Technical Report

UCAM-CL-TR-762
ISSN 1476-2986

Number 762

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Resource provisioning for virtualized server applications

Evangelia Kalyvianaki

November 2009

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

© 2009 Evangelia Kalyvianaki

This technical report is based on a dissertation submitted August 2008 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Lucy Cavendish College.

Technical reports published by the University of Cambridge Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

Resource Provisioning for Virtualized Server Applications

Evangelia Kalyvianaki
August 2008

Data centre virtualization creates an agile environment for application deployment. Applications run within one or more virtual machines and are hosted on various servers throughout the data centre. One key mechanism provided by modern virtualization technologies is *dynamic resource allocation*. Using this technique virtual machines can be allocated resources as required and therefore, occupy only the necessary resources for their hosted application. In fact, two of the main challenges faced by contemporary data centres, server consolidation and power saving, can be tackled efficiently by capitalising on this mechanism.

This dissertation shows how to dynamically adjust the CPU resources allocated to virtualized server applications in the presence of workload fluctuations. In particular it employs a reactive approach to resource provisioning based on *feedback control* and introduces five novel controllers. All five controllers adjust the application allocations based on past utilisation observations.

A subset of the controllers integrate the *Kalman filtering* technique to track the utilisations and based on which they predict the allocations for the next interval. This approach is particularly attractive for the resource management problem since the Kalman filter uses the evolution of past utilisations to adjust the allocations. In addition, the adaptive Kalman controller which adjusts its parameters online and dynamically estimates the utilisation dynamics, is able to differentiate substantial workload changes from small fluctuations for unknown workloads.

In addition, this dissertation captures, models, and builds controllers based on the CPU resource coupling of application components. In the case of multi-tier applications, these controllers collectively allocate resources to all application tiers detecting saturation points across components. This results in them acting faster to workload variations than their single-tier counterparts.

All controllers are evaluated against the Rubis benchmark application deployed on a prototype virtualized cluster built for this purpose.

Acknowledgements

I would like to thank my two supervisors, Steven Hand and Ian Pratt. I am very grateful to Steven Hand for his invaluable suggestions on this dissertation and helping me reach the finish line. As his student I have learnt so many things in such a short time. I am also grateful to Ian Pratt for his advice and help for most of my studies. My gratitude goes to Jon Crowcroft for his comments and enthusiasm on my research over the years. Similarly, to Rebecca Isaacs for an enjoyable summer during my internship at MSR Cambridge and for her advice from then on. I am grateful to Derek McAuley and Intel Research at Cambridge for funding the first years of my studies and to Derek McAuley for interesting discussions. I would like to thank my examiners Ian Leslie and Ian Wakeman for a very interesting discussion and suggestions.

I would like to thank Themistoklis Charalambous for fruitful discussions on control theory. In particular, for his contribution on forming array M and on the MIMO-UB stability proof in Section 5.2.1. Themis also introduced me to the Kalman filters and our discussions led to the design of the related controllers.

I am indebted to Steven Hand, Rebecca Isaacs, Jon Crowcroft, Alex Ho, Themistoklis Charalambous, Eleni Kalyvianaki, Faye Chalcraft, and Giuliano Casale for reading and commenting on drafts of this dissertation.

I would like to thank my friends and best office mates I could have ever asked for, the Volta Lounge residents: Anil Madhavapeddy, Christian Kreibich, Alex Ho and honorary member Euan Harris, for sharing so many good moments. To my colleagues over the years Eric (Yu-En) Lu, Evangelos Kotsovinos, Andrew Warfield, Cali Policroniades, Christopher Clark, Grzegorz Milos, Wenjun Hu, and Michael Fetterman for their company and fruitful research discussions.

A big thank you to: Stelios, Zacharias, Irine, Katerina, Athina, Ria, Nastja, Christos, Stavros, Lene, Chae-Young, and Peri for their friendship which they "practise" either in person or remotely.

For partially funding my studies, I would like to thank: Lucy Cavendish College, Schilizzi Foundation, Cambridge Philosophical Society, Board of Graduate Studies and Jeff Buzen and the Computer Measurement Group.

I am immensely grateful to my family, George, Niki, and Eleni for their support which they demonstrate in so many ways. Thank you!

Θα ήθελα να ευχαριστήσω θερμά την οικογένεια μου Γιώργο, Νίκη, και Ελένη για την συνεχή υποστηριξή τους, την οποία δείχνουν με τόσους τρόπους. Σας ευχαριστώ!

To my parents George and Niki,
my sister Eleni,
and my grandmother Georgia.

Στους γονείς μου Γιώργο, Νίκη,
στην αδερφή μου Ελένη,
και στην γιαγιά μου Γεωργία.

Contents

Summary 3

Acknowledgements 4

List of Figures 11

List of Tables 13

1 Introduction 15
- 1.1 Motivation . . . 15
- 1.2 Context . . . 18
- 1.3 Contributions . . . 19
- 1.4 Outline . . . 19
- 1.5 Publications/Awards . . . 20

2 Background and Motivation 23
- 2.1 Server Applications . . . 24
  - 2.1.1 Architecture . . . 24
  - 2.1.2 Performance Related Terminology . . . 26
  - 2.1.3 Workload Characteristics . . . 26
  - 2.1.4 Summary . . . 27
- 2.2 Resource Management . . . 27
  - 2.2.1 Deployment . . . 28
  - 2.2.2 Dedicated Hosting . . . 29
  - 2.2.3 Shared Hosting . . . 30
  - 2.2.4 Discussion . . . 31
  - 2.2.5 Summary . . . 32
- 2.3 Server Virtualization . . . 33
  - 2.3.1 Virtualization . . . 33
  - 2.3.2 New Generation Data Centres . . . 36
  - 2.3.3 Operations . . . 37
  - 2.3.4 Challenges . . . 39
  - 2.3.5 Commercial Solutions . . . 41
  - 2.3.6 Summary . . . 42
- 2.4 Feedback Control . . . 42

2.4.1 Overview 42
2.4.2 Feedback Control for Resource Management 44
2.5 Summary 45

3 Architecture and Tools **47**
3.1 Deployment Model 47
3.2 Evaluation Platform 48
3.3 Architecture 49
3.4 Rubis Benchmark 52
3.4.1 Introduction 52
3.4.2 Tier-Layout and Request Execution Path 53
3.4.3 Client Emulator 54
3.4.4 Summary 54
3.5 Xen 55
3.6 CPU Sharing and Management in Xen 56
3.7 Summary 57

