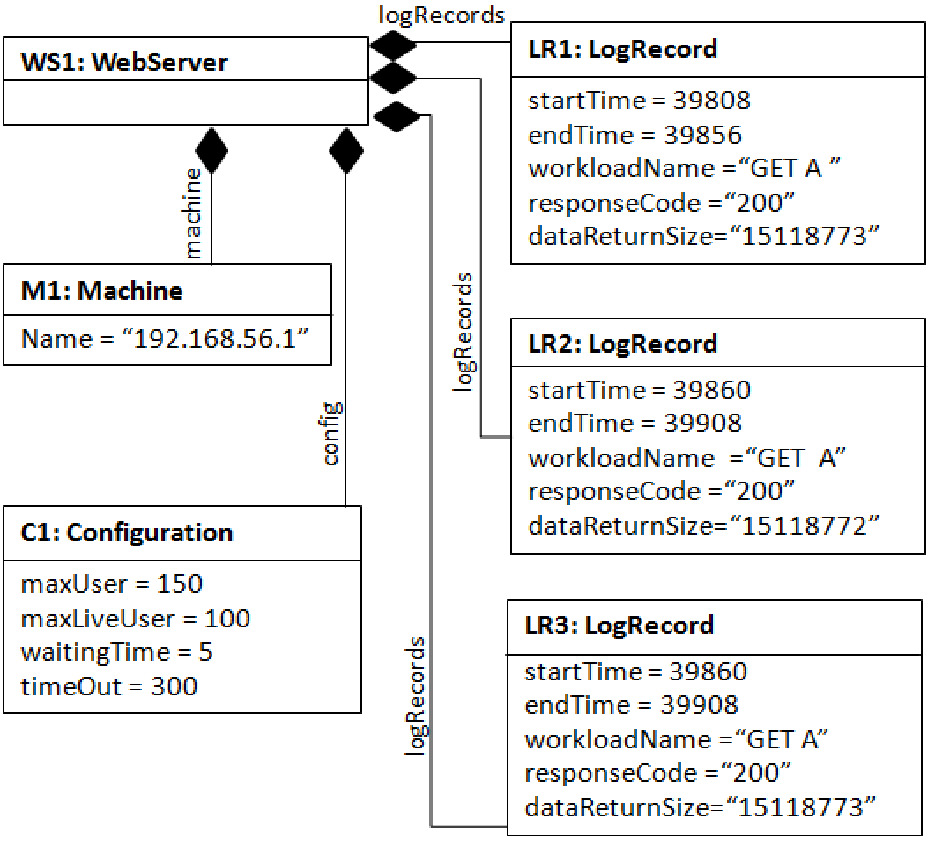


Figure 6.8: Example of RequestLog instance (for the DSML, see RequestLog DSML in Figure 6.7).

VBoxManage and creates *VBoxVMMetrics* models. The *DFLogDiskUsageModel* program produces *DiskUsageLog* models based on information recorded by the Ubuntu disk utilisation tool. The programs create the models in XML form with the information extracted from the log recording text. Samples of resource usage log recordings and its model in XML form are presented in Listing E.3 to E.6 in Appendix E. In the scenario of using single resource monitoring, a T2M transformation direct to the *ResourceLog* model is sufficient and step 4 in Figure 6.3 is not required.

Listing 6.3: EOL program (*DiskUtilToVBDisk.eol*) transfers storage information from *DiskUsageLog Model* to *VBoxVMMetrics Model*.

```
//vbLog=UtilisationLog objects of VBoxVMmetrics model;
//diskLog=LogRecord objects of DiskUsageLog model;
//fs=FileSystem objects of DiskUsageLog model;
var totalDisk = 0.00;
var tempVL : Set = new Set;

for (vl in vbLog.logRecords){
  var dl=diskLog.selectOne(l|l.time.floor()==
        vl.time.floor());
  if (dl.isDefined()){
        totalDisk = 0.00;
        for (fs in dl.filesystems){
                totalDisk = totalDisk + fs.usage;
        }
        // convert kilobyte to megabyte
        vl.Disk_Used = totalDisk/1024.00;
  } else {tempVL.add(vl);}
}
//to remove the last object if possible
if (tempVL.size()>0){vbLog.logRecords.removeall(tempVL);}
```

ii. **Manage Multi-Resource Models to a Resource Model**

The disk utilisation information captured in the *DiskUsageLog* model is transferred to the *VBoxVMMetrics* model by the *DiskUtilToVBDisk.eol* program. Later, a model-to model (M2M) transformation computes and selects the resource metrics from the attributes identified in the *VBoxVMMetrics* model. The code in Listing 6.3 shows the EOL script that transfers storage information from *DiskUsageLog Model* to *VBoxVMMetrics Model*. Listing 6.4 shows the ETL model-to-model

(M2M) transformation that transforms selected parameters as resource metrics into the *ResourceLog* model.

Listing 6.4: ETL program (*VBMetricsToResourceMetrics.etl*) transfers selected resource metrics parameter to *ResourceLog Model.*

```
rule vbMetricsToResourceMetrics
  transform s : vbModel!UtilisationLog
  to t : resModel!UtilisationLog {
     t. machine = s.machine;
     t.logRecords =s.logRecords.equivalent();
}

rule vbLogRecordToResLogRecord
  transform s : vbModel!LogRecord
  to t : resModel!LogRecord {
     t.time = s.time;
     t.CPU_Used = 100.00 - s.Guest_CPU_Idle;
     t.RAM_Used = s.Guest_RAM_Total - s.Guest_RAM_Free;
     t.Net_Incoming = s.Net_Incoming;
     t.Net_Outgoing = s.Net_Outgoing;
     t.Disk_Used = s.Disk_Used;
}
```

6.6 First Case Study: Media Stream

A prototype media web stream application was developed using WordPress[1] with a media plug-in called Stream Video Player[2]. WordPress is an open source blogging tool and content management system (CMS) based on PHP and MySQL. The Stream Video Player plug-in for WordPress is one of the most popular and most complete video-audio player WordPress plug-ins. Assuming the quality of the videos is the same, the workload for this application is characterised by number of users' request to view a live video. The implementation of MDE solutions for ReRA and ViRR processes for this case study is discussed in the following sections.

[1] http://wordpress.org/about/

[2] https://wordpress.org/plugins/stream-video-player/

6.6.1 ReRA of the Media Stream Application

Workload (request) information is retrieved from Apache's log file and a resource usage log file is generated using the VBManage monitoring tool which comes with VirtualBox. To carry out the ReRA process, additional configuration data is needed. This data can only be gathered from the specific application under test, and the experimental set-up (e.g., web server used, maximum number of users, number of CPUs, etc.). For this case study, the default Apache server configuration is used. The default configuration defines a request timeout of 300 seconds, a queuing time of 5 seconds, and an upper limit of 150 concurrent requests and 100 live requests. A virtual machine with 3 CPUs and 2GB of memory was used and VBoxManage was configured to capture and store resource usage measurements in a log. VBoxManage operates on the host server and therefore its operation does not interfere with that of the virtual machine.

This parameterised data is then transformed into *ResourceLog* models using the T2M transformations discussed in Section 6.3 (Resource Usage Logs to Resource Models). Artificial workloads involving concurrent requests, R, where $R = \{1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150\}$ were generated with JMeter, following the request pattern illustrated in Figure 6.9. The play time of the video file used was about 48 seconds and the application buffer the streaming shorter than play time. The configuration setting shows the request pattern for more than 100 concurrent requests, shown in Figure 6.9.

In this stage, the limitation of VBoxManage in recoding storage usage was not realised since the application was not making use of storage space. Therefore, the programs that were performing text-to-model transformations created *RequestLog* and *ResourceLog* models. *DiskUsageLog* and *VBoxVMMetrics* DSMLs together with their model management programs were not developed during the implementation of this case study.

The simulation of workloads created two log files; i) a *request log* file produced by the web server (access.log by Apache) and ii) a *resource usage log* file produced by the VM monitoring tool (*name*.vbm by VBoxMange). Both log files were transformed into *RequestLog* and *ResourceLog* models using text-to-model transformations. These two models were then sorted, compared and merged based on time, to produce a single model (*WorkloadRequestVsTime*). Resource usage and occupancy graphs are produced using a model-to-text transformation. These graphs are presented in Figures C.3, C.4, C.5, C.6, and C.7 respectively for CPU, memory, incoming network, outgoing net-

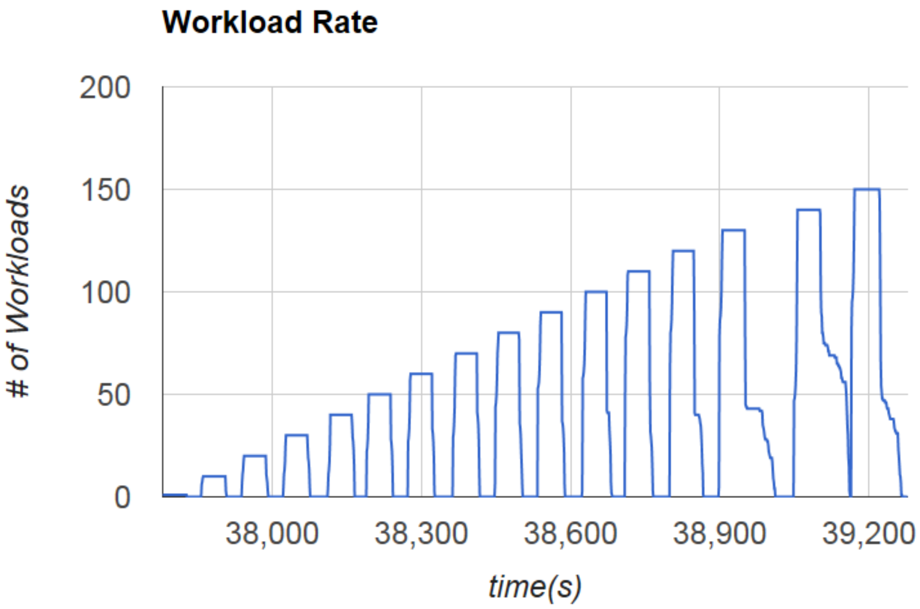


Figure 6.9: Media stream workload pattern simulated with JMeter.

work and storage. Subsequently, the *WorkloadResourceVsTime* model was transformed to a *ResourceVsWorkload* model by correlating the calculated occupancy and resource usage. The noise introduced due to technical constraints in the implementation environments (time synchronisation between host and VMs) is then removed by examining the *WorkloadPattern* model.

Then, statistical analysis was performed on the *ResourceVsWorkload* model in order to calculate formulas that can predict resource usage for arbitrary workloads. Figure 6.10 illustrates the statistical analysis performed to calculate the minimum, maximum, average and median resource requirements to process the workload. All other figures of the same type of Figure 6.10, which are available in Appendix C and D, hold the same. The workload presented in Figure 6.10 are the number of concurrent user requests to view a video file as. The *ResourceRequirementAnalysis* model captures the output of the statistical analysis and the mean values were transformed into a CSV file using model-to-text transformation. This CSV file was then analysed with equation extraction module to retrieve formulas for the resource metrics. In this research, Matlab was used and it is open to explore other equation extraction module such as machine learning as future work.

Figure 6.11 shows the average reading of resource metrics as output of model-to-text transformation which produces a CSV file. This CSV file is used as

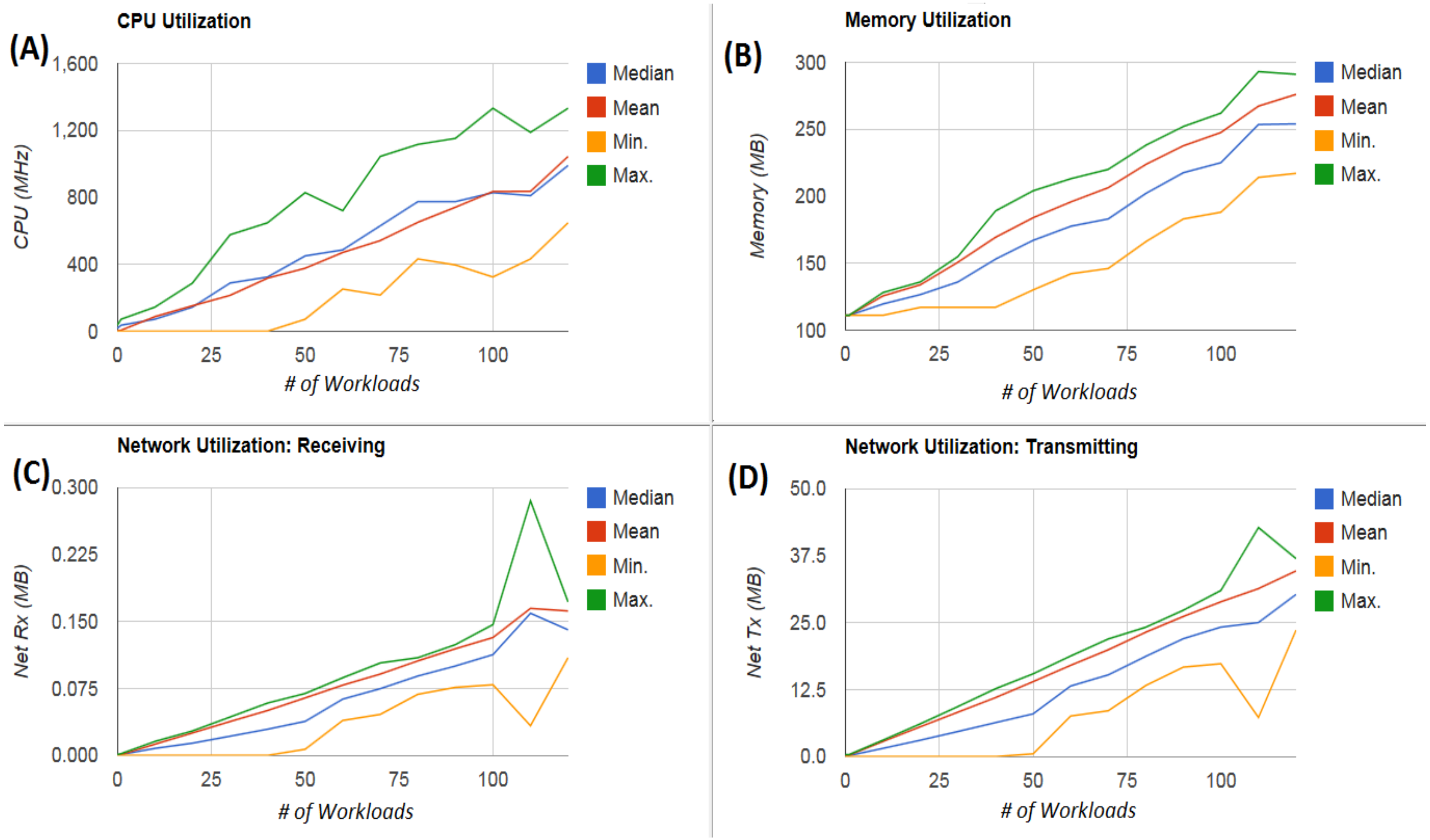
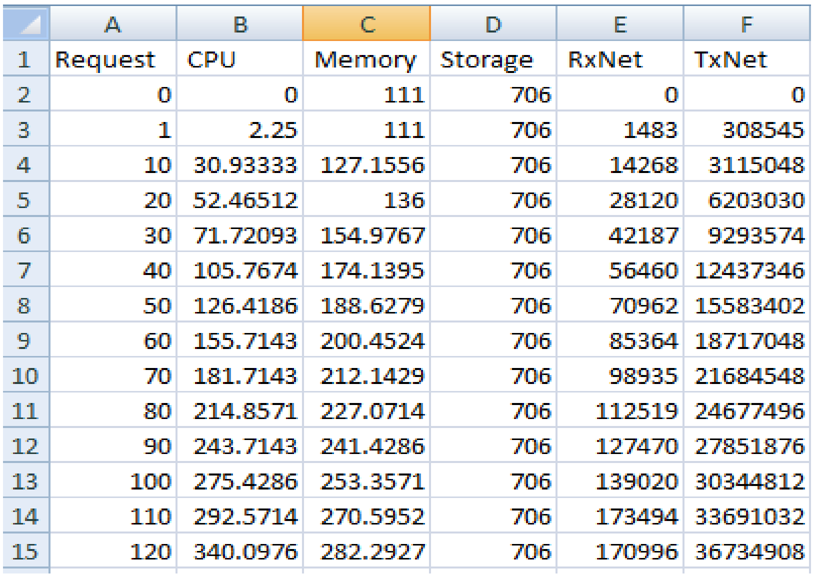


Figure 6.10: Statistical results of resource usage of CPU(A), memory(B), incoming network(C) and outgoing network(D) for the number of workloads (requests to view a video file).



	A	B	C	D	E	F
1	Request	CPU	Memory	Storage	RxNet	TxNet
2	0	0	111	706	0	0
3	1	2.25	111	706	1483	308545
4	10	30.93333	127.1556	706	14268	3115048
5	20	52.46512	136	706	28120	6203030
6	30	71.72093	154.9767	706	42187	9293574
7	40	105.7674	174.1395	706	56460	12437346
8	50	126.4186	188.6279	706	70962	15583402
9	60	155.7143	200.4524	706	85364	18717048
10	70	181.7143	212.1429	706	98935	21684548
11	80	214.8571	227.0714	706	112519	24677496
12	90	243.7143	241.4286	706	127470	27851876
13	100	275.4286	253.3571	706	139020	30344812
14	110	292.5714	270.5952	706	173494	33691032
15	120	340.0976	282.2927	706	170996	36734908

Figure 6.11: The table structure of the CVS file showing the statistical analysis of average reading of resource metrics for the request being processed.

input to the equation extraction tool (Matlab in this thesis). Figure 6.12 shows the outcome of the equation extraction module, which also holds for all other figures of the same type, which are available in Appendix C and D. In this case study, the equations extracted are as following: $CPU = 2.8 * \#requests + 15$, $Memory = 1.438 * \#requests + 120$, $IncomingNetwork = 1458.3 * \#requests$, $OutgoingNetwork = 30590 * \#requests$ and $Storage = 706$. The formulas can be used twofold: to both inform the current capacity planning process, but also to support prognostics, though of course many iterations of experiments must be carried out to increase confidence in the validity of the formulas. This is a simple case study where the only variable considered was the number of requests. Additional variables are considered in the subsequent case studies.

The following sections discuss the Virtual Resource Requirement (ViRR) process for the case study.

6.6.2 ViRR of the Media Stream Application

The resource requirement schedule to process the workload of an application is captured in *ResourceRequestPlan* model. It represents a daily resource request time table (*VMRequestPlan* and *DailyRequest*) further decomposed

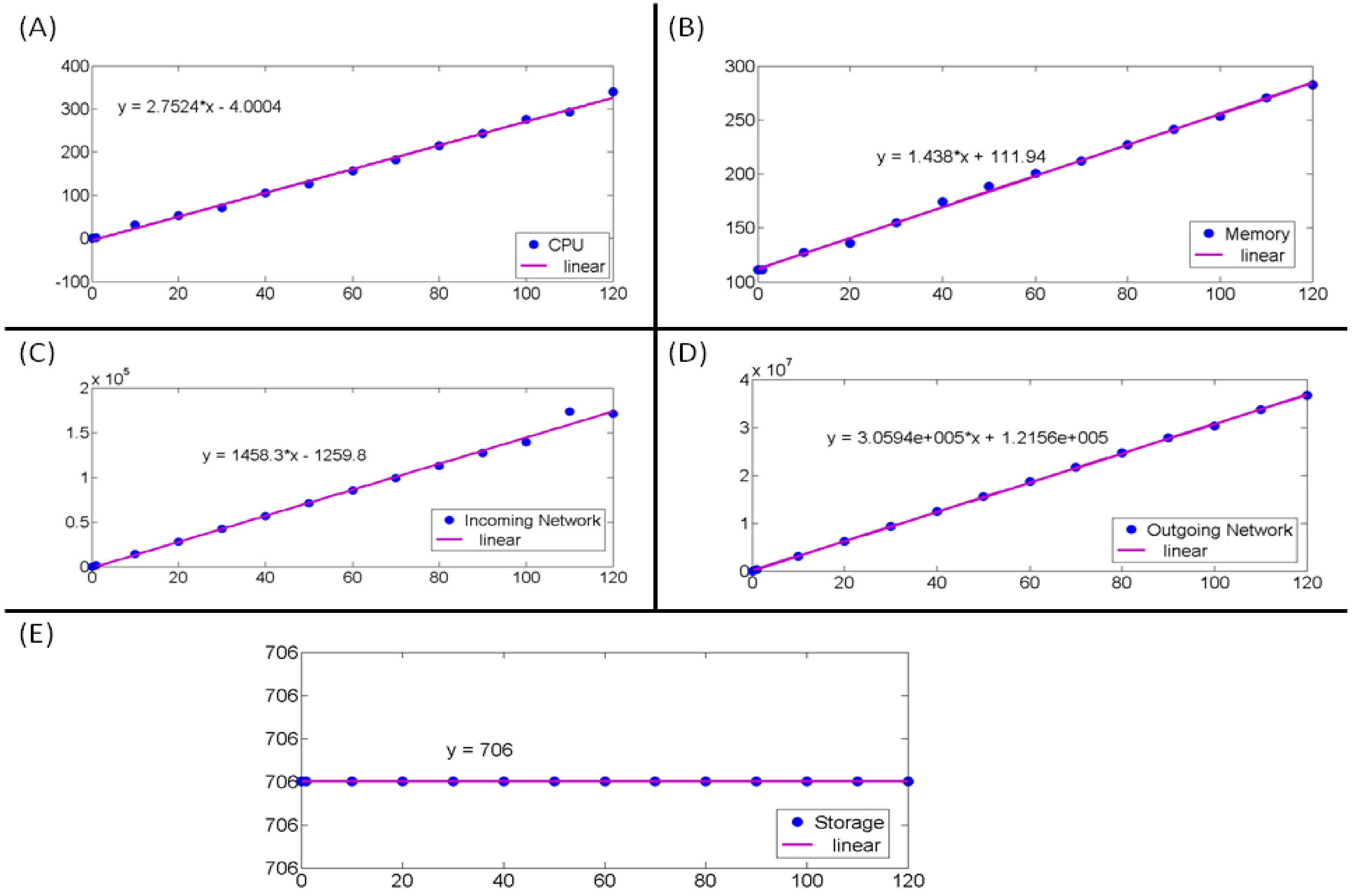


Figure 6.12: Extracted equations for resource metrics of the CVS data file.

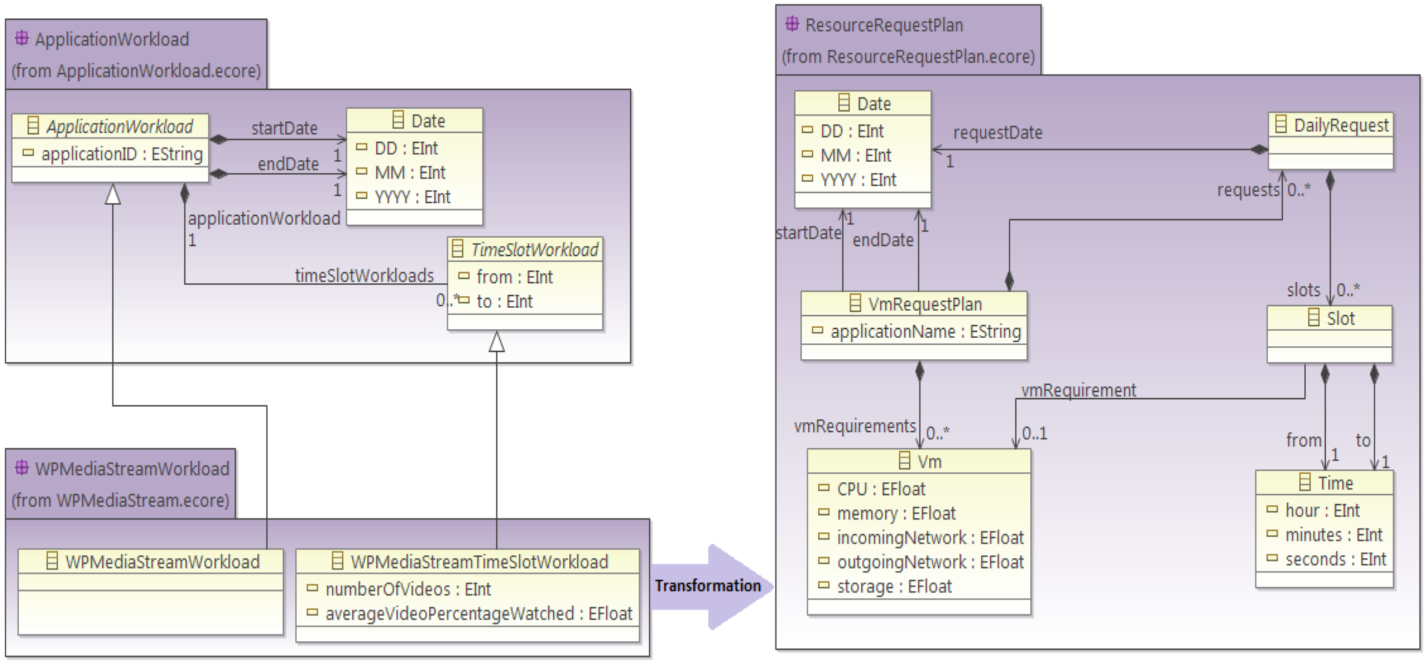


Figure 6.13: Overview of ViRR DSMLs for the Media Stream Application.

into multi-hour slots. Resource requirements are estimated based on information provided by the *ApplicationWorkload* model which defines the workload for each hourly slot (*TimeSlotWorkload*) of each calendar period of interest (*Date*). Since the structure of each application's workload is unique, a specific DSML is required to capture its features by extending core classes of the *ApplicationWorkload* DSML.

In this case study, the WordPress media stream workload (*WPMediaStream-Workload*) DSML was designed to capture the application workload. The number of user requests to view the file is captured with *numberOfVideos* however, there is a possibility that the users do not watch the entire video. Therefore, the average percentage of the video being watched online is also captured (*averageVideoPercentageWatched*). Figure 6.13 shows the overview of ViRR DSMLs for the WordPress Media Stream Application and Figure 6.14 shows a contrived *WPMediaStreamWorkload* model. The workload is divided into 5 time slots for days in the second half of the year 2014 (01/07/2014 to 31/12/2014).

The daily resource requirement schedule to manage the workload was captured in *ResourceRequestPlan* model. Therefore, *WPMediaStreamWorkload* model was transformed to *ResourceRequestPlan* model using formulas produced from the ReRA process performed above. Listing 6.5 and 6.6 illustrate the ETL transformation that computes the resource requirements of the application. As discussed above, the formula for resource metrics were retrieved from ReRA using Matlab. A custom Java tool (*tools.DateTool* at line 3 in Listing 6.5) was used to calculate number of days (line 15) and respective dates (line 24) for the selected calendar period. The complete ETL program combines these two rules is presented in Listing G.2 in Appendix G. For example, the outcome by transforming *WPMediaStreamWorkload* model (illustrated in Figure 6.14) with the rules defined in the ETL transformation is *ResourceRequestPlan* model (illustrated in Figure 6.15).

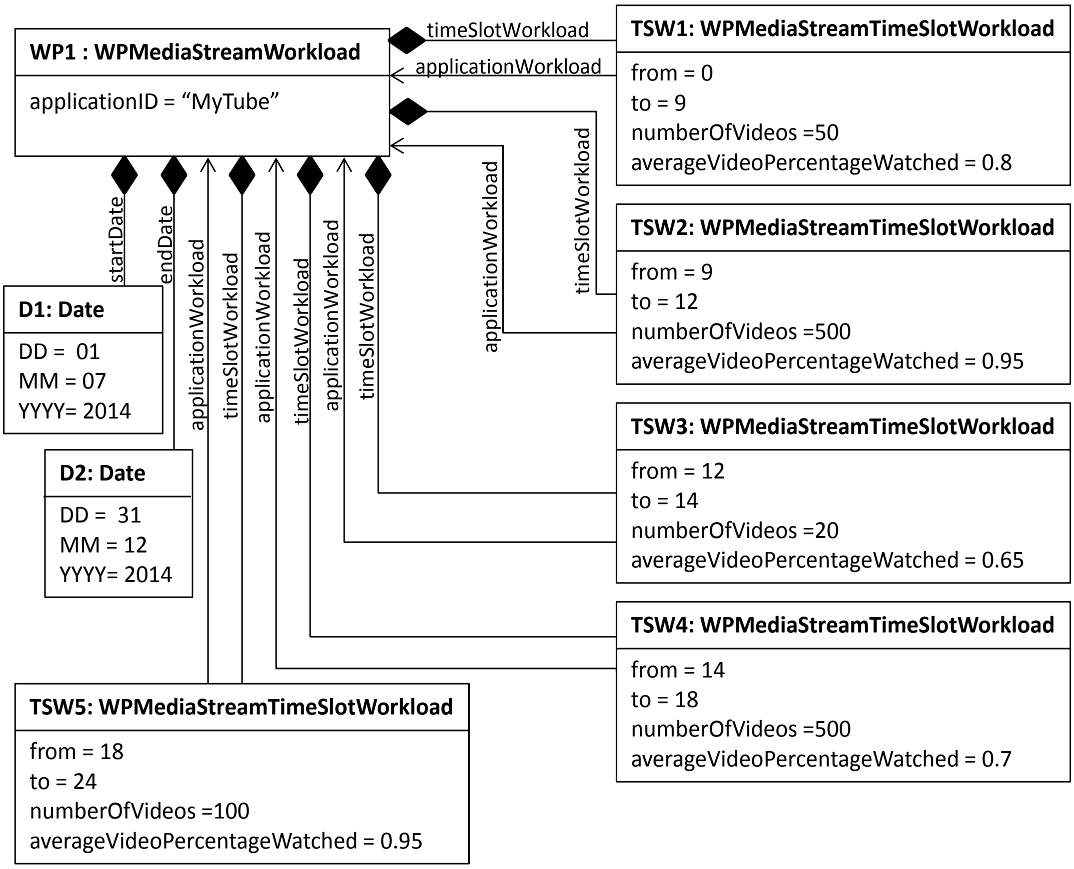

WP1 : WPMediaStreamWorkload
applicationID = "MyTube"
timeSlotWorkload
applicationWorkload
startDate
endDate
D1: Date
DD = 01
MM = 07
YYYY= 2014
D2: Date
DD = 31
MM = 12
YYYY= 2014
TSW1: WPMediaStreamTimeSlotWorkload
from = 0
to = 9
numberOfVideos =50
averageVideoPercentageWatched = 0.8
TSW2: WPMediaStreamTimeSlotWorkload
from = 9
to = 12
numberOfVideos =500
averageVideoPercentageWatched = 0.95
TSW3: WPMediaStreamTimeSlotWorkload
from = 12
to = 14
numberOfVideos =20
averageVideoPercentageWatched = 0.65
TSW4: WPMediaStreamTimeSlotWorkload
from = 14
to = 18
numberOfVideos =500
averageVideoPercentageWatched = 0.7
TSW5: WPMediaStreamTimeSlotWorkload
from = 18
to = 24
numberOfVideos =100
averageVideoPercentageWatched = 0.95

Figure 6.14: Object diagram represents *WPMediaStreamWorkload* model.

Listing 6.5: Rule to Transform the WordPress Media Stream Workloads to VM Request Plan.

```
pre {
  var myTubeWorkload = S!WPMediaStreamWorkload.all.first();
  var dateTool = new Native(”tools.DateTool”);
}

rule WPMediastreamWorkloadToVmRequestPlan
  transform s : S!WPMediaStreamWorkload
  to t : T!VmRequestPlan {

    t.applicationName = s.applicationID;
    t.vmRequirements.addAll(s.timeSlotWorkloads.equivalent());

    // defining start and end date

    noOfDays = dateTool.countDays(startDate, endDate);
    currentDate = startDate;

    for (i in Sequence{1..noOfDays}){
            //create daily request

            for (timeSlotWorkload in s.timeSlotWorkloads){
              t.createSlots(reqDate, timeSlotWorkload);
            }
            currentDate = dateTool.tomorrow(currentDate);
    }
  //assign start and end date
}
```

Listing 6.6: Rule to Transform the WordPress Media Stream Workload Time Slot to VM Resource Requirement.

```
rule WPTimeSlotWorkloadToVmRequirement
  transform timeSlotWorkload : S!WPMediaStreamTimeSlotWorkload
  to vm : T!Vm {
    var duration : Integer;
    //hours to seconds
    duration:=(timeSlotWorkload.`to`-timeSlotWorkload.from)
                *3600;

    //average CPU requirement (MHz)
    vm.CPU=(2.8*timeSlotWorkload.numberOfVideos+15.0)*
                timeSlotWorkload.averageVideoPercentageWatched
                   ;

    //average memory requirement (MB)
    vm.memory=(1.438*timeSlotWorkload.numberOfVideos+120.0)*
                timeSlotWorkload.averageVideoPercentageWatched
                   ;

    //average incoming network requirement (MB)
    vm.incomingNetwork=
                (1458.3*timeSlotWorkload.numberOfVideos)*
                timeSlotWorkload.averageVideoPercentageWatched
                   ;

    //average outgoing network requirement (MB)
    vm.outgoingNetwork=
                (30590.0*timeSlotWorkload.numberOfVideos)*
                timeSlotWorkload.averageVideoPercentageWatched
                   ;

    //average storage requirement (MB)
    vm.storage = 706.00;
}
```

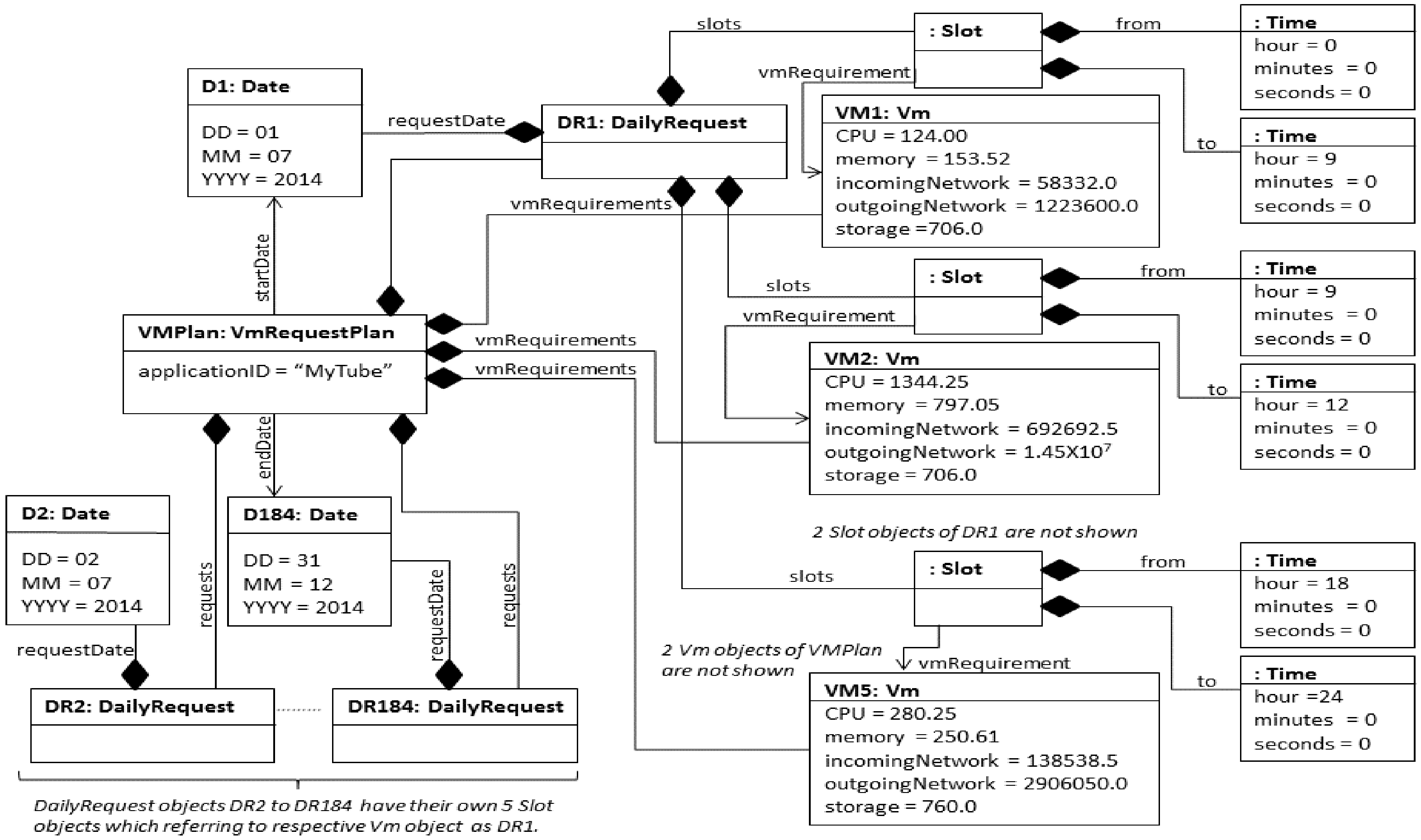


Figure 6.15: Object diagram represents *ResourceRequestPlan* model based on the *WPMediaStreamWorkload* model.

6.7 Second Case Study: Part of Speech Tagging

The second case study was developed using PHP and an open-source part-of-speech tagging tool named lexicon[1]. In a part-of-speech tagging application, a process of tagging up the words in a text as corresponding to a particular part of speech (e.g. verb, noun, adjective) is performed. This case study highly consume all the interested resource metrics. I Samples of tags are NN for noun, NNS for plural noun, VB for verb, VBD for verb past tense and others. For example the string, *"The purpose of this walk-through is to provide a computer role-playing game."* is tagged as *"The/DT purpose/NN of/IN this/DT walk-through/NN is/VBZ to/TO provide/VB a/DT computer/NN role/NN playing/VBG game/NN."*

The application receives a number of text files to be tagged, returns the tagged files to the user and also stores the received text files. This case study utilised all the interested resource metrics and very CPU intensive. The first case study is network, CPU and memory intensive while this case study include storage utilisation.

6.7.1 ReRA of the Part-of-Speech Tagger Application

The Apache 2 web server was used with its default configuration on a virtual machine with 4 CPUs and 2GB of memory. VBoxManage was configured to record resource usage into a log file. In conducting this case study, a limitation of VBoxManage was discovered, whereby the disk utilisation recording does not represent the changes in disk usage (it records the same value even though the storage usage changes). Therefore, the disk utility monitoring tool of the operating system was used to capture storage utilisation of the VM. In this case study, two resource monitoring tools were used and the resource utilisation log files were transformed into two models (*VBoxVM-Metrics* and *DiskUsageLog*). These models were synthesised to produce a *ResourceLog* model. The same DSMLs and model management techniques used in the first case study were reused on top of an additional DSML for resource utilisation recording.

Simulated workloads involving concurrent requests, R, where $R = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 100\}$ were generated with JMeter. There was a CPU

[1] `http://phpir.com/part-of-speech-tagging`

bottleneck in processing the workload of this application in the virtualised environment. A total of 4 CPU units were used to set the virtualisation environment on the physical machine and the VM hosted should not consume more than this. The experiments were also performed with smaller number of CPU units, however using the maximum of 4 CPU units shows better dependencies on the number of CPU. For example, the graphical illustration of CPU utilisation in Figure C.2 shows that the CPU utilisation reached 100% when the number of requests is equal to the number of CPU. The observation shows that each request fully utilised a CPU unit (as discussed above part-of-speech tagging is a CPU-heavy task). Therefore smaller numbers of concurrent requests were simulated to illustrated resource utilisation.

In addition to the number of concurrent requests, the size of the files processed had an impact on the resource usage. As proof-of-concept implementation, three files representing three file size categories, S, where $S = \{147035, 57765, 21291\}$ bytes with respective words count, C, where $C = \{24767, 10873, 3781\}$ were processed. Therefore, three sets of experiments were conducted to analyse resource usage of the simulated workloads for the three size categories (large, medium and small).

The ReRA process presented in Section 6.5 was fully utilised to analyse each workload of the part-of-speech tagger application. The outcomes of implementing ReRA with this application are presented in Appendix C. Three sets of graphs were produced by performing ReRA for each workload category. The first set of graphs show the resource utilisation and simulated concurrent requests of a particular workload. The second set of graphs shows the correlation between resource usage and number of concurrent workloads using basic statistical analysis (mean, median, minimum and maximum). The third set of graphs shows the outcome of equation extraction using the mean values of the resource usage with Matlab. The outcome of the equation extraction are presented in Figure C.10, C.19 and C.28 in Appendix C. These formulas were used in the ViRR process to estimate future resource requirements.

6.7.2 ViRR of the Part of Part-of-Speech Tagger Application

A *ResourceRequestPlan* model captures resource requirement schedule to process the workload of an application with a daily resource request time table (*VMRequestPlan* and *DailyRequest*) on an hourly basis with time (*Slot*). The resource requirement is estimated based on information provided by

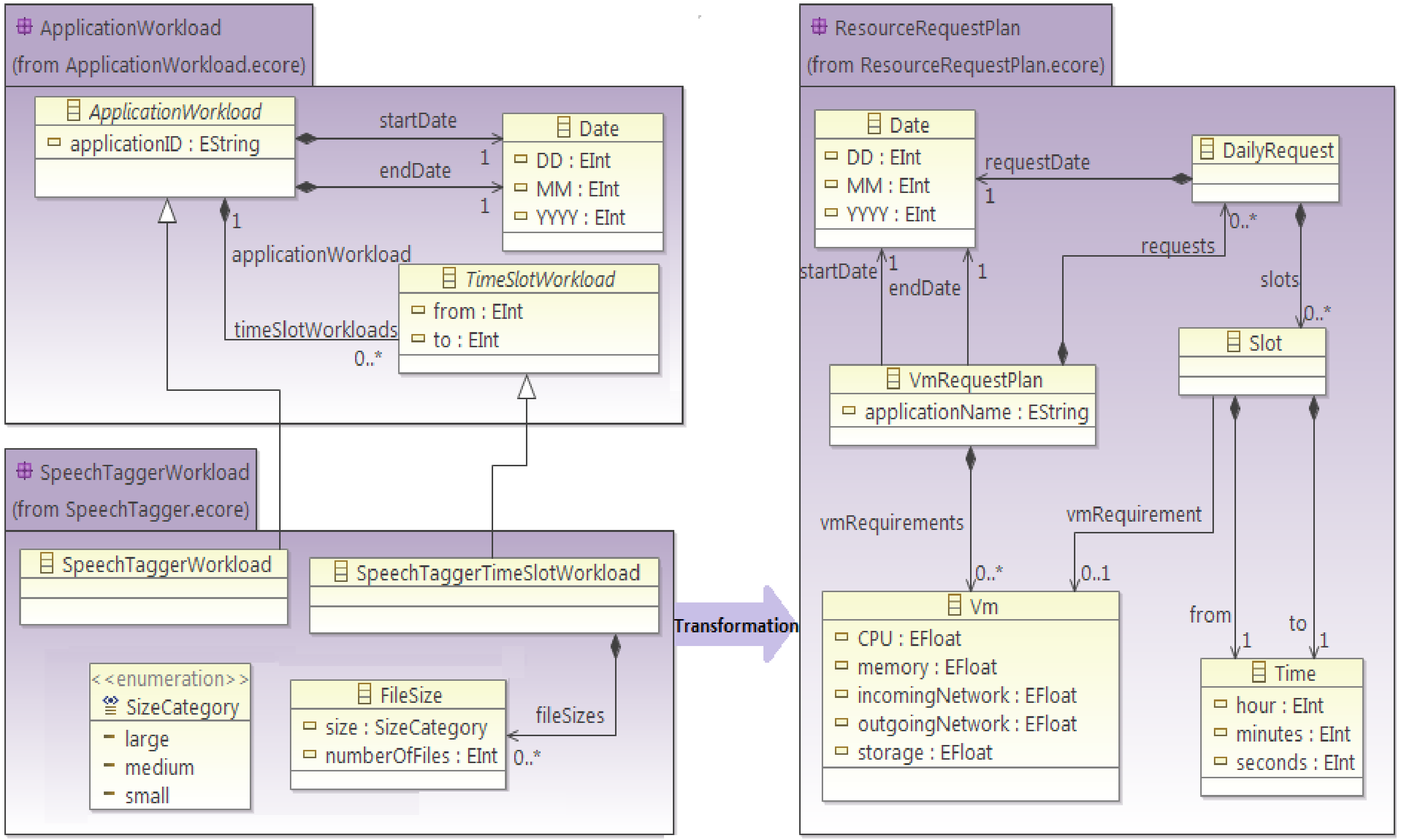


Figure 6.16: Overview of ViRR DSMLs for the part-of-speech tagger Application.

the *ApplicationWorkload* model, which defines the workload per hourly slot (*TimeSlotWorkload*) for the calendar period of interest (*Date*).