4 System Identification **59**
4.1 Introduction 59
4.2 QoS Target 60
4.3 Control Signals 64
4.4 System Modelling 64
4.5 Inter-Component Resource Coupling 67
4.6 Summary 70

5 Controllers Design **71**
5.1 Single-Tier Controllers 71
5.1.1 SISO Usage-Based Controller 73
5.1.2 The Kalman Filter 76
5.1.3 Kalman Filter Formulation 77
5.1.4 Kalman Basic Controller 79
5.1.5 Discussion 83
5.2 Multi-Tier Controllers 83
5.2.1 MIMO Usage-Based Controller 84
5.2.2 Process Noise Covariance Controller 88
5.2.3 Summary 89
5.3 Process Noise Adaptation 90
5.4 Discussion 91

6 Experimental Evaluation **93**
6.1 Preliminaries 93
6.2 Usage Based Controllers 97
6.2.1 SISO-UB 97
6.2.2 MIMO-UB 103
6.2.3 SISO-UB and MIMO-UB Comparison 107
6.2.4 Discussion 113
6.3 Kalman Based Controllers 113

6.3.1 KBC 115
6.3.2 PNCC 125
6.3.3 KBC and PNCC Comparison 127
6.3.4 APNCC 130
6.3.5 PNCC and APNCC Comparison 132
6.4 Discussion 136

7 Related Work 139
7.1 Control-Based Resource Provisioning 139
7.1.1 Single-Tier Feedback Control 139
7.1.2 Multi-Tier Feedback Control 141
7.1.3 Predictive Control 142
7.2 Filtering methods 143
7.3 Machine Learning in Resource Management 143
7.4 Resource Management in Grids 144

8 Conclusion 145
8.1 Summary 145
8.2 Future Work 147
8.3 Conclusions 149

A Steady-State Kalman Gain 151

Bibliography 153

List of Figures

2.1 Modern Enterprise Web Server 25
2.2 Operating System Server Virtualization 34
2.3 New Generation of Data Centres 36
2.4 Resource Management in Server Consolidation 40
2.5 Feedback Control System 43

3.1 Virtualized Prototype and Control System 49
3.2 Resource Management Architecture 50
3.3 Controller and Measurement Intervals 51
3.4 Rubis Tier Layout 53
3.5 Xen Architecture Layout 55

4.1 System Identification 61
4.2 System Identification (Response Distributions Summary) 62
4.3 System Identification (Tomcat CPU usage Distributions Summary) 63
4.4 `Extra` Allocation 66
4.5 Inter-Component Resource Coupling Example 68
4.6 Inter-Component Resource Coupling Experiments 69

5.1 SISO-UB Controller Layout 74
5.2 Kalman Filter Overview 78
5.3 KBC Controller Layout 80
5.4 MIMO-UB Controller Layout 84
5.5 PNCC Controller Layout 88
5.6 APNCC Controller Layout 90

6.1 SISO-UB Controllers Performance 99
6.2 SISO-UB Allocations for Stable Input. 103
6.3 MIMO-UB Controller Performance 104
6.4 MIMO-UB Allocations for Stable Input. 107
6.5 SISO-UB and MIMO-UB Comparison 108
6.6 SISO-UB and MIMO-UB Comparison for Different Parameter Values . . 110
6.7 Values of Utilisation Variances and Covariances 115
6.8 KBC Performance for Stable Workload and Q_0 Values 116
6.9 KBC Performance for Stable Workload and $Q_0/400$ Values 118
6.10 KBC Performance for Stable Workload and Different Q Values 119

6.11 KBC Allocations for Variable Workload and two Q Values 121
6.12 KBC Server Performance for Variable Workload and two Q Values 122
6.13 KBC Performance for Workload Increases and Different Q Values 124
6.14 Settling Times and Overshoot for KBC Controllers. 124
6.15 PNCC Performance for Variable Workload and $\mathbf{Q_0}/400$ Values 126
6.16 PNCC Allocations for Stable Input. 127
6.17 PNCC and KBC Comparison for E1(600,200) and Different x 128
6.18 PNCC and KBC Comparison for E2 Experiments and Different x 129
6.19 APNCC Performance for Variable Workload and $\mathbf{Q}/40$ Values 131
6.20 APNCC Gains for Variable Workload and $\mathbf{Q}/40$ Values. 132
6.21 PNCC and APNCC Comparison for E0(40,80,120) Experiments 133
6.22 PNCC and APNCC Comparison for E2 experiments 134
6.23 APNCC and PNCC Kalman Gains for $x = 8$ and $x = 400$ 135

List of Tables

4.1 Parameters of the Models of Components' Utilisation Coupling. 70

5.1 Controllers Notation. 72
5.2 Classification of Controllers 92

6.1 Performance Evaluation Experiments 94
6.2 Performance Evaluation Metrics 95
6.3 Input Parameter Configuration for SISO-UB Controllers 101
6.4 Input Parameter Configuration for MIMO-UB Controller 106
6.5 Experiment Description for the MIMO-UB and SISO-UB Comparison. . 110
6.6 `Additional` Resources Comparison 111
6.7 UB Controllers Comparison, SISO-UB $\lambda = 0.45$, MIMO-UB $\lambda = 0.12$ . . 112
6.8 UB Controllers Comparison, SISO-UB $\lambda = 0.75$, MIMO-UB $\lambda = 0.2$ . . . 112
6.9 KBC Server Performance for Stable Workload and Different x. 119

1 Introduction

This dissertation is concerned with the dynamic CPU resource provisioning of multi-tier virtualized server applications. It introduces feedback controllers that adapt resources based on recent observed utilisations, allowing server applications to meet their performance goals in the presence of workload fluctuations, while at the same time, resources become available for other applications to use.

This chapter starts by motivating the current problem (Section 1.1). It then provides the context of the work (Section 1.2) and enumerates the contributions in Section 1.3. Finally, it presents the structure for the rest of this dissertation in Section 1.4 and lists the publications/awards related to this work in Section 1.5.

1.1 Motivation

Resource Provisioning in Data Centres

Modern omnipresent server applications are complex programs that provide diverse services to thousands of users. Their demanding operations require a powerful deployment base delivered from contemporary data centres which are equipped with hundreds or thousands of commodity machine units. Machine resource provisioning is *central* for each application to comply with its service level agreements and to efficiently administer data centre machines hosting multiple applications.

Commonly, due to the low costs of commodity hardware, a set of machines is dedicated to a single application. The size of the machine group is subject to the resource demands

of the application. A common practice is to over-provision applications to cope with their most demanding workloads, however rare they may be. Other factors include load balancing and machine failure recovering. This resource management scheme, that hosts applications in non-overlapping sets of machines, provides *performance isolation* and *performance guarantees* for the applications.

Machine dedication in conjunction with over-provisioning has caused several implications stemming from the ever increasing data centre size required to host more applications that grow in size and complexity. Several reports show that machines are under-utilised most of the time. As quoted in [ser08] *"According to Tony Iams, Senior Analyst at D.H. Brown Associates Inc. in Port Chester, NY, servers typically run at 15-20% of their capacity"*. In addition, an IDC 2007 report [IDC07] shows that current enterprises have already spent $140B (capital expenses) more than needed to satisfy their current needs[1]. Furthermore, as the number of applications grows, the power and cooling costs for the servers increase too. IDC [SE07] reports that for every $1.00 (capital expenses) spent on new server equipment, another $0.50 is spent on power and cooling expenses.

To alleviate these issues data centres need fewer but more utilised machines. A practice known as *server consolidation* increases machine utilisation by running multiple applications on the same host machine. However, in the case of applications with strict performance requirements, resource sharing is efficient only when mechanisms that ensure performance isolation among the running applications exist. Without these means, server consolidation becomes inadequate to support application performance constraints.

Data centre machines usually run general purpose operating systems which lack these mechanisms. Recognising consolidation's potential, some have proposed prototype frameworks that support resource sharing for clusters running general purpose operating systems (e.g. [USR02, US04]). Despite these attempts, server consolidation in traditional data centres is not fully exploited or operates with non-strict application performance isolation.

Resource Management in the Virtual World

Recent advances in virtualizing commodity hardware (e.g. [BDF+03]) are changing the structure of the data centre. A physical machine is transformed into one or more virtual ones, each capable of hosting a different application. Each virtual machine is subject to management operations, such as, creation, deletion, migration between physical machines, as well as run-time resource allocation. Virtualizing the data centre enables resource sharing in arbitrary combinations between applications and physical servers and is now regarded as the key technology towards achieving efficient server consolidation. This is because virtualization: (a) is transparent to the applications since the underlying virtual machine monitor handles resource multiplexing; (b) provides almost native performance to virtual machines; (c) ensures performance isolation since each virtual machine is guaranteed resources; and (d) is widely applicable, as virtual machines run heterogeneous operating systems.

[1] Information taken from [vmw08a].

To capitalise on this technology, it is *essential* to adaptively provision virtualized applications with resources according to their workload demands. It is known that server applications operate under fluctuating workloads [AJ00, Vir02] that cause diverse and changing resource demands to their components. Adjustable resource allocations that follow the workload fluctuations for virtualized application components are important to create a high performance server consolidation environment. In this case, each application is provisioned resources as required, and therefore, there could be free resources for other applications to use. Efficiently managing virtual machine resources is also important for other high level management tasks in addition to server consolidation, such as power management and load balancing.

Current commercial resource management tools provide only partial solutions to this problem. VMware [vmw08b] and XenSource [xen08a], the two leading vendors in modern virtualization technologies, offer tools such as, the VMware Distributed Resource Scheduler (DRS) [vmw08d] and XenCenter [Xen08b] respectively, which provide resource management capabilities by forcing virtual machines allocations to be within certain limits. However, these tools do not address setting these limits with appropriate values for each application, or how they should be changed in case, for example, an application requires more resources than the upper limit. This dissertation address these limitations and builds resource management tools that dynamically adapt CPU allocations to workload fluctuations.

Resource Management and Feedback Control

There are two general approaches to dynamic resource allocation: *proactive* and *reactive*. Proactive allocation is based on resource utilisation predictions. In this case, utilisation patterns are learnt in advance and allocations are accordingly adjusted prior to any change. When predictions are accurate, this scheme provides very good performance (e.g. [XZSW06]). However, it fails when predictions are not possible (for instance when deploying a new application, or when the utilisations do not follow any predictable patterns) and/or are inaccurate (for instance in the case of unprecedented sharp changes in workloads, e.g. flash crowds, or in the case of very noisy workloads). In addition, utilisation predictions are expensive since they require workload data analysis and storage space.

With reactive schemes, allocation is adjusted on-demand based solely on recent behaviour; a change is detected and the allocations are adjusted accordingly. Reactive allocation is computationally attractive since it does not require extensive knowledge of the application's resource demands. However, its efficiency in practice depends on its ability to detect changes and adjust allocations in response to them in a timely fashion while smoothly handling transient fluctuations.

This dissertation employs a reactive allocation approach. It uses feedback control-based allocation that embodies the essence of reactive allocation. In parallel with this dissertation others have also approached the problem of resource provisioning of virtualized server applications using feedback control (e.g. [PSZ+07, WLZ+07]). This dissertation builds a set of controllers that provide a range of solutions for virtualized server applica-

tions, with diverse workload characteristics. In particular, it uses a filtering technique to deal with noisy utilisations and address the characteristics of multi-tier applications.

1.2 Context

This dissertation presents basic tools for the realisation of resource management in virtualized applications. This section discusses the context of the current work with respect to the ways it can be used for data center management.

The current controllers allocate CPU resources to virtualized server components using an adaptive upper-bound threshold approach; every time interval they adjust the maximum CPU allocation each virtualized component uses based on the application performance model. The purpose of this approach is to constanlty provision each virtualized application with resources to meet its performance goals. Additionally, this mechanism makes the available free resources easy to calculate and further use for co-hosting more applications subject to the total machine physical capacity.

Examples below indicate they way the current controllers can be used in conjunction with other tools for further data center management. These examples do not provide an exhaustive list of solutions, rather, they are used to further motivate the current controllers.

Assume a multi-step control architecture for data center management: at first, low-level CPU allocation controllers like the ones of this dissertation adjusts the allocations of the application according to its requirements and, later, high-level tools dictate the application placement on machines for resource sharing.[2] The controllers of this dissertation implement the low-level resource allocations and are built with characteristics towards facilitating the high-level tools. For instance, certain configurations in the case of the Kalman controllers presented later in Chapter 5 enable smooth CPU allocations despite transient fluctuations of the workload. This creates stable allocations when the workload is relatively stable with small fluctuations, as well as important allocation changes to happen only when the workload substantially changes. In this way, the high-level tools could operate under a relatively confident manner that the allocation of a running application changes only when absolutely necessary and, therefore, they could make plans to shuffle applications among machines based on the available free resources.[3]

In addition to smooth allocations, the controllers of this dissertation can also operate in a different way and allocate CPU resources in a way that reflects every resource requirement by the application at every time interval. In this case, it is hard for the management tools to plan for QoS driven applications placement since the available free resource

[2]Such schemes have been proposed, for example, Padala *et al.* present a two-layered controller to regulate the CPU resources of two instances of two-tier virtualized servers co-hosted on two physical servers [PSZ+07]. The first layer controller regulates the resources for each server tier and the second layer controller adjusts the allocations in cases of contention.