Since a specific DSML is required to capture unique features of application workloads, the *SpeechTaggerWorkload*) DSML was designed by extending the core classes of *ApplicationWorkload* DSML. The number of files involved for the part-of-speech tagging process is captured with (*numberOfFiles*). Files are categorised according to their sizes and the *FileSize* class is added to capture this feature. The file size is captured using three file categories (large, medium and small). It is assumed that the percentage of file categories being processed at each timeslot is different, therefore *percentages* is used to capture this information. Figure 6.16 shows the overview of ViRR DSMLs design for the part-of-speech tagger application. The *SpeechTaggerWorkload* DSML is specific to that application; however, it is an extension of *ApplicationWorkload* DSML.

Defining the workload timeslots is beyond the scope of this case study since it involves business planning for that particular application. However, for testing purposes, similar workload timeslots as in the first case study are used, whereby there are 5 time slots for each day between 01/07/2014 to 31/12/2014.

The *SpeechTaggerWorkload* model was transformed to a *ResourceRequestPlan* model using formulas produced through the ReRA process performed for this application. The transformation produced a daily resource requirement schedule for the workload in the *ResourceRequestPlan* model. Listing 6.7 illustrates the rule to transform the Part-of-Speech tagger application workloads to resource requirement plan. The formulas for resource metrics are retrieved from ReRA using statistical data analysis as discussed in Section 6.7.1. Listing 6.8 transforms the Part-of-Speech Tagger application workload time slot to VM resource requirement schedule. The complete concrete transformation of rules in Listing 6.7 and 6.8 is presented in Listing G.3 in Appendix G.

Listing 6.7: Rule to transform the Part-of-Speech Tagger Application Workloads to VM Request Plan.

```
pre {
 var mySpeechTaggerWorkload=S!SpeechTaggerWorkload.all.first()
     ;
 var dateTool=new Native(”tools.DateTool”);
}

rule SpeechTaggerWorkloadToVmRequestPlan
  transform s : S!SpeechTaggerWorkload
  to t : T!VmRequestPlan {
    t.applicationName = s.applicationID;
    t.vmRequirements.addAll(s.timeSlotWorkloads.equivalent());
    // defining start and end date
    noOfDays = dateTool.countDays(startDate, endDate);
    currentDate = startDate;
    for (i in Sequence{1..noOfDays}){
            //create daily request
            for (timeSlotWorkload in s.timeSlotWorkloads){
              t.createSlots(reqDate, timeSlotWorkload);
            }
            currentDate = dateTool.tomorrow(currentDate);
    }
  //assign start and end date
}
```

Listing 6.8: Rule to Transform the Part-of-Speech Tagger Application Workload Time Slot to VM Resource Requirement.

```
rule SpeechTaggerTimeSlotWorkloadToVmRequirement
 transform timeSlotWorkload : S!SpeechTaggerTimeSlotWorkload
 to vm : T!Vm {
  //declaration of variables;
  for(fileSize in timeSlotWorkload.fileSizes){
    //formulas retrived from ReRA to process small files are
        used (Figure 6.47 )
    if (fileSize.size == SizeCategory#small){
      smallFileCPU=(5.2152*fileSize.numberOfFiles.pow(3))
          -(188.59*fileSize.numberOfFiles.pow(2))+(2069.5*
          fileSize.numberOfFiles)-129.9;

      smallFileMemory=(0.2727*fileSize.numberOfFiles.pow(3))
          +(1.4333 *fileSize.numberOfFiles.pow(2))+(43.554*
          fileSize.numberOfFiles)+64.48;

      smallFileNetIn=(0.1271*fileSize.numberOfFiles.pow(3))
          -(3.7384*fileSize.numberOfFiles.pow(2))+(36.104*
          fileSize.numberOfFiles)-2.1564;

      smallFileNetOut=(0.1909*fileSize.numberOfFiles.pow(3))-(5.6324*fileSize.numberOfFiles.pow(3))+(54.541*fileSize.numberOfFiles.pow(3))-3.8293;

      smallFileStorage=1.3296*fileSize.numberOfFiles+1302.04;
    }

    //formulas retrived from ReRA to process medium files are used (Figure 6.38 )
    else if (fileSize.size == SizeCategory#medium){
      mediumFileCPU = ...//CPU formula
      mediumFileMemory = ...//memory formula
      mediumFileNetIn = ...//incoming network formula
      mediumFileNetOut = ...//outgoing network formula
      mediumFileStorage = ...//outgoing network formula
    }

    //formulas retrived from ReRA to process large files are used (Figure 6.29)
    else if (fileSize.size == SizeCategory#large){
      mediumFileCPU = ...//CPU formula
      mediumFileMemory = ...//memory formula
      mediumFileNetIn = ...//incoming network formula
      mediumFileNetOut = ...//outgoing network formula
      mediumFileStorage = ...//outgoing network formula
    }
  }

  //get resource requirements
  vm.CPU=getMaximum(smallFileCPU,mediumFileCPU,largeFileCPU);
  vm.memory=getMaximum(smallFileMemory,mediumFileMemory,largeFileMemory);
  vm.incomingNetwork=getMaximum(smallFileNetIn,mediumFileNetIn,largeFileNetIn);
  vm.outgoingNetwork=getMaximum(smallFileNetOut,mediumFileNetOut,largeFileNetOut);
  vm.storage=getMaximum(smallFileStorage,mediumFileStorage,largeFileStorage);
}

operation getMaximum(x,y,z) : Any{
    var max = x;
        if (y > max) { max = y; }
        if (z > max) { max = z; }
 return max;
}
```

6.8 Chapter Summary

In this chapter, two case studies were used to demonstrate that the DSMLs and semi-automated processes presented in the previous chapter can be used to support parts of the capacity planning process in different applications. In particular, the implementations demonstrate that valuable information in the form of resource utilisation graphs together with resource metrics formulas can be produced to aid in capacity planning. The application of proposed MDE solution with two case studies hosted in a virtualised environment shows that the developed DSMLs and model management techniques are modular and reusable. The modularity and reusability of the proposed MDE solutions were evaluated using an additional case study in the following chapter.

Chapter 7

Evaluation

This thesis has presented a couple of DSML sets that allow workloads to be captured precisely using models, as well as a transparent, automated and repeatable MDE process. This MDE approach assists the capacity planning manager to estimate the resource requirements from the application workload models. The MDE process exploits model transformation, comparison and merging as model management techniques. The core approaches were demonstrated in Chapter 5 with sets of DSMLs and model management techniques. In Chapter 6, the core approaches were extended by applying it to web applications hosted in a virtualisation environment. The implementation using two case studies shows that, the proposed MDE approach demonstrated in Chapter 5 are modular and reusable, making it possible to perform capacity planning for an application hosted in a virtualised environment. To evaluate further the modularity, reusability, completeness and extensibility of the proposed DSMLs and the MDE process for the framework, a third case study was conducted which is discussed in this chapter. This case study is also a web application hosted in the same virtualised environment to evaluate the extended solution demonstrated in Chapter 6. Sets of synthetic workloads were generated to conduct ReRA and the retrieved formulas were applied in the ViRR process, as presented in Chapter 6.

7.1 Evaluation Plan

In Section 6.2, an overview of the evaluation plan was discussed as the second phase of the experimental research method. Figure 7.1 illustrates the flow of the evaluation plan executed with an additional third case study.

The third case study was also a resource-intensive web application running in the virtualised environment configured for the previous two case studies discussed in Chapter 6. Therefore, the application type and the infrastructure used in exploratory and evaluation case studies are the same. However, the evaluation case study differs from the previous case studies in terms of the functionality of the application and the characteristics of the workload. The first case study related to videos, while the second related to text files and the third is related to images. The attributes of the workloads used to manage those entities in the respective web applications are different. For example, the third case study has a filter type attribute to differentiate the request types in addition of number of images and image size. The second case study considers text file size and number of text file, while the first considers the number of video and average video percentage watched.

The modularity and reusability of the proposed MDE solutions are the variables, as defined in the hypotheses. Additional variables such as the completeness and extensibility of the proposed MDE solutions were also evaluated. To evaluate the predictive approach, additional synthetic workloads were generated to compare the actual resource consumptions with the predicted resource requirements.

An analysis of the findings will be discussed to measure the fulfilment of the system level requirements presented in Chapter 6. The conclusion of the research will be elaborated in the following chapter.

7.2 Third Case Study: Image Filtering

An image filtering web application was exclusively used as a third case study to evaluate the desired variables of the proposed MDE solutions in performing capacity planning in virtualised environments. This application was developed using PHP's image filtering utilities[1]. Three types of image filters (grayscale, negate and default) were used in this application. The application receives a number of images to be filtered, applies the requested filters, sends the filtered files to the user and also stores them on the server. This application intensively utilises all resources (CPU, memory, network and storage) of interest. For the purpose of this case study, the images were categorised into three sizes; large, medium and small.

[1] http://www.php.net/manual/en/function.imagefilter.php

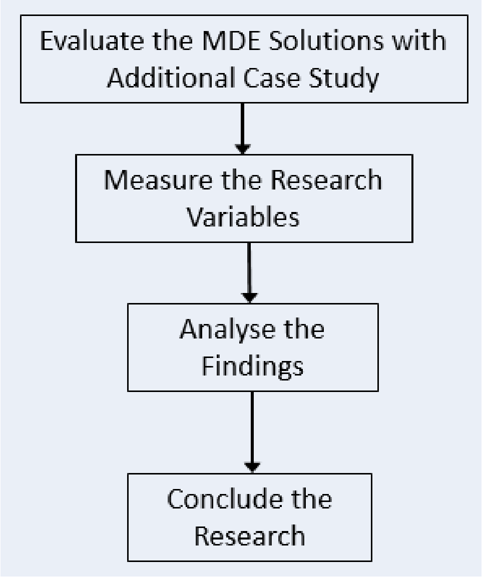

Evaluate the MDE Solutions with Additional Case Study
Measure the Research Variables
Analyse the Findings
Conclude the Research

Figure 7.1: Evaluation Plan of the Research.

The Image Filtering application was hosted in a VM running in the VirtualBox virtualisation environment as discussed in Chapter 6. As in the previous two case studies, Apache2 web server was used to host the Image Filtering application. Therefore, the same structure of log recording was configured to reuse the text-to-model programs which initiate the creation of the *RequestLog*, *VBoxVMMetrics* and *DiskUsageLog* models. Subsequently, all the programs and DSMLs discussed in Chapter 6 for ReRA and ViRR were utilised and the practicality was evaluated.

A virtual machine with 4 CPUs and 2GBs of RAM was used to host this application. A maximum of 4 CPU units was used to better highlight the dependency on the number of CPU units utilised. Our observations show that each request utilised a CPU unit until the selected image filtering process completed. For example, a CPU unit was fully utilised to process a request, 2 CPU units were fully utilised to process two concurrent requests, 3 CPU units were utilised to process three concurrent requests and the maximum of 4 CPU units were utilised to process four and above concurrent requests.

VBoxManage was configured to record resource usage into a log file. The disk utility monitoring tool of the operating system was used to capture storage utilisation of the VM. Therefore, two resource monitoring tools as in the second case study were used and the resource utilisation log files were transformed into two models (*VBoxVMMetrics* and *DiskUsageLog*). These

Image Categories	**# of Images**	**Original Size of Images (bytes)**	**Size After Filtering (bytes)**		
			Grayscale	**Negate**	**Default**
Large	200	822,244	760,241	577,887	1,825,686
Medium	500	247,539	213,287	130,371	421,941
Small	2,000	81,809	80,304	55,607	82,161

Table 7.1: Image Filter Case Study

models were synthesised to produce a *ResourceLog* model. The DSMLs and model management techniques used in the second case study were reused in ReRA and ViRR processes.

Workloads involving concurrent requests, R, where $R = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ were generated using JMeter. Since a CPU bottleneck was observed in the VM when processing the workload of this application, a small number of concurrent requests was simulated to measure the application workload. In addition to concurrent requests, the size of the images and the filter type also impact resource usage.

The image filtering application used for the evaluation was a synthetic case study. Therefore, to simplify the technicality, the size of the images was categorised into three types (large, medium and small). Since there are also three types of filter used to analyse resource requirement to process three types of image sizes, nine sets (3 types of images X 3 types of filters) of experiments were conducted. These experiments aimed to analyse resource usage triggered by the simulated workload categories by using DSMLs and model management techniques proposed in Chapter 6.

Images that need to be processed were stored in a client site (in this configuration, the host). For example, 200 large images were used and the locations of the images (path), together with the filter type to be implemented were captured in a csv file. This csv file was used in the JMeter to simulate respective workloads. The time taken to process smaller images was less, therefore the number of files was increased for smaller images to ensure sufficient logs were collected for analysis. Filtered images were returned to the client and were also stored in the server (VM). Table 7.1 shows the number of files used to analyse resource usage for each image category. The size of images before and after filtering is also presented in Table 7.1 since these values influence network usage and storage.

The experiments undertaken to collect empirical data and the outcomes of performing ReRA activities are discussed in the following section.

7.2.1 ReRA of Image Filtering Application

The workloads for the image filtering application were categorised into 9 types (3 filter types x 3 image categories). The ReRA process was used to retrieve formulas for the resource metrics from the nine identified workloads. In this section, the practicality of the process and the reusability of the proposed MDE solutions are evaluated.

The ReRA process presented in Chapter 6 was fully utilised to analyse each workload of the image filtering application. The outcomes of implementing ReRA with this application are presented in Appendix D. Three sets of graphs were produced by performing ReRA for each workload category. The first set of graphs show the resource utilisation and simulated concurrent requests of a particular workload. The second set of graphs shows the correlation between resource usage and number of concurrent workloads using basic statistical analysis (mean, median, minimum and maximum). The third set of graphs shows the outcome of further analysis with Matlab using the mean values of the resource usage to produce formulas using equation extracting module of Matlab.

The designed DSMLs for ReRA process and the developed programs to perform model management techniques (text-to-model, model-to-model and model-to-text) were reused without modifications. The ReRA process was utilised systematically and the obtained formulas for the resource metrics of the generated workloads are presented in Figure D.3, D.6, D.9, D.12, D.15, D.18, D.21, D.24 and D.27 in Appendix D. These formulas were used in the ViRR process to estimate future resource requirements.

7.2.2 ViRR of the Image Filtering Application

The resource requirements schedule for processing the workloads of the application of interest, in the form of a daily resource request time table is captured in the *ResourceRequestPlan* model. Resource requirements are estimated based on information provided by the *ApplicationWorkload* model, which defines the workload for time-long slots (*TimeSlotWorkload*) for each calendar period (*startDate* and *endDate*) of interest. The time-long slots (*from* and *to* attributes in *TimeSlotWorkload*) can be used to support various levels of granularity (such as hours, minutes or seconds). The selection of the time unit affects the transformation and the extensibility of the related transformation program is presented in lines 230-252 of Listing G.4 for *createTime* operation. In this section, the application of the proposed MDE solutions of the ViRR process is presented.

An *ImageFilterWorkload* DSML was designed by extending classes of the *ApplicationWorkload* DSML. Only this DSML was designed to replace the application workload DSML of the previous two case studies since it is specific to the structure and behaviour of this application. In a time period, various categories of workloads (*workload*) may be processed. To capture this information, the start time (*from*) and end time (*to*) of processing the image filtering workloads were stored in *ImageFilterTimeSlotWorkload*. The workloads in this case study were defined using: the number of images to be processed (*numberOfImages*), the image size category (*size*), and the type of filter to be used to process the images (*filter*). Images are categorised according to their sizes (using the values large, medium and small); the *SizeCategory* class is designed to capture these values. The filter type is defined in the *Filter* enumeration with negate, grayscale and default as literals. Figure 7.2 shows the overview of the ViRR DSLMs designed for the image filtering application.

Listing 7.1: Rule to transform the Image Filtering Application Workloads to VM Request Plan.

```
pre {
  var myIFWorkload = S!ImageFilterWorkload.all.first();
  var dateTool = new Native("tools.DateTool");
}

rule ImageFilterWorkloadToVmRequestPlan
  transform s : S!ImageFilterWorkload
  to t : T!VmRequestPlan {
        t.applicationName = s.applicationID;
    t.vmRequirements.addAll(s.timeSlotWorkloads.equivalent());
    // defining start and end date
    noOfDays = dateTool.countDays(startDate, endDate);
    currentDate = startDate;
    for (i in Sequence{1..noOfDays}){
            //create daily request
            for (timeSlotWorkload in s.timeSlotWorkloads){
              t.createSlots(reqDate, timeSlotWorkload);
            }
```

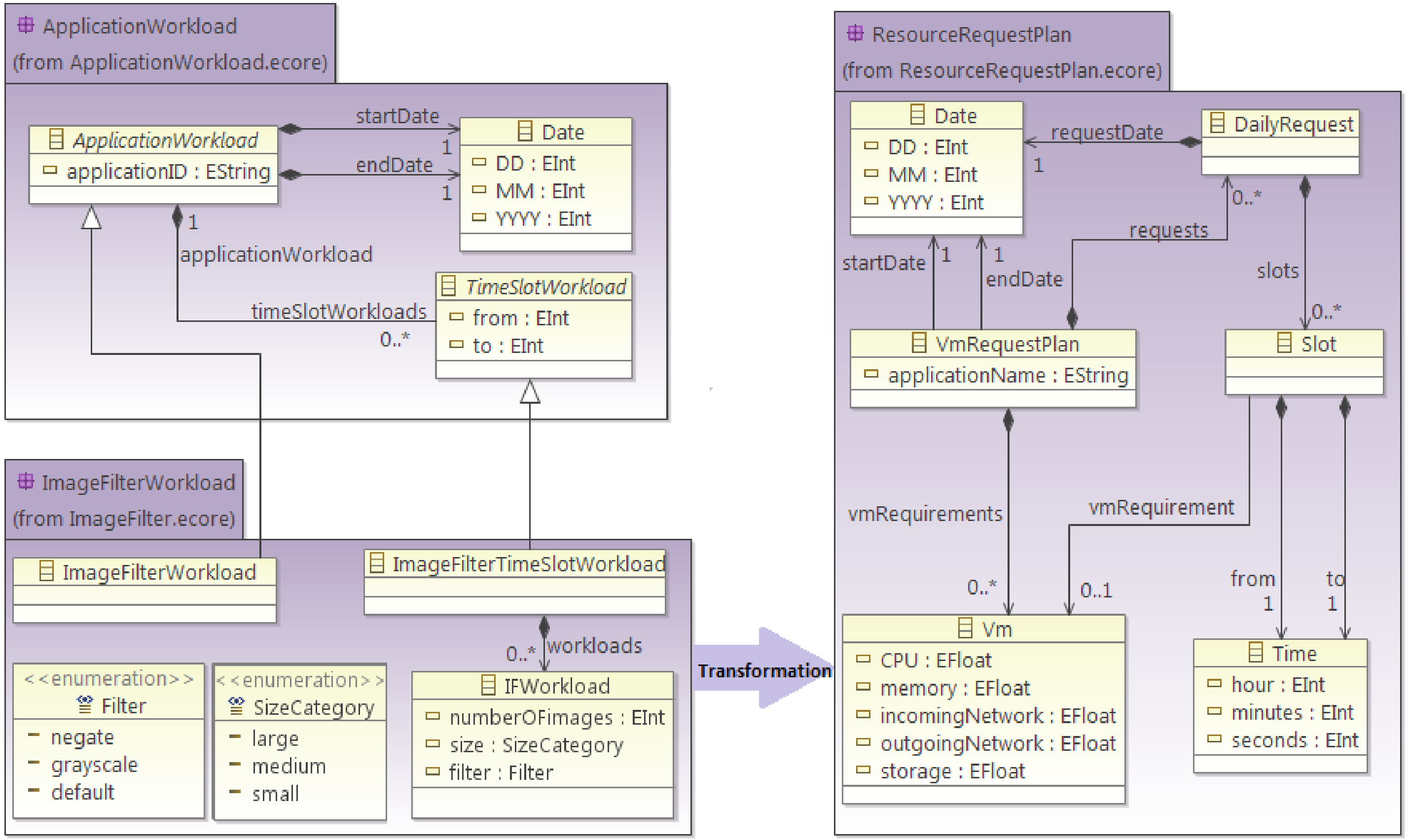


Figure 7.2: Overview of ViRR DSMLs for the Image Filtering Application.

```
                currentDate = dateTool.tomorrow(currentDate);
        }
    //assign start and end date
}
```

The formulas of resource metrics produced from the ReRA process in Section 7.2.1 were used for transforming *ImageFilterWorkload* model to *ResourceRequestPlan* models. Listing 7.1 illustrates the rule to transform the image filtering application workloads to resource requirement plan. The application processes various workloads within a day and this behaviour is repeated throughout the selected period. This behaviour is similar to the other two case studies, therefore the ETL transformation that translates workload to a VM request plan remains the same as for the two previous case studies. Listing 7.2 illustrates an outline of the rule that transforms each time slot of the image filtering application to respective VM resource requirements. The formulas for resource metrics are retrieved from ReRA using statistical data analysis as discussed in Appendix D. In the complete transformation, the formulas are applied within lines 5 to 39 depending on the type of file size and the type of the filter. The complete concrete transformation is presented in Listing G.4 in Appendix D.

Listing 7.2: Rule to Transform the Image Filtering Application Workload Time Slot to VM Resource Requirement.

```
1  rule ImageFilterTimeSlotWorkloadToVmRequirement
2    transform timeSlotWorkload : S!ImageFilterTimeSlotWorkload
3    to vm : T!Vm {
4      //declaration of variables
5      for(wl in timeSlotWorkload.workloads){
6        if (wl.size == SizeCategory#small){
7          if (wl.filter == Filter#‘default‘){
8          //apply formulas retrieved from ReRA to process small
               images with default filter (Figure C.21)
9          }
10         else if (wl.filter == Filter#negate){
11         //apply formulas retrieved from ReRA to process small
               images with negate filter (Figure C.27)
12         }
13         else if (wl.filter == Filter#grayscale) {
14         //apply formulas retrieved from ReRA to process small
               images with grayscale filter (Figure C.24)
15          }
16       }
17       else if (wl.size == SizeCategory#medium){
18         if (wl.filter == Filter#‘default‘){
19         //apply formulas retrieved from ReRA to process medium
```

```
            images with default filter (Figure C.12)
        }
        else if (wl.filter == Filter#negate){
        //apply formulas retrieved from ReRA to process medium
            images with negate filter (Figure C.18)
         }
         else if (wl.filter == Filter#grayscale) {
         //apply formulas retrieved from ReRA to process
            medium images with grayscale filter (Figure C.15)
         }
       }
       else if (wl.size == SizeCategory#large){
         if (wl.filter == Filter#'default'){
         //apply formulas retrieved from ReRA to process large
             images with default filter (Figure C.3)
         }
         else if (wl.filter == Filter#negate){
         //apply formulas retrieved from ReRA to process large
             images with negate filter (Figure C.9 )
         }
         else if (wl.filter == Filter#grayscale) {
         //apply formulas retrieved from ReRA to process large
             images with grayscale filter (Figure C.6)
         }
       }
    }

    //get resource requirements
    vm.CPU =  getMaximum(parameters);
    vm.memory =  getMaximum(parameters);
    vm.incomingNetwork = getMaximum(parameters)
    vm.outgoingNetwork = getMaximum(parameters);
}
```

7.3 Evaluation of Modularity

MDE which utilise modularity, responds quickly to the changes in requirements, and also increases productivity, maintainability, and portability [40, 41]. In this thesis, a couple of DSML sets and model management techniques were proposed to facilitate capacity planning in virtualised environments. The design of DSMLs for ReRA and ViRR processes, as presented in Chapter 5, is self-contained to ensure that every step of the capacity planning process is modular in nature. The applications of the proposed MDE approach with two case studies, presented in Chapter 6, demonstrate that the steps are modular. Every automated/semi-automated step of ReRA and ViRR was portable and responded rapidly to the changes in requirements. The input and output of the DSMLs and their structure are specified in a rigorous manner to support modularity.

7.3.1 Modularity in ReRA

In Chapter 5, six DSMLs and nine model management activities were designed to enable modularity by being self-containing within the step. The input and output of the DSMLs were managed by respective model management activities. In Chapter 6, two additional DSMLs (*VBoxMetrics* and *DiskUsageLog*) were designed to capture the resource usages from two log files into corresponding models. Model management programs were developed to produce the required output from those models to feed into the core *ResourceLog* model. The modification involved in capturing the resource usage logs does not affect other steps in the ReRA process due to modularity.

Modularity allows several steps to be processed simultaneously since it minimises dependency. Capturing workload records into an *ApplicationWorkloadLog* model and resource usage records into a *ResourceLog* model can be conducted concurrently. These two steps are opening activities that initiate the ReRA process. Modular design aids those two separate activities to be executed concurrently.

Other steps require input from previous activity, but those steps may be extended to produce the required core models. For example, a selected statistical method can be applied to the *ResourceVsWorkload* model to generate a *ResourceRequirementAnalysis* model. The structure of the *ResourceRequirementAnalysis* DSML may differ from the design proposed in the thesis, to align it with the applied statistical method. These changes only affect the model management techniques related to that DSML.

Modularity also allows different equation extraction tools to be used in producing formulas for the resource metrics. The ReRA process provides a modular component for equation extraction which can be replaced with appropriate implementations such as machine learning. The design of the *ResourceRequirementAnalysis* DSML may be customised according to the selected equation extraction tools.

The modularity qualifies selected DSMLs to be used for other purposes. For example, it is also possible to use *ApplicationWorkloadLog* and *ResourceLog* models for various analyses related to application workload and resource monitoring respectively. Modularity also makes selected steps optional. For, example, cleansing activity can be excluded if the data collection does not produce any noise.

7.3.2 Modularity in ViRR

In the ViRR process, the *ApplicationWorkload* model captures the predicted workload of an application and is application specific. This model will be transformed into a *ResourceRequirementPlan* model to estimates the resource requirements for the defined schedule with model-to-model transformation. The changes in *ApplicationWorkload* DSML specific to an application only affect the transformation program which generates a *ResourceRequirementPlan* model. The modularity feature applied in ViRR does not affect changes to the *ResourceRequirementPlan* DSML although there are changes in the *ApplicationWorkload* DSML, as demonstrated by all three case studies.

7.4 Evaluation of Reuse

This section discusses the level of reusability of the proposed core MDE solutions presented in Chapter 5 and also of the applied MDE solutions. Table F.1 summaries the average percentages of reusability of both MDE solutions. The first column in Table F.1, list the two processes developed in this research (ReRA and ViRR), the second column represents the artefacts categories which were grouped by DSMLs and model management techniques, the third column represents the individual core MDE artefacts discussed in Chapter 5 and the fourth column identifies the core artefacts that had been reused and the additional artefacts developed during implementation, as demonstrated in Chapter 6. The average reuse percentages of the core MDE artefacts are presented in the fifth column (*% of Reuse Compared to Core MDE Artefact*) with three sub-columns which evaluate the individual artefacts, artefacts categories (DSMLs and model management techniques separately) and processes (artefacts used in the process), accordingly.

Table F.1 also shows the average reuse percentages of the applied MDE solutions with the third case study discussed in Section 7.2. The third case study was used to measure the level of reusability of the extended MDE solutions specific to web applications operating in the VirtualBox virtualised environment. This set-up has been used to represent cloud-based applications at the time of writing that are suitable for ReRa and ViRR processes as i) the applications run in a virtualised environment, ii) precise resource utilisation metrics can be extracted using VM monitoring and iii) application requests and responses and their associated times are recorded by default by the web server. The reuse of concrete artefacts demonstrated in Chapter 6 is presented in the sixth column (*Artefact for evaluation Case Study*) and the reuse percentage of applied MDE artefacts in the third case study is presented in the seventh column by dividing it into artefact, artefact categories and process.

7.4.1 Reuse in ReRA

In applying the core MDE solutions to automate the ReRA process, an additional *RequestLog* DSML was designed for capturing web application workloads by extending the *ApplicationWorkloadLog* core DSML. Due to certain limitations (related to the recording of storage utilisation) of the VBoxManage resource monitoring tool in VirtualBox, two additional DSMLs (*DiskUsageLog* and *VBoxVMMetrics*) were designed to consolidate those resource DSMLs, and a model-to-model model transformation was developed to transform multi-resource models into a single model that conform to the core *ResorceLog* DSML. These improvements were presented in Chapter 6 by applying the core MDE with web applications' case studies in VirtualBox virtualisation environment. The rest of the core MDE artefacts proposed for the ReRA process were fully reused (66.67% of DSMLs and 90% of the model management programs were reused, as presented in Table F.1). On average, 78.33% of the core DSMLs and model management programs applied in ReRA reuse the core MDE solutions. The statistics values are influenced by the effort required to customise text-to-model transformation programs to capture log recording structure to the respective log models. The structures of those log files differ across infrastructure although the required values are recorded. Once those text-to-model transformation programs are implemented for that infrastructure, the programs are reusable for other applications run in that infrastructure.

The presented reusability figures are heavily influenced by the similarity of the applications used in the three case studies. Applications with heterogeneous/irregular workloads are more likely to require substantial extension/rework, hence reducing the level of reuse.

In Section 7.2, the applied MDE artefacts were further evaluated with the third case study and it demonstrates that all of the MDE artefacts used in previous case studies were fully reused without additional work being required to automate the ReRA process. This shows that the developed MDE artefact of ReRA may be completely reused in other web applications running in the VirtualBox virtualisation environment. *DiskUsageLog* and *VBoxVMMetrics* DSMLs are specific to VirtualBox due to the storage reading limitation in VirtualBox. These DSMLs might not be required in the situation when another virtualised platform is used. The *ResorceLog* DSML is sufficient to capture the resource utilisation records. The implementation of core MDE solutions for ReRA is specific to the **type of application** and also to the capabilities of the resource monitoring tool used to record resource utilisation. The level of reusability might differ slightly for other technology infrastructures and all of the core MDE solutions may be reusable for various technologies.

The required effort for implementing the proposed ReRA solutions in different platforms consists primarily of in customising the text-to-model transformation programs that read log recordings of workloads and resource utilization according to the log structure. The functionality of those programs is the same. The design of proposed core DSMLs and other model management techniques are reusable for various applications across platforms.

7.4.2 Reuse in ViRR

To automate the ViRR process, both core DSMLs were reused and an application specific workload DSML had to be developed. Moreover, to transform the application workload into a resource request plan, the core model-to-model transformation had to be customised. This customisation is related to the behaviour of an application workload for the time slots and also implements the retrieved formulas from ReRA. Therefore, out-of-the-box reuse only reached 43.33% for the application of ViRR in this case study as presented in Table F.1. However, the *ResourceRequestPlan* DSML was reused without modification for all case studies.

The level of reuse of the core MDE solutions is higher for ReRA than for ViRR. The implementation of the ViRR process is specific **to the application itself** since the workloads of an application are unique. For example, the MDE solutions of ViRR for the Media Stream case study and the Part-of-Speech case study presented in Chapter 6 significantly differed, despite the fact that both of them are web applications. However, the ViRR process for both applications was the same where the core MDE solutions were reused with additional MDE solutions specific to those applications. The complexity of application specific *ApplicationWorkload* DSML affects the model-to-model transformation program at implementation level. The effort to customise the model-to-model transformation program increases with the complexity of the application. The formulas retried from the ReRA process need to be applied in the transformation program; therefore, the reusability of this program is reduced, although the functionality is the same.

7.5 Evaluation of Extensibility

This section evaluates the extensibility of the MDE solutions for the ReRA and ViRR processes. The modular feature of the proposed solution is the key enabler for this feature.

7.5.1 Extensibility in ReRA

The case studies demonstrate that the core MDE solutions of ReRA are extensible. Referring to Table F.1, 3 additional DSMLs (*RequestLog*, *DiskUsageLog* and *VBVMMetrics*) were developed to capture log information for the web applications hosted in VirtualBox. The *RequestLog* DSML was designed specifically to capture the log information of a web application by extending the *ApplicationWorkload* DSML. *DiskUsageLog* and *VBVMMetrics* DSMLs were designed to capture the resource usage logs in VirtualBox in order eventually to produce a *ResourceLog* model. Model-to-model transformations were used to merge the *DiskUsageLog* and *VBVMMetrics* models to form a single resource model conforming to the core *ResourceLog* DSML.

7.5.2 Extensibility in ViRR

The MDE solutions of the ViRR process are application-specific and highly dependent on the workloads of the application of interest. More specifically, the core *ApplicationWorkload* DSML needs to be extended to fit the structure and behaviour of the application in question. This DSML is highly abstract and fully reusable and extensible. The *ResourceRequestPlan* DSML was reused completely without modification. The model-to-model transformation involved in this process also needs to be extended for the application of interest, as discussed in Section 5.3.2.

7.6 Evaluation of Completeness

This section evaluates the completeness of the ReRA and ViRR processes. The completeness was measured by the level of reusability. The MDE artefact which was fully reused without modification is complete.

7.6.1 Completeness in ReRA

Referring to Table F.1, an additional 3 DSMLs (*RequestLog*, *DiskUsageLog* and *VBVMMetrics*) were developed to capture logs information by applying the core MDE solutions for web applications hosted in VirtualBox. The core DSMLs and model management programs developed to automate the ReRA process were completely reused. The applied DSMLs specific to web applications were also fully reused with the third case study. Therefore, those DSMLs are classified as complete regarding the requirement to capture the information for the ReRA process.

In the scenario of applying the proposed MDE solutions to other types of application running under a different virtualisation technology, the initial text-to-model transformation programs used to capture log information into the respective log models needed to be developed. However, the functionality of those text-to-model transformation programs is similar, as discussed in Section 5.2.2. The core DSMLs of ReRA are reusable and able to be extended for various technology infrastructures. On the other hand, the developed model management programs are semi-automated. The success in applying the semi-automated ReRA process with the 3 case studies demonstrates the completeness of the proposed MDE solutions.

In applying different statistical methods, which is suggested as future study, the *ResourceRequirementAnalysis* DSML might need modifications to capture the features of that method. There is also flexibility in adopting other equation extracting tools/methods based on data gathered from the *ResourceRequirementAnalysis* model. Applying other statistical methods or adopting different equation extracting tools will affect the model management techniques applied to the *ResourceRequirementAnalysis* model and it may be implemented without affecting the other steps in ReRA.

7.6.2 Completeness in ViRR

In automating the ViRR process, the designed *ResourceRequestPlan* DSML that acts as a resource request calendar for the capacity planning period was applied as it is and reused without modification in all 3 case studies. Therefore, the *ResourceRequestPlan* DSML can be classified as complete as evaluated with the case studies. The *ApplicationWorkload* DSML which captures the predicted application workload is application-specific and was extended to additional DSML, specific to the application of interest. However, the core *ApplicationWorkload* DSML was reused by extending it to application-specific DSML which captures the workloads of a particular application. This DSML is an expandable and the model management program that transforms the *ApplicationWorkload* model to *ResourceRequestPlan* model needs to be updated accordingly together with the formulas retrieved from ReRA.

7.7 Predictive Capability Evaluation

The evaluation discussed in Sections 7.3 to 7.6, evaluates the variables of the proposed MDE artefacts in conducting capacity planning in virtualised environments. Additionally, to evaluate the predictive capability by utilising the proposed MDE solutions, a small experiment was conducted with the image filtering application.

As to proof the concept, at random, five workloads for the image filtering application were simulated with JMeter. The simulation was conducted for 600 seconds. Each workload was processed continuously for 60 seconds. Table 7.2 summarises the simulated workloads. The first column represents the name given to the simulated workload to enable traceability with Figure 7.3 and Table 7.3. The second and third columns represent the start and end times of that workload respectively. These time units were captured as seconds of the day. The fourth column represents the number of concurrent workloads being processed and the last column gives a description of the workloads.

A virtualisation setup that able to accurately monitor resource usage is required for evaluation. After unsuccessful experimentation with more complex infrastructures and applications (e.g. Eucalyptus and OpenStack for virtualisation and JPetStore as a use-case), a decision was made to use VirtualBox and simpler, uniform workload applications. A major implication of using VirtualBox is that workloads can only be executed on a single physical machine, thus limiting the capacity for larger-scale evaluation. Despite the limitations of the evaluation setup, the principles of the proposed processes are also relevant to larger scale setups of uniform workload applications as long as workload logs can be captured and be reliably co-related with resource utilisation. On the other hand, the applicability of the proposed processes to applications with heterogeneous workloads cannot be assessed at all through the evaluation as conducted.

The information on the simulated workloads presented in Table 7.2 was captured in the *ImageFilterWorkload* model. Figure 7.3 illustrates the *ImageFilterWorkload* model as an object diagram that has five workloads with the features defined in Table 7.2. The workload names (*Workload Name*) in Table 7.2 are presented with small rectangles on the right side of the *ImageFilterTimeSlotWorkload* objects in Figure 7.3.

The *ImageFilterWorkload* model was transformed to a *ResourceRequestPlan* model using the formulas produced through the ReRA process, as discussed in Section 7.2.1. The formulas used to estimate resource requirements in transforming the application workload to resource requirements are presented in Table 7.3. Column "Workload Name", "Start Time", "End Time" and "Workload Description" in Table 7.3 represent the same columns in Table 7.2. Column "Total Concurrent Workloads" represents the total workloads being processed for various workload types and graph Workload Rate in Figure 7.4 shows this value. The number of concurrent requests for each workload represented as "x" in Table 7.3 and the respective formulas using this value. Rows in Table 7.3 have been grouped according to the workload name. For example, w1 to w3 represent a single request being processed at a time and values for each resource type (CPU, memory, incoming network and outgoing network) are calculated with resource metrics formulas retrieved from ReRA. Workloads w4 and w5 managed three different types of requests at a time.

Workload Name	**Start Time**	**End Time**	**# of Workload**	**Description**
wl1	33518	33578	3	3 concurrent requests to process *small images* with *negate filter*.
wl2	33638	33698	2	2 concurrent requests to process *large images* with *grayscale filter*.
wl3	33758	33818	4	4 concurrent requests to process *medium images* with *default filter*.
wl4	33878	33938	2	2 concurrent requests to process *medium images* with *negate filter*, 1 concurrent requests to process *large images* with *default filter* and 3 concurrent requests to process small images with *grayscale filter*.
wl5	33998	34058	3	3 concurrent requests to process *small images* with *negate filter*, 2 concurrent requests to process *large images* with *grayscale filter* and 4 concurrent requests to process *medium images* with *default filter*.

Table 7.2: Image Filter Workloads

Therefore the rows for these workloads are further divided according to the request type being processed. In this work, maximum resource requirement for each resource metric to run the process the workload are used as benchmarks. This is to ensure that the resource allocation is sufficient to run the application. Therefore, the maximum values which are highlighted in bold in Table 7.3 were used for estimation. The transformation rules also apply this method to predict resource requirements.

The transformation of Listing G.4 was then used to produce a daily resource requirement schedule to predict the workload in the *ResourceRequestPlan* model. In this thesis, the maximum resource requirement was used for prediction and other possible statistical methods are open to explore to improvise the prediction. The implementation gathering maximum resource requirement was stated between line 195 and 204 in Listing G.4 and the values were presented in bold in Table 7.3.

The predicted resource requirements and the actual resource consumption are presented in Figure 7.4. The results observed are generally consistent with the predictions made through ReRA and ViRR. The estimated CPU resource requirements are slightly lower than actual CPU consumption. The estimated memory requirements are slightly higher than the actual memory consumption. The estimated incoming and outgoing network resource requirements are in line with the actual resource consumption. The underestimation of resources will affect the application performance and overestimation will increase the cost. This evaluation was conducted to demonstrate that it is possible to apply the proposed approach in estimating the resource requirements. However, further improvement through applying a better statistical method such as machine learning to achieve a more precise estimation is suggested. In addition, evaluation with cloud-based applications running in large scale virtualised environments offers another avenue for future work.

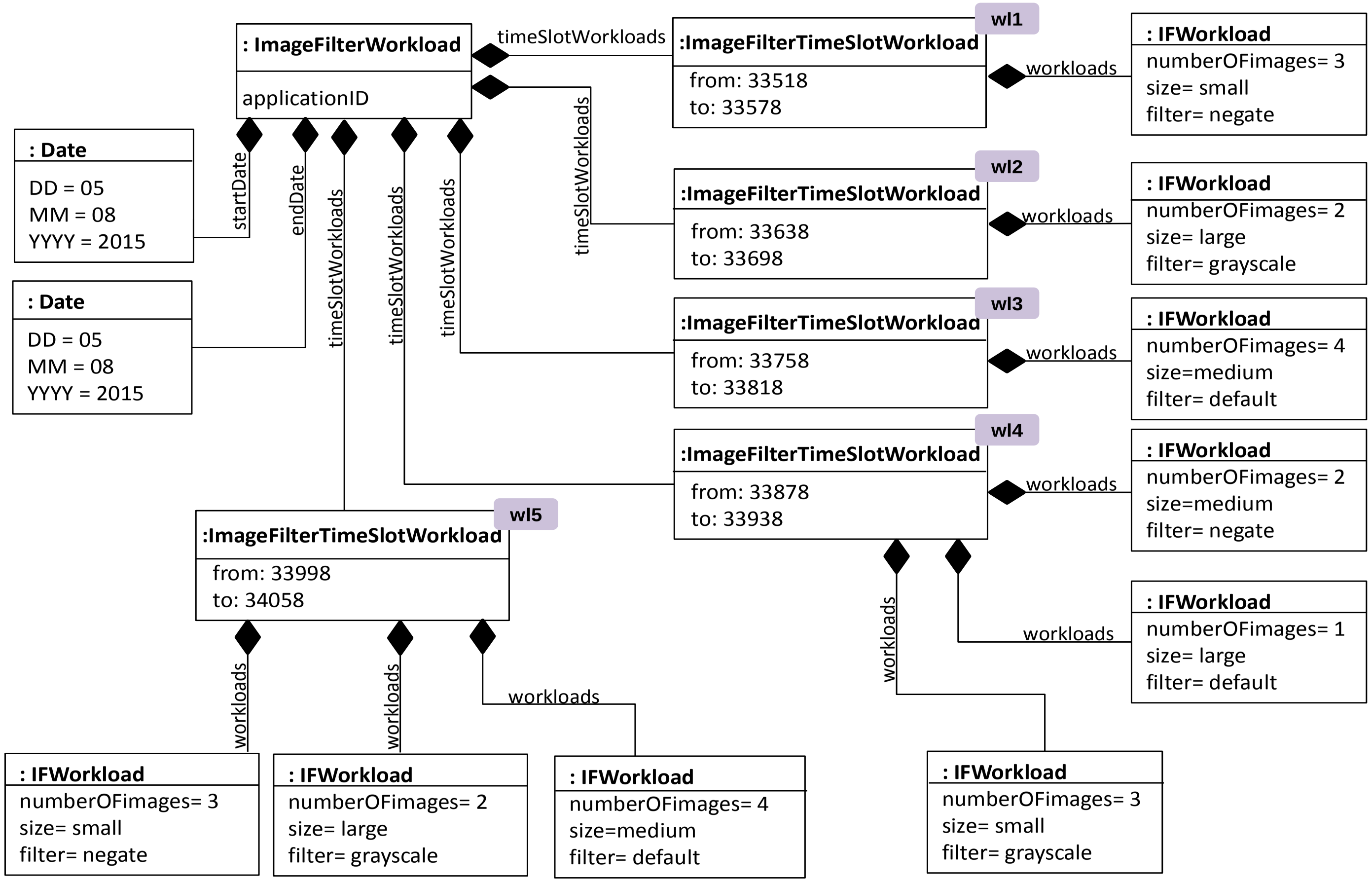
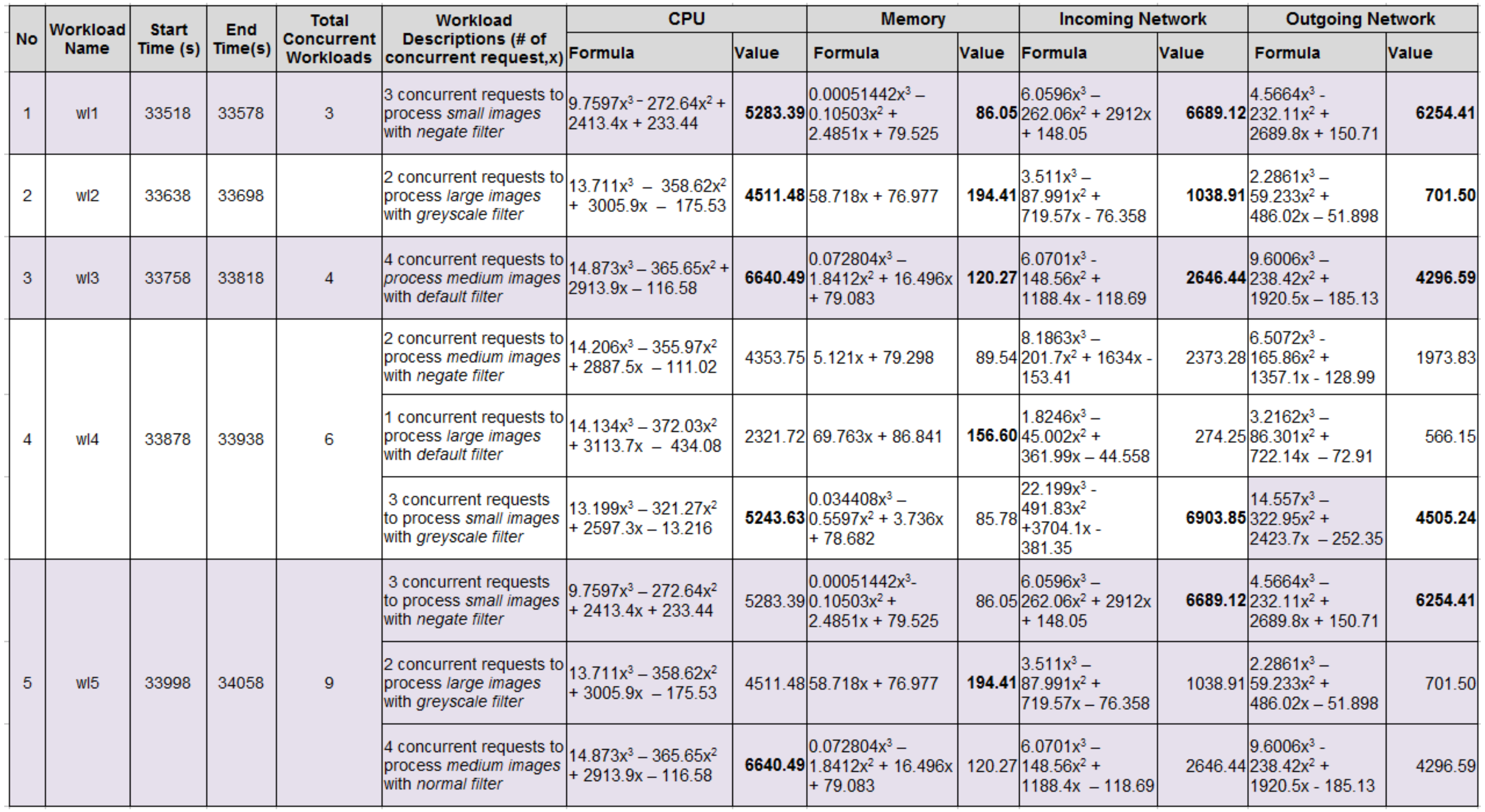


Figure 7.3: Object diagram of predictive capability evaluation.



No	Workload Name	Start Time (s)	End Time(s)	Total Concurrent Workloads	Workload Descriptions (# of concurrent request,x)	CPU		Memory		Incoming Network		Outgoing Network	
						Formula	Value	Formula	Value	Formula	Value	Formula	Value
1	wl1	33518	33578	3	3 concurrent requests to process *small images* with *negate filter*	$9.7597x^3 - 272.64x^2 + 2413.4x + 233.44$	**5283.39**	$0.00051442x^3 - 0.10503x^2 + 2.4851x + 79.525$	**86.05**	$6.0596x^3 - 262.06x^2 + 2912x + 148.05$	**6689.12**	$4.5664x^3 - 232.11x^2 + 2689.8x + 150.71$	**6254.41**
2	wl2	33638	33698		2 concurrent requests to process *large images* with *greyscale filter*	$13.711x^3 - 358.62x^2 + 3005.9x - 175.53$	**4511.48**	$58.718x + 76.977$	**194.41**	$3.511x^3 - 87.991x^2 + 719.57x - 76.358$	**1038.91**	$2.2861x^3 - 59.233x^2 + 486.02x - 51.898$	**701.50**
3	wl3	33758	33818	4	4 concurrent requests to *process medium images* with *default filter*	$14.873x^3 - 365.65x^2 + 2913.9x - 116.58$	**6640.49**	$0.072804x^3 - 1.8412x^2 + 16.496x + 79.083$	**120.27**	$6.0701x^3 - 148.56x^2 + 1188.4x - 118.69$	**2646.44**	$9.6006x^3 - 238.42x^2 + 1920.5x - 185.13$	**4296.59**
4	wl4	33878	33938	6	2 concurrent requests to process *medium images* with *negate filter*	$14.206x^3 - 355.97x^2 + 2887.5x - 111.02$	4353.75	$5.121x + 79.298$	89.54	$8.1863x^3 - 201.7x^2 + 1634x - 153.41$	2373.28	$6.5072x^3 - 165.86x^2 + 1357.1x - 128.99$	1973.83
					1 concurrent requests to process *large images* with *default filter*	$14.134x^3 - 372.03x^2 + 3113.7x - 434.08$	2321.72	$69.763x + 86.841$	**156.60**	$1.8246x^3 - 45.002x^2 + 361.99x - 44.558$	274.25	$3.2162x^3 - 86.301x^2 + 722.14x - 72.91$	566.15
					3 concurrent requests to process *small images* with *greyscale filter*	$13.199x^3 - 321.27x^2 + 2597.3x - 13.216$	**5243.63**	$0.034408x^3 - 0.5597x^2 + 3.736x + 78.682$	85.78	$22.199x^3 - 491.83x^2 + 3704.1x - 381.35$	**6903.85**	$14.557x^3 - 322.95x^2 + 2423.7x - 252.35$	**4505.24**
5	wl5	33998	34058	9	3 concurrent requests to process *small images* with *negate filter*	$9.7597x^3 - 272.64x^2 + 2413.4x + 233.44$	5283.39	$0.00051442x^3 - 0.10503x^2 + 2.4851x + 79.525$	86.05	$6.0596x^3 - 262.06x^2 + 2912x + 148.05$	**6689.12**	$4.5664x^3 - 232.11x^2 + 2689.8x + 150.71$	**6254.41**
					2 concurrent requests to process *large images* with *greyscale filter*	$13.711x^3 - 358.62x^2 + 3005.9x - 175.53$	4511.48	$58.718x + 76.977$	**194.41**	$3.511x^3 - 87.991x^2 + 719.57x - 76.358$	1038.91	$2.2861x^3 - 59.233x^2 + 486.02x - 51.898$	701.50
					4 concurrent requests to process *medium images* with *normal filter*	$14.873x^3 - 365.65x^2 + 2913.9x - 116.58$	**6640.49**	$0.072804x^3 - 1.8412x^2 + 16.496x + 79.083$	120.27	$6.0701x^3 - 148.56x^2 + 1188.4x - 118.69$	2646.44	$9.6006x^3 - 238.42x^2 + 1920.5x - 185.13$	4296.59

Table 7.3: Predicted Resource Requirements (maximum values were selected, shown in bold).

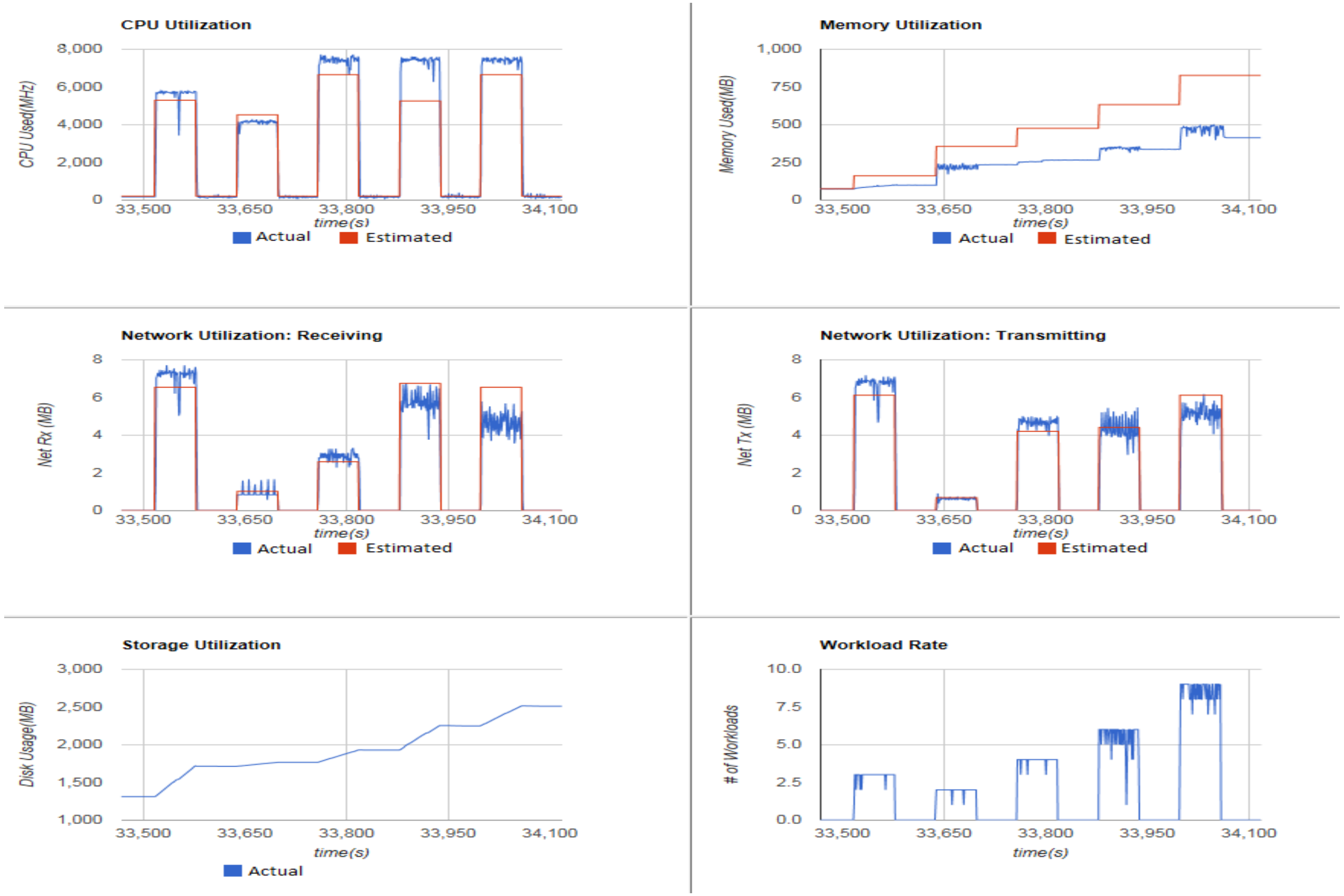


Figure 7.4: Actual and estimated resource usage.

7.8 Evaluation of the System Requirements

The usability of the proposed MDE approach can be achieved by satisfying the defined system requirements together with the possible implementation of components to automate the approach. The system level requirements of the research were defined in Section 6.3 together with its components and the tools/methods used in this research. The system level requirements listed in Section 6.3 are as below:

i. *Requirement 1*: The system shall provide accurate predictions of capacity;

ii. *Requirement 2*: The system shall support multiple statistical analysis techniques;

iii. *Requirement 3*: The system shall make use of rigorously-defined models for all configuration and data collection activities; and

iv. *Requirement 4*: The system shall make use of model management techniques for all validation and transformation activities.

The evaluation of predictive capability was conducted with the third case study as discussed in Section 7.7. This demonstrates that the proposed approach is possible to be used in estimating the resource requirements. Significant results were produce by comparing actual and estimated values for resource utilisation. Therefore, *Requirement 1* was satisfied; however, improvement with machine learning techniques is suggested as future work.

The modularity of DSMLS of ReRA process enables changes to a component without effecting other components. The modular component enables multiple statistical analysis techniques to be implemented. An equation extraction been used can be replaced with appropriate implementations such as applying machine learning. This capability can fulfil *Requirement 2* however the evaluation of this requirement is lay beyond the scope of this research.

Sets of DSMLs were defined and applied with three case studies for ReRA and ViRR. Rigorously-defined models conforming to the proposed DSMLs were used in the entire research for all configurations and data collection activities. This demonstrates the fulfilment of *Requirement 3* as presented in Chapter 5, Chapter 6 and Section 7.2.

Text-to-Model, Model-to-Model and Model-to-Text transformations together with model validation were utilised as the model management techniques. These model management techniques were used for handling the models and also the input/output of the related models. This demonstrates that *Requirement 4* was achieved, as presented in Chapter 5, Chapter 6 and Section 7.2.

Additional requirements that lay beyond the scope of this research are listed below:

i. The system shall enable full automation of the capacity planning process.

ii. The system shall be applicable for an application running on multiple virtual machines.

The requirements that lie outside the scope of this research are discussed in Section 8.4. The applications of the proposed MDE approach with two case studies, presented in Chapter 6 and also with the evaluation case studies in Section 7.2, generally demonstrate that all of the system level requirements listed in Section 6.3 were satisfied.

The usability of the proposed approach was demonstrated by applying it to three web applications running in a virtualised environment, as discussed in Chapter 6 and Section 7.2. In this research, applications that receive many concurrent and homogeneous requests, which are compatible within the capacity of the hosting infrastructure to serve were used. The case studies in this work cannot be used to evaluate the applicability of the approach to applications that receive irregular and/or heterogeneous requests. To achieve this, more complex applications (e.g. enterprise resource planning systems) involving heterogeneous workloads should be deployed and monitored. Despite the limited scope of the evaluation, there are still elements of practical usefulness, as applications with similar properties to those used in the evaluation case studies exist in the real world (e.g. document conversion services such as Convert API and video transcoding services such as Amazon's Elastic Transcoder). To demonstrate further real-world relevance, additional case studies should be conducted on such large-scale systems of this type.

The usability of the approach can also not be fully evaluated in the absence of experiments that involve real users (e.g. capacity managers). To generalise further the usability of the proposed solution, additional research or evaluation is required with applications hosted in larger and more complex virtualisation setups. Collaboration with industries that manage cloud-based applications which receive concurrent and homogeneous requests is suggested as additional research. The usability of the proposed approach can be achieved by satisfying the defined system requirements together with the possible implementation of components that are suitable for the selected environment.

7.9 Chapter Summary

The transparent and systematic capacity planning process proposed in this thesis was evaluated with an image filter web application as a third case study. The ReRA and ViRR processes were conducted with the proposed DSMLs and associated model management techniques to evaluate the usage of the proposed MDE solutions. A comparison between the actual and estimated resource utilisation was performed and discussed. The satisfaction at achieving the defined system requirement was discussed in this chapter.

Chapter 8

Conclusion

8.1 Introduction

This thesis presents a systematic and semi-automated MDE approach to supporting the initial phases of capacity planning in a virtualised environment. Two processes (one for resource requirements analysis and the second for resource requirements estimation) were designed, along with a set of DSMLs and model management techniques to support reuse and some automation. Resource requirement analysis (ReRA) analyses the resource usage pattern, demonstrates various types of graphs and produces formulas which correlate resource usage with application workloads. The second process estimates the virtual resource requirement (ViRR) based on formulas driven from ReRA and also known workload patterns of an application. The practicality and potential of the proposed approach was demonstrated through web applications running in VirtualBox virtualised environment. The use of DSMLs allows engineers to abstract information and characteristics from concrete tools (for implementing web applications, for monitoring, for analysing logs), and to automate analysis tasks in a precise, flexible and systematic way. The research in this thesis has been carried out to investigate the following hypothesis, stated in Section 1.4:

> *The hypothesis of this work is that MDE and DSML techniques can be used to support modular and reusable capacity planning in virtualised environments. In this context:*
>
> *i.* modular *means that every step of the capacity planning process is self-contained and the structure of its expected inputs*

and outputs is specified in a rigorous manner;

ii. reusable *means that steps/components can be shared between different capacity planning processes.*

To answer this hypothesis, the following objectives were defined in Section 1.5:

i. To identify capacity planning phases processes in virtualised environments based on a systematic literature review.

ii. To design systematic and model-based processes with a focus on the initial capacity planning phases.

iii. To design and implement DSMLs and model management techniques to support the identified processes.

iv. To evaluate modularity and reusability of the proposed DSMLs and model management techniques.

The following sections summarise the contributions of the thesis in relation to the hypothesis and research objectives. The limitations of the research and future research work are discussed in the following sections.

8.2 Research Contribution

The ReRA and ViRR processes integrate the first and second phases of the capacity planning framework as discussed in Section 4.2. In this thesis, MDE was utilised to automate the processes to enable the capacity modelling of applications running in virtualised environments. This was successfully achieved by automating the ReRA and ViRR processes with 3-level metamodelling architecture for three web applications running in a virtualised environment. Therefore, the contributions made in this thesis are as listed below:

i. The thesis contributes a three-phase framework for capacity planning in virtualised environments. Capacity planning involves predicting future computing resource requirements by monitoring a system's resource usage patterns against different workloads [6]. Capacity planning in virtualised environments is a multi-phase process that involves various stakeholders. In this thesis, the capacity planning in virtualised environments were clearly defined with a high level framework which integrates the identified processes in three phases.

ii. The thesis contributes ReRA and ViRR process to integrate the initial two phases. The identified process in the first two phases of the framework were systematically defined to provide transparent and organised flow. The resource requirement analysis (ReRA) process, analysed the resource usage pattern, demonstrated various types of graphs and produced formulas which correlated resource usage with application workloads. The second process, virtual resource requirement (ViRR) estimated the required resources based on formulas driven from ReRA together with known workload pattern of an application.

iii. The thesis contributes a couple of DSML sets and model management techniques to automate ReRA and ViRR processes. Utilising a Model-Driven Engineering (MDE) approach allows for abstraction of the respective concrete tools and to automate analysis tasks in a precise, flexible and systematic way by using model management languages and tools. Sets of DSMLs and model management techniques were designed, implemented and evaluated to semi-automate the processes. The DSMLs allow resource and request logs, as well as workloads, to be precisely captured using models, as well as a transparent, automated and repeatable MDE process for generating predictions for resource usage from workload models. The MDE process, which exploits model transformation, comparison and merging, is modularised so that it can be configured for different kinds of capacity planning applications and technical infrastructures.

iv. The thesis also contributes the practicality and potential of applying the proposed MDE solutions through three web applications running in VirtualBox virtualised environment. The proposed core MDE solutions are modular, reusable and extensible to various technologies.

v. The thesis opens new research opportunities by implementing the DSMLs using EMF/Ecore standardised technologies which makes the DSMLs potentially usable or reusable by others. This in turn can make it easier for developers to define *reusable connections* between tools to support analysis (and hence capacity planning), thus enriching the automated capability for the capacity planning community in general.

8.3 Research Limitation

The limitations of the research are as follows:

i. The limitation of the evaluation was the implementation of the proposed MDE solutions with web applications running on single virtual machine with limited physical resources. Several initiatives were taken to setup a private cloud environment in order to have complete control of the infrastructure. Initially, Eucalyptus and OpenStack were explored with three servers. Due to technical constrains, setting up the environment was time consuming so, as an alternative, VirtualBox was configured with a server. The limited resources prevented the evaluation of application running in more than a VM.

ii. The resource requirement analysis (ReRA) was conducted to correlate resource requirement for numbers of workloads being processed. The formulas produced by using this process are dependent on a single parameter (number of concurrent workloads). An application's workloads are categorised by considering all the possible combinations of the workload's components. For example, nine categories of workload were identified to analyse all of the possible workload combinations (3 filters x 3 file categories). The more components there are, the more ReRA analysis needs to be conducted and as such, many formulas will be produced. Writing the transformation program in ViRR is dependent on the formulas produced from the ReRA. Having many formulas makes the transformation program extensive and complex. Therefore, these approaches are most suitable for more manageable numbers of workload categories.

iii. The thesis presents three-phase framework of capacity planning in virtualised environments. The proposed ReRA and ViRR processes are initial processes of the framework which integrate first two phases. Several more detail processes to fully automate the three-phase framework are not covered in this research.

iv. Statistical analyses by calculate mean, median, maximum and minimum resource utilisation were observed. In this thesis, mean value was selected to produce resource requirements formulas for the interested resource metrics. Although the estimated resource requirements based on produced formulas were significant, more precise statistical method is possible to be implemented and this is proposed as future study.

8.4 Future Work

This section briefly describes potential extensions to the research.

Fully automated process. The ReRA process that we described in the thesis is not yet fully automated: configuration data (e.g., number of CPUs being used by the application, number of users) must be provided at the application level. At present, this is achieved manually, by editing the models, but automating this further by providing a bespoke editor (e.g., implemented using update-in-place model transformations) might prove useful.

Applying a better statistical method for prediction. Although the accuracy of the generated formula is beyond the scope of this research, using better statistical methods such as machine learning is suggested in future work to provide better resource requirement estimations. In this research work, basic statistical analysis was used to produce the resource requirement formulas. Machine learning methods normally include statistical analysis features which can be used to analyse the resource requirement of the workloads. The accuracy of resource requirements prediction depends on the statistical analytic capability of the analytic method been used to produce the equations. Applying better statistical analysis methods such as machine learning might provide more accurate prediction.

Usability evaluation on large scale and complex environments. The implementation with three case studies demonstrates the usability of the proposed approach in small virtualised environments. To evaluate further the usability of the proposed MDE solutions, case studies with cloud-base applications in large and complex virtualised environments are suggested. Collaboration with industry is one option for implementing this evaluation.

Evaluation with more than 1 VM. Further enhancement by implementing the MDE solutions with multiple VMs is suggested, since many applications in the industry are run with several VMs.

Improvement in estimating storage requirement. Time taken to process the workloads was captured in the related models. It may be possible to estimate the storage requirement by considering the storage consumed after processing each workload and time taken to complete the process. However, this consideration needs further analysis, development and evaluation.

8.5 Closing Remarks

A framework to integrate the identified three phases in capacity planning for visualisation was discussed in Chapter 4. The use of domain specific modelling to facilitate the integration of identified processes in the initial phases were proposed. The DSMLs and the model management techniques for the suggested ReRA and ViRR processes were discussed in Chapter 5. In Chapter 6, the implementation of proposed MDE solutions to perform capacity planning specifically for web applications running in VirtualBox visualisation environment were conducted with media stream and speech tagger web applications. Further, to evaluate the proposed MDE solutions, an additional case study with an image filtering web application was presented in Chapter 7. Finally, this chapter concluded the overall research work with a discussion on research contributions and the limitations of the proposed solutions. The potential future work to improve and extend the research were also discussed.

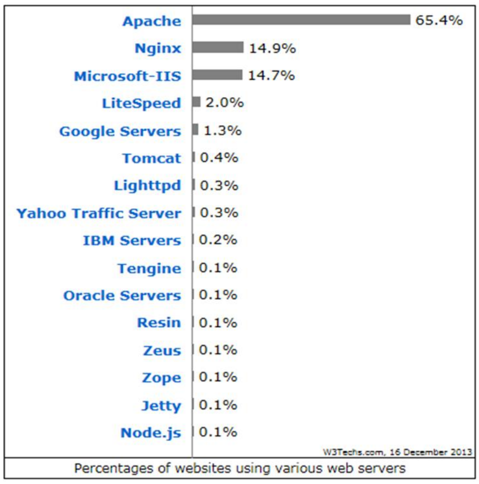
Appendices

Appendix A

Web Servers Analysis

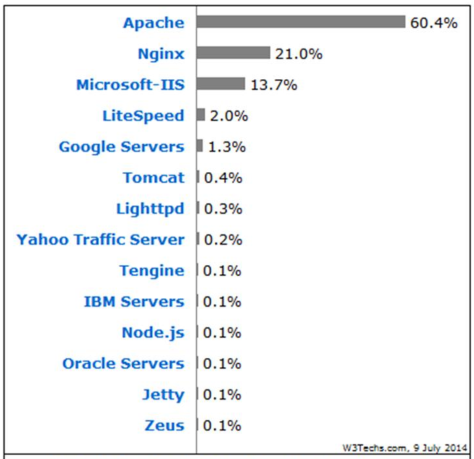
Analysis of Web Servers

The list of available web servers with the percentage of usage of various website can be retrieved from “http://w3techs.com/technologies/overview/web_server/all ” . Figures were selected from the website. The list on the left was taken on 16 December 2013 and the right on 9 July 2014.



W3Techs.com, 16 December 2013

Percentages of websites using various web servers



W3Techs.com, 9 July 2014

Percentages of websites using various web servers

Both listings show that Apache, Nginx, Microsoft-IIS, LiteSpeed, Lightttpd and Tomcat are among top in the list. Along with this, the log files which capture the web request information of these web servers were analysed. Table below shows the comparison and the proposed *RequestLog* DSML can be used for these web servers.

Information From Request Log	Web Server’s Request Parameters					
	Apache	Nginx	IIS	LiteSpeed	lighttpd	Tomcat
Time to process the request	%D, %T	$request_time	Time taken	%D, %T	%D, %T	%D, %T
Request start time	%t	$time_iso8601, $time_local	Time	%t	%t	%t
Request	%r	$request_length	Request type, Parameters	%r	%r	%r
Request status	%>s	$status	Service status code	%>s	%s	%s
Return data size	%b	$bytes_sent	Server bytes sent	%b	%b, %B	%b, %B
Receive data size	-	-	Client bytes sent	-	%I	-

Appendix B

VBoxManage Resource Parameters

Resource Metrics in VirtualBox : VBoxManage metrics list

Resources	Metrics	Unit	Host	VMs
CPU (actual, average, minimum & maximum for each metrics)	CPU/Load/User	%	Percentage of processor time spent in user mode.	Percentage of processor time spent in user mode by the VM process.
	CPU/Load/Kernel	%	Percentage of processor time spent in kernel mode.	Percentage of processor time spent in kernel mode by the VM process.
	CPU/Load/Idle	%	Percentage of processor time spent idling.	X
	CPU/MHz	MHz	Average of current frequency of all processors.	X
	Guest/CPU/Load/User	%	X	Percentage of processor time spent in user mode as seen by the guest.
	Guest/CPU/Load/Kernel	%	X	Percentage of processor time spent in kernel mode as seen by the guest.
	Guest/CPU/Load/Idle	%	X	Percentage of processor time spent idling as seen by the guest.
Memory (actual, average, minimum & maximum for each metrics)	RAM/Usage/Total	kB	Total physical memory installed.	X
	RAM/Usage/Used	kB	Physical memory currently occupied.	Size of resident portion of VM process in memory.
	RAM/Usage/Free	kB	Physical memory currently available to applications.	X
	RAM/VMM/Used	kB	Total physical memory used by the hypervisor.	X
	RAM/VMM/Free	kB	Total physical memory free inside the hypervisor.	X
	RAM/VMM/Ballooned	kB	Total physical memory ballooned by the hypervisor.	X
	RAM/VMM/Shared	kB	Total physical memory shared between VMs.	X
	Guest/RAM/Usage/Total	kB	X	Total amount of physical guest RAM.
	Guest/RAM/Usage/Free	kB	X	Free amount of physical guest RAM.
	Guest/RAM/Usage/Balloon	kB	X	Amount of ballooned physical guest RAM.
	Guest/RAM/Usage/Shared	kB	X	Amount of shared physical guest RAM.
	Guest/RAM/Usage/Cache	kB	X	Total amount of guest (disk) cache memory.

Storage (actual, average, minimum & maximum for each metrics)	FS/{/}/Usage/Total	mB	Root file system size.	X
	FS/{/}/Usage/Used	mB	Root file system space currently occupied.	X
	FS/{/}/Usage/Free	mB	Root file system space currently empty.	X
	Disk/ua006-root/Load/Util	%	Percentage of time disk was busy serving I/O requests.	X
	Disk/ua006-root/Usage/Total	mB	Disk size.	X
	Disk/Usage/Used	mB	X	Actual size of all VM disks combined.
	Guest/Pagefile/Usage/Total	kB	X	Total amount of space in the page file.
Network (actual, average, minimum & maximum for each metrics) Note: For Host, eth1 is primary interface and eth0 might be vboxnet0 for local network).	Net/eth1/LinkSpeed	mbit/s	Physical link speed.	X
	Net/eth1/Load/Rx	%	Percentage of network interface receives bandwidth used.	X
	Net/eth1/Load/Tx	%	Percentage of network interface transmits bandwidth used.	X
	Net/eth0/LinkSpeed	mbit/s	Physical link speed.	X
	Net/eth0/Load/Rx	%	Percentage of network interface receives bandwidth used.	X
	Net/eth0/Load/Tx	%	Percentage of network interface transmits bandwidth used.	X
	Net/Rate/Rx	B/s	X	Network receive rate.
	Net/Rate/Tx	B/s	X	Network transmit rate.

Appendix C

Part-of-Speech Tagger Case Study Graphs

C.1 Introduction

Presented in this appendix are the outcomes of implementing ReRA process with proposed MDE solutions for the Part-of-Speech Tagger case study. Mainly, the outcomes were presented with graphs which summarise the resource usage for the identified workloads. Three types of workloads were defined to process three categories of file sizes. The ReRA process was conducted and the outcomes were presented with three sets of graphs for each workload. The first set shows the resource utilisation to process concurrent workloads. The second set presents the correlation between workloads and resource consumption for a selected statistical unit (mean, median, maximum and minimum). The third set shows the outcome of applying equation extracted to produce formulas analysing the mean value.

C.2 Experiment with Large Files Size

This experiment involved a file size of 147,035 bytes with a 24,767 word count and 100 similar files were generated, each with a unique name. The list of 100 files was processed for each concurrent request generated. The request pattern is illustrated in Figure C.1. The time taken to process concurrent requests of more than 4 was longer due to the CPU bottleneck as illustrated in Figure C.2. The bottleneck occurs due to the resource constrain in the

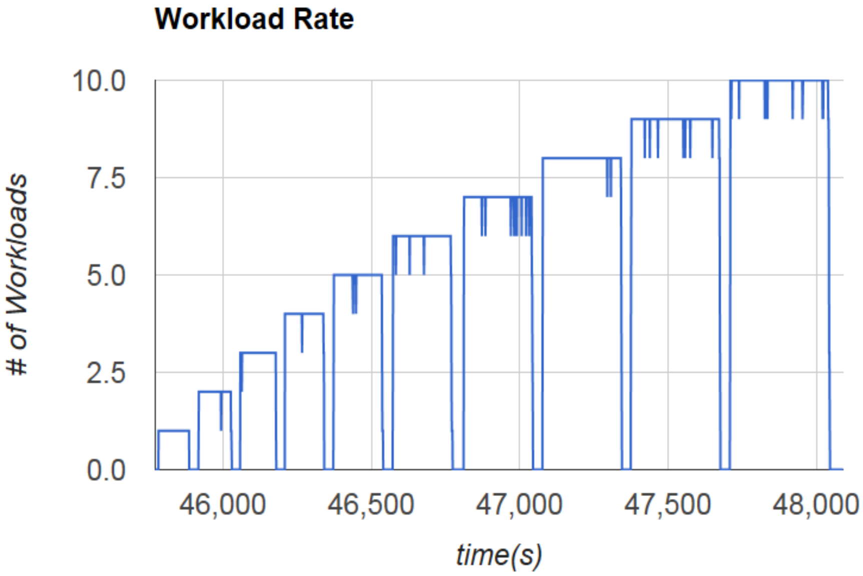


Figure C.1: Part-of-speech tagger requests pattern for large file size.

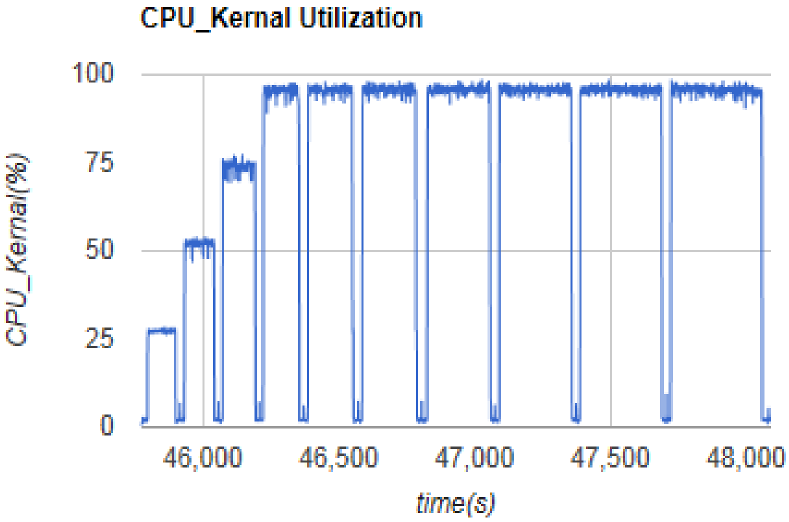


Figure C.2: The CPU usage (%) observation with VBoxManage for the simulated requests to process large files.

experimental environment. The host used to set-up the virtualised environment only had 4 CPU units. Therefore, a VM runs in that virtualised environment can only have maximum of 4 CPU units.

Observation of the application behaviour shows that the number of CPU

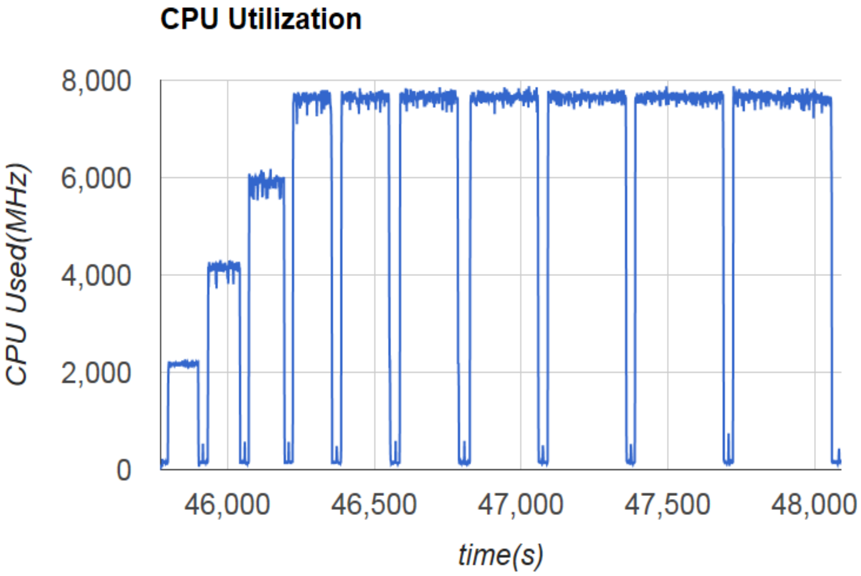


Figure C.3: The CPU usage of the simulated part-of-speech tagger requests to process large files.

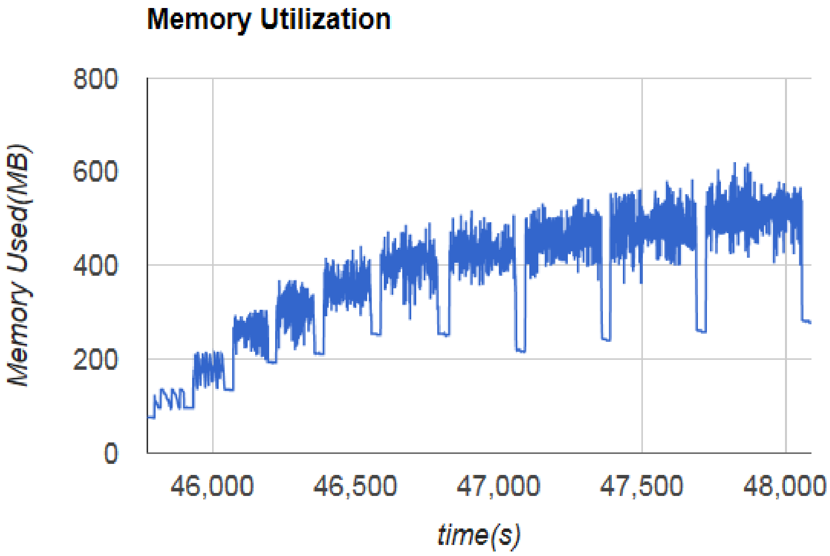


Figure C.4: The memory utilisation of the simulated part-of-speech tagger requests to process large files.

units fully utilised is related to the number of concurrent requests. Since 4 CPU units were allocated to this VM, the application fully utilised the available CPUs when 4 concurrent requests were being processed. Therefore, the

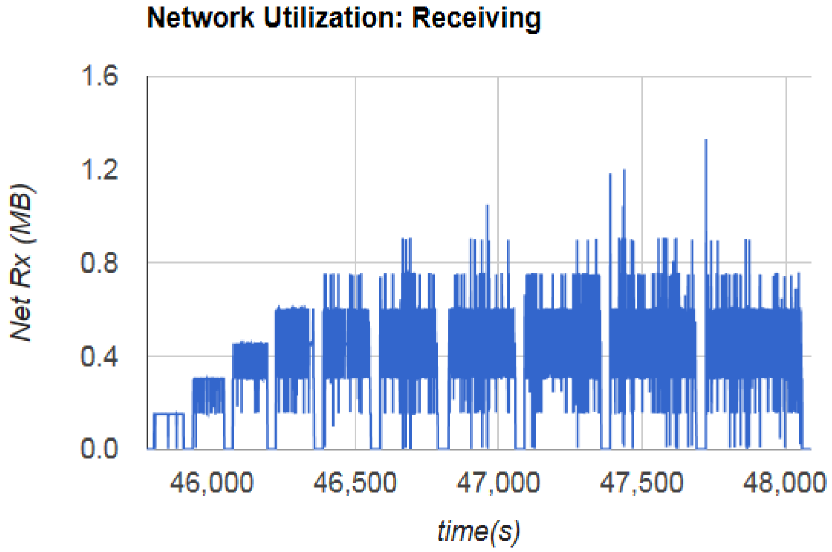


Figure C.5: The incoming network traffic received by the application based on the simulated part-of-speech tagger requests to process large files.

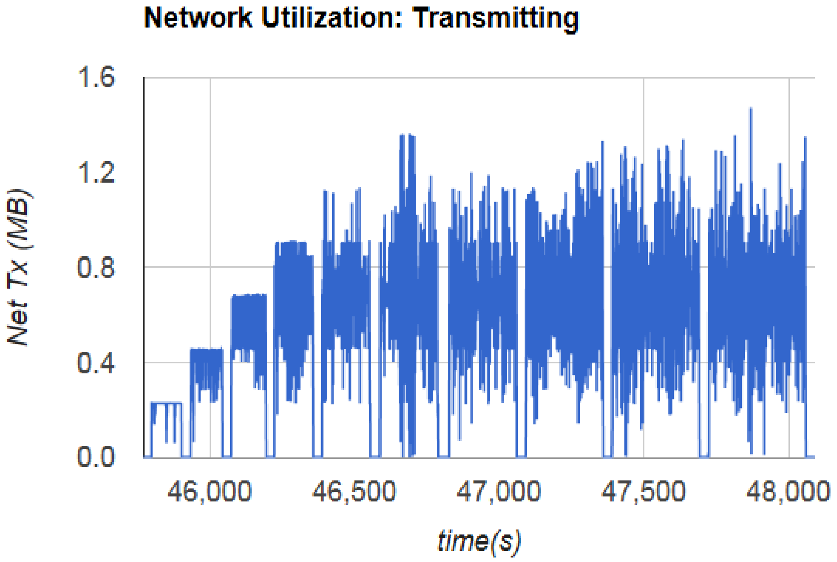


Figure C.6: The outgoing network traffic transferred by the application based on the simulated part-of-speech tagger requests to process large files.

time taken to process more than 4 concurrent requests is longer since additional requests are waiting for CPU availability. Figures C.3 to C.7 show the utilisation of CPU, memory, incoming network, outgoing network and stor-

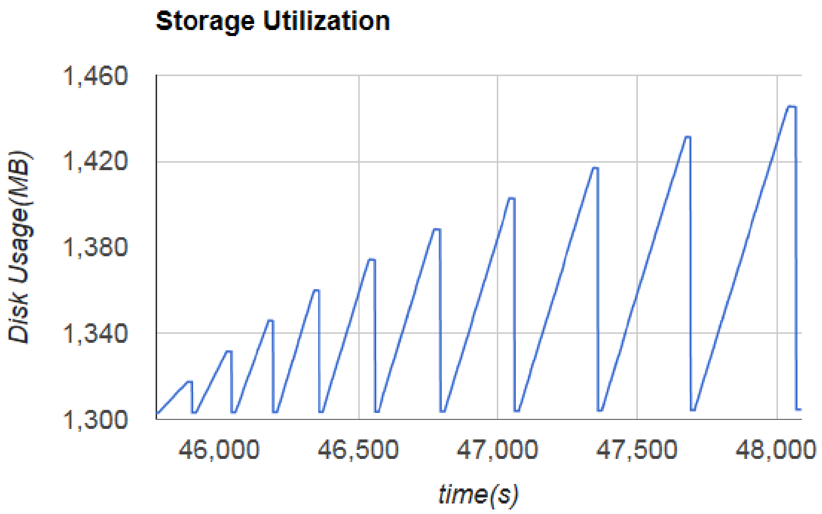
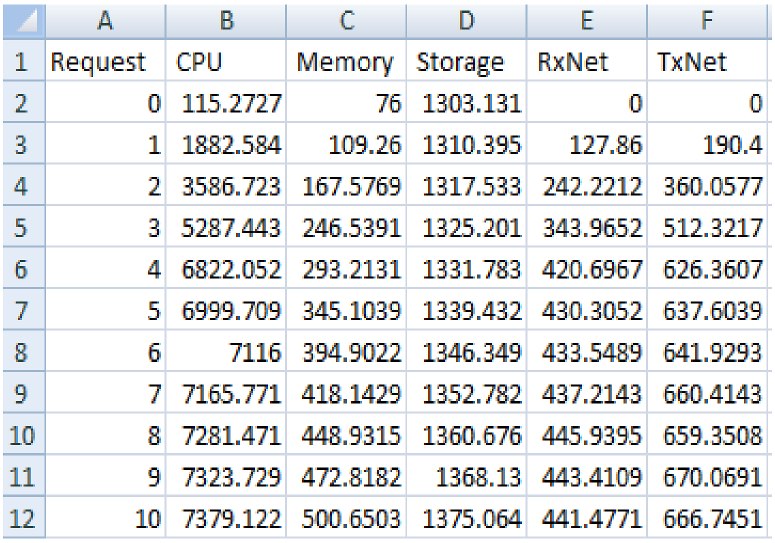


Figure C.7: The storage usage recorded based of the simulated part-of-speech tagger requests to process large files.

age. The statistical analysis of these resource utilisations were synthesized and a CSV file as presented in Figure C.8 was produced. The data from the CSV file was used to generate a set of resource usage graphs as presented in Figure C.9.



	A	B	C	D	E	F
1	Request	CPU	Memory	Storage	RxNet	TxNet
2	0	115.2727	76	1303.131	0	0
3	1	1882.584	109.26	1310.395	127.86	190.4
4	2	3586.723	167.5769	1317.533	242.2212	360.0577
5	3	5287.443	246.5391	1325.201	343.9652	512.3217
6	4	6822.052	293.2131	1331.783	420.6967	626.3607
7	5	6999.709	345.1039	1339.432	430.3052	637.6039
8	6	7116	394.9022	1346.349	433.5489	641.9293
9	7	7165.771	418.1429	1352.782	437.2143	660.4143
10	8	7281.471	448.9315	1360.676	445.9395	659.3508
11	9	7323.729	472.8182	1368.13	443.4109	670.0691
12	10	7379.122	500.6503	1375.064	441.4771	666.7451

Figure C.8: Table structure of CVS file for the statistical analysis of average reading of resource metrics to process large size files.