[3]There are of course several additional issues to be considered by the high-level tools, such as, resource contention, migration costs, etc.

could change frequently based on momentarily fluctuations. However, there are applications that could still benefit for this type of allocations. Consider for instance a CPU-intensive scientific application with no strict QoS requirements. This kind of applications can use CPU resources as they become available and can be used for example in cases where the free resources are not enough to host another application with QoS requirements. In fact, the high-level tools can switch between the different low-level tools based on the availability of resources and application requirements.

The controllers of this dissertation focuss on providing adaptive CPU allocations for virtualized server applications. This is an integral part to data center management and can be used as standalone tools or in conjunction with high-level tools for data center management.

1.3 Contributions

The work of this dissertation is evaluated based on a prototype virtualized cluster created for this purpose. The cluster consists of four server machines. Three of them run the Xen virtualization technology and host the multi-tier Rice University Bidding System (Rubis) benchmark [ACC+02]. Rubis is a prototype auction web site server application which models eBay.com and implements the basic operations of such a site: selling, browsing, and bidding. The fourth machine runs the Rubis Client Emulator that emulates clients generating different types of requests to the application. The cluster uses a prototype implementation of the resource management control software that monitors components' utilisations and remotely controls their allocations.

In particular, this dissertation makes the following contributions:

1. a general architecture for resource management of virtualized server applications;
2. a system identification analysis for multi-tier virtualized server applications. Analysis shows that there exists (a) a correlation between the resource allocation and the CPU utilisations and (b) a resource coupling between components;
3. a black-box approach to modelling resource coupling of virtualized server components;
4. the integration of the Kalman filtering technique to feedback resource provisioning;
5. controllers that manage the CPU allocations of individual virtual machines; and
6. controllers that collectively manage the CPU allocations of server components.

1.4 Outline

Chapter 2 further motivates the work of this dissertation and describes the background. In particular it elaborates on four areas that constitute the current context, namely server applications, resource management, server virtualization and feedback control.

Chapter 3 describes the resource management architecture and all the components of the evaluation platform. It describes the prototype cluster, the Xen virtualization technology, and discusses resource management related issues. Finally, it presents the multi-tier Rubis benchmark server application.

Chapter 4 discusses the system identification analysis upon which all controllers are based. The application is subjected to different workload conditions and three system models are defined. The additive and the multiplicative model describe the relationship between allocation and utilisation for maintaining good server performance. The resource coupling between components' utilisation is also identified. These models are used to build the controllers in the next chapter.

Chapter 5 presents five novel CPU allocation controllers. Two controllers, the SISO-UB and the KBC, allocate CPU resource to application tiers individually. The KBC controller in particular is based on the Kalman filtering technique. The other two controllers, the MIMO-UB and the PNCC, allocate resources to all application components collectively. The MIMO-UB controller is based on the resource coupling model identified in the previous chapter. The PNCC controller extends the KBC design to multiple components. Finally, the APNCC controller extends the PNCC to adapt its parameters online.

Evaluation results on all controllers are shown in Chapter 6. Evaluation is performed on a per-controller basis and comparisons between controllers are also given.

Chapter 7 covers recent related work in the area of resource provisioning for single-tier and multi-tier virtualized server applications. In addition, it discusses filtering methods used for resource allocation in resource sharing environments. Furthermore, it presents other methods for performance modelling based on machine learning. Finally, it introduces concepts of resource management in the Grid environment.

Finally, Chapter 8 provides a summary, conclusions, and discusses future work.

1.5 Publications/Awards

The following publications/awards are based on this dissertation:

1. Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters. To appear in the *6th International Conference on Autonomic Computing and Communications (ICAC)*, 2009.

   This paper presents the Kalman filtering based controllers and extensive evaluation.

2. Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Applying Kalman Filters to Dynamic Resource Provisioning of Virtualized Server Applications. In *Proceedings of the 3rd International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2008)*, 2008.

This paper presented the Kalman filtering based controllers and preliminary evaluation.

3. Evangelia Kalyvianaki and Themistoklis Charalambous. On Dynamic Resource Provisioning for Consolidated Servers in Virtualized Data Centres, In *Proceedings of the 8th International Workshop on Performability Modelling of Computer and Communication Systems, (PMCCS-8)*, 2007.

   This paper presented initial system identification analysis, the MIMO-UB, the KBC controllers and some preliminary results.

4. The proposal of this dissertation titled "*Tuning Server Applications with OS Virtualization Support using Feedback Control*" is awarded the Computer Measurement Group (CMG) Graduate Fellowship 2006. The proposal also suggested the migration of server components in shared clusters to maximise total resource utilisation.

5. Parts of this dissertation will be published on *The Computer Measurement Group (CMG) Journal*.

2

Background and Motivation

During the past three decades, server applications have evolved with respect to both increasing complexity and resource requirements. At the same time, the hardware and software technological developments of the underlying hosting platforms have led to different resource management capabilities. This chapter follows the parallel development of server applications and hardware/software advances from single machine hosting to modern virtualized data centres. Modern server virtualization provides the mechanisms for secure and adaptive resource sharing of server applications while guaranteeing performance. However, as server applications exhibit diverse resource demands, efficient use of these mechanisms for high performance server virtualization imposes unique challenges.

This chapter presents the background and motivation of this dissertation and identifies *resource management* as an integral part of high performance server virtualization. Here, a control-based direction towards adaptive resource provisioning is employed; its motivation is given towards the end of this chapter.

In particular, this chapter is divided into four sections, which correspond to the four different dimensions that shape this dissertation:

1. *Server Applications.* Section 2.1 provides an overview of the architecture and workload characteristics of server applications.
2. *Resource Management.* Section 2.2 identifies the significance of resource management in making efficient use of data centres while achieving individual application performance goals. This section also describes the two models of hosting, dedicated and shared, and presents their advantages and limitations in traditional data centres.

3. *Server Virtualization.* The next section presents virtualization and its use in modern data centres. This section shows that virtualization provides the means for efficient resource sharing.
4. *Control Theory.* Finally, Section 2.4 motivates the use of control theory to obtain efficient resource management.