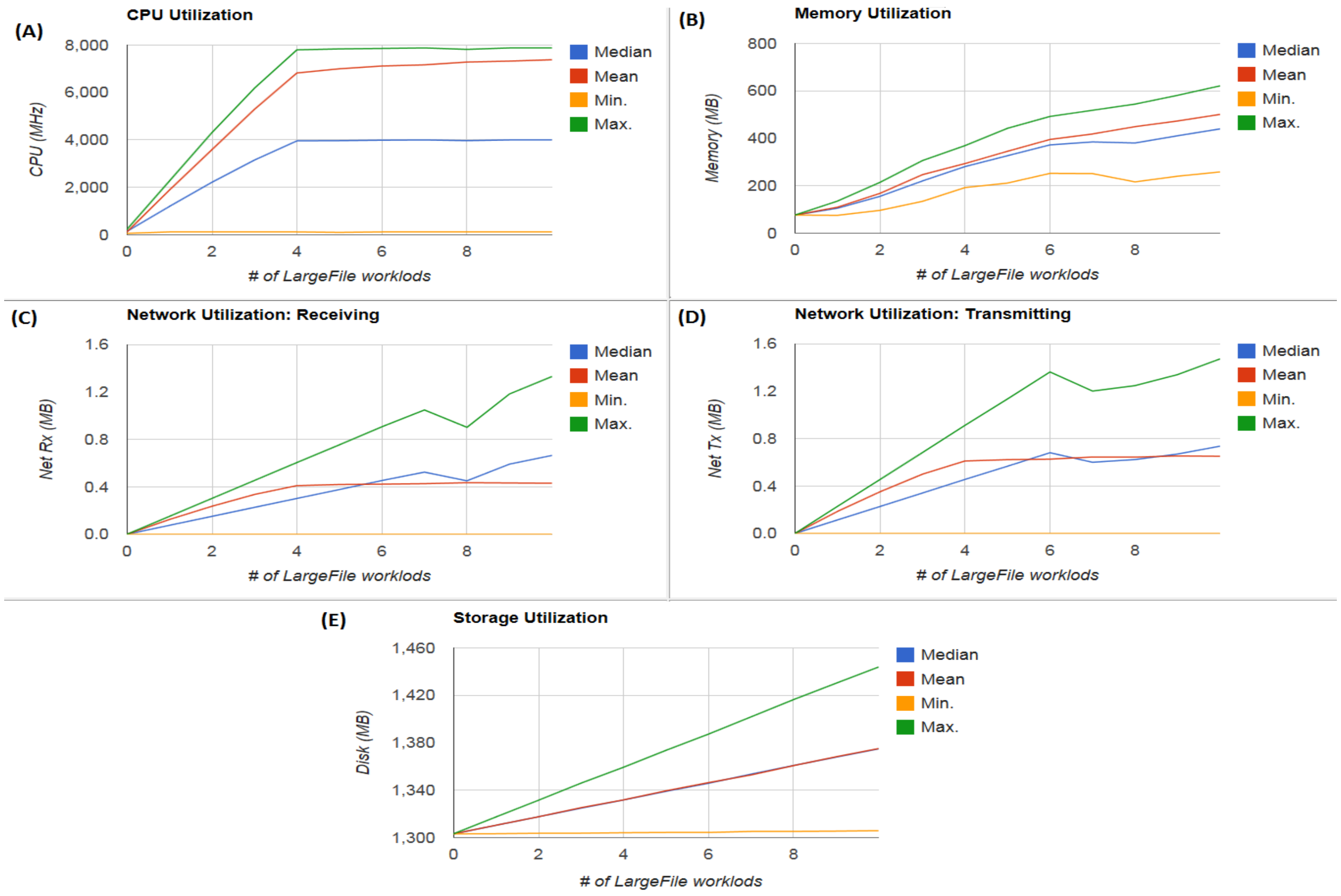


Figure C.9: Statistical results of resource usage to process large files.

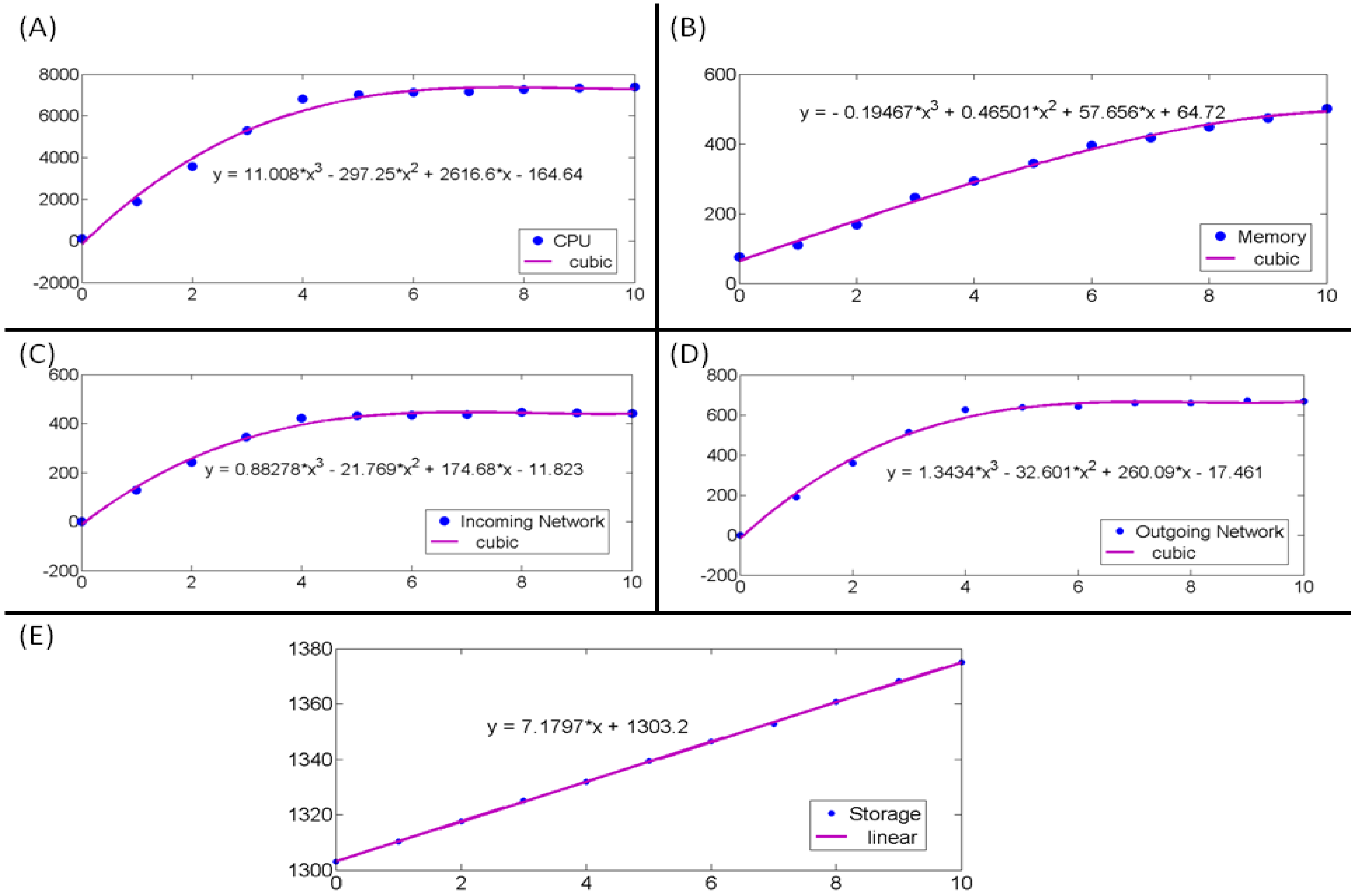


Figure C.10: Formula retrieved to process large files based on average (mean) resource usage.

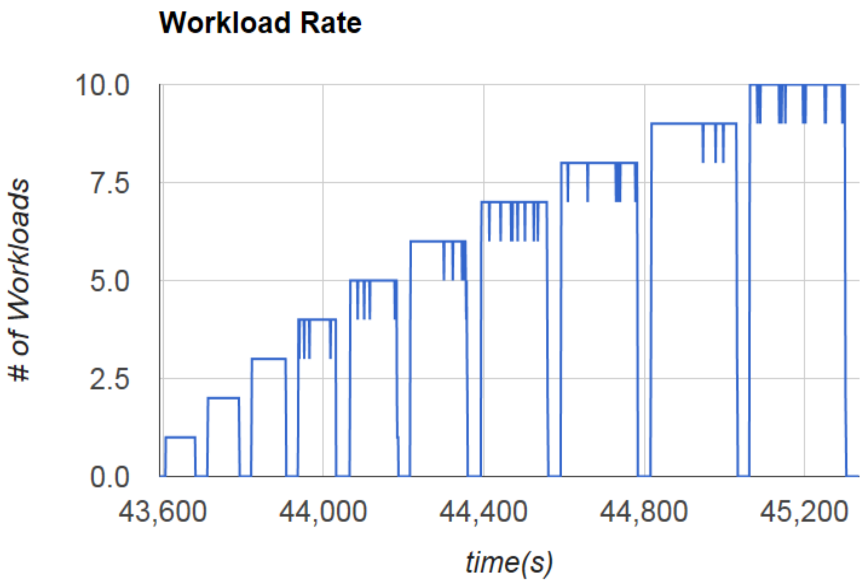


Figure C.11: Part-of-speech tagger requests pattern for medium file size.

C.3 Experiment with Medium Files Size

In this experiment, a list of 100 files with a size of 57,765 bytes and a 10,873 word count were used. The requests were simulated similar to the above experiment and Figure C.11 shows the request pattern. The resource utilisation graphs of each of the resource metrics are illustrated in Figures C.12 to C.16 respectively for CPU, memory, incoming network, outgoing network and storage utilisation to process simulated workloads. The figures show that, storage and memory utilisation grows linearly with the size of the workload and network utilisation is affected by the number of available CPUs.

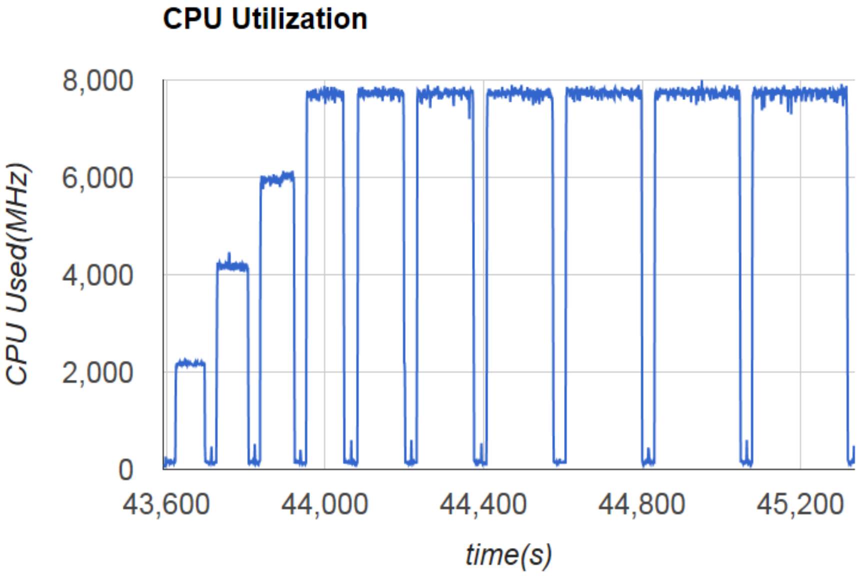


Figure C.12: The CPU usage of the simulated part-of-speech tagger requests to process medium files.

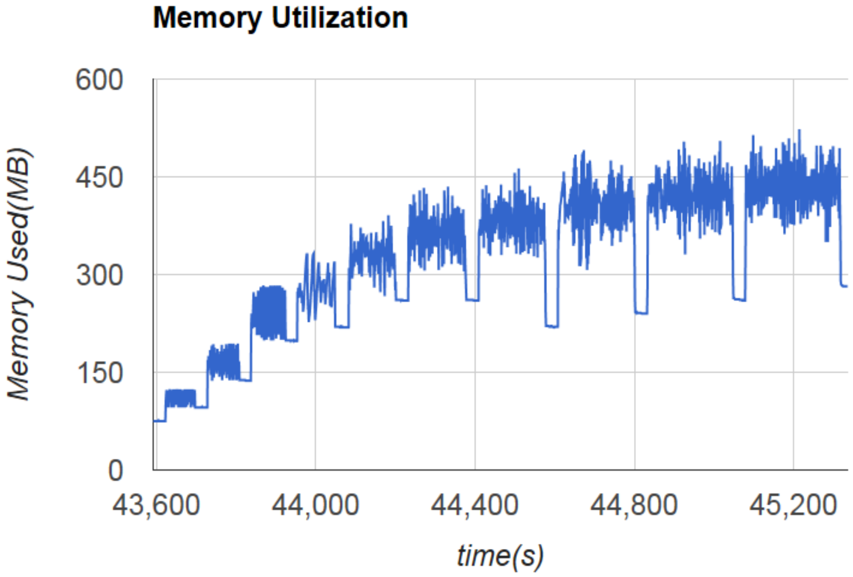


Figure C.13: The memory utilisation of the simulated part-of-speech tagger requests to process medium files.

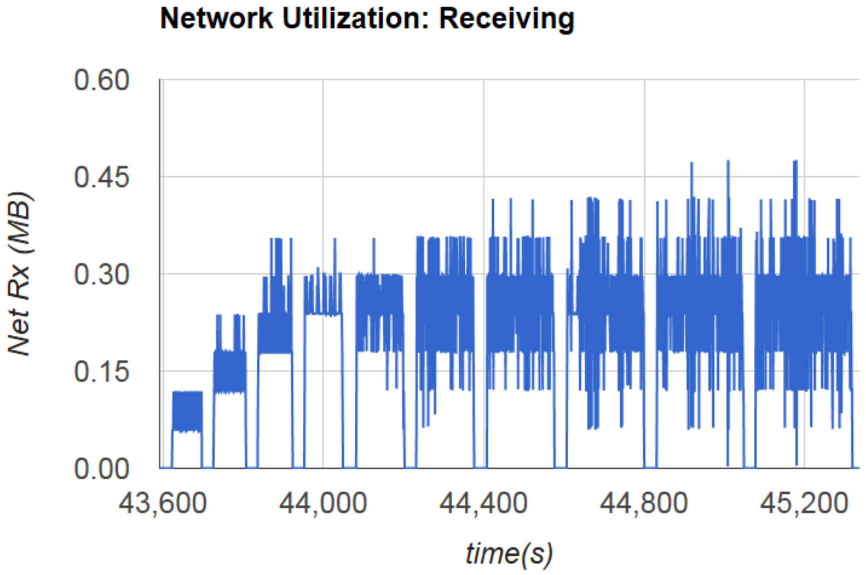


Figure C.14: The incoming network traffic received by the application based on the simulated part-of-speech tagger requests to process medium files.

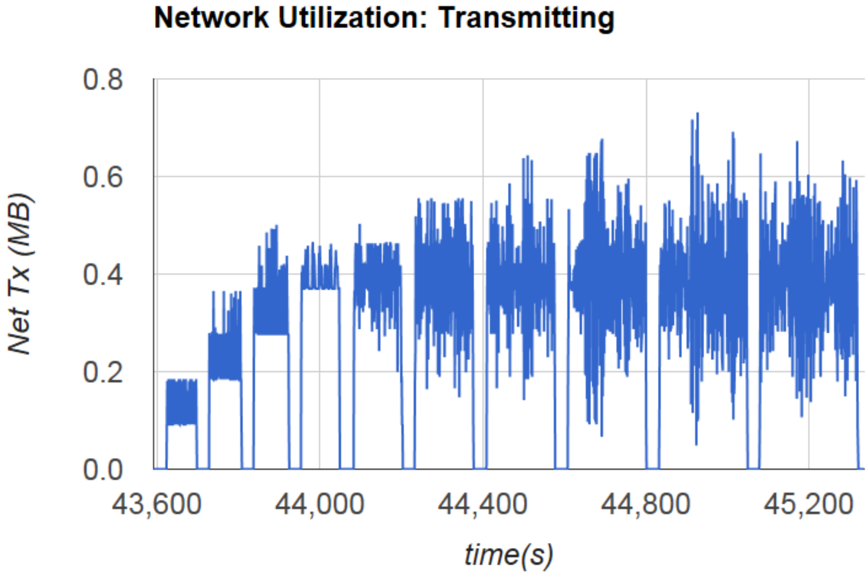


Figure C.15: The outgoing network traffic transferred by the application based on the simulated part-of-speech tagger requests to process medium files.

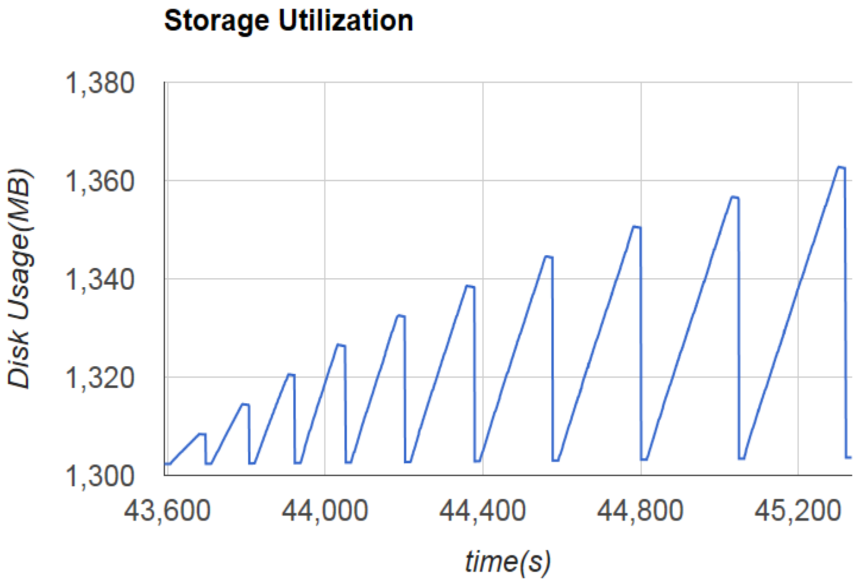
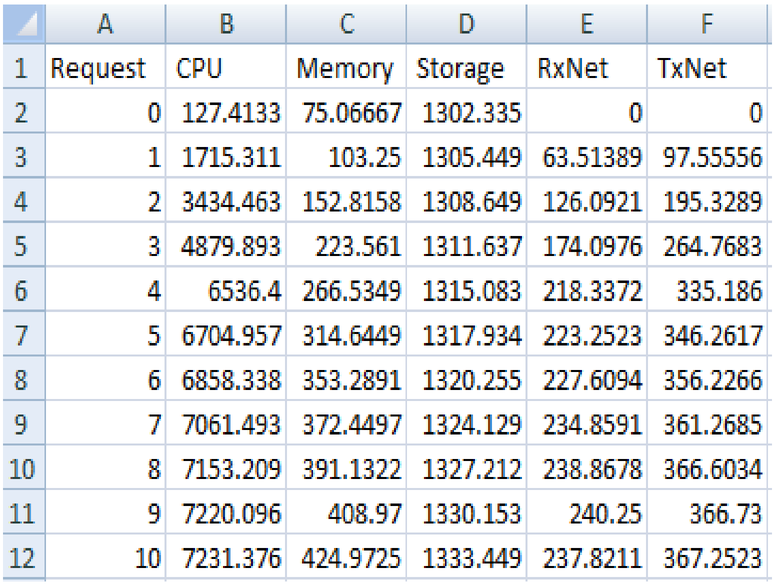


Figure C.16: The storage usage recorded based of the simulated part-of-speech tagger requests to process medium files.



	A	B	C	D	E	F
1	Request	CPU	Memory	Storage	RxNet	TxNet
2	0	127.4133	75.06667	1302.335	0	0
3	1	1715.311	103.25	1305.449	63.51389	97.55556
4	2	3434.463	152.8158	1308.649	126.0921	195.3289
5	3	4879.893	223.561	1311.637	174.0976	264.7683
6	4	6536.4	266.5349	1315.083	218.3372	335.186
7	5	6704.957	314.6449	1317.934	223.2523	346.2617
8	6	6858.338	353.2891	1320.255	227.6094	356.2266
9	7	7061.493	372.4497	1324.129	234.8591	361.2685
10	8	7153.209	391.1322	1327.212	238.8678	366.6034
11	9	7220.096	408.97	1330.153	240.25	366.73
12	10	7231.376	424.9725	1333.449	237.8211	367.2523

Figure C.17: Table structure of CVS file for the statistical analysis of average reading of resource metrics to process medium size files.

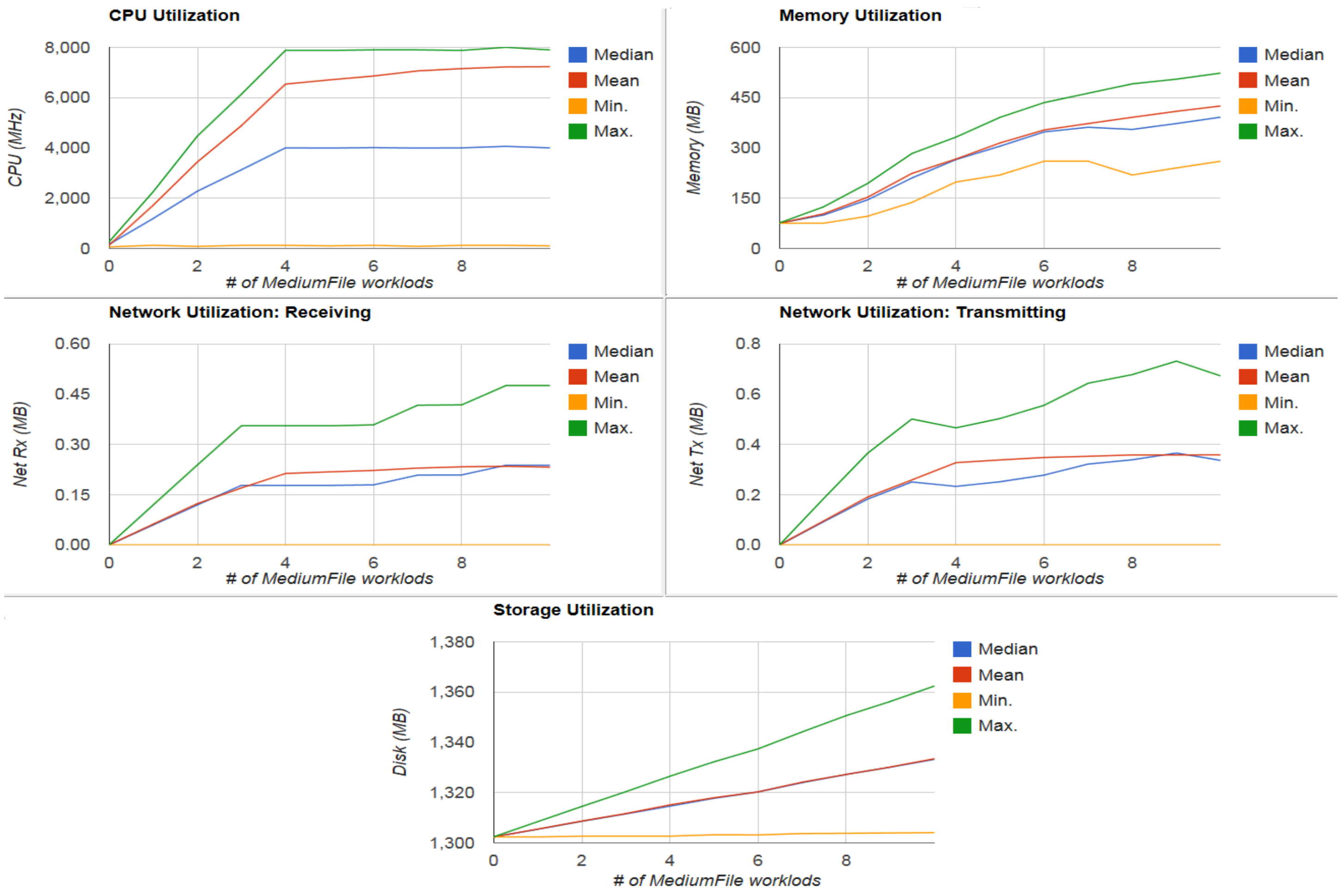


Figure C.18: Statistical results of resource usage to process medium files.

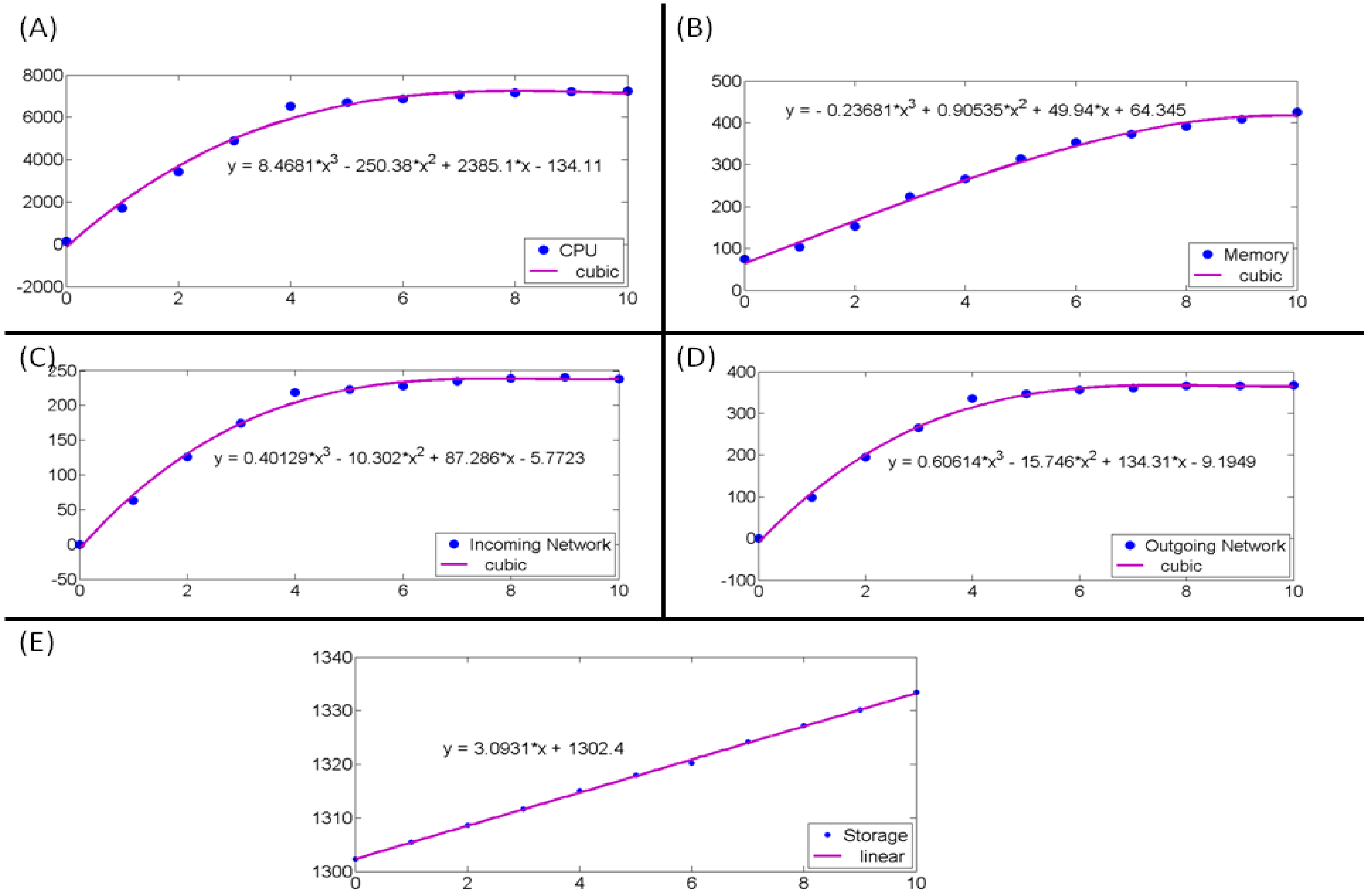


Figure C.19: Formula retrieved to process medium files based on average (mean) resource usage.

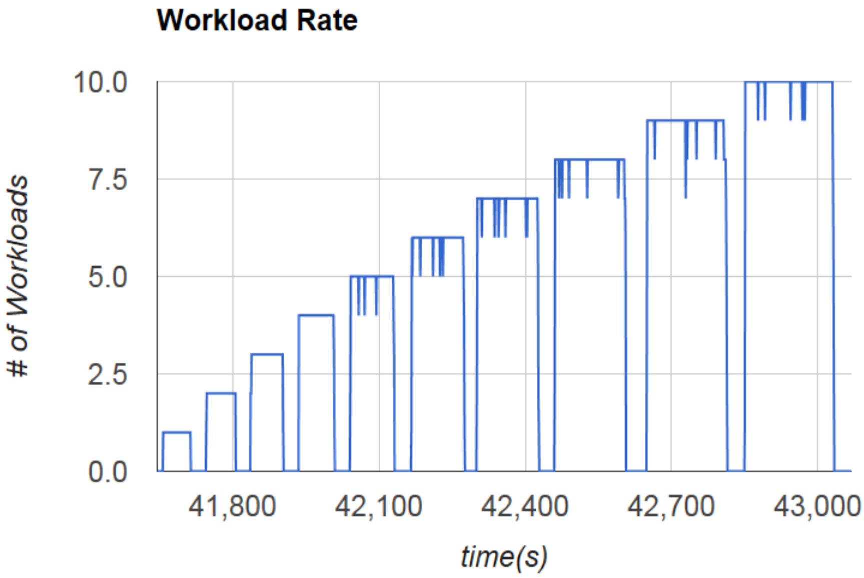


Figure C.20: Part-of-speech tagger requests pattern for small file.

C.4 Experiment with Small Files Size

For the final experiment, a list of 100 smaller files were generated. The size of the files was 21,291 bytes and the word count was 3,781. The requests were simulated similar to the above experiment and Figure C.20 shows the request pattern. The resource utilisation graphs of each resource metrics are illustrated in Figures C.21 to C.25. Looking at the figures, storage and memory utilisation grows linearly with the size of the workload and network utilisation is affected by the number of available CPUs.

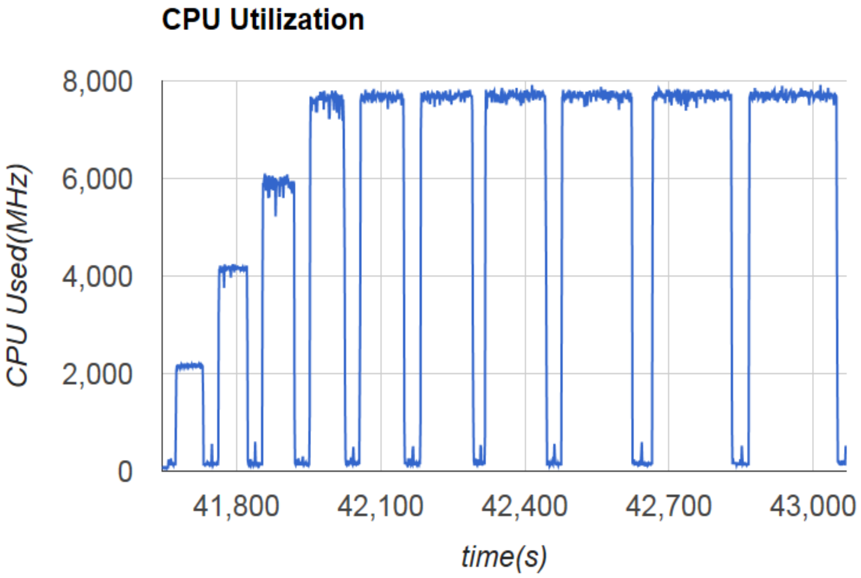


Figure C.21: The CPU usage of the simulated part-of-speech tagger requests to process small files.

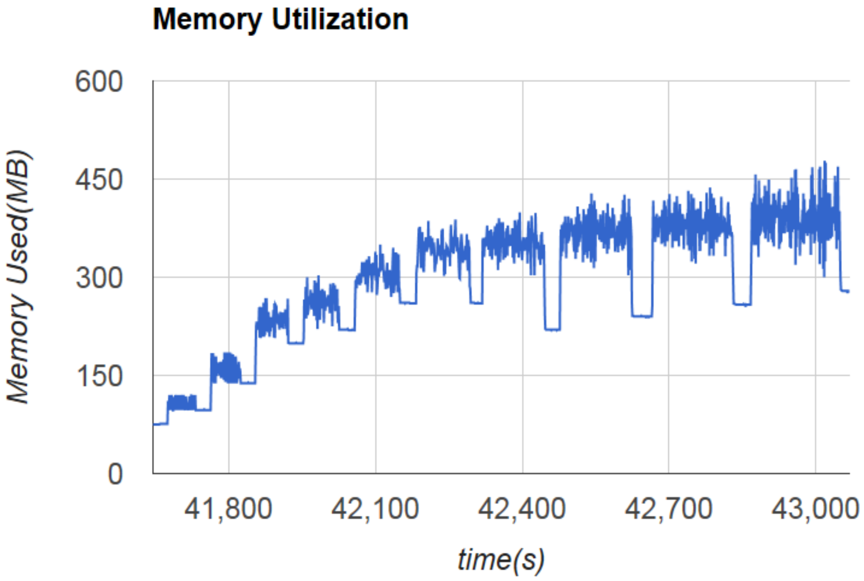


Figure C.22: The memory utilisation of the simulated part-of-speech tagger requests to process small files.

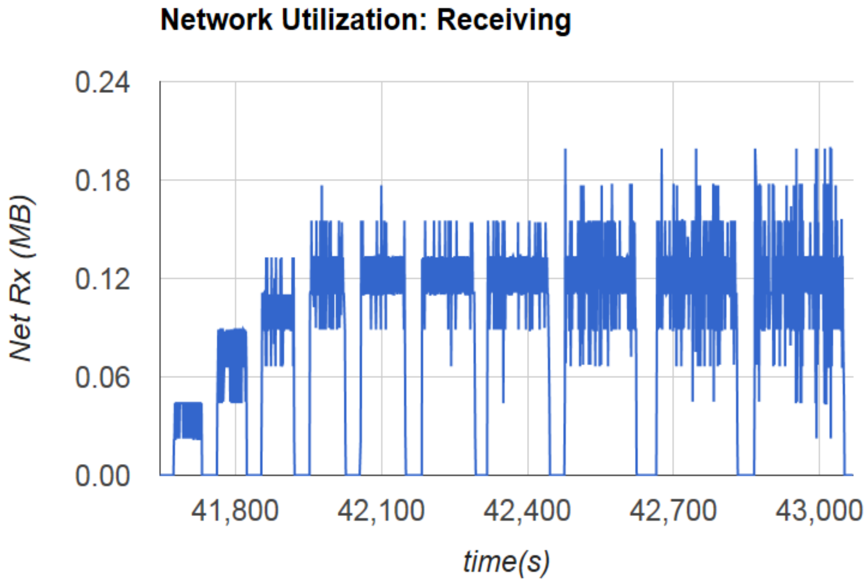


Figure C.23: The incoming network traffic received by the application based on the simulated part-of-speech tagger requests to process small files.

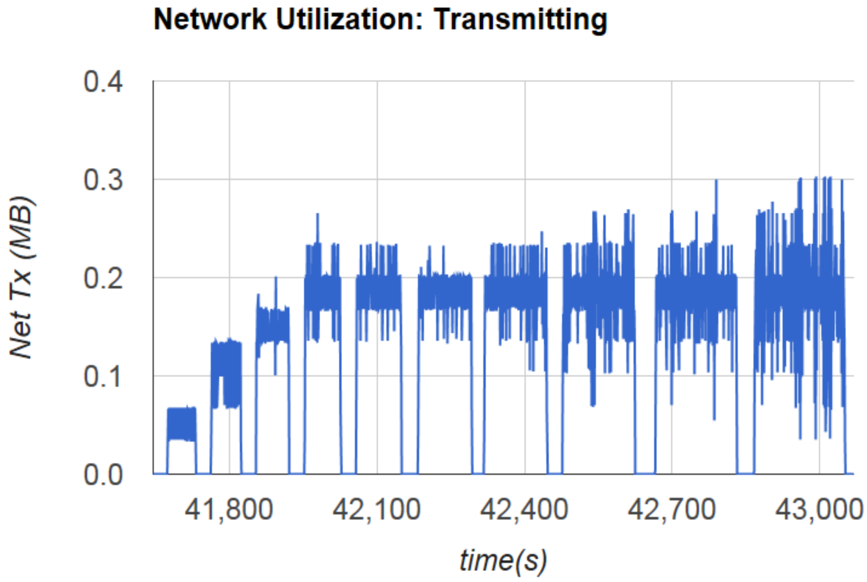


Figure C.24: The outgoing network traffic transferred by the application based on the simulated part-of-speech tagger requests to process small files.

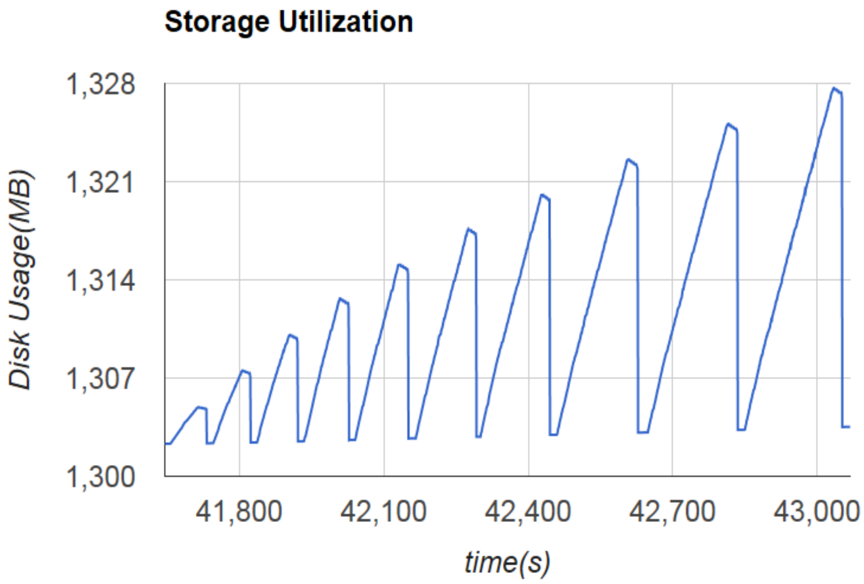
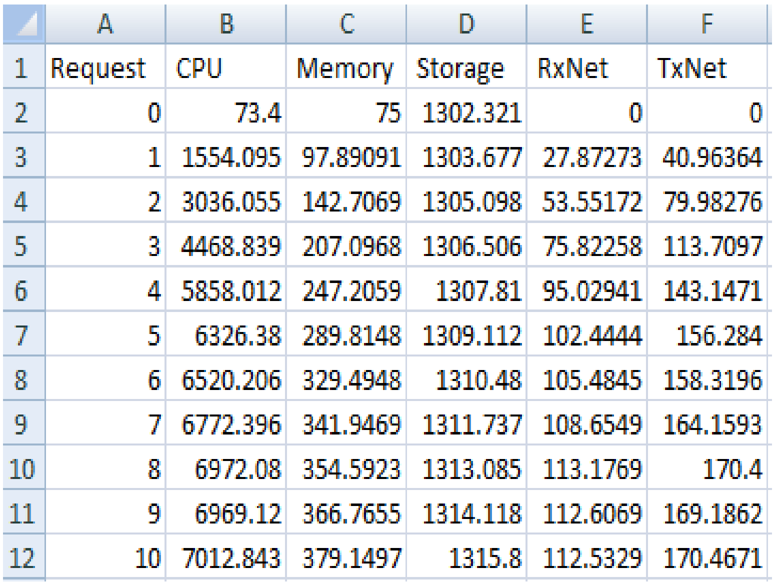


Figure C.25: The storage usage recorded based of the simulated part-of-speech tagger requests to process small files.



	A	B	C	D	E	F
1	Request	CPU	Memory	Storage	RxNet	TxNet
2	0	73.4	75	1302.321	0	0
3	1	1554.095	97.89091	1303.677	27.87273	40.96364
4	2	3036.055	142.7069	1305.098	53.55172	79.98276
5	3	4468.839	207.0968	1306.506	75.82258	113.7097
6	4	5858.012	247.2059	1307.81	95.02941	143.1471
7	5	6326.38	289.8148	1309.112	102.4444	156.284
8	6	6520.206	329.4948	1310.48	105.4845	158.3196
9	7	6772.396	341.9469	1311.737	108.6549	164.1593
10	8	6972.08	354.5923	1313.085	113.1769	170.4
11	9	6969.12	366.7655	1314.118	112.6069	169.1862
12	10	7012.843	379.1497	1315.8	112.5329	170.4671

Figure C.26: Table structure of CVS file for the statistical analysis of average reading of resource metrics to process small size files.

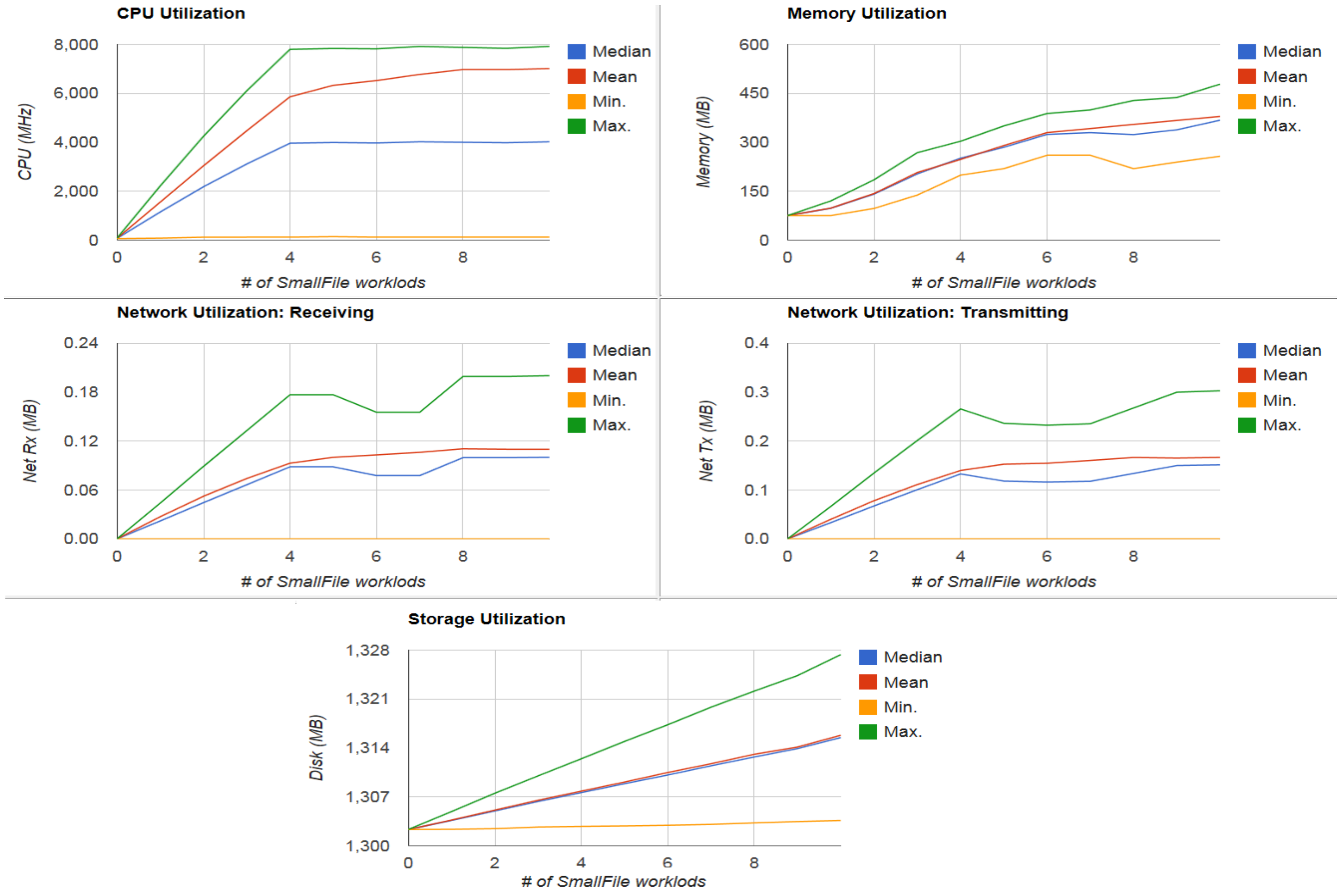


Figure C.27: Statistical results of resource usage to process small files.