2.1 Server Applications

Server applications are programs designed to provide access to data and/or execute operations on it on behalf of a group of users, referred to as *clients*. They are widely deployed — for instance, Netcraft reports almost 65 million active web sites [net08], hosted by a huge number of web servers — and perform a variety of diverse operations. There are many types of servers, e.g. e-commerce, video-streaming, database, corporate-specific, file-system servers, and so on.

One of the most common server applications is a web server. A simple web server stores and provides access to information in the form of static HTML web pages. The server application executes on one or more machines and clients access its resources using the Internet. Using the server's publicly-known URL and the HTTP protocol, a client requests some content. Upon receiving the request, the server retrieves the requested document from its storage and sends back the response to the initiating client. Today's web servers are very complex applications that provide a wide range of services ranging from retrieving static HTML pages to uploading data or accessing multimedia content.

The rest of this section presents the characteristics of server applications related to the context of this dissertation and defines the terminology. It starts by describing the multi-tier server architecture (Section 2.1.1), proceeds with the metrics used to characterise performance (Section 2.1.2), and concludes by presenting the unique workload characteristics of web server applications (Section 2.1.3).

2.1.1 Architecture

One of the main characteristics of server applications' internal architecture is their modularity. The *multi-tier* model has become the predominant way of building server applications, where the application logic implementation is distributed into several different parts, referred to as *tiers* or *components*. A web server application typically employs the three-tier model which consists of (a) the *client-side* user interface tier, used by the clients to issue requests to the server, (b) the application logic tier responsible for the server specific functions such as manipulating user data and executing actions upon them on behalf of the clients, and (c) the storage tier which handles the server data. The last two tiers are also referred to as the *server-side* tiers.

As server applications grow in complexity by offering different services to clients and grow in size by serving thousands of requests per second, the diversity and the number of server-side tiers also increases. For example, the application-logic tier can be further

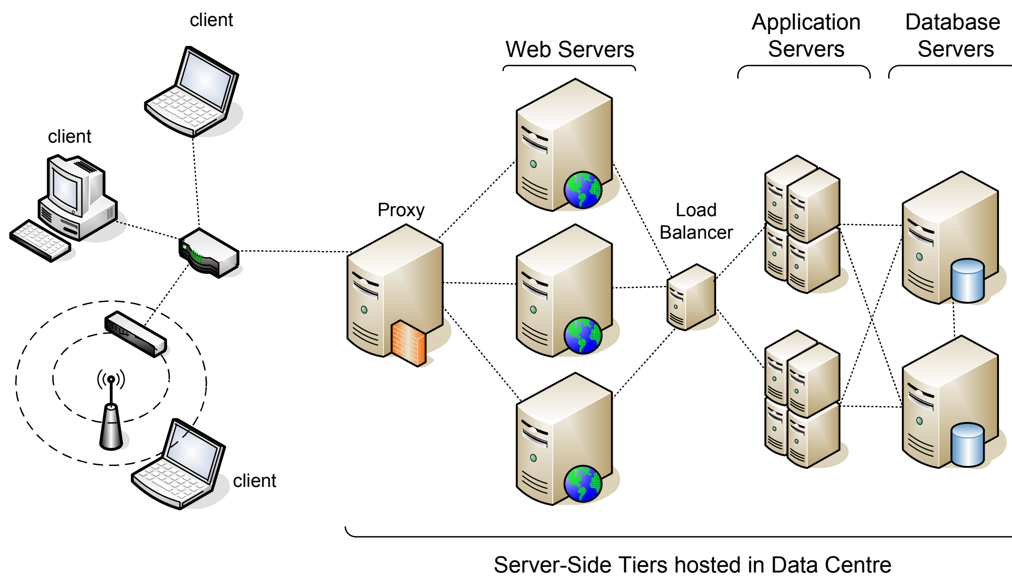


Figure 2.1: Modern Enterprise Web Server. Today's server applications employ a multi-tier architecture to cope with the complexity of their operations. Server components span across multiple tiers and components within each tier could also be replicated to face demanding workloads. Server-side tiers are hosted within a data centre.

divided into a web server tier[1] handling the HTTP incoming requests and one or more application-logic tiers, each one implementing different server functions (Figure 2.1). In this case, the server application is composed from one or multiple components.

In this dissertation, the *application tier layout* is defined as the group of server-side tiers that the server application is composed from. The exact tier layout is the result of a wide spectrum of decisions that span the application's lifetime. For instance, there is a large range of application and database servers, e.g. JBoss [jbo08], Jonas [jon08], MySQL [mys08], and Oracle [ora08], from which the application architect can choose based on both the application's and the servers' specifications. Each decision might result in a different tier layout. Also, the server administrator can choose to replicate tiers either in advance or at run-time in order to accommodate changing workloads. The process of defining the application tier layout has been the subject of research for many years and continues to face new challenges as server applications and available middle-ware platforms become more complex. This dissertation focuses on applications with static tier layout.

[1]Note that the term *web server* is used in two ways: (a) to denote the type of the application itself and (b) to identify the first tier of the application.

2.1.2 Performance Related Terminology

Server application load is usually described and measured by its *workload*. The workload is a set of parameters and their values that describe various client and application operations over a time interval. For example, in an online bookstore server, two workload parameters could be: (a) the number of buy requests for travel books per day, and (b) the web server's CPU utilisation per minute.

The term workload is also loosely used to characterise the overall behaviour of the client-server application. When the client requests do not significantly vary in frequency, the server is said to be under a *stable workload*. In contrast, when the request types or frequency change over time, the server is under a *dynamic or variable workload*. In addition, the term *workload demand* refers to the resource requirements needed to serve client requests.

The performance of a server application is usually measured by its *throughput* — the number of completed requests per time interval — and/or request *response times* — the time elapsed between the arrival of a client request to the server and the server's response to the client. It is important to sustain the performance of commercial web servers at certain levels. In fact, there are dedicated contracts, called *Service Level Agreements* (SLAs) [MA98, page 101] that denote the *Quality of Service* (QoS) level the server should provide to its clients. The QoS is expressed by a number of performance/workload metrics and their appropriate values. For example, in a news web server, a clause in an SLA could be: 90% of requests accessing news in the economic sector must have response times less than 2 seconds. SLAs are used in commercial hosting providers to charge server applications for their resources. A SLA violation indicates that the required resource provisioning is not adequate and that further capacity planning is needed.