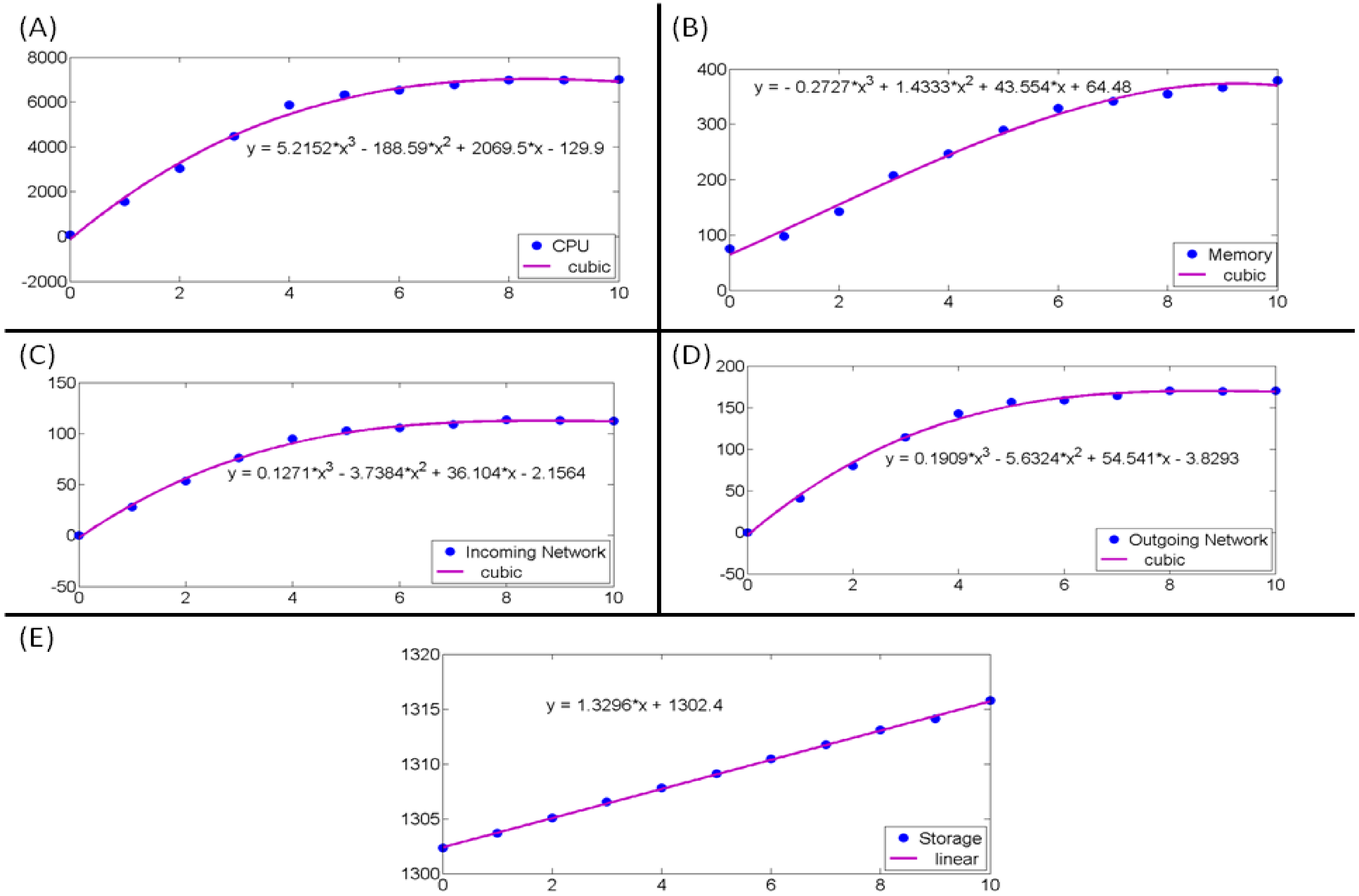


Figure C.28: Formula retrieved to process small files based on average (mean) resource usage.

Appendix D

Image Filter Case Study Graphs

D.1 Introduction

Presented in this appendix are the outcomes of evaluating the ReRA process with proposed MDE solutions for the image filter case study. Mainly, the outcomes were presented with graphs which summarise the resource usage for the identified workloads. Nine types of workloads were defined to process three categories of images, with three 3 types of filters. The ReRA process was conducted and the outcomes were presented with three sets of graphs for each workload. The first set shows the resource utilisation to process concurrent workloads. The second set presents the correlation between workloads and resource consumption for a selected statistical unit (mean, median, maximum and minimum). The third set shows the outcome of equation extraction module which produce resource metrics formulas by analysing the mean value.

D.2 Large Images

Three types of filters were used to process 200 large images and the logs recordings were analysed. The following shows the output of the ReRA process for the three experiments conducted respectively for default, grayscale and negate image filters.

i. **Experiment with Default Filter**

The resource consumption for the workloads which process large images with the default filter is presented in Figure D.1. Basic statistical analyses were performed and the outcome for the workload which process large images with the default filter is presented in Figure D.2. Figure D.3 show the graphs and the formulas which correlate the number of workloads with the resource metrics.

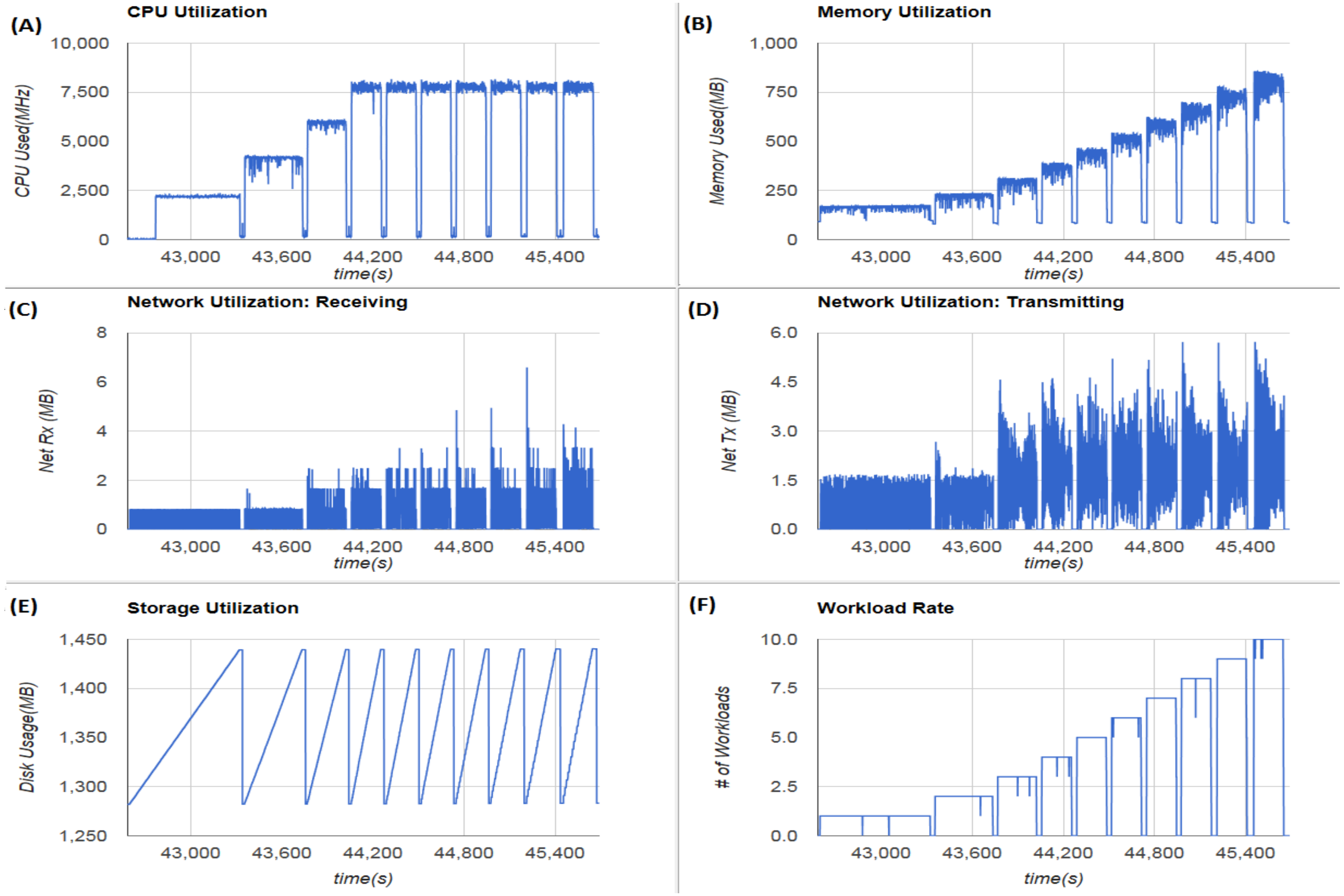


Figure D.1: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *large images* with *default filter*.

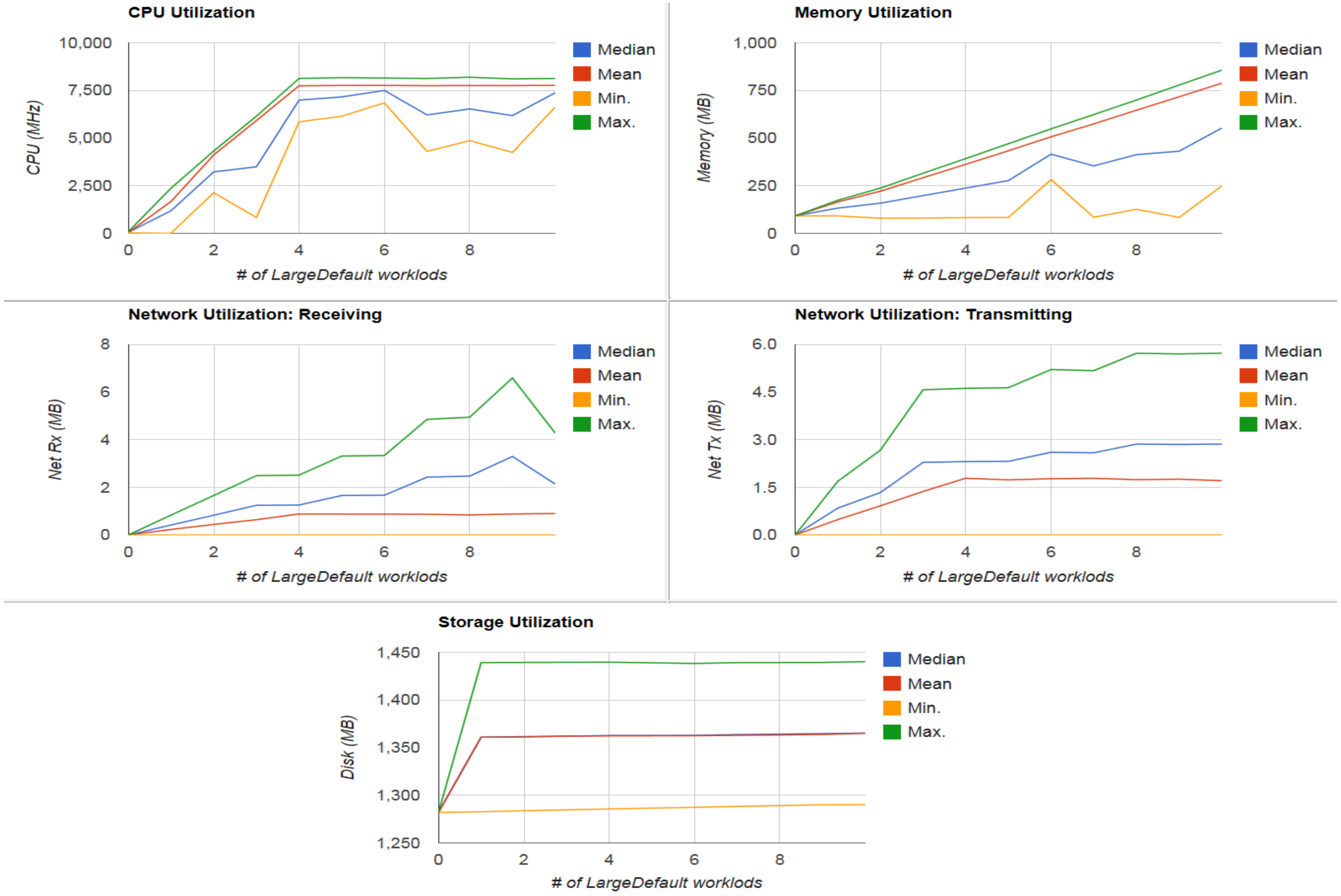


Figure D.2: Statistical analysis of resource utilisation to process *large images* with *default filter*.

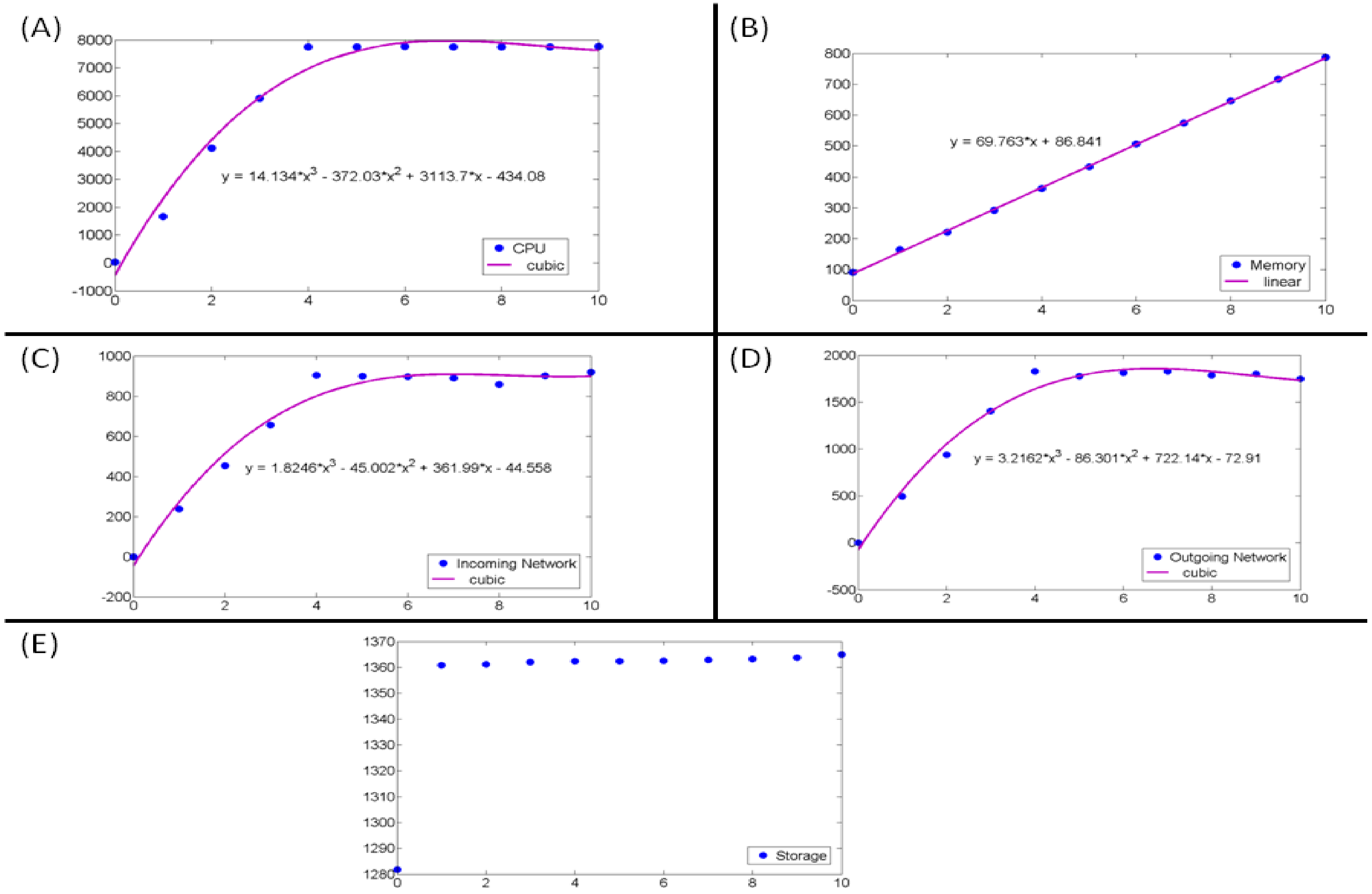


Figure D.3: Formula retrieved to process *large images* with *default filter* based on average (mean) resource usage.

ii. **Experiment with Grayscale Filter**

A set of graphs in Figure D.4 show the resource utilisation of resource metrics and the number of concurrent requests being processed based on time. Figure D.5 shows the outcome, along with statistical analyses, which were performed with model management techniques in the ReRA process and also the outcome for the workload to process large images with the grayscale filter. The average resource consumption for the resource metrics were analysed to correlate the number of workloads with resource requirement.

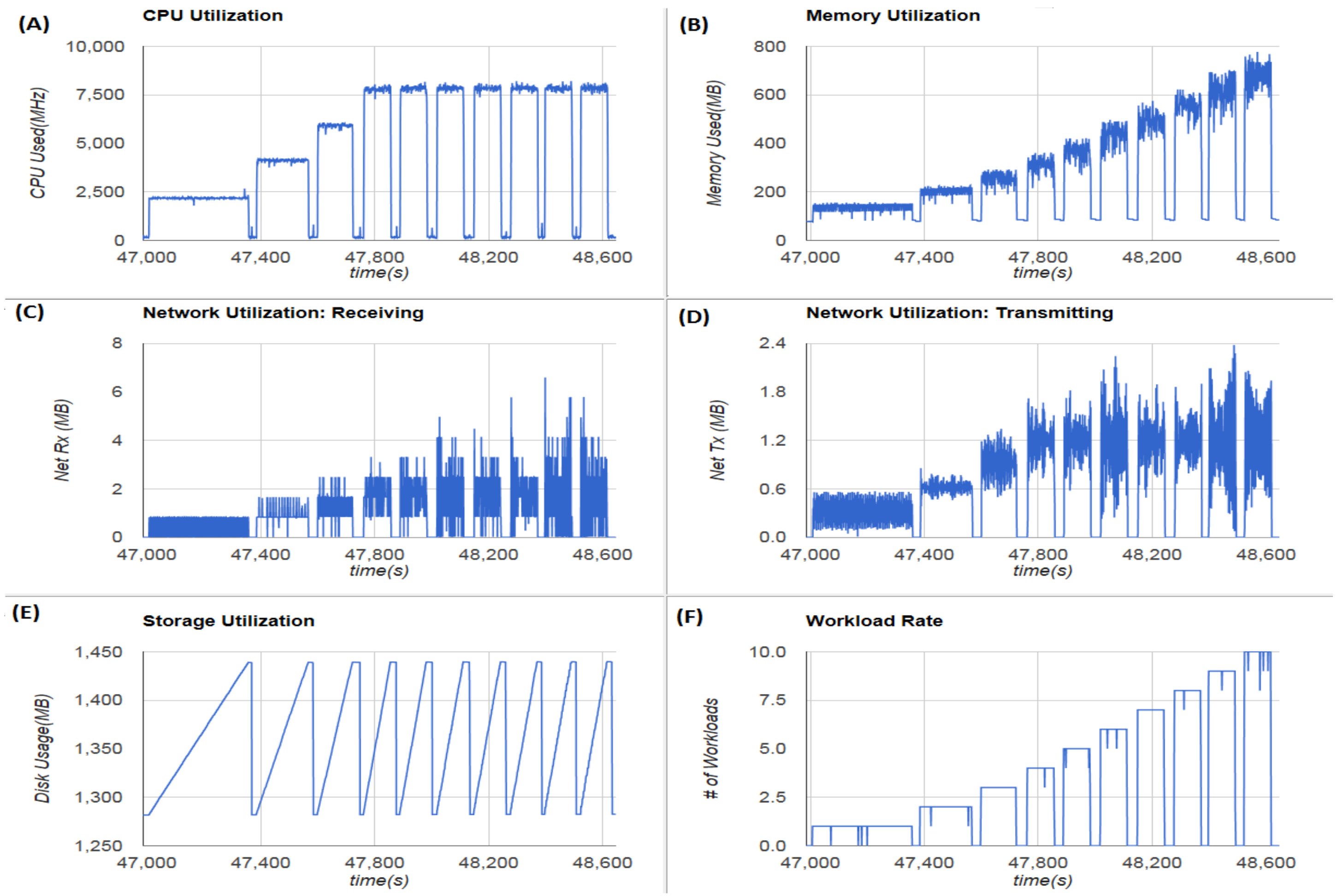


Figure D.4: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *large images* with *GrayScale filter*.

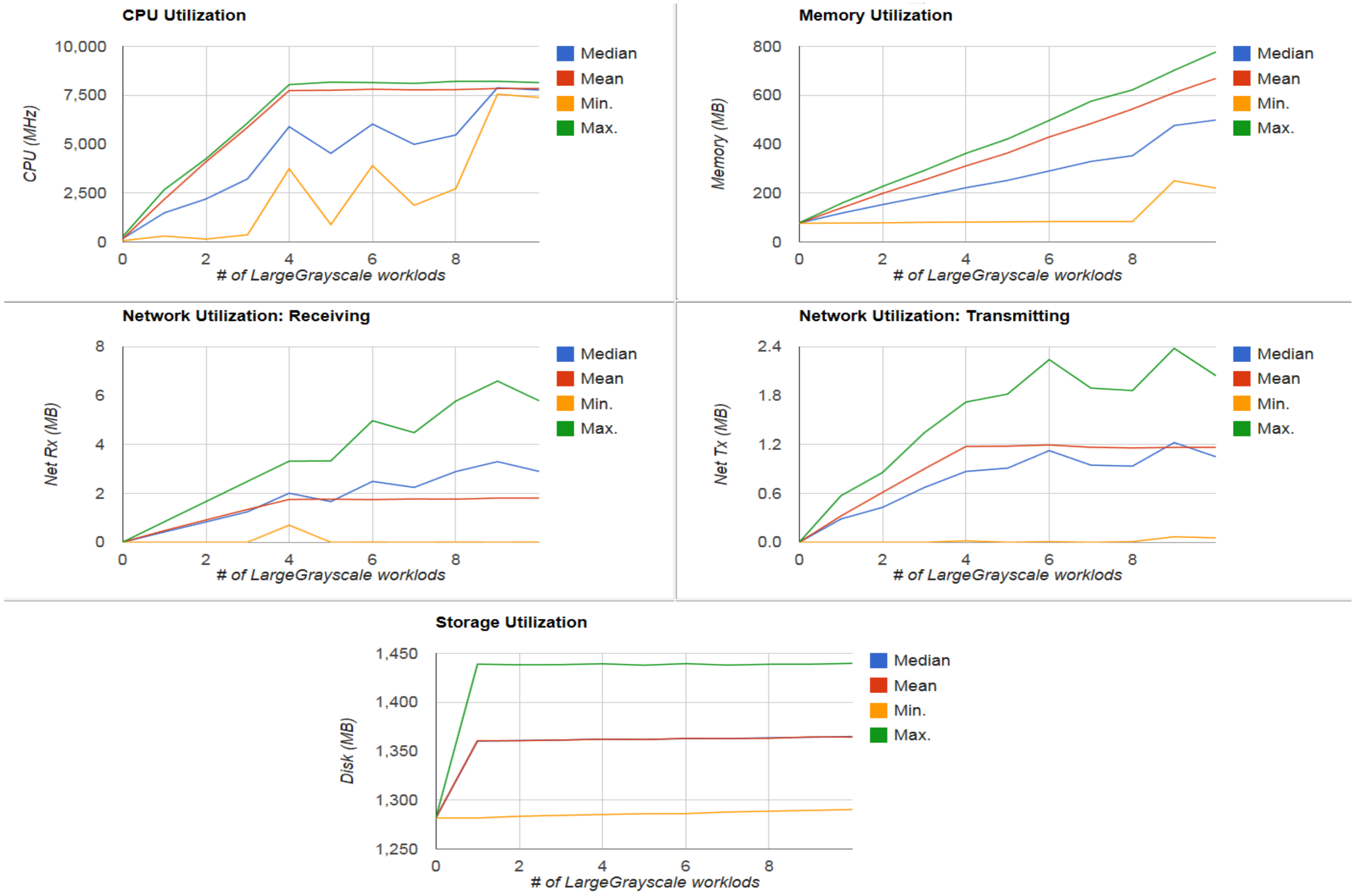


Figure D.5: Statistical analysis of resource utilisation to process *large images* with *GrayScale filter*.

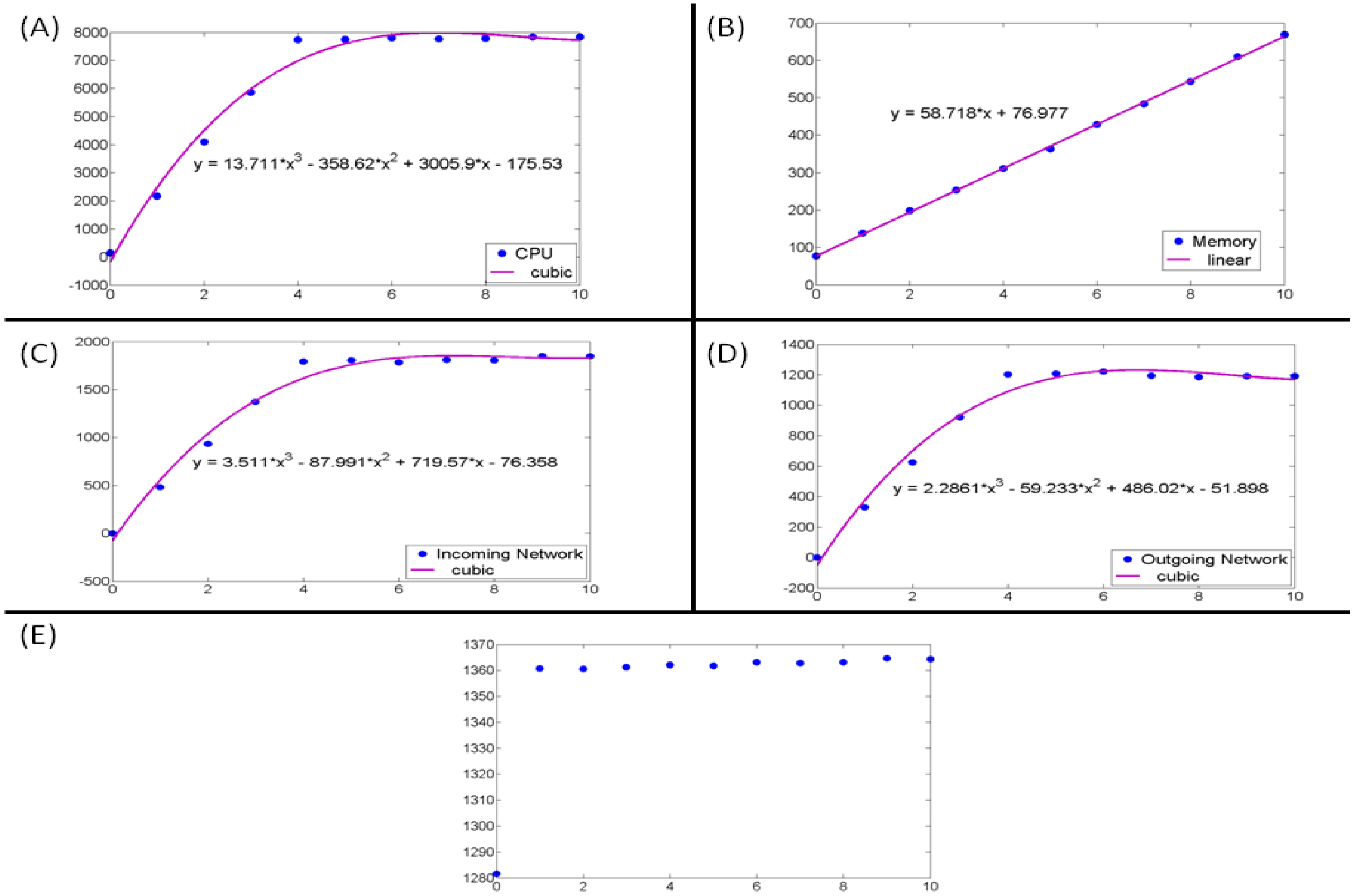


Figure D.6: Formula retrieved to process *large images* with *GrayScale filter* based on average (mean) resource usage.

iii. **Experiment with Negate Filter**
Resource utilisation of each resource metric and the number of concurrent workloads which process large images with the negate filter is presented in Figure D.7. The outcome of statistical analysis for the workload is presented in Figure D.8 and Figure D.9 shows a set of formula which correlate number of workloads with resource requirement.

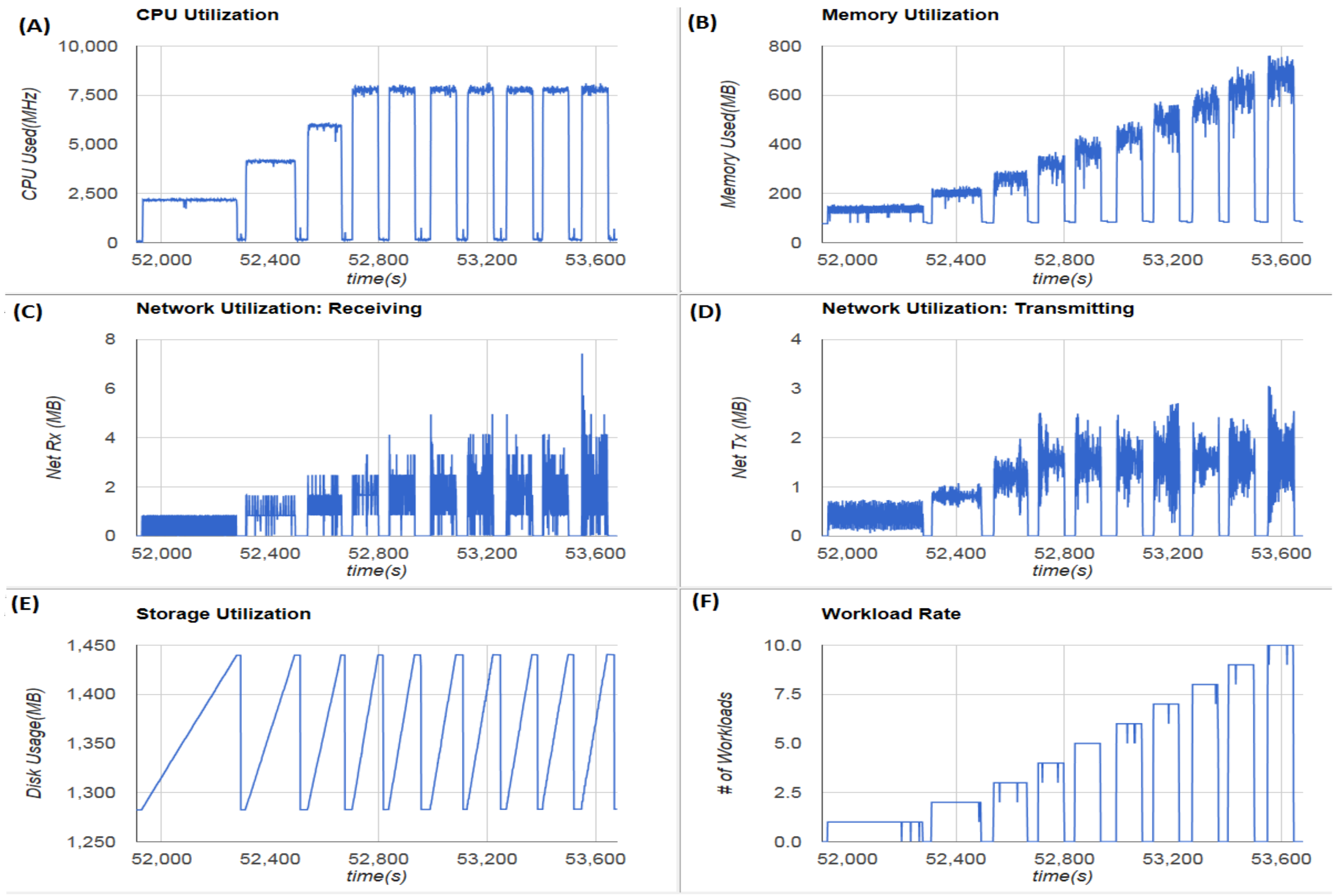


Figure D.7: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *large images* with *Negate filter*.

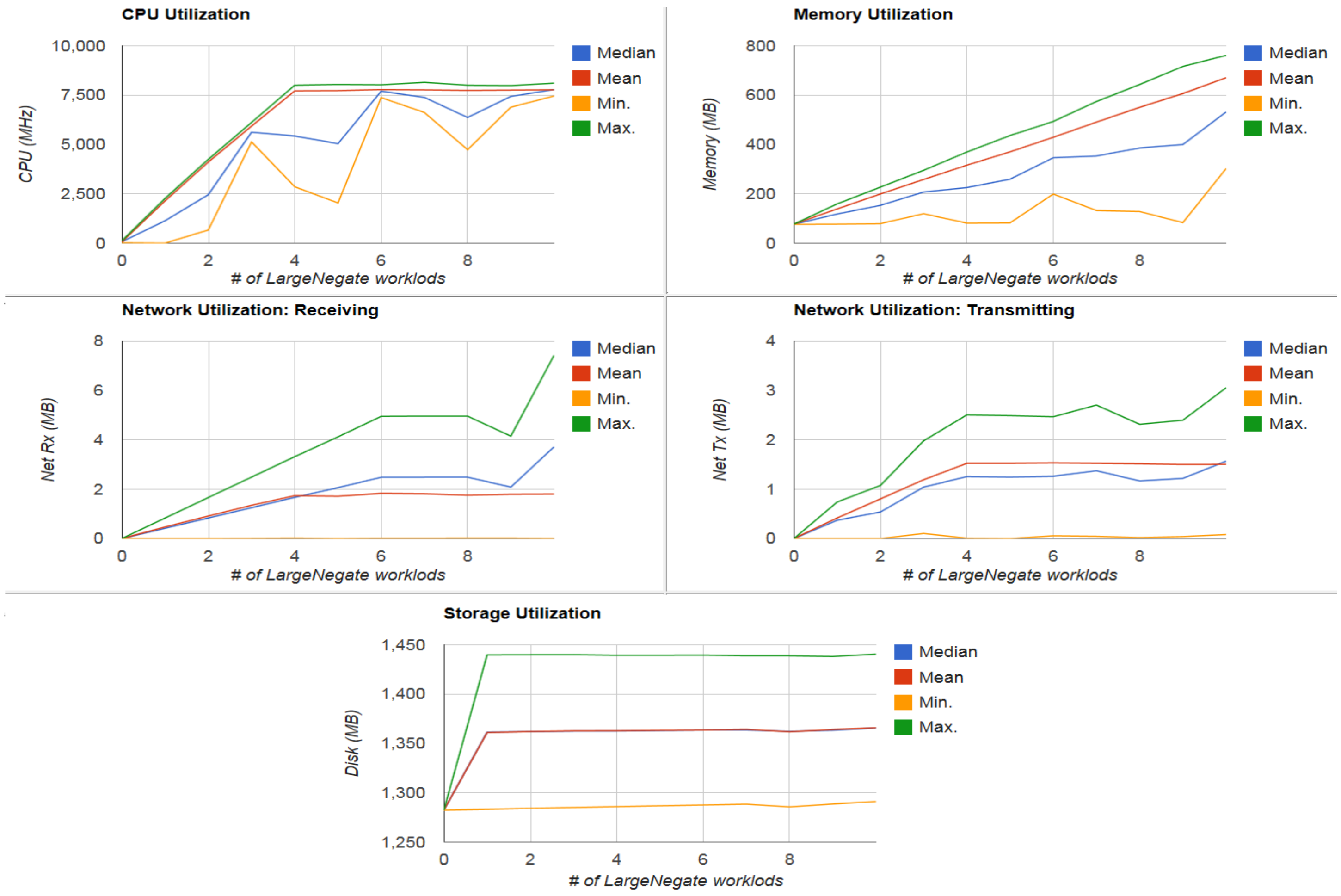


Figure D.8: Statistical analysis of resource utilisation to process *large images* with *Negate filter*.

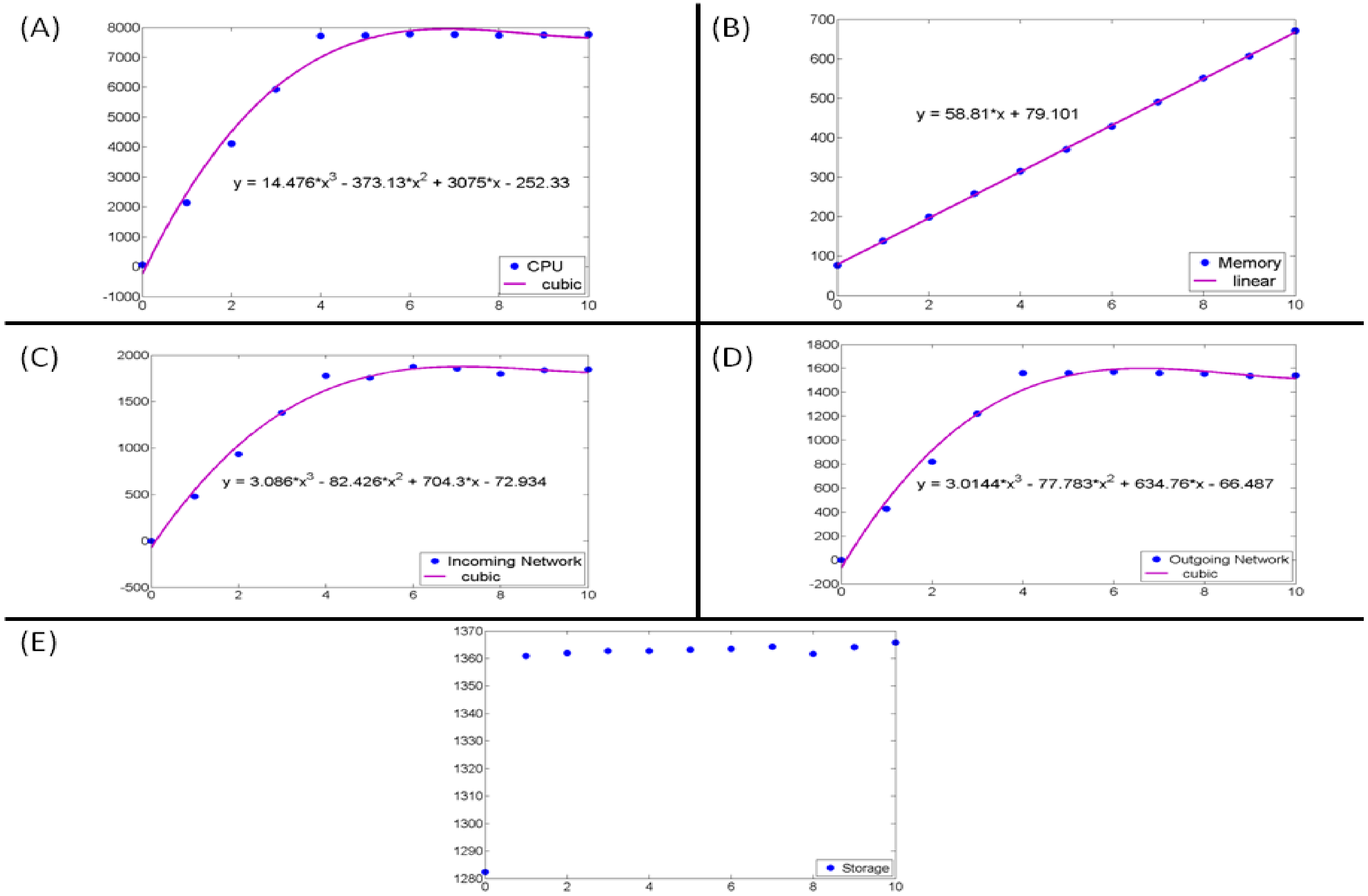


Figure D.9: Formula retrieved to process *large images* with *Negate filter* based on average (mean) resource usage.

D.3 Medium Images

Medium images in this case study were categorised with a size that was approximately half the size of the large image. The number of images were doubled, as it took shorter time to process as the large images. Therefore, 500 images with a size of 247,539 bytes were used to simulated workloads to process the images with the identified filters. The following are a graphical representation of the output of the ReRA process for the experiments conducted respectively for default, grayscale and negate image filters.

i. **Experiment with Default Filter**
   Graphs in Figure D.10 show resource utilisation of each resource metric and the number of concurrent workloads to process medium images with default filter. The outcome of statistical analysis for the workload is presented in Figure D.11 and Figure D.12 show a set of formula which correlate number of workloads with resource requirement.

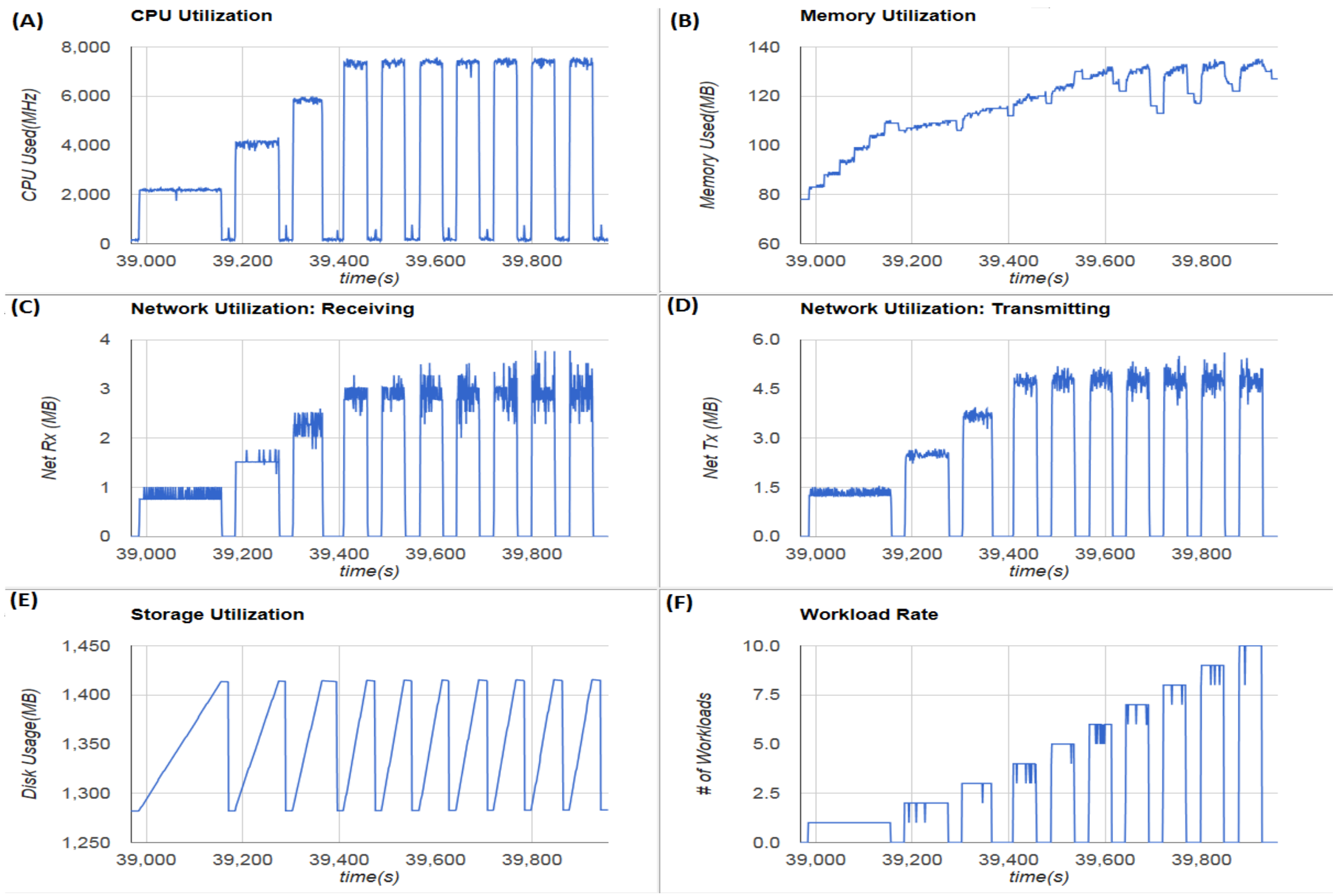


Figure D.10: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *medium images* with *default filter*.

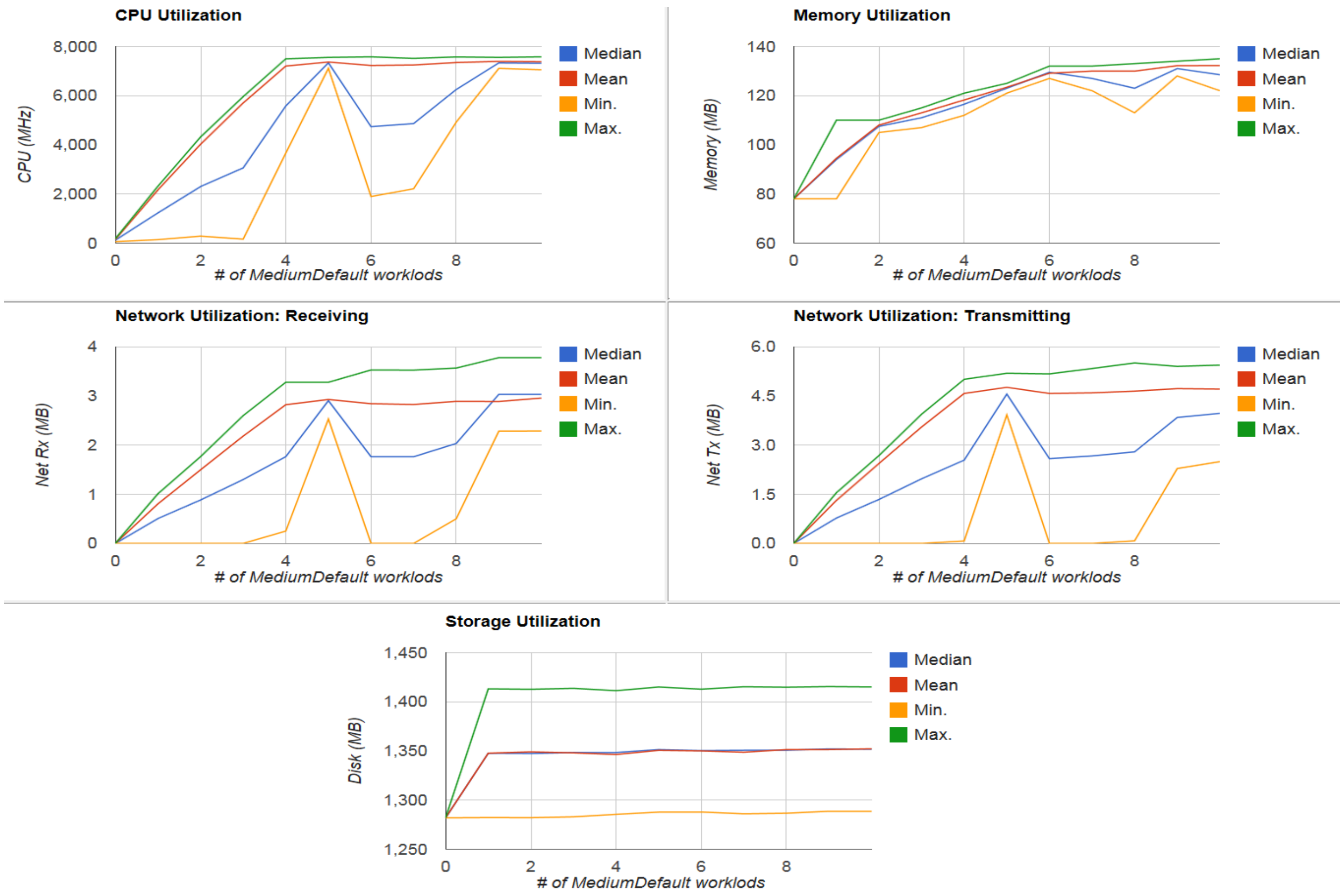


Figure D.11: Statistical analysis of resource utilisation to process *medium images* with *default filter*.

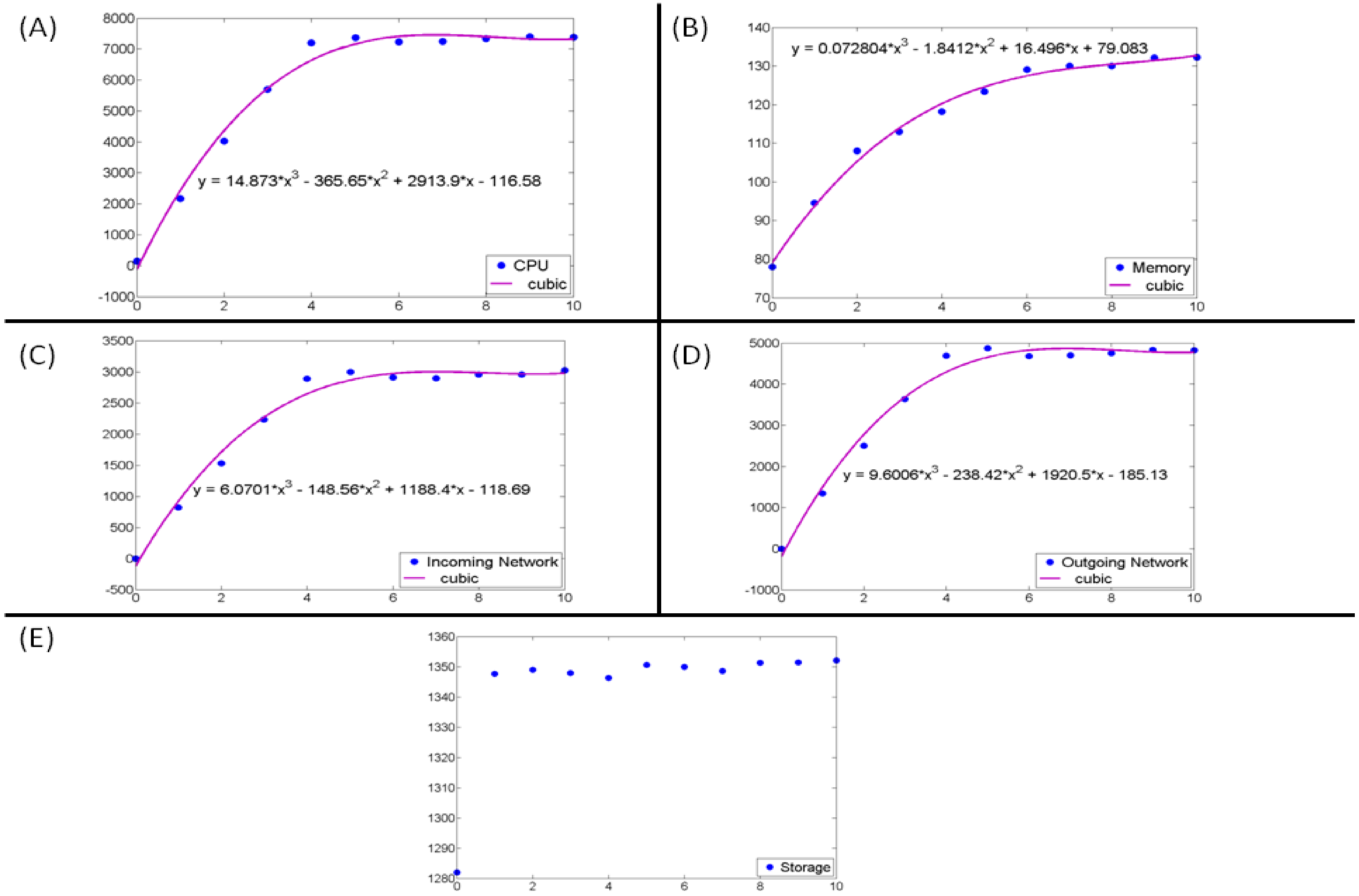


Figure D.12: Formula retrieved to process *medium images* with *default filter* based on average (mean) resource usage.

ii. **Experiment with Grayscale Filter**

Graphs in Figure D.13 show resource utilisation of each resource metric and the number of concurrent workloads to process medium images with grayscale filter. The outcome of statistical analysis for the workload is presented in Figure D.14 and Figure D.15 show a set of formula which correlate number of workloads with resource requirement.

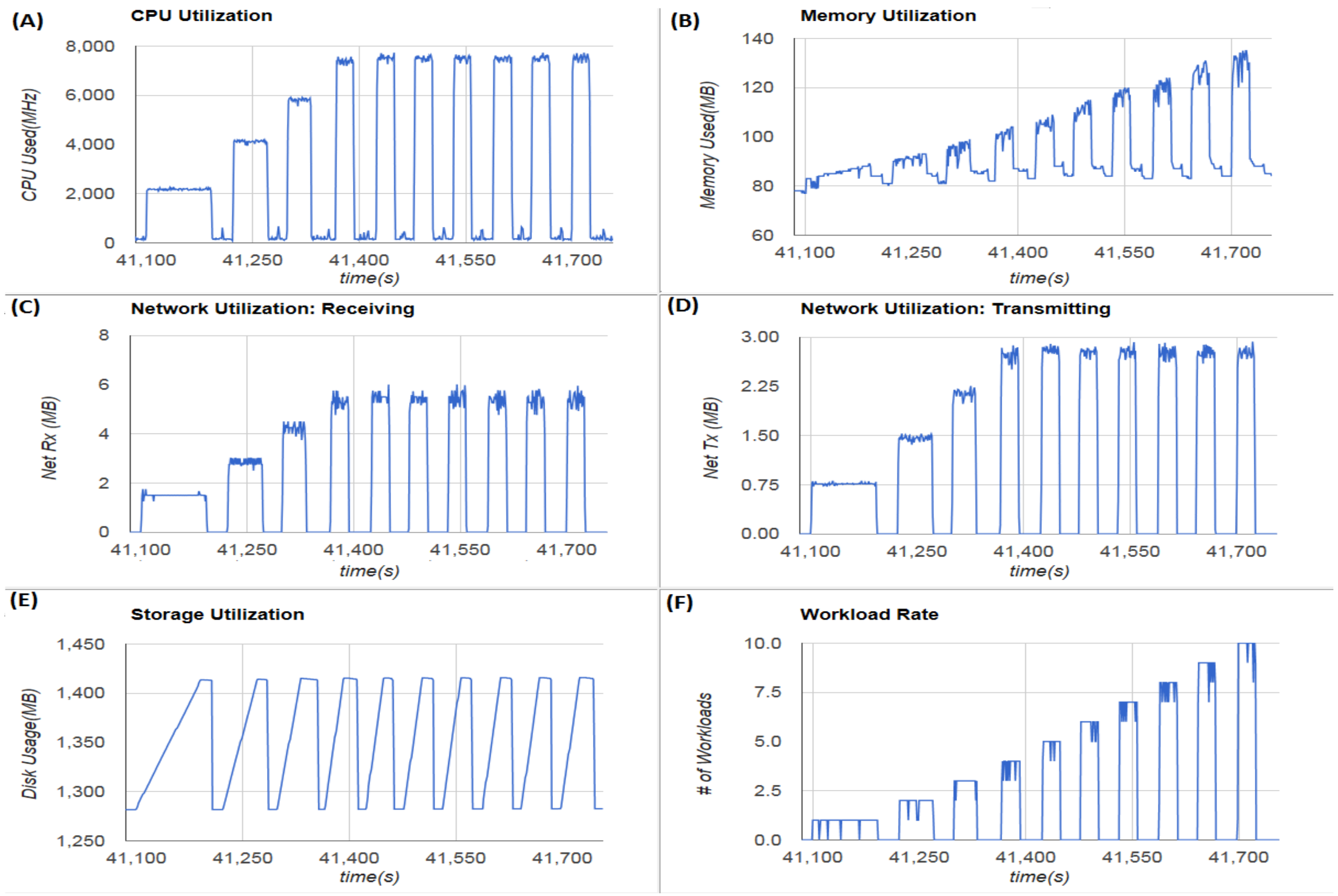


Figure D.13: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *medium images* with *Grayscale filter*.

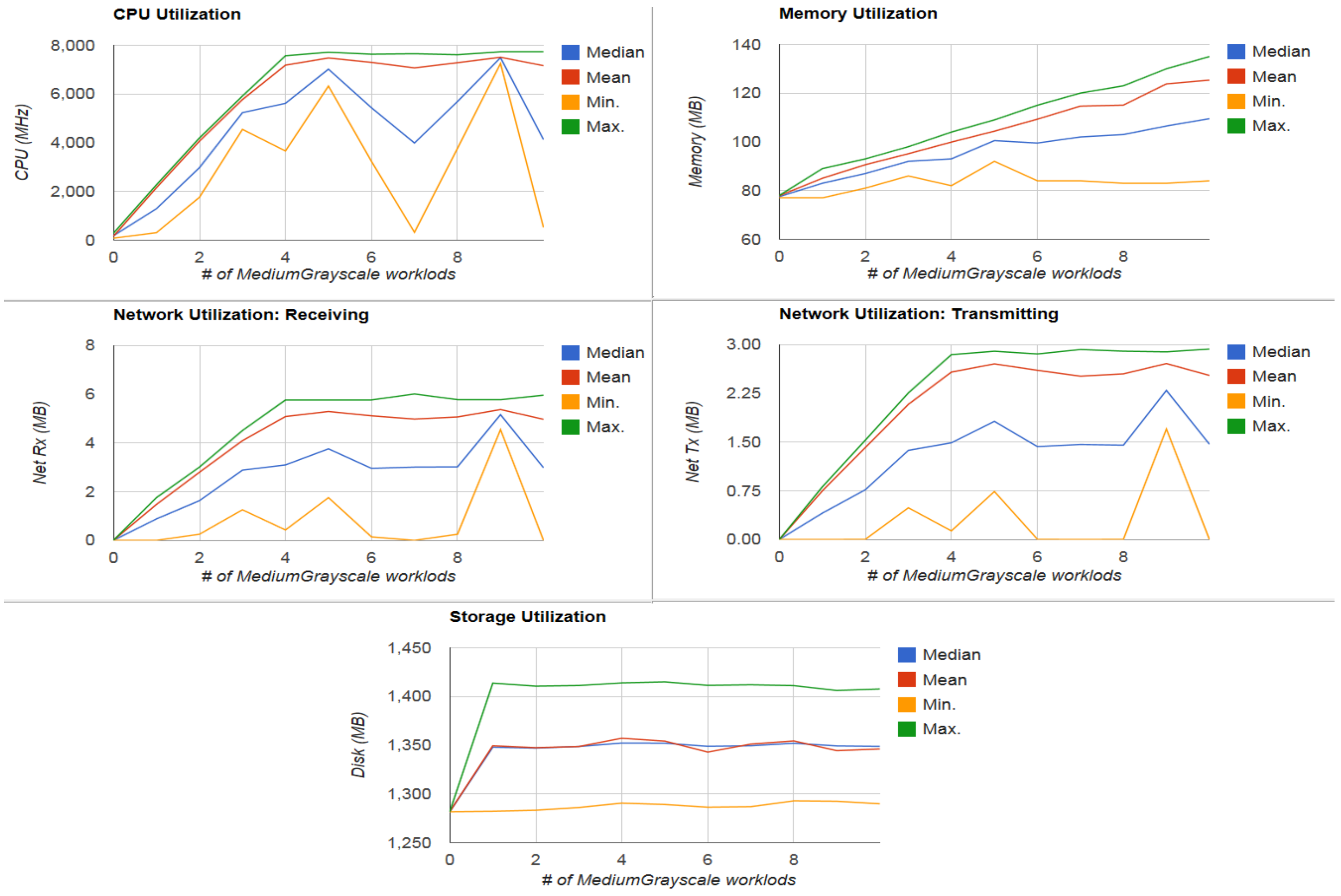


Figure D.14: Statistical analysis of resource utilisation to process *medium images* with *Grayscale filter*.

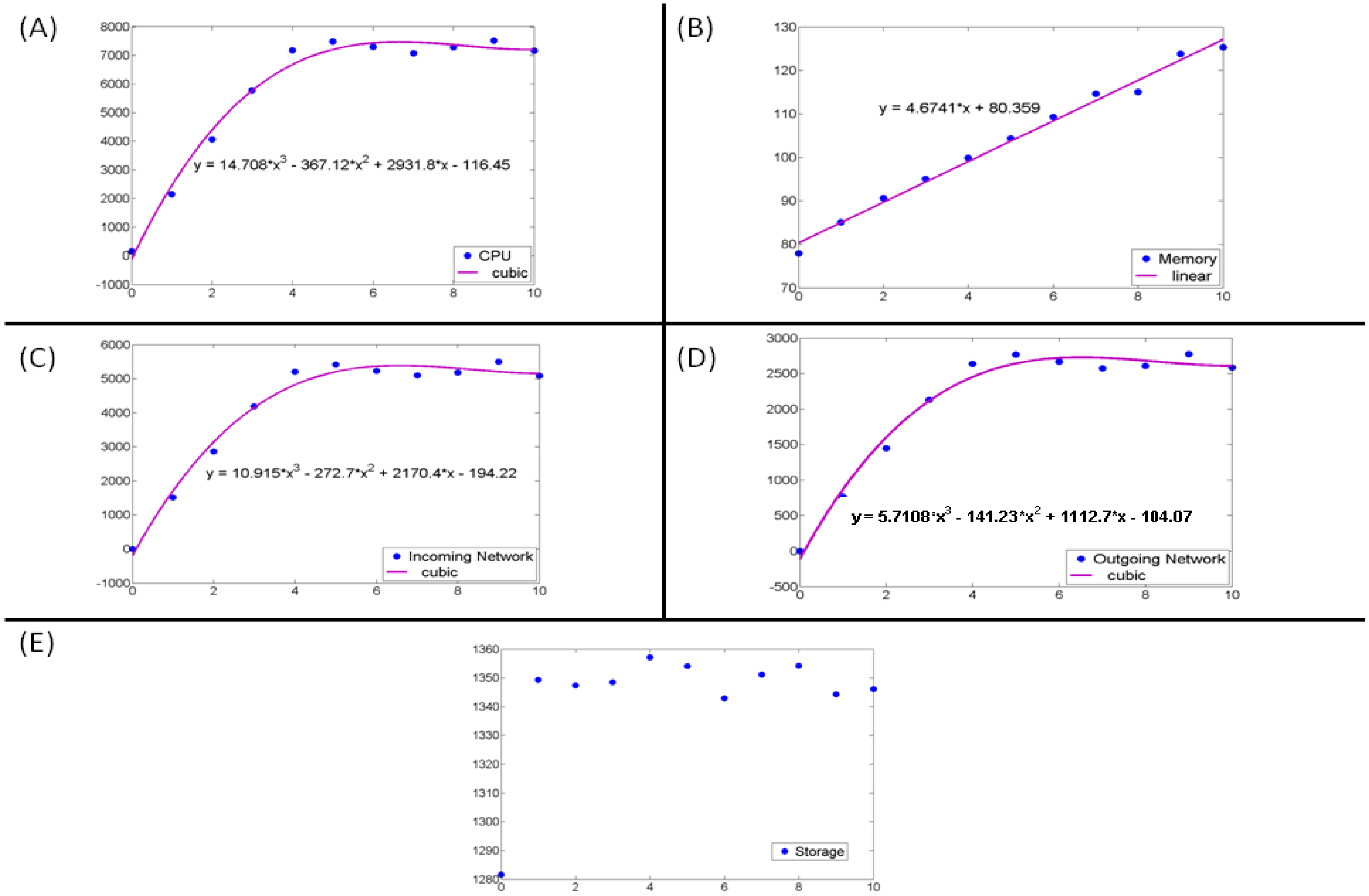


Figure D.15: Formula retrieved to process *medium images* with *Grayscale filter* based on average (mean) resource usage.

iii. **Experiment with Negate Filter**
Graphs in Figure D.16 show resource utilisation of each resource metric and the number of concurrent workloads to process medium images with negate filter. The outcome of statistical analysis for the workload is presented in Figure D.17 and Figure D.18 show a set of formula which correlate number of workloads with resource requirement.

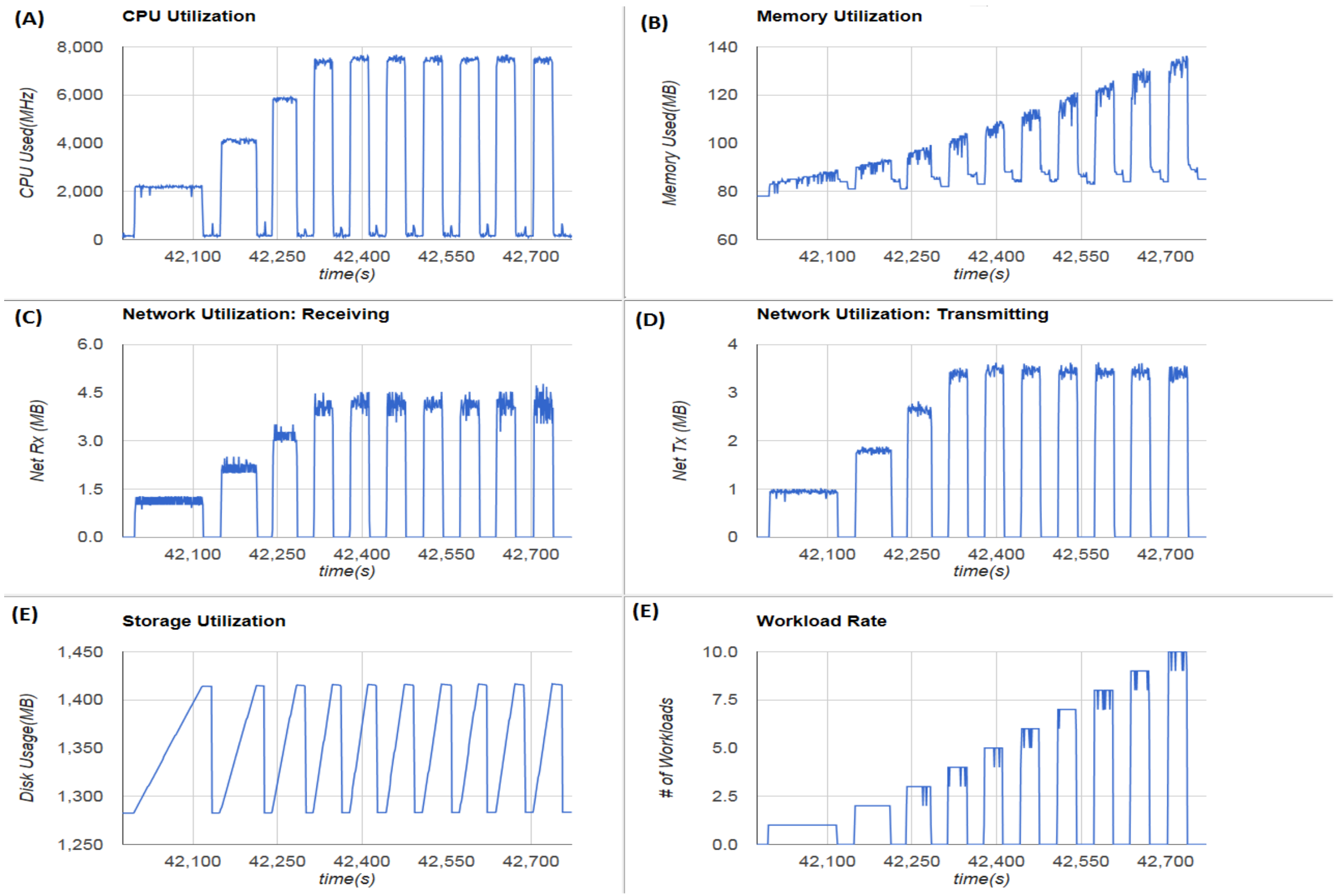


Figure D.16: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *medium images* with *Negate filter*.

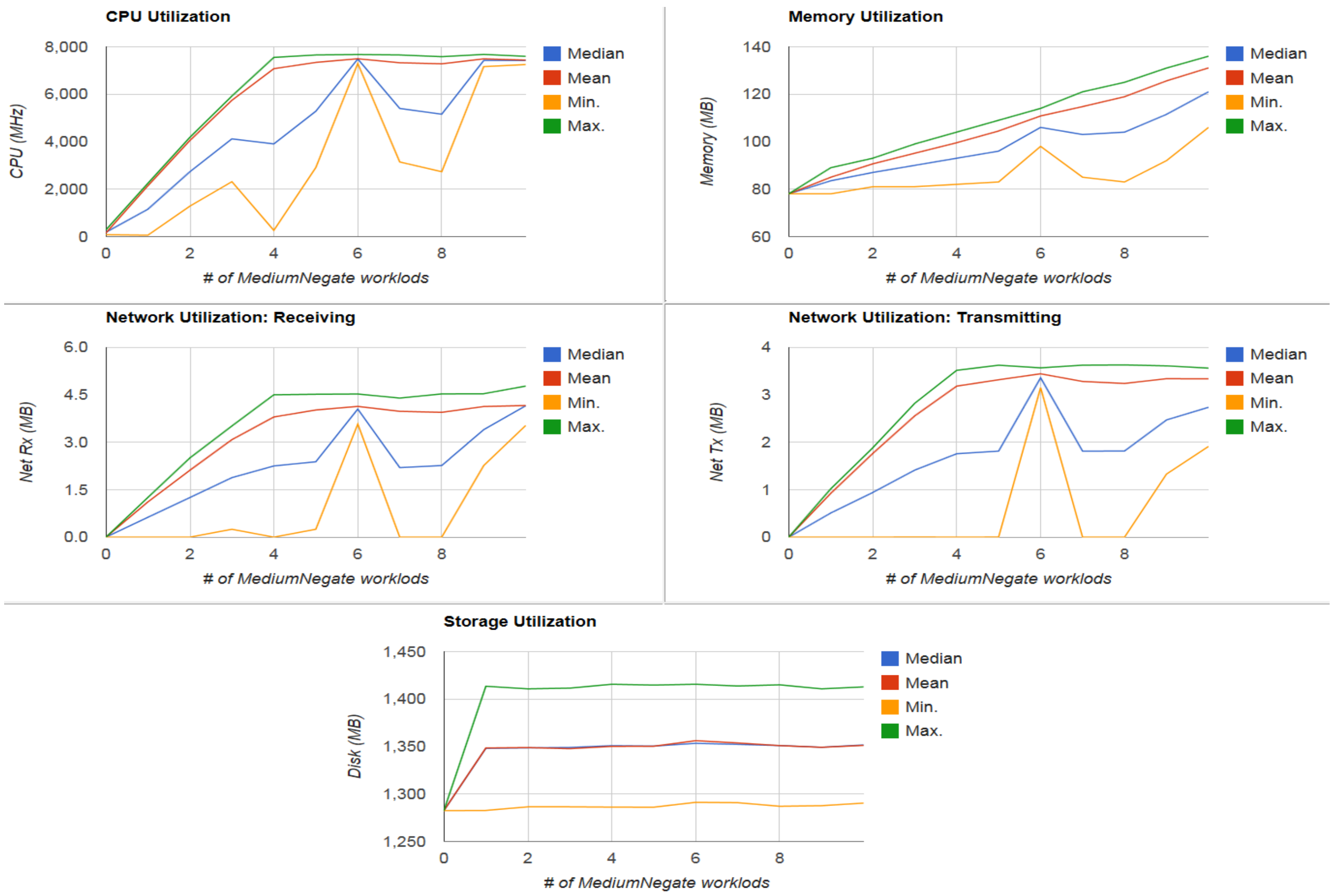


Figure D.17: Statistical analysis of resource utilisation to process *medium images* with *Negate filter*.

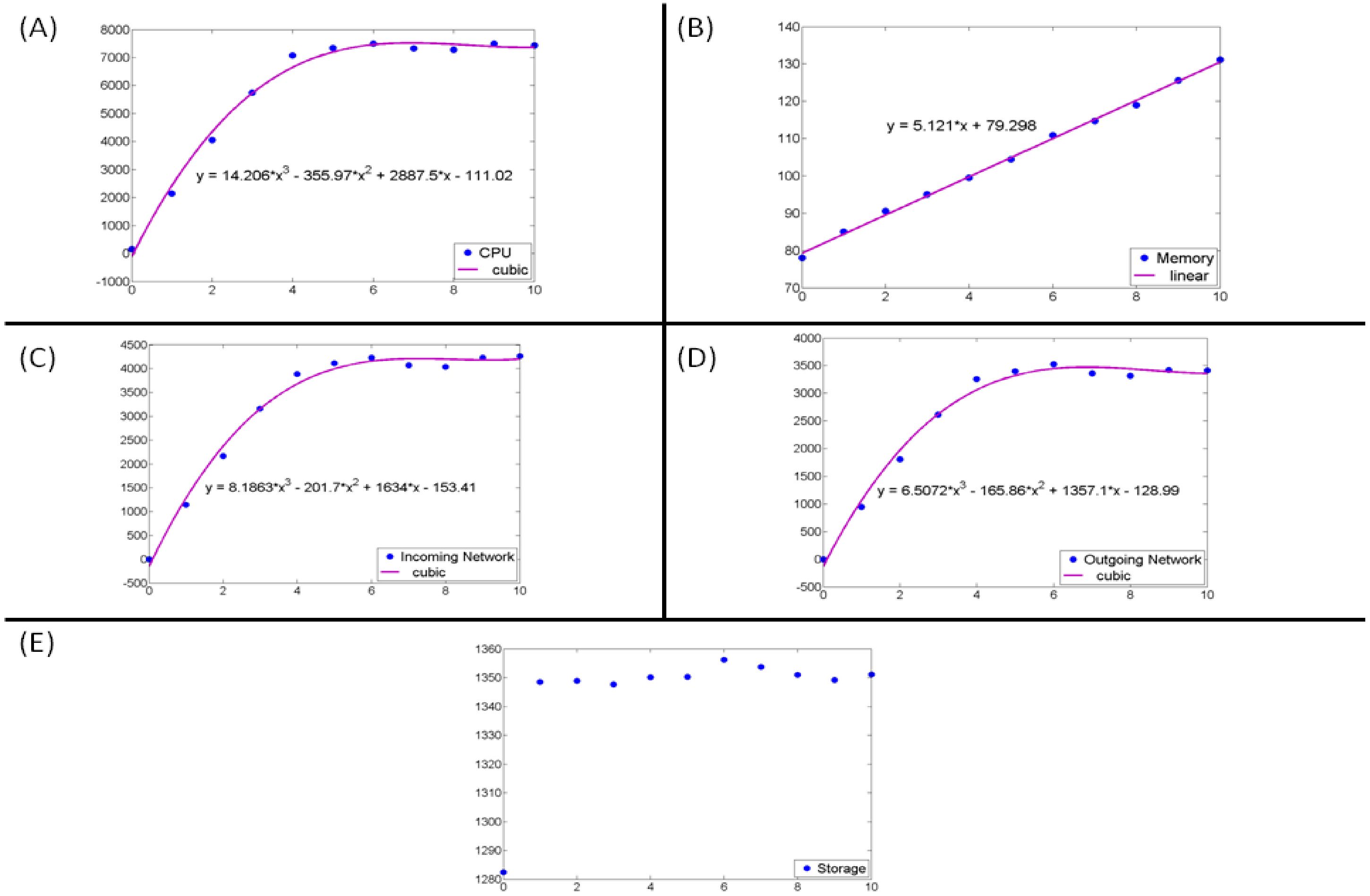


Figure D.18: Formula retrieved to process *medium images* with *Negate filter* based on average (mean) resource usage.

D.4 Small Images

In previous experiments for medium size images, the size of images was half of the large size images. In the following experiments, small size images were categorised as images which are approximately half of the size of the medium image. The total number images to be processed increased since the time needed to process these images was shorter than the medium images. Therefore, 2000 images with the size of 81,809 bytes were used to simulated workloads for the identified filters. The following are the graphical representation of the outputs of the ReRA process for the experiments conducted respectively for default, grayscale and negate image filters.

i. **Experiment with Default Filter**
The resource utilisation of the resource metrics for the simulated workload of small images with the default filter is presented in Figure D.19. Basic statistical analyses were performed and the outcome for the workload to process small images with the default filter is presented in Figure D.20. The average resource consumption for the resource metrics were further analysed to get the correlation between the number of workloads with resource requirement. Figure D.21 show the graphs and the formulas which correlate the number of workloads with the resource metrics.

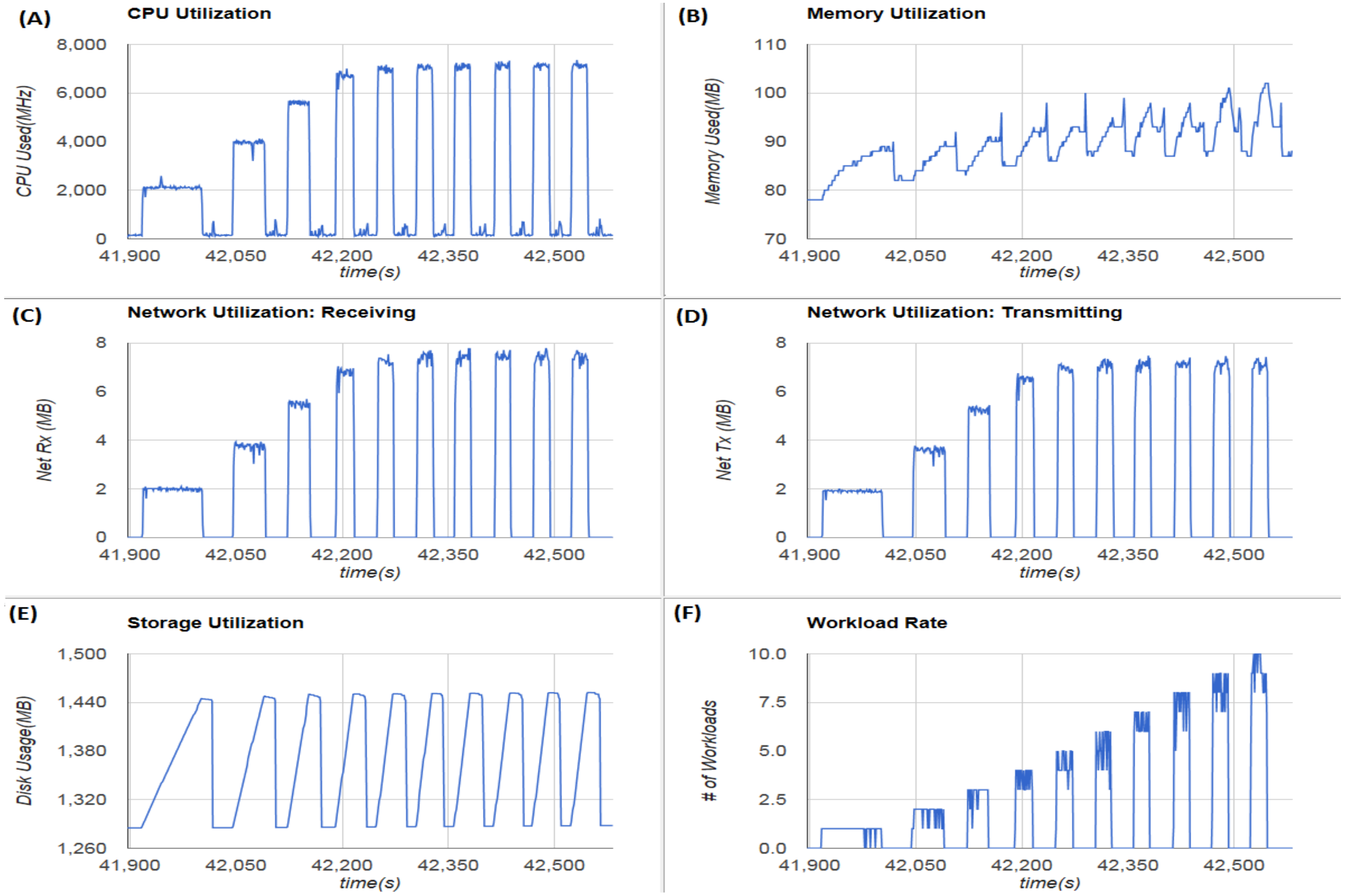


Figure D.19: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *small images* with *default filter*.

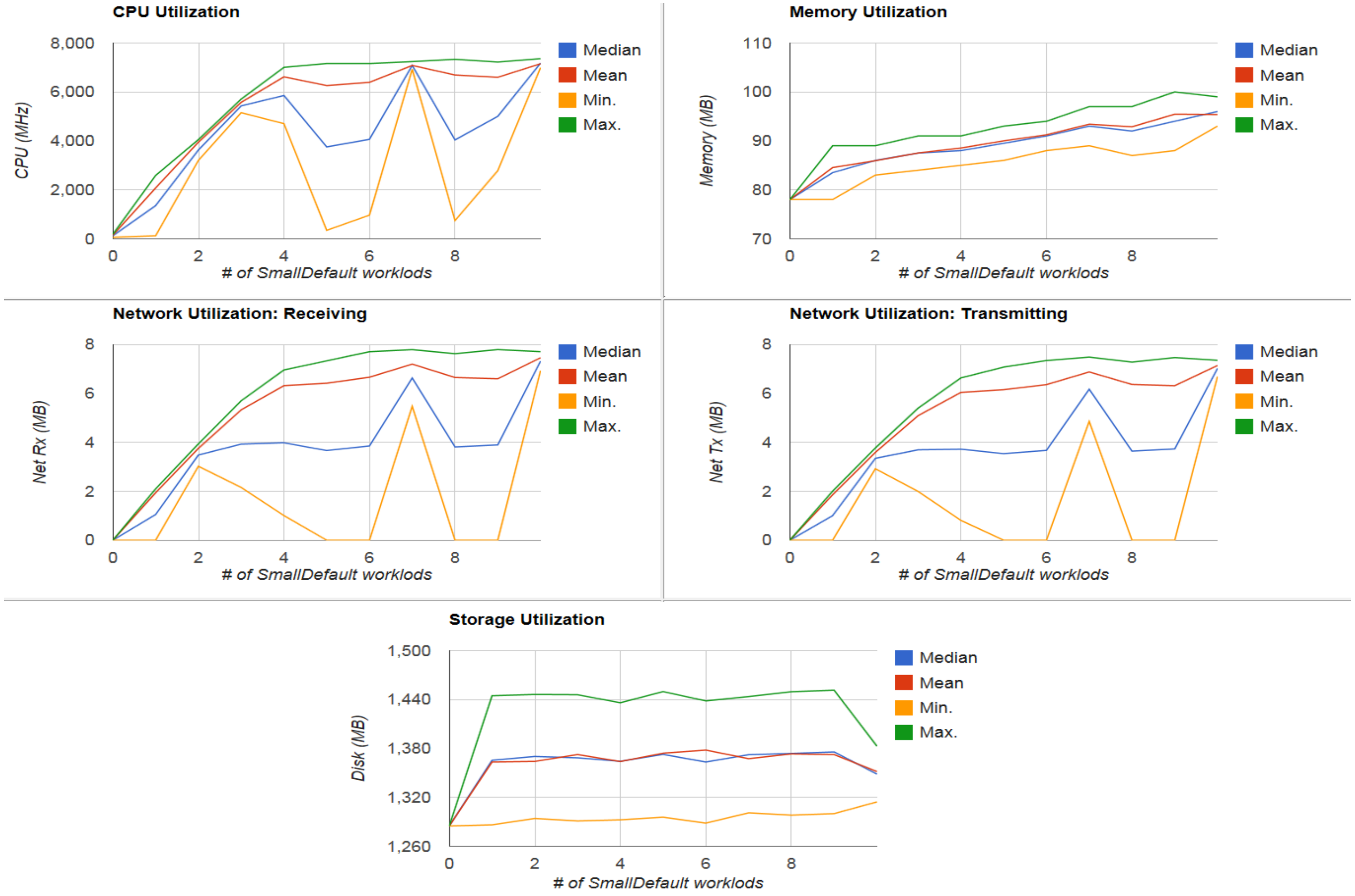


Figure D.20: Statistical analysis of resource utilisation to process *small images* with *default filter*.

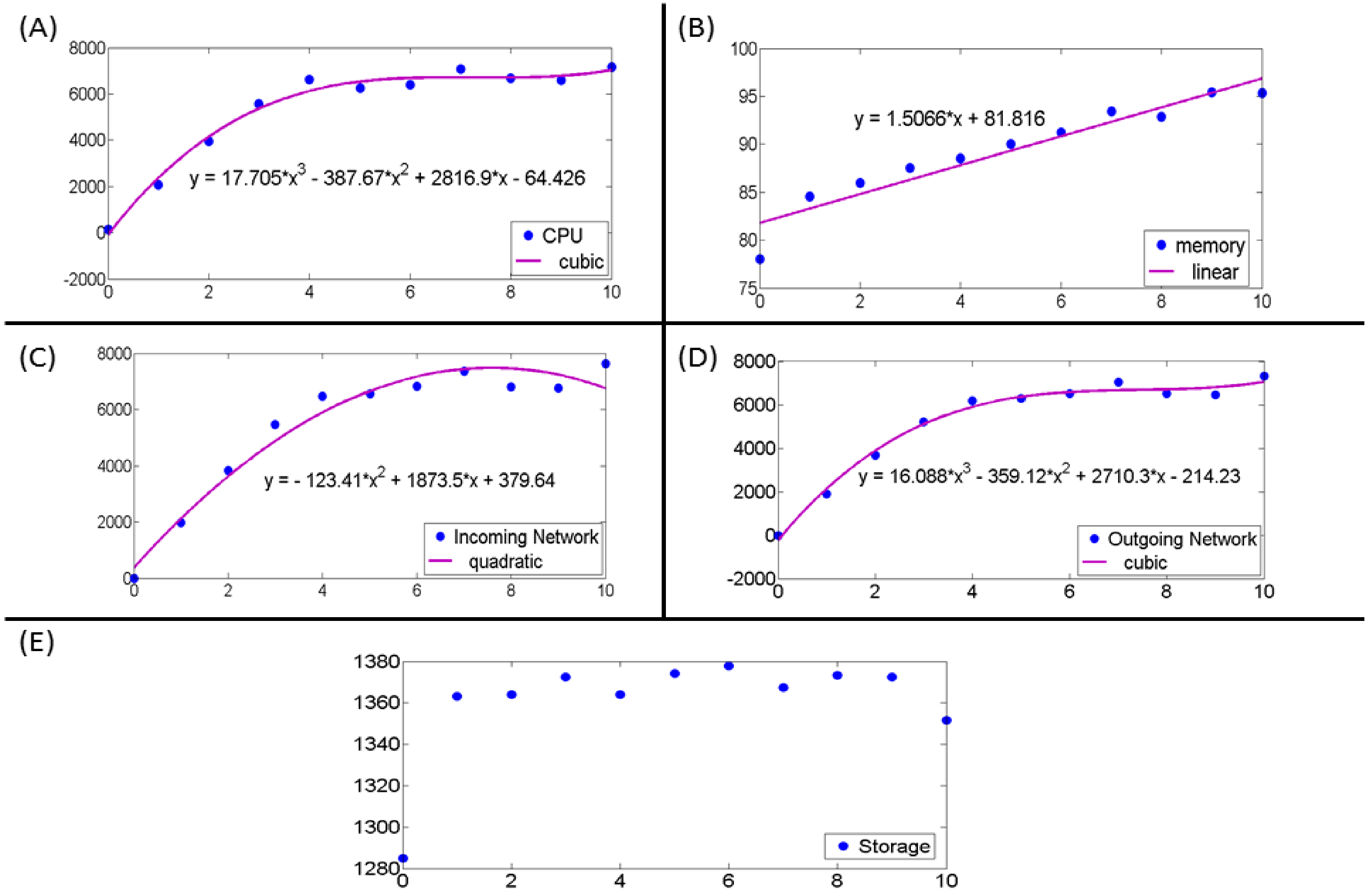


Figure D.21: Formula retrieved to process *small images* with *default filter* based on maximum resource usage.

ii. **Experiment with Grayscale Filter**

Graphs in Figure D.22 show resource utilisation of the resource metrics and number of concurrent workloads to process small images with the grayscale filter. The outcome of statistical analysis for the workload is presented in Figure D.23 and Figure D.24 show a set of formula which correlate the number of workloads with resource requirement of resource metrics.