2.1.3 Workload Characteristics

Web server applications exhibit highly dynamic, diverse and bursty workloads. The arrival rate and the types of server requests vary in time causing variable resource demands at the server tiers. In particular special patterns on the requests' arrival rates have been observed during the same hours every day. For example, the load to a news web server increases during the afternoon hours. Additionally, extreme server loads also occur in the form of *flash crowds* where an unusual increase in the number of clients causes unique resource demands at the server side. Some flash crowds have been observed because of very popular, known in advance events, such as the World Cup [AJ00] or the Olympic Games [ICDD00, page 17]. There are also cases, however, where unprecedented events, such as the September 11th attack in the United States or a very important political event, might cause extreme loads to a news web server. Even under "normal" conditions, web servers have bursty resource utilisations across their tiers due to the variation of the operations over incoming requests.

2.1.4 Summary

This section introduced the architecture and the workload characteristics of web server applications. Although emphasis so far has been given to the description of web servers, this dissertation is concerned with the resource management of any server application with similar architecture and workload characteristics. The next section discusses the importance of resource management for efficient server applications and provides an overview of two different approaches developed over the years.

2.2 Resource Management

Resource management or *resource provisioning* is one of the most important tasks in server application deployment and management. It involves the suitable provisioning of resources — CPU time, memory, disk, and network bandwidth — in order for the application to meet its QoS performance goals in the presence of time-varying workloads. Unless properly provisioned to capture changing resource demands, applications fail or delay serving incoming requests with consequences such as loss of revenue (e.g. in e-commerce applications). The following are examples of resource provisioning questions:

1. What is the change in CPU demand as the number of clients in a video distribution server increases by 20% within a period of 5 minutes?
2. What are the memory resources required across all server tiers for all the requests that involve purchasing an item in an e-commerce server?

Planning for the server resources involves two main steps: *workload characterisation* and *system modelling*. The types and characteristics of incoming requests are analysed and modelled in workload characterisation, while in system modelling, a model of the application's resource demands is derived. Specifically, system modelling is a process that associates the server's operations (e.g. request serving path) and physical characteristics (e.g. number of CPUs) with various performance metrics (e.g. throughput, response time). The modelling method, the level of model detail, and the performance metrics all depend upon the specific resource provisioning task in question and the available tools. Together, system modelling and workload characterisation provide a thorough view of the server's performance and identify the major contributing blocks.

Resource provisioning is achieved either in a *proactive* and/or in a *reactive* manner. In proactive allocation, all resources are provided in advance of workload changes, which can be predicted with workload forecasting. Future request demands are predicted and, using the system model, the corresponding resource demands are estimated. Resource provisioning is traditionally linked to the proactive way of resource estimation. However, since accurate workload forecasting is not always possible, resource corrective actions can also happen in a reactive manner. In reactive allocation, resources are updated after a workload change is detected. In either case, the aim is to update the resource allocations in a *timely* fashion by minimising the deviation from the QoS performance goals during and after the workload change.

In addition, resource provisioning highly depends on the server's workload. If the workload's resource demands exhibit small variation, the allocation of resources is a simple process. Using either off-line or on-line monitoring methods, utilisations are measured, and the relating allocations are set to appropriate levels. In a dynamic workload case, however, utilisations fluctuate over time, making any static allocation scheme inadequate.

The rest of this section starts by presenting the deployment infrastructure of modern server applications (Section 2.2.1). It then overviews the two main resource allocation schemes adopted throughout the years — dedicated hosting (Section 2.2.2) and shared hosting (Section 2.2.3) — and concludes by evaluating the efficacy of the different schemes across several dimensions (Section 2.2.4).

2.2.1 Deployment

In the 1970s, server applications were run on mainframes. They were deployed by large institutions, corporations, and governments, which could afford the expensive mainframes.

In the mid 1980s, there was a shift in both the types of deployed server applications and the available hardware solutions for hosting them. With hardware becoming less expensive, personal computers (PCs) appeared. A few years later, small server applications (when compared to the ones running in large corporations) appeared on the Internet. In addition, many small to medium enterprises deployed server applications to manage their own data.

Today, hardware has become increasingly less expensive and more powerful. In particular, commodity server machines can be used as a group to host server applications with diverse and demanding computational and storage demands. With this cheap solution at hand, enterprises adjust to computational demands simply by adding or removing server machines as needed.

Modern server applications are complex programs that offer diverse functionality and span across several machines (Figure 2.1). To address management and power considerations, server machines are usually deployed in dedicated places equipped with special cooling facilities, called *data centres*. In addition to corporate owned data centres, there are specialised companies that provide hardware resources under payment for third-party server deployment, called *hosting platforms*. A hosting platform can be used in many different ways: (a) by a company that lacks the resources to deploy and manage its own data centre, (b) by a company that wishes to outsource some of its services, (c) for handling excess service requests, and (d) for taking advantage of the geographical position of the platform (e.g. placing the server application closer to the end users).

This section continues by describing the different deployment models used to host server applications in modern data centres. There are two different types of hosting platforms: (a) *dedicated hosting*, where disjoint sets of machines are dedicated to different applications, and (b) *shared hosting* where applications are co-located on the same machines and share physical resources.

2.2.2 Dedicated Hosting

As mentioned earlier, server applications exhibit highly unpredictable and bursty workloads. To avoid dropping incoming requests, a simple approach to resource provisioning has been adopted the last few years: a dedicated group of server machines is used for each application with enough total resource capacity to accommodate requests at peak rates. Consider, for example, a news web server in which incoming requests have been shown to follow the time-of-day effect: between 5pm and 11pm, the number of incoming requests is almost twice as much as during the rest of the day. Performance analysis has shown that during peak times the application requires two server machines, while the rest of the day one will suffice. To provide 24 hour availability over the server's content, the simplest approach to resource management is to always dedicate two machines. Although some of the resources are under-utilised most of the time, this approach is easy to implement and relatively cost effective because of the low price of commodity server machines. This is a very simple and administratively efficient approach. If the current server machine group is still not adequate to serve incoming requests, another machine is manually added to the group, or an upgrade is scheduled.

Dedicated hosting is widely adopted because of its simplicity and the low cost of commodity machines. Nevertheless, it constitutes a rigid deployment model, and to this end additional techniques such as: load balancing in replicated tiers; admission control in overload conditions; and dynamic server distribution have been developed to enhance its performance and flexibility. These are further discussed below.