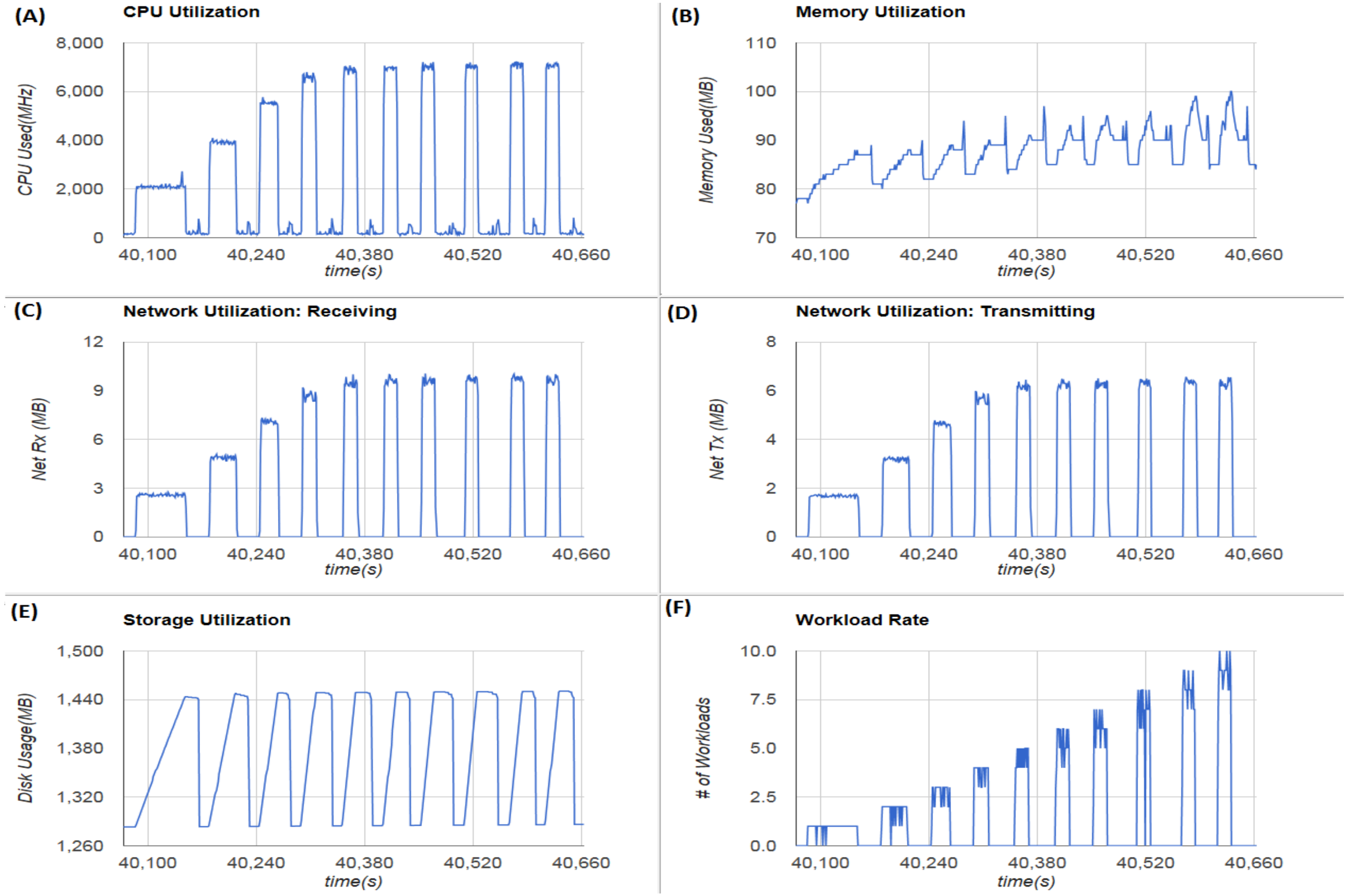


Figure D.22: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *small images* with *Grayscale filter*.

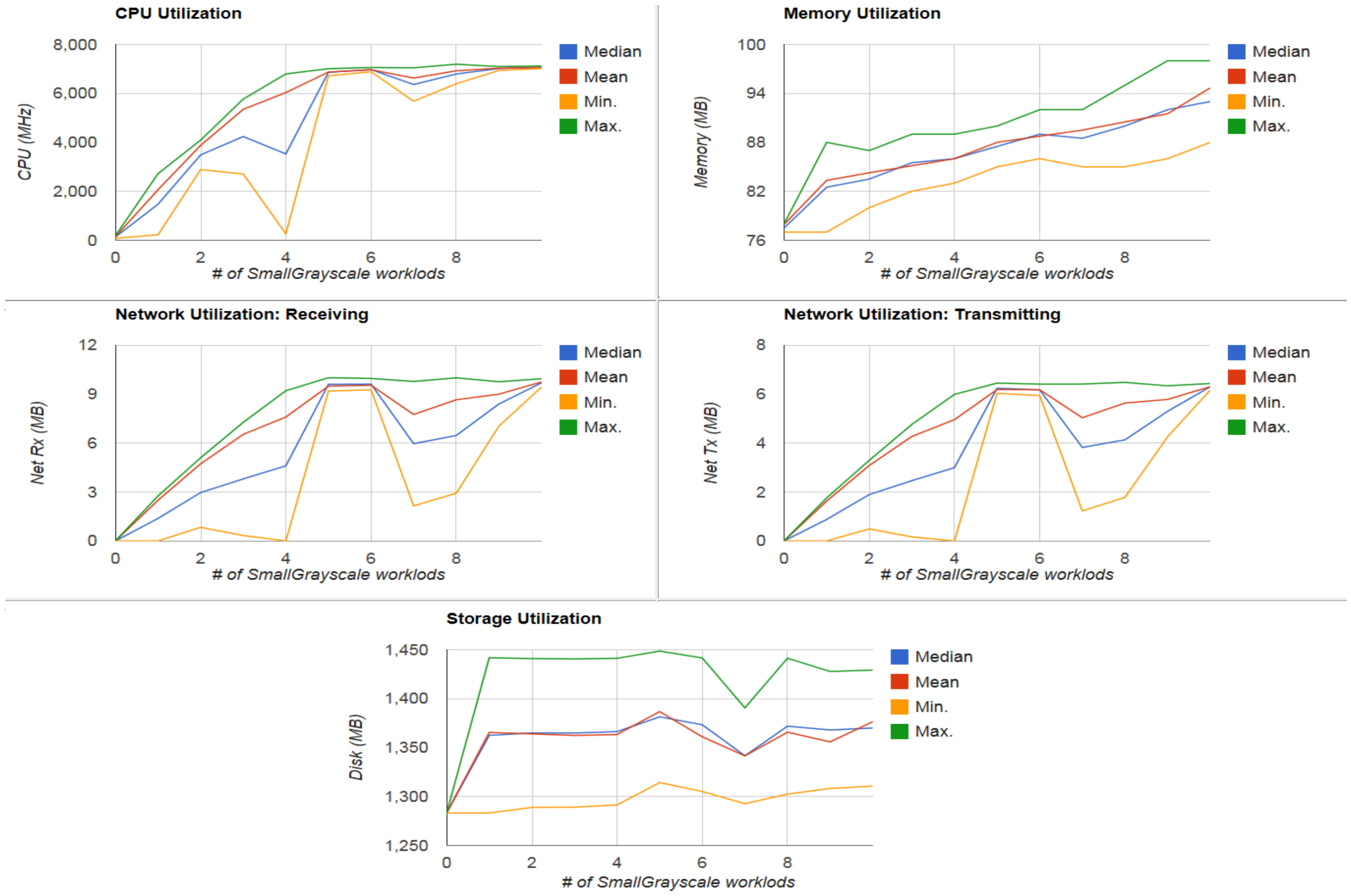


Figure D.23: Statistical analysis of resource utilisation to process *small images* with *Grayscale filter*.

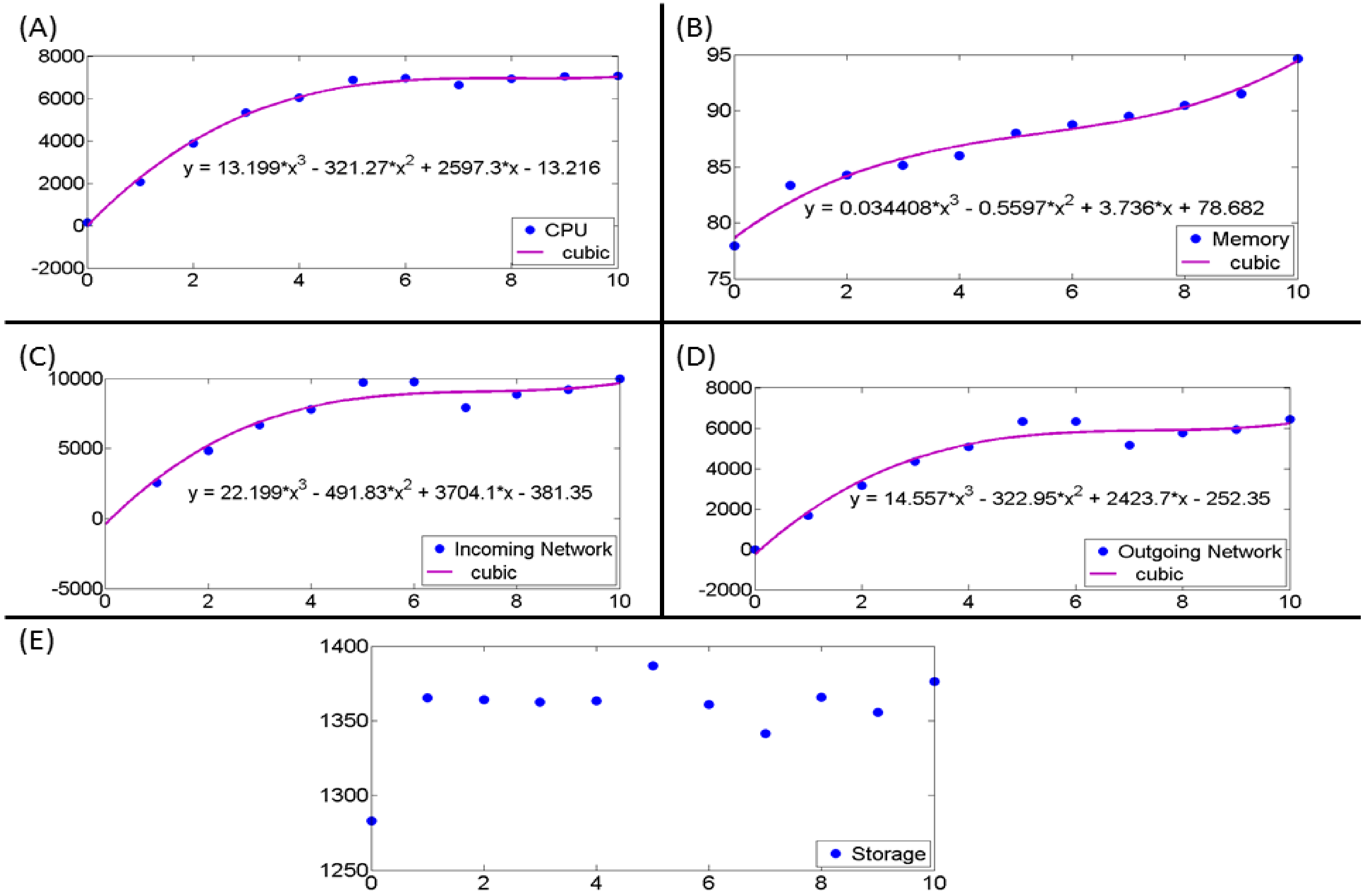


Figure D.24: Formula retrieved to process *small images* with *Grayscale filter* based on maximum resource usage.

iii. **Experiment with Negate Filter**

The graphs in Figure D.25 show the resource utilisation of the resource metrics and number of concurrent workloads being processed based on time. Figure D.26 shows the outcome, including statistical analyses which were performed with model management techniques in the ReRA process for the workload to process small images with negate filter. The average resource consumption for the resource metrics were analysed to correlate number of workloads with resource requirement. The relation is presented in Figure D.27 for the resource metrics.

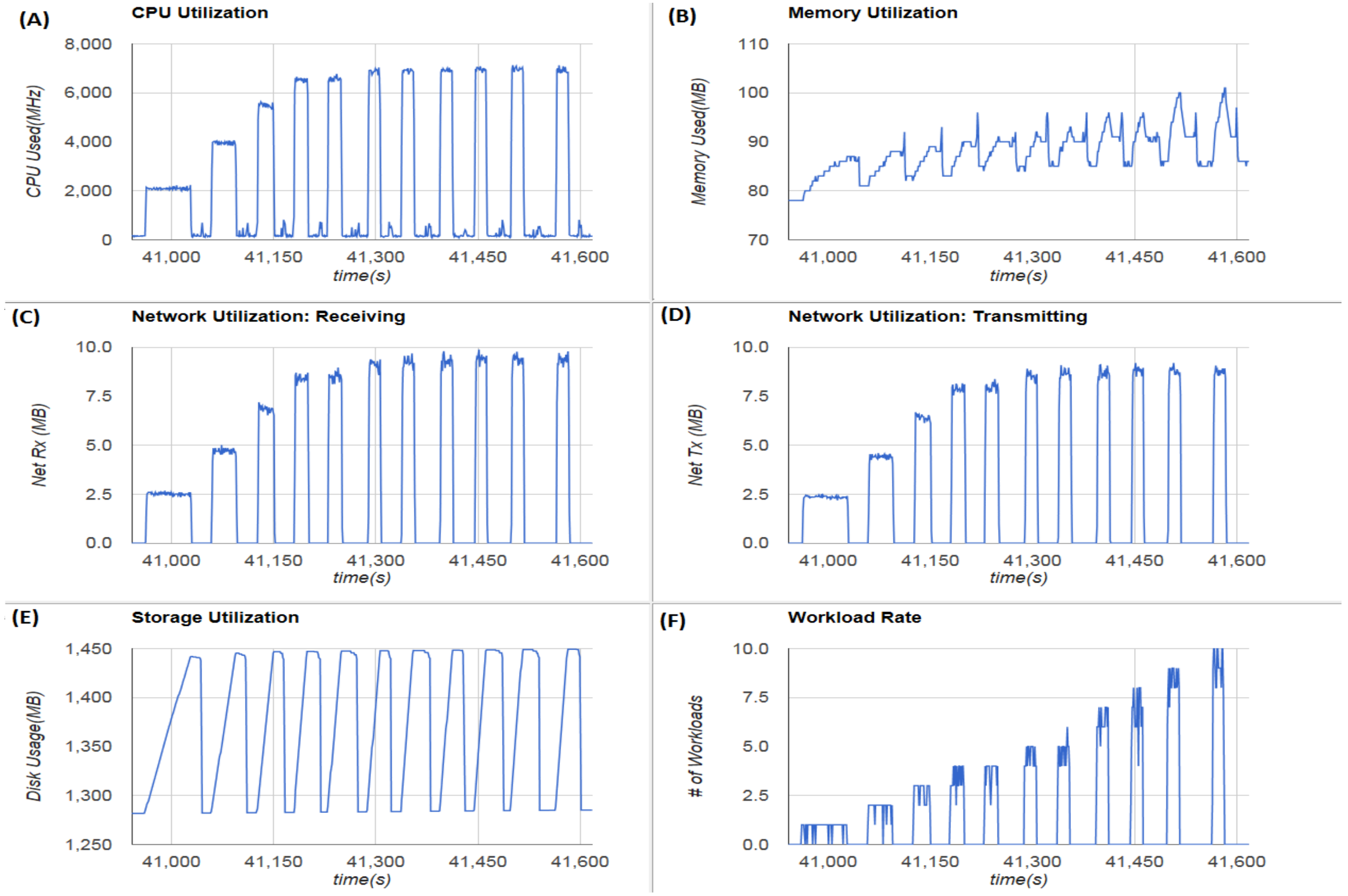


Figure D.25: Resource utilisation graphs (A-E) and simulated workload pattern(F) for *small images* with *Negate filter*.

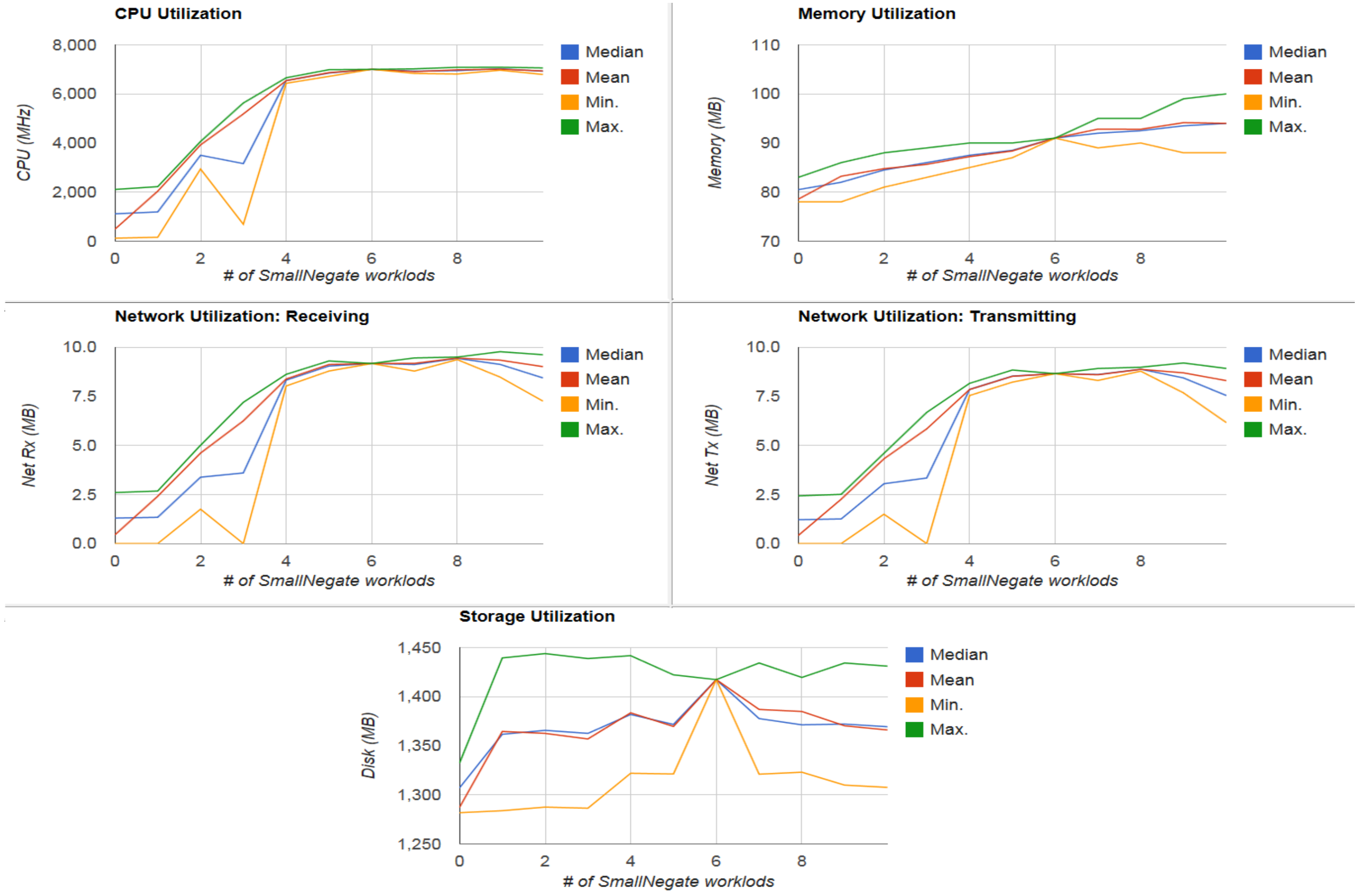


Figure D.26: Statistical analysis of resource utilisation to process *small images* with *Negate filter*.

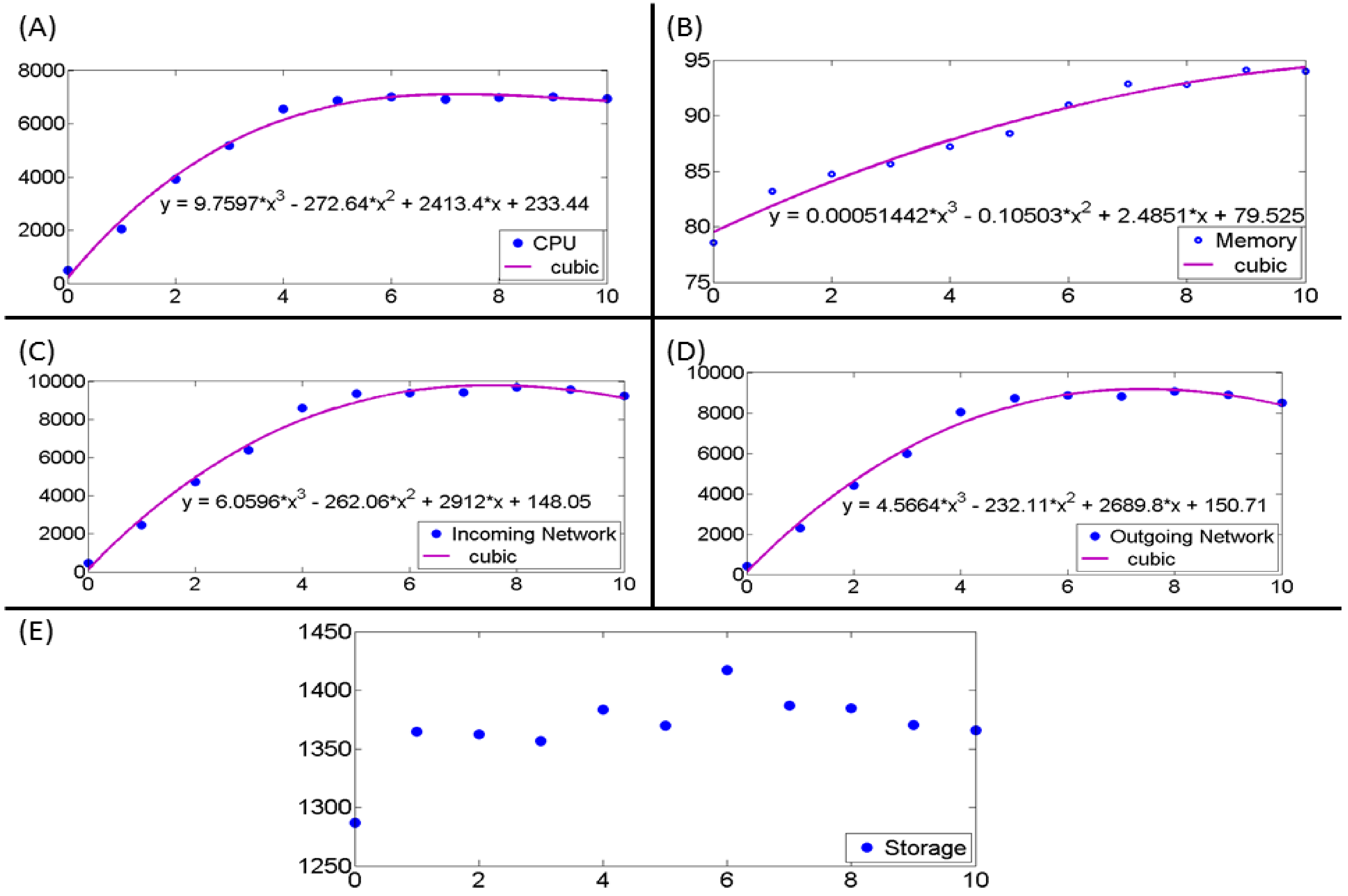


Figure D.27: Formula retrieved to process *small images* with *Negate filter* based on average (mean) resource usage.

Appendix E

Sample of Log Recordings to Log Models

Listing E.1: Sample of request log recording in access.log file in Apache web server

```
192.168.56.1 - - [18/Jun/2014:11:49:58 +0100] 025764 3755927 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827717 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:01 +0100] 810923 3691593 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:05 +0100] 538371 3552740 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:09 +0100] 105526 3579073 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:12 +0100] 694718 3717118 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:16 +0100] 422656 3713344 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:20 +0100] 146139 3694977 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:23 +0100] 851114 3692522 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:27 +0100] 553377 3670565 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:31 +0100] 234672 3561506 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:34 +0100] 807302 3560109 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:38 +0100] 377371 3573324 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:41 +0100] 960344 3574463 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:45 +0100] 545633 3564690 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
192.168.56.1 - - [18/Jun/2014:11:50:49 +0100] 120301 3582029 "
    POST /imagefilter/index1.php HTTP/1.1" 200 1827716 "-" "
    Apache-HttpClient/4.2.3 (java 1.5)"
```

Listing E.2: Example of the *RequestLog* model in XML form generated based on request log in Listing E.1

```
<?xml version=”1.0” encoding=”ASCII”?>
<WebServer xmi:version=”2.0” xmlns:xmi=”http://www.omg.org/XMI
     ” xmlns:xsi=”http://www.w3.org/2001/XMLSchema–instance”
     xmlns=”RequestLog”>
  <machine xsi:type=”Machine” name=”ubuntu05”/>
  <config xsi:type=”Configuration” maxUser=”150” maxLiveUser
      =”100” waitingTime=”300.0” timeOut=”5.0”/>
  <logRecords xsi:type=”LogRecord” startTime=”42598.025764”
      endTime=”42601.781691” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827717”/>
  <logRecords xsi:type=”LogRecord” startTime=”42601.810923”
      endTime=”42605.502516” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42605.538371”
      endTime=”42609.091111” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42609.105526”
      endTime=”42612.684599” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42612.694718”
      endTime=”42616.411836” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42616.422656”
      endTime=”42620.136000000006” workloadName=”LargeDefault”
       responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42620.146139”
      endTime=”42623.841115999996” workloadName=”LargeDefault”
       responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42623.851114”
      endTime=”42627.543635999995” workloadName=”LargeDefault”
       responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42627.553377”
      endTime=”42631.223942” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42631.234672”
      endTime=”42634.796178” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42634.807302”
      endTime=”42638.367411” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42638.377371”
      endTime=”42641.950695” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42641.960344”
      endTime=”42645.534807” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42645.545633”
      endTime=”42649.110323” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
  <logRecords xsi:type=”LogRecord” startTime=”42649.120301”
      endTime=”42652.70233” workloadName=”LargeDefault”
      responseCode=”200” dataSizeReturn=”1827716”/>
</WebServer>
```

Listing E.3: Sample of VBoxMetrics resource usage log recording

```
Time stamp    Object     Metric                    Value
------------  ---------  ------------------------  ----------
10:49:57.454  ubuntu05   CPU/Load/User             0.50%
10:49:57.454  ubuntu05   CPU/Load/Kernel           0.00%
10:49:57.454  ubuntu05   RAM/Usage/Used            1321748 kB
10:49:57.454  ubuntu05   Disk/Usage/Used           2959 mB
10:49:57.454  ubuntu05   Net/Rate/Rx               0 B/s
10:49:57.454  ubuntu05   Net/Rate/Tx               0 B/s
10:49:57.454  ubuntu05   Guest/CPU/Load/User       0.00%
10:49:57.454  ubuntu05   Guest/CPU/Load/Kernel 2.00%
10:49:57.454  ubuntu05   Guest/CPU/Load/Idle   98.00%
10:49:57.454  ubuntu05   Guest/RAM/Usage/Total 2050900 kB
10:49:57.454  ubuntu05   Guest/RAM/Usage/Free 1957564 kB
10:49:57.454  ubuntu05   Guest/RAM/Usage/Balloon 0 kB
10:49:57.454  ubuntu05   Guest/RAM/Usage/Shared 0 kB
10:49:57.454  ubuntu05   Guest/RAM/Usage/Cache 292376 kB
10:49:57.454  ubuntu05   Guest/Pagefile/Usage/Total 2097148 kB
------------  ---------  ------------------------  ----------
10:49:58.466  ubuntu05   CPU/Load/User             0.25%
10:49:58.466  ubuntu05   CPU/Load/Kernel           0.00%
10:49:58.466  ubuntu05   RAM/Usage/Used            1321732 kB
10:49:58.466  ubuntu05   Disk/Usage/Used           2959 mB
10:49:58.466  ubuntu05   Net/Rate/Rx               0 B/s
10:49:58.466  ubuntu05   Net/Rate/Tx               0 B/s
10:49:58.466  ubuntu05   Guest/CPU/Load/User       0.00%
10:49:58.466  ubuntu05   Guest/CPU/Load/Kernel 0.00%
10:49:58.466  ubuntu05   Guest/CPU/Load/Idle   100.00%
10:49:58.466  ubuntu05   Guest/RAM/Usage/Total 2050900 kB
10:49:58.466  ubuntu05   Guest/RAM/Usage/Free 1957572 kB
10:49:58.466  ubuntu05   Guest/RAM/Usage/Balloon 0 kB
10:49:58.466  ubuntu05   Guest/RAM/Usage/Shared 0 kB
10:49:58.466  ubuntu05   Guest/RAM/Usage/Cache 292376 kB
10:49:58.466  ubuntu05   Guest/Pagefile/Usage/Total 2097148 kB
------------  ---------  ------------------------  ----------
10:49:59.476  ubuntu05   CPU/Load/User             0.00%
10:49:59.476  ubuntu05   CPU/Load/Kernel           0.00%
10:49:59.476  ubuntu05   RAM/Usage/Used            1321716 kB
10:49:59.476  ubuntu05   Disk/Usage/Used           2959 mB
10:49:59.476  ubuntu05   Net/Rate/Rx               863647 B/s
10:49:59.476  ubuntu05   Net/Rate/Tx               6337 B/s
10:49:59.476  ubuntu05   Guest/CPU/Load/User       1.00%
10:49:59.476  ubuntu05   Guest/CPU/Load/Kernel 1.00%
10:49:59.476  ubuntu05   Guest/CPU/Load/Idle   98.00%
10:49:59.476  ubuntu05   Guest/RAM/Usage/Total 2050900 kB
10:49:59.476  ubuntu05   Guest/RAM/Usage/Free 1879640 kB
10:49:59.476  ubuntu05   Guest/RAM/Usage/Balloon 0 kB
10:49:59.476  ubuntu05   Guest/RAM/Usage/Shared 0 kB
```

```
10:49:59.476 ubuntu05   Guest/RAM/Usage/Cache 293300 kB
10:49:59.476 ubuntu05   Guest/Pagefile/Usage/Total 2097148 kB
------------ ---------- -------------------- ----------
10:50:00.486 ubuntu05   CPU/Load/User         0.00%
10:50:00.486 ubuntu05   CPU/Load/Kernel       0.00%
10:50:00.486 ubuntu05   RAM/Usage/Used        1321700 kB
10:50:00.486 ubuntu05   Disk/Usage/Used       2959 mB
10:50:00.486 ubuntu05   Net/Rate/Rx           0 B/s
10:50:00.486 ubuntu05   Net/Rate/Tx           0 B/s
10:50:00.486 ubuntu05   Guest/CPU/Load/User   0.00%
10:50:00.486 ubuntu05   Guest/CPU/Load/Kernel 1.00%
10:50:00.486 ubuntu05   Guest/CPU/Load/Idle   98.00%
10:50:00.486 ubuntu05   Guest/RAM/Usage/Total 2050900 kB
10:50:00.486 ubuntu05   Guest/RAM/Usage/Free 1878636 kB
10:50:00.486 ubuntu05   Guest/RAM/Usage/Balloon 0 kB
10:50:00.486 ubuntu05   Guest/RAM/Usage/Shared 0 kB
10:50:00.486 ubuntu05   Guest/RAM/Usage/Cache 293304 kB
10:50:00.486 ubuntu05   Guest/Pagefile/Usage/Total 2097148 kB
------------ ---------- -------------------- ----------
.
.
.
```

Listing E.4: Example of the *VBoxVMMetrics* model in XML form generated based on resource usage log in Listing E.3

```
<?xml version="1.0" encoding="ASCII"?>
<UtilisationLog xmi:version="2.0" xmlns:xmi="http://www.omg.
       org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance" xmlns="ResourceLog" xmlns:_1="VBoxVMmetrics">
    <machine xsi:type="_1:Machine" name="ubuntu05" cpuSpeed
         ="2000.0" cpuUnit="4" memory="2048.0"/>
    <logRecords time="42597.454" CPU_Used="0.5" RAM_Used="91.0"
          Disk_Used="2959" Net_Incoming="0.0" Net_Outgoing
         ="0.0"/>
   <logRecords time="42598.466" CPU_Used="0.25" RAM_Used="91.0"
         Disk_Used="2959" Net_Incoming="0.0" Net_Outgoing
        ="0.0"/>
    <logRecords time="42599.476" CPU_Used="0" RAM_Used="167.0"
         Disk_Used="2959" Net_Incoming="843.0" Net_Outgoing
         ="6.0"/>
    <logRecords time="42600.486" CPU_Used="0" RAM_Used="168.0"
         Disk_Used="2959" Net_Incoming="0.0" Net_Outgoing
         ="0.0"/>
.
.
.
</UtilisationLog>
```

Listing E.5: Sample of disk utilisation log recording with df utility

```
[TIME: 11:49:56.415] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1285588 6314500 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
[TIME: 11:49:57.434] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1285592 6314496 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
[TIME: 11:49:58.450] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1286400 6313688 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
[TIME: 11:49:59.490] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1286400 6313688 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
[TIME: 11:50:00.508] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1286400 6313688 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
[TIME: 11:50:01.530] Output: Filesystem 1K-blocks Used
    Available Use% Mounted on /dev/mapper/ubuntu05-root
    8006820 1286400 6313688 17% / udev 1015908 4 1015904 1% /
    dev tmpfs 410180 320 409860 1% /run none 5120 0 5120 0% /
    run/lock none 1025448 0 1025448 0% /run/shm /dev/sda1
    233191 26737 194013 13% /boot
.
.
.
```

Listing E.6: Example of the *DiskUsageLog* model in XML form generated based on disk usage log in Listing E.5

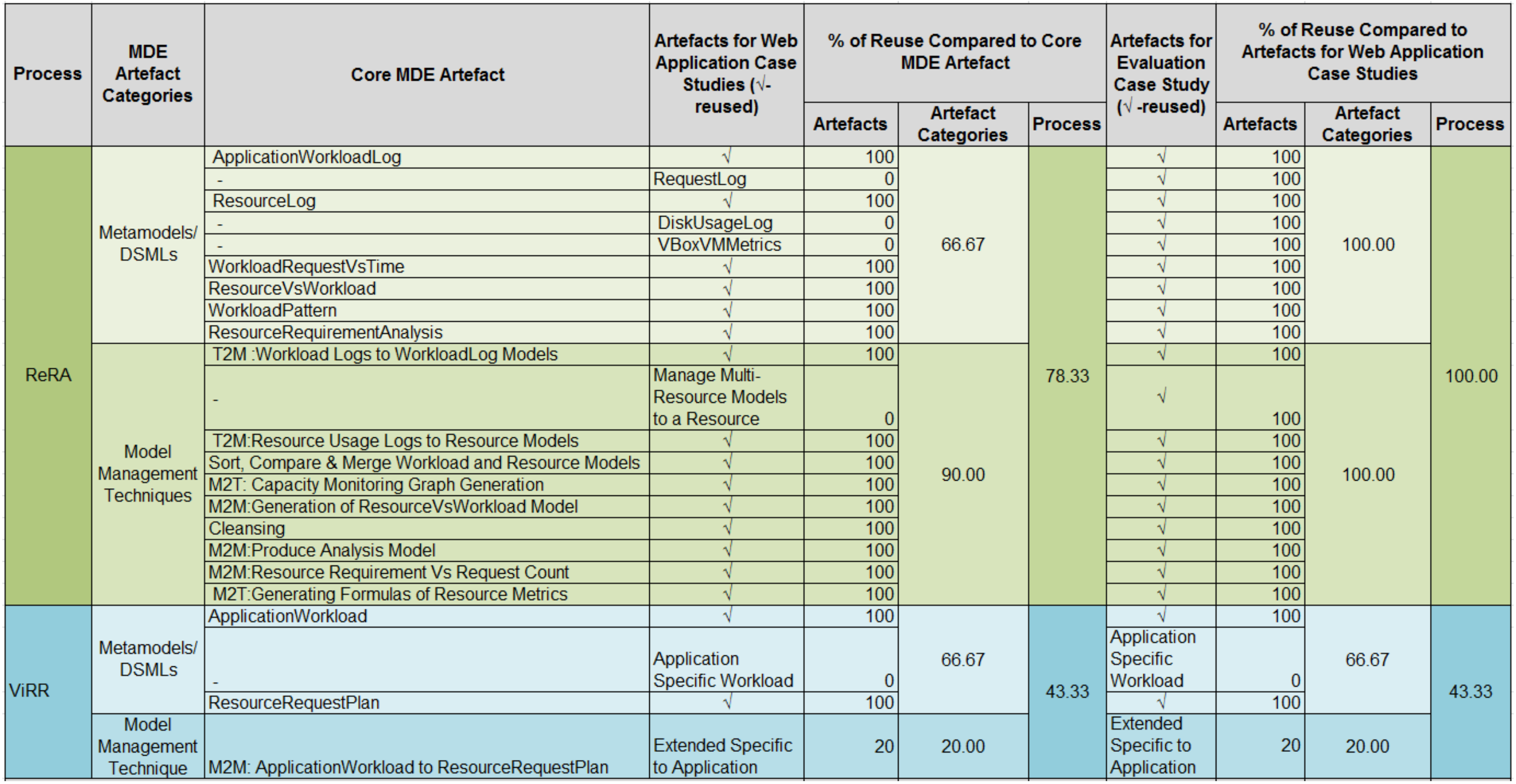
```
<?xml version=”1.0” encoding=”ASCII”?>
<DiskUtilisation xmi:version=”2.0” xmlns:xmi=”http://www.omg.org/XMI” xmlns=”DiskUsageLog”>
<machine name=”ubuntu05”/>
<logRecords time=”42596.415”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1285588.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
<logRecords time=”42597.434”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1285592.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
<logRecords time=”42598.45”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1286400.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
<logRecords time=”42599.49”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1286400.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
<logRecords time=”42600.508”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1286400.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
<logRecords time=”42601.53”> <filesystems name=”/dev/mapper/ubuntu05-root” usage=”1286400.0”/> <filesystems name=”udev” usage=”4.0”/> <filesystems name=”tmpfs” usage=”320.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”none” usage=”0.0”/> <filesystems name=”/dev/sda1” usage=”26737.0”/> </logRecords>
.
.
.
</DiskUtilisation>
```

Listing E.7: Example of the *ResourceLog* model in XML form generated by merging *DiskUsageLog* and *VBoxVMMetrics* models that presented in Listing E.6 and E.4 respectively

```xml
<?xml version="1.0" encoding="ASCII"?>
<UtilisationLog xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="ResourceLog" xmlns:_1="VBoxVMmetrics">
<machine xsi:type="_1:Machine" name="ubuntu05" cpuSpeed="2000.0" cpuUnit="4" memory="2048.0"/>
  <logRecords time="42597.454" CPU_Used="0.5" RAM_Used="91.0" Disk_Used="1281.8876953125" Net_Incoming="0.0" Net_Outgoing="0.0"/>
  <logRecords time="42598.466" CPU_Used="0.25" RAM_Used="91.0" Disk_Used="1282.6767578125" Net_Incoming="0.0" Net_Outgoing="0.0"/>
  <logRecords time="42599.476" RAM_Used="167.0" Disk_Used="1282.6767578125" Net_Incoming="843.0" Net_Outgoing="6.0"/>
  <logRecords time="42600.486" RAM_Used="168.0" Disk_Used="1282.6767578125" Net_Incoming="0.0" Net_Outgoing="0.0"/>
  <logRecords time="42601.498" RAM_Used="159.0" Disk_Used="1282.6767578125" Net_Incoming="4.0" Net_Outgoing="244.0"/>
.
.
.
</UtilisationLog>
```

Appendix F

Reusability Analysis



Process	MDE Artefact Categories	Core MDE Artefact	Artefacts for Web Application Case Studies (√-reused)	% of Reuse Compared to Core MDE Artefact			Artefacts for Evaluation Case Study (√ -reused)	% of Reuse Compared to Artefacts for Web Application Case Studies		
				Artefacts	Artefact Categories	Process		Artefacts	Artefact Categories	Process
ReRA	Metamodels/ DSMLs	ApplicationWorkloadLog	√	100	66.67	78.33	√	100	100.00	100.00
		-	RequestLog	0			√	100		
		ResourceLog	√	100			√	100		
		-	DiskUsageLog	0			√	100		
		-	VBoxVMMetrics	0			√	100		
		WorkloadRequestVsTime	√	100			√	100		
		ResourceVsWorkload	√	100			√	100		
		WorkloadPattern	√	100			√	100		
		ResourceRequirementAnalysis	√	100			√	100		
	Model Management Techniques	T2M :Workload Logs to WorkloadLog Models	√	100	90.00		√	100	100.00	
		-	Manage Multi-Resource Models to a Resource	0			√	100		
		T2M:Resource Usage Logs to Resource Models	√	100			√	100		
		Sort, Compare & Merge Workload and Resource Models	√	100			√	100		
		M2T: Capacity Monitoring Graph Generation	√	100			√	100		
		M2M:Generation of ResourceVsWorkload Model	√	100			√	100		
		Cleansing	√	100			√	100		
		M2M:Produce Analysis Model	√	100			√	100		
		M2M:Resource Requirement Vs Request Count	√	100			√	100		
		M2T:Generating Formulas of Resource Metrics	√	100			√	100		
ViRR	Metamodels/ DSMLs	ApplicationWorkload	√	100	66.67	43.33	√	100	66.67	43.33
		-	Application Specific Workload	0			Application Specific Workload	0		
		ResourceRequestPlan	√	100			√	100		
	Model Management Technique	M2M: ApplicationWorkload to ResourceRequestPlan	Extended Specific to Application	20	20.00		Extended Specific to Application	20	20.00	

Table F.1: Average Percentages of Reusability of Proposed MDE Solutions.

Appendix G

Programs

Listing G.1: Complete DiskUtilToVbDisk.eol.

```
var vbLog = VBm!UtilisationLog.all.first();
var diskLog = Disk!LogRecord.all;
var fs = Disk!FileSystem.all;
var totalDisk = 0.00;
var countRemove = 0;
var tempVL : Set = new Set; //[VBm!LogRecord];

fs.size().println("fs size is : ");

//***********Details of Log Files************************
vbLog.logRecords.size().println("vbm log count is : ");
diskLog.size().println("disk log count is : ");
diskLog.at(0).filesystems.println("disk log is : ");
//rename the webserver request name to an identified request name
for ( vl in vbLog.logRecords){
          var dl = diskLog.selectOne(l|l.time.floor()== vl.time.floor());

          if (dl.isDefined()){
                //dl.at(0).println();
                totalDisk = 0.00;
                 for (fs in dl.filesystems){
            totalDisk = totalDisk + fs.usage;
        }
        totalDisk.println();
        vl.Disk_Used = totalDisk/1024.00; // convert kilobyte to megabyte
      } else {
         countRemove = countRemove + 1;
         tempVL.add(vl);
         vl.println(” is removed ”);
      }
  }

countRemove.println(”countRemove = ”);

//to remove the last object if possible
if (tempVL.size()>0){
        vbLog.logRecords.removeall(tempVL);
}
```

Listing G.2: Complete ETL Program with Rules to Transform the Media Stream Application Workload to VM Resource Requirement Plan.

```
pre {
        var myTubeRequest = S!MediaStreamRequest.all.first();
        var dateTool = new Native(”tools.DateTool”);
}

//Rule to transform the Media Stream Application Workloads to VM Request Plan
rule MediastreamRequestToVmRequestPlan
  transform s : S!MediaStreamRequest
  to t : T!VmRequestPlan {

  t.applicationName = s.applicationID;
  t.vmRequirements.addAll(s.timeSlotRequests.equivalent());

  var noOfDays : Integer;
  var startDate : String;
  var endDate : String;
  var currentDate: String;

  startDate = s.startDate.DD + ”/” + s.startDate.MM + ”/” + s.startDate.YYYY;
  endDate = s.endDate.DD + ”/” + s.endDate.MM + ”/” + s.endDate.YYYY;

  noOfDays = dateTool.countDays(startDate, endDate);
  currentDate = startDate;

  for (i in Sequence{1..noOfDays}){
    //create daily request
    var reqDate : T!Date := new T!Date;
    reqDate.DD = currentDate.substring(0,2);
    reqDate.MM = currentDate.substring(3,5);
    reqDate.YYYY = currentDate.substring(6,10);
    t.createDailyRequest(reqDate);

    for (timeSlotRequest in s.timeSlotRequests){
      t.createSlots(reqDate, timeSlotRequest);
    }
    currentDate = dateTool.tomorrow(currentDate);
  }
  //assign start and end date
  t.startDate = t.requests.at(0).requestDate;
  t.endDate = t.requests.at(noOfDays -1).requestDate;
}


```

```
//Rule to Transform the Media stream Application Workload Time Slot to VM Resource Requirement.
rule TimeSlotRequestToVmRequirement
  transform timeSlotRequest : S!MediaStreamTimeSlotRequest
  to vm : T!Vm {

  var duration : Integer;

  duration := (timeSlotRequest.`to` - timeSlotRequest.from)* 60 * 60; //seconds

  //formula to calculate everage bandwidth (MegaByte)
  vm.bandwidth = myTubeRequest.averageVideoSize * timeSlotRequest.numberOfVideos * timeSlotRequest.averageVideoPercentageWacth;

  //average network requirement (MB/s)
  vm.network = vm.bandwidth/duration ;

  //average CPU requirement (GHz per seconds)
        vm.CPU = (timeSlotRequest.numberOfVideos * myTubeRequest.averageVideoSize * myTubeRequest.decodingTimeRate * timeSlotRequest.averageVideoPercentageWacth)/duration;

  //average memory requirement (MegaByte)
  vm.memory = (timeSlotRequest.numberOfVideos * myTubeRequest.bufferTime * myTubeRequest.bitRate)/duration;
}


operation T!VmRequestPlan createDailyRequest(day : T!Date){
        var r = new T!DailyRequest;
        r.requestDate= day;
        self.requests.add(r);
}

operation T!VmRequestPlan createSlots(day : T!Date, ts : Any ){
   for (dayReq in self.requests){
            if (dayReq.requestDate == day){
               var s = new T!Slot;
               s.from = createTime(ts.from);
               s.`to` = createTime(ts.`to`);
                   s.vmRequirement = ts.equivalent();
                   dayReq.slots.add(s);
                }
        }
}
```

```

operation createTime(i) : T!Time {
        var time = new T!Time;
        time.hour = i;
        time.minutes = 0;
        time.seconds = 0;
        return time;
}
```

Listing G.3: Complete ETL Program with Rules to Transform the Part-of-Speech Tagger Application Workload to VM Resource Requirement Plan.

```
pre {
        var mySPWorkload = S!SpeechTaggerWorkload.all.first();
        var dateTool = new Native("tools.DateTool");
}

//Rule to transform the Part-of-Speech Tagger  Application Workloads to VM Request Plan
rule SpeechTaggerWorkloadToVmRequestPlan
  transform s : S!SpeechTaggerWorkload
  to t : T!VmRequestPlan {

  t.applicationName = s.applicationID;
  t.vmRequirements.addAll(s.timeSlotWorkloads.equivalent());

  var noOfDays : Integer;
  var startDate : String;
  var endDate : String;
  var currentDate: String;

  startDate = s.startDate.DD + "/" + s.startDate.MM + "/" + s.startDate.YYYY;
  endDate = s.endDate.DD + "/" + s.endDate.MM + "/" + s.endDate.YYYY;

  noOfDays = dateTool.countDays(startDate, endDate);
  currentDate = startDate;

  for (i in Sequence{1..noOfDays}){
    //create daily request
    var reqDate : T!Date := new T!Date;
    reqDate.DD = currentDate.substring(0,2);
    reqDate.MM = currentDate.substring(3,5);
    reqDate.YYYY = currentDate.substring(6,10);
    t.createDailyRequest(reqDate);

      for (timeSlotWorkload in s.timeSlotWorkloads){
        t.createSlots(reqDate, timeSlotWorkload);
      }
      currentDate = dateTool.tomorrow(currentDate);
   }
   //assign start and end date
   t.startDate = t.requests.at(0).requestDate;
   t.endDate = t.requests.at(noOfDays -1).requestDate;
}


```

```
//Rule to Transform the Part-of-Speech Tagger Application Workload Time Slot to VM Resource Requirement.
rule SpeechTaggerTimeSlotWorkloadToVmRequirement
  transform timeSlotWorkload : S!SpeechTaggerTimeSlotWorkload
  to vm : T!Vm {

  //hours to seconds
  duration := (timeSlotWorkload.`to` - timeSlotWorkload.from)* 60 * 60; //seconds

  //declaration of variables
  var duration : Integer = 0;
  var smallFileCPU  : Any = 0.0;
  var mediumFileCPU : Any = 0.0;
  var largeFileCPU : Any = 0.0;
  var smallFileMemory : Any = 0.0;
  var mediumFileMemory : Any = 0.0;
  var largeFileMemory : Any = 0.0;
  var smallFileNetIn : Any = 0.0;
  var mediumFileNetIn : Any = 0.0;
  var largeFileNetIn : Any = 0.0;
  var smallFileNetOut : Any = 0.0;
  var mediumFileNetOut : Any = 0.0;
  var largeFileNetOut : Any = 0.0;
  var smallFileStorage : Any = 0.0;
  var mediumFileStorage : Any = 0.0;
  var largeFileStorage : Any = 0.0;

  for(fileSize in timeSlotWorkload.fileSizes){
    //formulas retrived from ReRA to process small files are used (Figure 6.47 )
    if (fileSize.size == SizeCategory#small){
      smallFileCPU = (5.2152 * fileSize.numberOfFiles.pow(3)) - (188.59 * fileSize.numberOfFiles.pow(2)) + (2069.5 * fileSize.numberOfFiles) - 129.9;

      smallFileMemory = (0.2727 * fileSize.numberOfFiles.pow(3)) + (1.4333 * fileSize.numberOfFiles.pow(2)) + (43.554* fileSize.numberOfFiles) + 64.48;

      smallFileNetIn = (0.1271 * fileSize.numberOfFiles.pow(3)) - (3.7384 * fileSize.numberOfFiles.pow(2))+ (36.104 * fileSize.numberOfFiles)  - 2.1564;

      smallFileNetOut = (0.1909 * fileSize.numberOfFiles.pow(3)) - (5.6324 * fileSize.numberOfFiles.pow(2)) + (54.541 * fileSize.numberOfFiles) - 3.8293;

```

```
       smallFileStorage =  1.3296 * fileSize.numberOfFiles + 1302.04 ;
    }

     //formulas retrived from ReRA to process medium files are used (Figure 6.38 )
     else if (fileSize.size == SizeCategory#medium){
      mediumFileCPU = (8.4681 * fileSize.numberOfFiles.pow(3)) - (250.38 * fileSize.numberOfFiles.pow(2)) + (2385.1 * fileSize.numberOfFiles) - 134.11;

       mediumFileMemory = (0.23681 * fileSize.numberOfFiles.pow(3)) + (0.90535 * fileSize.numberOfFiles.pow(2)) + (49.94 * fileSize.numberOfFiles) + 64.345;

       mediumFileNetIn = (0.40129 * fileSize.numberOfFiles.pow(3)) - (10.302 * fileSize.numberOfFiles.pow(2)) + (87.286 * fileSize.numberOfFiles) - 5.7723;

       mediumFileNetOut = (0.60614 * fileSize.numberOfFiles.pow(3)) - (15.746 * fileSize.numberOfFiles.pow(2)) + (134.31 * fileSize.numberOfFiles) - 9.1949;

       mediumFileStorage = 3.0931 * fileSize.numberOfFiles + 1302.4;
    }

    //formulas retrived from ReRA to process large files are used (Figure 6.29)
    else if (fileSize.size == SizeCategory#large){
       largeFileCPU = (11.008 * fileSize.numberOfFiles.pow(3)) - (97.25 * fileSize.numberOfFiles.pow(2)) + (2616.6 * fileSize.numberOfFiles) - 164.64;

       largeFileMemory = (0.9467 * fileSize.numberOfFiles.pow(3)) + (0.46501 * fileSize.numberOfFiles.pow(2)) + (57.656 * fileSize.numberOfFiles) + 64.72;

       largeFileNetIn = (0.88278 * fileSize.numberOfFiles.pow(3)) - (21.769 * fileSize.numberOfFiles.pow(2)) + (174.68 * fileSize.numberOfFiles) - 11.823;

       largeFileNetOut = (1.3434 * fileSize.numberOfFiles.pow(3)) - (32.601 * fileSize.numberOfFiles.pow(2)) + (260.09 * fileSize.numberOfFiles) - 17.461;

       largeFileStorage = 7.1797 * fileSize.numberOfFiles + 1303.2;
    }
```

```
 }

  //get CPU requirement (MHz)
  vm.CPU=getMaximum(smallFileCPU,mediumFileCPU,largeFileCPU);

  //get memory requirement (MB)
  vm.memory=getMaximum(smallFileMemory,mediumFileMemory,largeFileMemory);

  //get incomming network requirement (MB)
  vm.incomingNetwork=getMaximum(smallFileNetIn,mediumFileNetIn,largeFileNetIn);

  //get outgoing network requirement (MB)
  vm.outgoingNetwork=getMaximum(smallFileNetOut,mediumFileNetOut,largeFileNetOut);

  //get storage requirement (MB)
  vm.storage=getMaximum(smallFileStorage,mediumFileStorage,largeFileStorage);
}

operation T!VmRequestPlan createDailyRequest(day : T!Date){
        var r = new T!DailyRequest;
        r.requestDate= day;
        self.requests.add(r);
}

operation T!VmRequestPlan createSlots(day : T!Date, ts : Any ){
   for (dayReq in self.requests){
            if (dayReq.requestDate == day){
               var s = new T!Slot;
               s.`to` = createTime(ts.`to`);
               s.from = createTime(ts.from);
                   s.vmRequirement = ts.equivalent();
                   dayReq.slots.add(s);
                }
        }
}

operation createTime(i) : T!Time {
        var time = new T!Time;
        time.hour = i;
        time.minutes = 0;
        time.seconds = 0;
        return time;
}

```

```
operation getMaximum(x, y, z) : Any{
    var max = x;
        if (y > max) { max= y; }
        if (z > max) { max=z; }
 return max;
}
```

Listing G.4: Complete ETL Program with Rules to Transform the Image Filter Application Workload to VM Resource Requirement Plan.

```
pre {
  var myIFWorkload = S!ImageFilterWorkload.all.first();
  var dateTool = new Native(”tools.DateTool”);
}

//Rule to transform the Image Filtering Application Workloads
   to VM Request Plan
rule ImageFilterWorkloadToVmRequestPlan
  transform s : S!ImageFilterWorkload
  to t : T!VmRequestPlan {

    t.applicationName = s.applicationID;
    t.vmRequirements.addAll(s.timeSlotWorkloads.equivalent());

    var noOfDays : Integer;
    var startDate : String;
    var endDate : String;
    var currentDate: String;

    startDate = s.startDate.DD + ”/” + s.startDate.MM + ”/” +
       s.startDate.YYYY;
    endDate = s.endDate.DD + ”/” + s.endDate.MM + ”/” + s.
       endDate.YYYY;

    noOfDays = dateTool.countDays(startDate, endDate);
    currentDate = startDate;

    for (i in Sequence{1..noOfDays}){
      //create daily request
      var reqDate : T!Date := new T!Date;
      reqDate.DD = currentDate.substring(0,2);
      reqDate.MM = currentDate.substring(3,5);
      reqDate.YYYY = currentDate.substring(6,10);
      t.createDailyRequest(reqDate);

      for (timeSlotWorkload in s.timeSlotWorkloads){
        t.createSlots(reqDate, timeSlotWorkload);
      }
      currentDate = dateTool.tomorrow(currentDate);
   }
   //assign start and end date
   t.startDate = t.requests.at(0).requestDate;
   t.endDate = t.requests.at(noOfDays −1).requestDate;
}


```

```
//Rule to Transform the Image Filtering Application Workload Time Slot to VM Resource Requirement.
rule ImageFilterTimeSlotWorkloadToVmRequirement
  transform timeSlotWorkload : S!ImageFilterTimeSlotWorkload
  to vm : T!Vm {

    //declaration of variables
    var smallDefaultCPU : Any = 0.0;
    var smallNegateCPU : Any = 0.0;
    var smallGrayCPU : Any = 0.0;
    var mediumDefaultCPU : Any = 0.0;
    var mediumNegateCPU : Any = 0.0;
    var mediumGrayCPU : Any = 0.0;
    var largeDefaultCPU : Any = 0.0;
    var largeNegateCPU : Any = 0.0;
    var largeGrayCPU : Any = 0.0;
    var smallDefaultMemory  : Any = 0.0;
    var smallNegateMemory : Any = 0.0;
    var smallGrayMemory : Any = 0.0;
    var mediumDefaultMemory  : Any = 0.0;
    var mediumNegateMemory : Any = 0.0;
    var mediumGrayMemory : Any = 0.0;
    var largeDefaultMemory : Any = 0.0;
    var largeNegateMemory : Any = 0.0;
    var largeGrayMemory : Any = 0.0;
    var smallDefaultNetIn : Any = 0.0;
    var smallNegateNetIn : Any = 0.0;
    var smallGrayNetIn : Any = 0.0;
    var mediumDefaultNetIn: Any = 0.0;
    var mediumNegateNetIn : Any = 0.0;
    var mediumGrayNetIn : Any = 0.0;
    var largeDefaultNetIn : Any = 0.0;
    var largeNegateNetIn : Any = 0.0;
    var largeGrayNetIn : Any = 0.0;
    var smallDefaultNetOut : Any = 0.0;
    var smallNegateNetOut : Any = 0.0;
    var smallGrayNetOut : Any = 0.0;
    var mediumDefaultNetOut: Any = 0.0;
    var mediumNegateNetOut : Any = 0.0;
    var mediumGrayNetOut : Any = 0.0;
    var largeDefaultNetOut : Any = 0.0;
    var largeNegateNetOut : Any = 0.0;
    var largeGrayNetOut : Any = 0.0;


    for(wl in timeSlotWorkload.workloads){
      if (wl.size == SizeCategory#small){
        if (wl.filter == Filter#`default`){
```

```
        //formulas retrieved from ReRA to process small images with default filter(Figure C.21 )
         smallDefaultCPU = (17.705 * wl.numberOFimages.pow(3)) - (387.67 * wl.numberOFimages.pow(2)) + (2816.9 * wl.numberOFimages) - 64.426;

         smallDefaultMemory = (1.5066 * wl.numberOFimages) + 81.816;

         smallDefaultNetIn = (123.41 * wl.numberOFimages.pow(2)) + (1873.5 * wl.numberOFimages) + 379.64;

         smallDefaultNetOut = (16.088 * wl.numberOFimages.pow(3)) - (359.12 * wl.numberOFimages.pow(2))+ (2710.3 * wl.numberOFimages) - 214.23;
        }

        else if (wl.filter == Filter#negate){
        //formulas retrieved from ReRA to process small images with negate filter(Figure C.27 )
         smallNegateCPU = (9.7597 * wl.numberOFimages.pow(3)) - (272.64 * wl.numberOFimages.pow(2)) + (2413.4 * wl.numberOFimages) + 233.44;

         smallNegateMemory = (0.00051442 * wl.numberOFimages.pow(3)) - ( 0.10503 * wl.numberOFimages.pow(2)) + (2.4851 * wl.numberOFimages) + 79.525;

         smallNegateNetIn = (6.0596 * wl.numberOFimages.pow(3)) - (262.06 * wl.numberOFimages.pow(2)) + (2912 * wl.numberOFimages) + 148.05;

         smallNegateNetOut = (4.5664 * wl.numberOFimages.pow(3)) - (232.11 * wl.numberOFimages.pow(2)) + (2689.8 * wl.numberOFimages)  + 150.71;
        }

        else if (wl.filter == Filter#grayscale) {
        //formulas retrieved from ReRA to process small images with grayscale filter(Figure C.24 )
         smallGrayCPU = (13.199 * wl.numberOFimages.pow(3)) - (321.27 * wl.numberOFimages.pow(2)) + (2597.3 * wl.numberOFimages) - 13.216;

         smallGrayMemory = (0.034408 * wl.numberOFimages.pow(3)) - (0.5597 * wl.numberOFimages.pow(2)) + (3.736 * wl.numberOFimages) + 78.682;

```

```
           smallGrayNetIn = (22.199 * wl.numberOFimages.pow(3))
              - (491.83 * wl.numberOFimages.pow(2)) + (3704.1 *
               wl.numberOFimages) - 381.35;

           smallGrayNetOut = (14.557 * wl.numberOFimages.pow(3))
               - (322.95 * wl.numberOFimages.pow(2)) + (2423.7
              * wl.numberOFimages) - 252.35;
           }
        }
        else if (wl.size == SizeCategory#medium){
          if (wl.filter == Filter#‘default‘){
          //formulas retrieved from ReRA to process medium
             images with default filter(Figure C.12 )
           mediumDefaultCPU = (14.873 * wl.numberOFimages.pow(3)
              ) - (365.65 * wl.numberOFimages.pow(2)) + (2913.9
               * wl.numberOFimages) - 116.58;

           mediumDefaultMemory = (0.072804 * wl.numberOFimages.
              pow(3)) - (1.8412 * wl.numberOFimages.pow(2)) +
              (16.496 * wl.numberOFimages) + 79.083;

           mediumDefaultNetIn = (6.0701 * wl.numberOFimages.pow
              (3)) - (148.56 * wl.numberOFimages.pow(2)) +
              (1188.4 * wl.numberOFimages)  - 118.69;

           mediumDefaultNetOut = (9.6006 * wl.numberOFimages.pow
              (3)) - (238.42 * wl.numberOFimages.pow(2))+
              (1920.5 * wl.numberOFimages) - 185.13;
          }

          else if (wl.filter == Filter#negate){
          //formulas retrieved from ReRA to process medium
             images with negate filter(Figure C.18 )
           mediumNegateCPU = (14.206 * wl.numberOFimages.pow(3))
               - (355.97 * wl.numberOFimages.pow(2)) + (2887.5
              * wl.numberOFimages) - 111.02;

           mediumNegateMemory = (5.121 * wl.numberOFimages) +
              79.298;

           mediumNegateNetIn = (8.1863 * wl.numberOFimages.pow
              (3)) - (201.7 * wl.numberOFimages.pow(2)) + (1634
               * wl.numberOFimages)  - 153.41;

           mediumNegateNetOut = (6.5072 * wl.numberOFimages.pow
              (3)) - (165.86 * wl.numberOFimages.pow(2)) +
              (1357.1 * wl.numberOFimages) - 128.99;
           }

```

```
      else if (wl.filter == Filter#grayscale) {
      //formulas retrieved from ReRA to process medium images with grayscale filter(Figure C.15 )
      mediumGrayCPU = (14.708 * wl.numberOFimages.pow(3)) - (367.12 * wl.numberOFimages.pow(2)) + (2931.8 * wl.numberOFimages) - 116.45;

      mediumGrayMemory =  (4.6741 * wl.numberOFimages) + 80.359;

      mediumGrayNetIn = (10.915 * wl.numberOFimages.pow(3)) - (272.7 * wl.numberOFimages.pow(2)) + (2170.4 * wl.numberOFimages) - 194.22;

      mediumGrayNetOut =(5.7108 * wl.numberOFimages.pow(3)) - (141.23 * wl.numberOFimages.pow(2)) + (1112.7 * wl.numberOFimages) - 104.07;
      }
    }

    else if (wl.size == SizeCategory#large){
      if (wl.filter == Filter#`default`){
      //formulas retrieved from ReRA to process large images with default filter(Figure C.3 )
      largeDefaultCPU = (14.134 * wl.numberOFimages.pow(3)) - (372.03 * wl.numberOFimages.pow(2)) + (3113.7 * wl.numberOFimages) - 434.08;

      largeDefaultMemory = (69.763 * wl.numberOFimages) + 86.841;

      largeDefaultNetIn = (1.8246 * wl.numberOFimages.pow(3)) - (45.002 * wl.numberOFimages.pow(2)) + (361.99 * wl.numberOFimages) - 44.558;

      largeDefaultNetOut = (3.2162 * wl.numberOFimages.pow(3)) - (86.301 * wl.numberOFimages.pow(2)) + (722.14 * wl.numberOFimages) - 72.91;
      }

      else if (wl.filter == Filter#negate){
      //formulas retrieved from ReRA to process large images with negate filter(Figure C.9 )
      largeNegateCPU = (14.476 * wl.numberOFimages.pow(3)) - (373.13 * wl.numberOFimages.pow(2)) + (3075 * wl.numberOFimages) - 252.33;

      largeNegateMemory = (58.81 * wl.numberOFimages) + 79.101;
```

```

            largeNegateNetIn = (3.086 * wl.numberOFimages.pow(3))
                 - (82.426 * wl.numberOFimages.pow(2)) + (704.3 *
                 wl.numberOFimages) - 72.934;

            largeNegateNetOut = (3.0144 * wl.numberOFimages.pow
                (3)) - (77.783 * wl.numberOFimages.pow(2)) +
                (634.76 * wl.numberOFimages) - 66.487;
            }

            else if (wl.filter == Filter#grayscale) {
            //formulas retrieved from ReRA to process large
                images with grayscale filter(Figure C.6 )
            largeGrayCPU = (13.711 * wl.numberOFimages.pow(3)) -
                (358.62 * wl.numberOFimages.pow(2)) + (3005.9 *
                wl.numberOFimages) - 175.53;

            largeGrayMemory = (58.718* wl.numberOFimages) +
                76.977;

            largeGrayNetIn = (3.511 * wl.numberOFimages.pow(3)) -
                 (87.991 * wl.numberOFimages.pow(2)) + (719.57 *
                wl.numberOFimages) - 76.358;

            largeGrayNetOut = (2.2861 * wl.numberOFimages.pow(3))
                 - (59.233 * wl.numberOFimages.pow(2)) + (486.02
                * wl.numberOFimages) - 51.898;
            }
        }
    }


    //get CPU requirement (MHz)
    vm.CPU =  getMaximum(smallDefaultCPU,smallNegateCPU,
        smallGrayCPU,mediumDefaultCPU,mediumNegateCPU,
        mediumGrayCPU,largeDefaultCPU,largeNegateCPU,
        largeGrayCPU);

    //get memory requirement (MB)
    vm.memory =  getMaximum(smallDefaultMemory,
        smallNegateMemory,smallGrayMemory,mediumDefaultMemory,
        mediumNegateMemory,mediumGrayMemory,largeDefaultMemory
        ,largeNegateMemory,largeGrayMemory);

    //get incomming network requirement (MB)
    vm.incomingNetwork = getMaximum(smallDefaultNetIn,
        smallNegateNetIn,smallGrayNetIn,mediumDefaultNetIn,
        mediumNegateNetIn,mediumGrayNetIn,largeDefaultNetIn,
        largeNegateNetIn,largeGrayNetIn);

    //get outgoing network requirement (MB)
    vm.outgoingNetwork = getMaximum(smallDefaultNetOut, smallNegateNetOut,smallGrayNetOut,mediumDefaultNetOut, mediumNegateNetOut,mediumGrayNetOut,largeDefaultNetOut ,largeNegateNetOut,largeGrayNetOut);

}


operation T!VmRequestPlan createDailyRequest(day : T!Date){
        var r = new T!DailyRequest;
        r.requestDate= day;
        self.requests.add(r);
}


operation T!VmRequestPlan createSlots(day : T!Date, ts : Any )
   {
   for (dayReq in self.requests){
            if (dayReq.requestDate == day){
               var s = new T!Slot;
               s.from = createTime(ts.from);
               s.`to` = createTime(ts.`to`);
                   s.vmRequirement = ts.equivalent();
                   dayReq.slots.add(s);
                }
        }
}


operation createTime(i) : T!Time {
        var time = new T!Time;

        time.hour = 0;
        time.minutes = 0;
        time.seconds = 0;

        if (i >= 3600) {
                time.hour = (i/3600).ceiling();
                i = i - (time.hour * 3600);
        }

        if (i >= 60 ) {
                time.minutes = (i/60).ceiling();
                i = i - (time.minutes * 60);
        }

        if (i > 0 ) {
                time.seconds = i;
        }

        return time;
}


operation getMaximum(a,b,c,d,e,f,g,h,i) : Any{
    var max = a;
        if (b > max) { max = b; }
        if (c > max) { max = c; }
        if (d > max) { max = d; }
        if (e > max) { max = e; }
        if (f > max) { max = f; }
        if (g > max) { max = g; }
        if (h > max) { max = h; }
        if (i > max) { max = i; }
 return max;
}
```

List of Abbreviations

1.	**PiP**	Physical Infrastructure Provider.
2.	**ViP**	Virtual Infrastructure Service Provider.
3.	**MDE**	Model-Driven Engineering.
4.	**DSM**	Domain Specific Modelling.
5.	**DSML**	Domain Specific Modelling Language.
6.	**SLA**	Service Level Agreement.
7.	**VM**	Virtual Machine.
8.	**PM**	Physical Machine.
9.	**ReRA**	Resource Requirement Analysis.
10.	**ViRR**	Virtual Resource Requirement.
11.	**M2M**	model-to-model.
12.	**T2M**	text-to-model.
13.	**M2T**	model-to-text.

References

[1] Epsilon Model Connectivity. www.eclipse.org. URL http://www.eclipse.org/epsilon/doc/emc/.

[2] An Overview of Virtualization Techniques. http://www.virtuatopia.com/, June 2009. URL http://www.virtuatopia.com/index.php/An_Overview_of_Virtualization_Techniques.

[3] Information technology - Object Management Group Meta Object Facility (MOF) Core. ISO/IEC 19508, April 2014. URL http://www.omg.org/spec/MOF/.

[4] Bruno Abrahao, Virgilio Almeida, Jussara Almeida, Alex Zhang, Dirk Beyer, and Fereydoon Safai. Self-Adaptive SLA-Driven Capacity Management for Internet Services. In *10th IEEE/IFIP Network Operations and Management Symposium NOMS 2006*, pages 557–568, April 2006. doi: 10.1109/NOMS.2006.1687584.

[5] Keith Adams and Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. *SIGARCH Comput. Archit. News*, 34 (5):2–13, October 2006. ISSN 0163-5964. doi: 10.1145/1168919.1168860. URL http://doi.acm.org/10.1145/1168919.1168860.

[6] John Allspaw. *The Art of Capacity Planning.* O'Reilly, 2008.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, Electrical Engineering and Computer Sciences University of California at Berkeley, February 2009. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html.

[8] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph,

Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010. ISSN 0001-0782. doi: 10.1145/1721654.1721672. URL `http://doi.acm.org/10.1145/1721654.1721672`.

[9] Arshdeep Bahga and Vijay Krishna Madisetti. Synthetic Workload Generation for Cloud Computing Applications. *Journal of Software Engineering and Applications*, 4(7):396–410, July 2011. doi: doi:10.4236/jsea.2011.47046.

[10] L.A. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12):33–37, December 2007.

[11] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3 – 22, 2009. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2008.03.066. URL `http://www.sciencedirect.com/science/article/pii/S0164121208001015`. Special Issue: Software Performance - Modeling and Analysis.

[12] Jean Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, May 2005. ISSN 1619-1366. doi: 10.1007/s10270-005-0079-0. URL `http://dx.doi.org/10.1007/s10270-005-0079-0`.

[13] Jean Bézivin, Frédéric Jouault, Ivan Kurtev, and Patrick Valduriez. Model-based DSL frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, OOPSLA '06, pages 602–616, New York, NY, USA, 2006. ACM. ISBN 1-59593-491-X. doi: 10.1145/1176617.1176632. URL `http://doi.acm.org/10.1145/1176617.1176632`.

[14] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. Model-Driven Software Engineering in Practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182, 2012. doi: 10.2200/S00441ED1V01Y201208SWE001. URL `http://dx.doi.org/10.2200/S00441ED1V01Y201208SWE001`.

[15] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. Modeling Parameter and Context Dependencies in Online Architecture-level Performance Models. In *Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering*, CBSE '12, pages 3–12, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1345-2.

doi: 10.1145/2304736.2304740. URL http://doi.acm.org/10.1145/2304736.2304740.

[16] Fabian Brosig, Nikolaus Huber, and Samuel Kounev. Architecture-level software performance abstractions for online performance prediction. *Science of Computer Programming*, 90:71 – 92, 2014. ISSN 0167-6423. doi: https://doi.org/10.1016/j.scico.2013.06.004. URL http://www.sciencedirect.com/science/article/pii/S0167642313001421. Special Issue on Component-Based Software Engineering and Software Architecture.

[17] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5–13, September 2008. doi: 10.1109/HPCC.2008.172.

[18] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Inter-Cloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In Ching-Hsien Hsu, LaurenceT. Yang, JongHyuk Park, and Sang-Soo Yeo, editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-13118-9. doi: 10.1007/978-3-642-13119-6_2. URL http://dx.doi.org/10.1007/978-3-642-13119-6_2.

[19] Nicholas Carr. *The Big Switch: Rewiring the World from Edison to Google*. W. W. Norton & Compan, 2008.

[20] Brian J.S. Chee and Jr. Curtis Franklin. *Cloud Computing: Technologies and Strategies of the Ubiquitous Data Center*. CRC Press, 2010.

[21] Ludmila Cherkasova, Wenting Tang, and Sharad Singhal. An SLA-Oriented Capacity Planning Tool for Streaming Media Services. In *Proceedings of the 2004 International Conference on Dependable Systems and Networks*, pages 743–752, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2052-9. URL http://dl.acm.org/citation.cfm?id=1009382.1009790.

[22] Vanea Chiprianov, Yvon Kermarrec, Siegfried Rouvrais, and Jacques Simonin. Extending Enterprise Architecture Modeling Languages for Domain Specificity and Collaboration: Application to Telecommunication Service Design. *Software & Systems Modeling*, 13(3):963–974,

2014. ISSN 1619-1374. doi: 10.1007/s10270-012-0298-0. URL http://dx.doi.org/10.1007/s10270-012-0298-0.

[23] Tony Clark, Andy Evans, Stuart Kent, and Paul Sammut. The MMF Approach to Engineering Object-Oriented Design Languages. In *Proceedings of the Workshop on Language Descriptions, Tools and Applications*, April 2001. URL http://eprints.mdx.ac.uk/6249/1/clarkpatterns.pdf.

[24] Krzysztof Czarnecki and Simon Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3): 621 – 645, April 2006. URL http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5386627.

[25] Christina Delimitrou and Christos Kozyrakis. Cross-Examination of Datacenter Workload Modeling Techniques. In *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 72 –79, june 2011. doi: 10.1109/ICDCSW.2011.45.

[26] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35: 26–36, June 2000. ISSN 0362-1340. doi: http://doi.acm.org/10.1145/352029.352035. URL http://doi.acm.org/10.1145/352029.352035.

[27] Brian Dougherty, Jules White, and Douglas C. Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, 28(2):371–378, 2012. ISSN 0167-739X. doi: 10.1016/j.future.2011.05.009. URL http://www.sciencedirect.com/science/article/pii/S0167739X11000902.

[28] Jorge Ejarque, Marc de Palol, Inigo Goiri, Ferran Julia, Jordi Guitart, Rosa M. Badia, and Jordi Torres. SLA-Driven Semantically-Enhanced Dynamic Resource Allocator for Virtualized Service Providers. In *Fourth IEEE International Conference on eScience*, pages 8–15, December 2008. doi: 10.1109/eScience.2008.15.

[29] Renee Elio, Jim Hoover, Ioanis Nikolaidis, Mohammad Salavatipour, Lorna Stewart, and Ken Wong. About Computing Science Research Methodology, 2011.

[30] Matthew Emerson and Janos Sztipanovits. Techniques for Metamodel Composition. In *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling*, pages 123–139. ACM, ACM Press, 2006.

[31] Jean-Marie Favre. Towards a Basic Theory to Model Model Driven

Engineering. In *Proceedings of the 3rd Workshop in Software Model Engineering (WiSME)*, 2004.

[32] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop (GCE '08)*, pages 1–10, November 2008. doi: 10.1109/GCE.2008.4738445.

[33] Roman Frigg and Stephan Hartmann. Models in Science. Stanford Encyclopedia of Philosophy website, 2006. URL `http://www.science.uva.nl/~seop/archives/spr2009/entries/models-science/`. http://www.science.uva.nl/ seop/archives/spr2009/entries/models-science/.

[34] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-Driven Workload Modeling for the Cloud. In *IEEE 26th International Conference on Data Engineering Workshops (ICDEW)*, pages 87–92. IEEE, March 2010. doi: 10.1109/ICDEW.2010.5452742.

[35] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. Minimizing data center SLA violations and power consumption via hybrid resource provisioning. In *International Green Computing Conference and Workshops (IGCC)*, pages 1–8, July 2011. doi: 10.1109/IGCC.2011.6008611.

[36] Saurabh Kumar Garg, Steve Versteeg, and Rajkumar Buyya. A framework for ranking of cloud computing services. *Future Generation Computer Systems*, 29(4):1012–1023, June 2013. ISSN 0167-739X. doi: 10.1016/j.future.2012.06.006. URL `http://dx.doi.org/10.1016/j.future.2012.06.006`.

[37] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Capacity Management and Demand Prediction for Next Generation Data Centers. In *IEEE International Conference on Web Services*, pages 43–50. IEEE, July 2007. doi: 10.1109/ICWS.2007.62. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4279581&tag=1`.

[38] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM Comput. Commun. Rev.*, 39(1):68–73, December 2008. ISSN

0146-4833. doi: 10.1145/1496091.1496103. URL http://doi.acm.org/10.1145/1496091.1496103.

[39] Emily H. Halili. *Apache JMeter*. Pacckt Publishing, 2008.

[40] John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-Driven Engineering Practices in Industry. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 633–642, May 2011. doi: 10.1145/1985793.1985882. URL http://dl.acm.org/citation.cfm?doid=1985793.1985882.

[41] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical Assessment of MDE in Industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0445-0. doi: 10.1145/1985793.1985858. URL http://doi.acm.org/10.1145/1985793.1985858.

[42] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012. ISSN 0167-739X. doi: http://dx.doi.org/10.1016/j.future.2011.05.027. URL http://www.sciencedirect.com/science/article/pii/S0167739X11001129.

[43] Kris Jamsa. *Cloud Computing: SaaS, Paas, Iaas, Virtualization, Business Models, Mobile, Security, and More.* Jones & Bartlett Learning, 2013.

[44] Advocate Jim Elliott. The Evolution of IBM Mainframes and VM. SHARE Session 9140, August 2004. URL http://www.linuxvm.org/Present/SHARE103/S9140jea.pdf.

[45] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation.* Wiley-IEEE Computer Society Pr, March 2008. ISBN 0470036664. URL http://www.worldcat.org/isbn/0470036664.

[46] Hamzeh Khazaei, Jelena Misic, and Vojislav B. Misic. Modelling of Cloud Computing Centers Using M/G/m Queues. In *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 87–92, June 2011. doi: 10.1109/ICDCSW.2011.13.

[47] Dimitrios Kolovos, Louis Rose, and Richard Paige. *The Epsilon Book*. www.eclipse.org/epsilon/, 2011. URL http://www.eclipse.org/epsilon/doc/book/.

[48] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. The Epsilon Object Language (EOL). In Arend Rensink and Jos Warmer, editors, *Model Driven Architecture Foundations and Applications*, volume 4066 of *Lecture Notes in Computer Science*, pages 128–142. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-35909-8. doi: 10.1007/11787044_11. URL `http://dx.doi.org/10.1007/11787044_11`.

[49] Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack. The Epsilon Transformation Language. In Antonio Vallecillo, Jeff Gray, and Alfonso Pierantonio, editors, *Theory and Practice of Model Transformations*, volume 5063 of *Lecture Notes in Computer Science*, pages 46–60. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69926-2. doi: 10.1007/978-3-540-69927-9_4. URL `http://dx.doi.org/10.1007/978-3-540-69927-9_4`.

[50] Jonathan G. Koomey. Estimating Total Power Consumption by Servers in The U.S. and the World. Technical report, Stanford University, February 2007.

[51] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. The Descartes Modeling Language. Technical report, Department of Computer Science, University of Wuerzburg, October 2014. URL `http://www.descartes-research.net/dml/`.

[52] Ivan Kurtev, Jean Bézivin, Frédéric Jouault, and Patrick Valduriez. Model-based DSL Frameworks. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 602–616. ACM, 2006.

[53] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M. Rumble, Eyal de Lara, Michael Brudno, and Mahadev Satyanarayanan. SnowFlock: Rapid Virtual Machine Cloning for Cloud Computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 1–12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-482-9. doi: 10.1145/1519065.1519067. URL `http://doi.acm.org/10.1145/1519065.1519067`.

[54] Ming Mao, Jie Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 41–48, Oct 2010. doi: 10.1109/GRID.2010.5697966.

[55] C.C.T. Mark, D. Niyato, and Tham Chen-Khong. Evolutionary Op-

timal Virtual Machine Placement and Demand Forecaster for Cloud Computing. In *IEEE International Conference on Advanced Information Networking and Applications (AINA)*, pages 348–355, March 2011. doi: 10.1109/AINA.2011.50.

[56] Joaquin Miller and Jishnu Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), June 2003. URL `http://www.omg.org/cgi-bin/doc?omg/03-06-01`.

[57] Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Modeling Workloads, SLAs and their Violations in Cloud Computing. In Christopher M. Poskitt, editor, *Fourth York Doctoral Symposium on Computer Science*. Department of Computer Science, The University of York, UK, October 2011. URL `http://www.cs.york.ac.uk/ftpdir/reports/2011/YCS/468/YCS-2011-468.pdf`.

[58] Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Cloud Computing Workload and Capacity Management Using Domain Specific Modelling. In *CloudMDE Workshop, co-located with ECMFA 2012*. CEUR Proceedings, July 2012. URL `http://www2.imm.dtu.dk/conferences/ECMFA-2012/workshops/?page=CloudMDE`.

[59] Rafidah Pakir Mohamad, Dimitrios S. Kolovos, and Richard F. Paige. Resource Requirement Analysis for Web Applications Running in a Virtualised Environment. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 632–637, Dec 2014. doi: 10.1109/CloudCom.2014.134. URL `http://www.computer.org/csdl/proceedings/cloudcom/2014/4093/00/4093a632-abs.html`.

[60] Greg Nordstrom, Janos Sztipanovits, Gabor Karsai, and Akos Ledeczi. Metamodeling-rapid design and evolution of domain-specific modeling environments. In *Engineering of Computer-Based Systems, 1999. Proceedings. ECBS '99. IEEE Conference and Workshop on*, pages 68–74, Mar 1999. doi: 10.1109/ECBS.1999.755863. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=755863`.

[61] Richard F. Paige and Louis M. Rose. Lies, Damned Lies and UML2Java. *Journal of Object Technology*, 12(1), 2013. URL `http://blog.jot.fm/2013/01/25/lies-damned-lies-and-uml2java/`.

[62] Richard F. Paige, Phillip J. Brooke, and Jonathan S. Ostroff. Metamodel-based Model Conformance and Multiview Consistency Checking. *ACM Transactions on Software Engineering and Methodo-*

logy, 16(3), July 2007. ISSN 1049-331X. doi: 10.1145/1243987.1243989. URL http://doi.acm.org/10.1145/1243987.1243989.

[63] Freeman Parkhill. *The Challenge of the Computing Utility*. Addison-Wesley, 1966.

[64] Luis Pedro, Matteo Risoldi, Didier Buchs, and Vasco Amaral. Developing Domain-specific Modeling Languages by Metamodel Semantic Enrichment and Composition: A Case Study. In *Proceedings of the 10th Workshop on Domain-Specific Modeling*, DSM '10, pages 16:1–16:6, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0549-5. doi: 10.1145/2060329.2060364. URL http://doi.acm.org/10.1145/2060329.2060364.

[65] Diego Perez-Palacin, Radu Calinescu, and José Merseguer. log2cloud: Log-based Prediction of Cost-Performance Trade-offs for Cloud Deployments. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 397–404, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1656-9. doi: 10.1145/2480362.2480442. URL http://doi.acm.org/10.1145/2480362.2480442.

[66] Matthew Portnoy. *Virtualization Essentials*. Wiley, 2012. ISBN 978-1-118-17671-9. URL http://www.amazon.co.uk/Virtualization-Essentials-Matthew-Portnoy-ebook/dp/B007RT24QK/.

[67] Andres Quiroz, Hyunjoo Kim, Manish Parashar, Nathan Gnanasambandam, and Naveen Sharma. Towards Autonomic Workload Provisioning for Enterprise Grids and Clouds. In *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing*. IEEE & ACM, 2009.

[68] Louis M. Rose, Richard F. Paige, Dimitrios S. Kolovos, and FionaA.C. Polack. The Epsilon Generation Language. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture Foundations and Applications*, volume 5095 of *Lecture Notes in Computer Science*, pages 1–16. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69095-5. doi: 10.1007/978-3-540-69100-6_1. URL http://dx.doi.org/10.1007/978-3-540-69100-6_1.

[69] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 500–507, July 2011. doi: 10.1109/CLOUD.2011.42.

[70] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven En-

gineering. *Computer*, 39(2):25–31, February 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.58. URL http://dx.doi.org/10.1109/MC.2006.58.

[71] Bran Selic. The Pragmatics of Model-Driven Development. *Software, IEEE*, 20(5):19–25, September 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231146. URL http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/papers/SelicPragmatics.pdf.

[72] Bran Selic. What will it take? A view on adoption of model-based methods in practice. *Software & Systems Modeling*, 11(4):513–526, 2012. ISSN 1619-1366. doi: 10.1007/s10270-012-0261-0. URL http://dx.doi.org/10.1007/s10270-012-0261-0.

[73] Shane Sendall and Wojtek Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. *Software, IEEE*, 20(5):42–45, Sept 2003. ISSN 0740-7459. doi: 10.1109/MS.2003.1231150. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1231150.

[74] Barrie Sosinsky. *Cloud Computing Bible.* Wiley Publishing, 2011.

[75] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, HotPower'08, Berkeley, CA, USA, 2008. USENIX Association. URL http://dl.acm.org/citation.cfm?id=1855610.1855620.

[76] Thomas Stahl, Markus Vølter, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-Driven Software Development: Technology, Engineering, Management.* Wiley, 2006.

[77] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF Eclipse Modeling Framework. The Eclipse Series.* Addison-Wesley, second edition, 2009.

[78] Xin Sun, Sen Su, Peng Xu, Shuang Chi, and Yan Luo. Multi-dimensional Resource Integrated Scheduling in a Shared Data Center. In *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 7–13, June 2011. doi: 10.1109/ICDCSW.2011.27.

[79] Yifeng Sun, Yingwei Luo, Xiaolin Wang, Zhenlin Wang, Binbin Zhang, Haogang Chen, and Xiaoming Li. Fast Live Cloning of Virtual Machine Based on Xen. In *High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on*, pages 392–399, June 2009. doi: 10.1109/HPCC.2009.97.

[80] Jian Tan, P. Dube, Xiaoqiao Meng, and Li Zhang. Exploiting Resource Usage Patterns for Better Utilization Prediction. In *31st International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 14–19, June 2011. doi: 10.1109/ICDCSW.2011.53.

[81] Matias Ezequiel Vara Larsen and Arda Goknil. Railroad Crossing Heterogeneous Model. In *GEMOC workshop 2013 - International Workshop on The Globalization of Modeling Languages*, Miami, Florida, United States, September 2013. URL `https://hal.inria.fr/hal-00867316`. This research was supported by ANR GEMOC project.

[82] Paola Inverardi Vittorio Cortellessa, Antinisca Di Marco. *Model-Based Software Performance Analysis.* Springer, 2011. doi: 10.1007/978-3-642-13621-4.

[83] Martin Ward. *Proving Program Refinements and Transformations.* PhD thesis, Oxford University, 1989. URL `http://www.cse.dmu.ac.uk/~mward/martin/thesis/index.html`.

[84] Sanford Weisberg. *Applied Linear Regression.* New York : Wiley, 1985. doi: ISBN0471879576. URL `http://yorsearch.york.ac.uk/primo_library/libweb/action/dlDisplay.do?docId=44YORK_ALMA_DS21197277240001381`.

[85] James R. Williams. *A Novel Representation for Search-Based Model-Driven Engineering.* PhD thesis, Department of Computer Science, University of York, September 2013.

[86] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Bo Cheng, Zexiang Mao, Chunhong Liu, Lisha Niu, and Junliang Chen. A cost-aware auto-scaling approach using the workload prediction in service clouds. *Information Systems Frontiers*, 16(1):7–18, 2014. ISSN 1387-3326. doi: 10.1007/s10796-013-9459-0. URL `http://dx.doi.org/10.1007/s10796-013-9459-0`.