Servers at any tier might be replicated to cope with increasing number of requests (Figure 2.1). Load balancing techniques have been developed to efficiently distribute the load of incoming requests among the available machines for low response times and high throughput. A number of different dispatching techniques in cluster-based server applications have been developed to route requests to the front-tiers [ZBCS99, ASDZ00, CRS99] or back-end database servers [PAB+98, ACZ03].

Despite over-provisioning, there may be cases where the request load exceeds the overall server capacity and where augmenting the computing power is not feasible. In order to sustain a manageable resource utilisation of the server resources and achieve controlled performance, admission control techniques have been developed to manage the rate and the type of incoming requests that eventually get forwarded to the server, e.g. [ENTZ04, KMN04, WC03]

In general, large data centres offer dedicated hosting to applications from many different parties. To ensure efficient and flexible use of machines, there are different techniques that dynamically allocate servers amongst the running applications. For example, the *Océano* prototype [AFF+01] aims to re-distribute servers among multiple web applications based on their resource usage and their SLAs. It collects information regarding the servers' load and predefined performance metrics, such as overall response time. Based on this information and when a SLA is violated, corrective actions (e.g. re-distribution of servers) are performed. *Cluster-On-Demand* (COD) [MIG+02] further extends Océano's concept to provide a pool of servers for applications with diverse software requirements. An application runs on an isolated group of servers, called a *virtual cluster*, with com-

plete control over its resources. Resources are managed locally within each virtual cluster, while nodes' allocations and releases are performed in coordination with the central COD management.

Summary

To summarise, dedicated hosting provides performance guarantees among the running applications. A high QoS is achieved, as each application is provisioned with resources to meet its resource close to its peak loads. In addition, isolation among applications is sustained since a hardware or a software failure, or a denial-of-service attack on any of the running applications does not affect the rest, since they use disjoint sets of servers.

However, there are two main drawbacks to dedicated hosting. First, it is commonly found that commodity server machines run at a low 15-20% utilisation rate due to over-provisioning (e.g. from [sys08] and references therein). Second, there are cases where the number of applications exceeds the number of physical servers in dedicated hosting platforms. To address these limitations, shared hosting is used.

2.2.3 Shared Hosting

With shared hosting, applications are co-located on server machines sharing the physical resources. This practice is also referred to as *server consolidation*.

The simplest way to perform application consolidation is to co-host applications on the same machine running a general purpose operating system, provided that the machine is equipped with the proper libraries and software packages required by all applications. For instance, two different database applications can be hosted on the same database server. In this way, (a) server utilisation is increased by increasing the number of running applications; (b) management is easier as the administrator needs to update and maintain one hardware and software platform instead of two; and (c) licence costs are decreased as the two applications share the same database server licence. However, unless this scheme is equipped with QoS differentiation mechanisms, one of the applications could monopolise the use of physical resources, leaving the second one starving.

High performance server consolidation requires performance isolation among the running applications, and mechanisms for QoS differentiation. Recognising the potential of server consolidation, some research in the late 1990s and early 2000s studied various aspects of it, such as (a) enhancing general purpose operating systems with resource schedulers for performance isolation; (b) detailed resource accounting for server activities; and (c) developing mechanisms for resource sharing across clusters. These contributions are discussed in the next three paragraphs.

Resource schedulers that guarantee performance isolation among applications are important to resource sharing as they ensure that each application is allocated resources as required. Many different schedulers that provide QoS guarantees have been developed for allocating CPU time [JRR97, LMB$^+$96, WW94], network [GVC96] and disk bandwidth [SV98], and for managing memory [VGR98, Wal02]. Although these schedulers

were initially developed for multimedia and real-time operating systems, they have also been used in the context of resource scheduling for co-hosted server applications on single machines and shared clusters [US04].

QoS differentiation uses resource accounting to schedule running applications. Fine-grained resource accounting for operations in multi-threaded servers running on a general purpose operating system is challenging. A client request usually invokes several user-level server threads and kernel activities. As general purpose operating systems usually provide resource accounting at the process granularity, methods have been proposed to accurately account for the resource utilisation from all the different entities involved in a single server operation. *Resource containers* [BDM99] are abstract entities used to account for system resources (e.g. CPU time, memory, sockets, etc) associated with a particular server activity such as serving a client request in monolithic kernels. The SILK mechanism [BVW+02] creates a *vertical slice* entity that encapsulates the flow of data and the processing actions associated with a networking operation of sending and receiving packets from the network based on the Scout operating system *paths* [MP96]. Similar accounting mechanisms have been developed for real-time operating system (e.g. *activities* in Rialto [JLDJSB95]) and multimedia operating system (e.g. *domains* in Nemesis [LMB+96]).

Finally, different approaches to resource sharing for applications deployed on cluster machines were also developed. Urgaonkar *et al.* [US04] propose the *Sharc* system that manages CPU time and network bandwidth across applications in shared clusters. Sharc provides a generic mechanism for multi-component application placement on cluster machines and resource trading across components of the same application. Aron *et al.* [ADZ00] extend the notion of resource containers in single machines for clustered applications by introducing *cluster reserves*. They aim to differentiate between classes of requests. Their system maintains global allocations per service class while adjusting the allocations per node as indicated by local utilisations, where service classes "steal" resources from under-utilised classes. Chase *et al.* [CAT+01] present the *Muse* system that manages resource sharing among shared servers with emphasis given to energy management. Muse uses an economic model to efficiently allocate resources which maximises the performance gain for each application while minimising the necessary power.

2.2.4 Discussion

Despite some specific solutions to resource sharing, today's data centres still suffer from under-utilisation with further implications to other areas such as energy consumption. The problem escalates as server applications grow in both size and complexity, and so does their deployment base; reports show that the current installed base is around 35 million [SE07, Figure 1 from IDC, 2007]. This section outlines the problems faced by contemporary data centres.

Resource Under-Utilisation

Numerous reports show that current data centres are poorly utilised. In fact many of today's commodity server machines are known to use only $15-20\%$ of their CPU capacity. This is mainly the result of application over-provisioning: server machines are allocated to cope with demanding but infrequent workloads. As a result, most of the time the server application runs with average workloads and the server machines are under-utilised. An IDC 2007 report [IDC07] shows that current entrerprises have already spent $140B (capital expense) more than needed to satisfy their current needs; or differently, the current infrastructure can in principle support application needs for the next three years, without purchasing any additional servers[2].

Management Expenditures

As applications grow in both size and complexity, the administrative costs for the servers increase too. IDC [IDC07] reports that for the years between 2001 and 2007, almost 50% of the total spendings on the server market worldwide was for people related administrative costs[3]. The same report predicts that this percentage will further increase over the next 4 years. As servers number increase and applications become more demanding with complex workloads, managing the data centres to deliver QoS has proved to be a challenge. Human intervention is necessary to configure the machine infrastructure and the applications themselves to achieve the QoS goals set. In addition, further maintenance for server machines is required to keep them aligned with the latest operating system and application server software patches and updates for both performance and security reasons.

Energy Consumption

IDC [SE07] reports that for every $1.00 (capital expense) spent on new server equipment, another $0.50 is spent on power and cooling expenses. According to the same report, this amount has increased over the last years and is predicted to further increase in the next three. In fact, the power allocation scheme follows the capacity planning model, according to which, power is allocated to cope with the most demanding workloads. This results in peak power consumption, while the servers are under-utilised most of the time. Finally, part of the administrative costs are due to managing energy in data centres and finding ways to reduce the energy costs.

2.2.5 Summary

Resource management of server applications is challenging because of the time-varying and bursty workloads that cause diverse resource demands across tiers. There are two

[2]Information taken from [vmw08a].
[3]Information taken from [CMH08, Figure 1-1].

main models of deployment in data centres. Dedicated hosting offers performance isolation among running applications, yet, due to over-provisioning, resources are underutilised. In shared hosting, applications are co-located and share the physical resources increasing the overall resource utilisation. Despite the benefits from shared hosting, current ad-hoc solutions are not widely adopted and data centres face a significant amount of loss in revenue with consequences in energy and management costs.

Nowadays, server virtualization is considered as a means to combine the advantages of both dedicated and shared hosting: on one hand benefiting from performance isolation and on the other increasing the resource utilisation. The next section introduces virtualization and discusses the ways it is shaping future data centres.

2.3 Server Virtualization

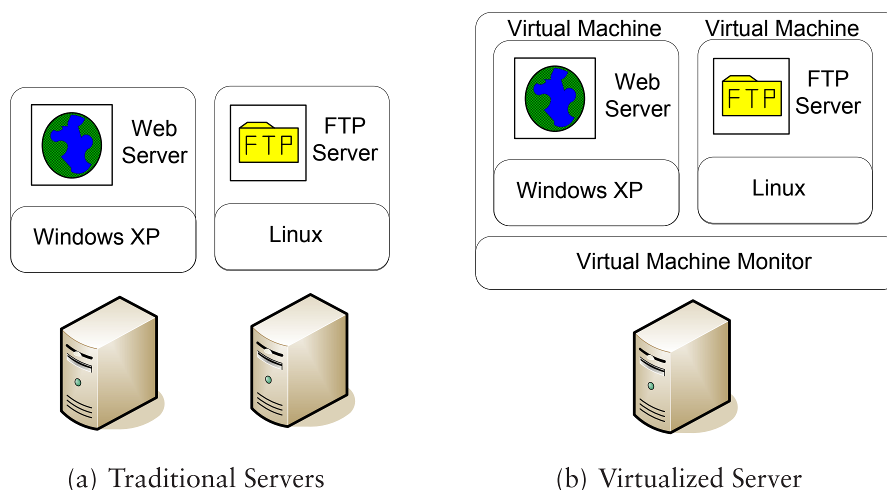
Server virtualization constitutes an abstract and secure method for server applications to share physical resources. This section starts by introducing the concept of virtualization and then discusses modern system-level virtualization technologies. It then provides an overview of next generation of data centres and concludes by identifying *adaptive resource management* as an integral part of efficient server virtualization.

2.3.1 Virtualization

Virtualization is a technique that transforms physical resources into one or more logical versions that can be used by end users or process applications in exactly the same way as if traditionally using the physical ones. A good example of this technique is memory management in an operating system. Virtualization is used to allow multiple processes to simultaneously access the physical memory in a secure and transparent way via the concept of virtual memory. The physical memory is mapped onto multiple virtual address spaces, one for each process. Each process uses pages from its own address space and behaves as if it owns all of the physical address space. The memory manager is responsible for the translation between the virtual and the physical space; it ensures isolation between the processes and provides an abstract way for each process to access physical memory.

History

System virtualization or *operating system virtualization* has its origins in the *time-sharing* concept for mainframes, which appeared in the late 1950s. In the case of a mainframe, time-sharing meant allowing multiple users to simultaneously use its expensive resources. Time-sharing kept the mainframe busy most of the time; whenever an executing task would wait for user input, another one was scheduled. Users prepared their tasks using remote consoles. The next task to be executed was selected among those that were ready. In this way, users on average were executing their programs faster. In 1961, the



(a) Traditional Servers (b) Virtualized Server

Figure 2.2: Operating System Server Virtualization. Traditionally, each server is hosted on a separate machine. With operating system virtualization, different Virtual Machines are created that run heterogeneous operating systems. Different server applications run within each Virtual Machine.

Compatible Time-Sharing System (CTSS) deployed on an IBM 709 was the first such system developed. The next step was the development of *Virtual Machines* (VMs). VMs were execution environments where users would run their programs and gave them the illusion of being the only user of the machine. The first such system was developed in mid 1960s on a specially modified IBM System/360 Model 30 with memory address translation capabilities. The Virtual Machine Control Program (CP) controlled the execution and time-sharing among 14 Virtual Machines, each one executing the Cambridge Monitor System (CMS). The system was referred to as CP/CMS [MS70].

Since the first appearance of VMs almost five decades ago, virtualization in mainframes has evolved into a mature technology. It has also gained significant attention during the last decade as one of the most promising technologies for commodity machines. The key point to its resurrection has been the virtualization of the popular Intel Pentium commodity hardware by the two leading vendors in virtualization, VMware [vmw08b] and XenSource [xen08a].

Figure 2.2 illustrates the basic concepts of modern operating system virtualization in a server deployment example. In a traditionally non-virtualized system, a simple server application is deployed on a machine, and the application uses the hardware resources via the operating system layer. In the virtualized case, there is an additional layer between the operating system and the hardware, called the *Virtual Machine Monitor* (VMM). The VMM creates the different execution environments, the VMs, interposes between the hardware and the running operating systems, and handles *resource multiplexing* and *isolation* between the VMs. In this particular example, each VM hosts a technologically different operating system and each one of them uses the hardware resources in an isolated manner. Server applications run in the VMs as if running on a traditional operating system.

Techniques

Initially, virtualization was synonymous with *full virtualization*, where a functionally equivalent image of the hardware is created by the VMM. Any operating system can run on the VM provided by the VMM without any modifications since the VMM provides virtualized versions of the hardware resources. VMs can use these resources as they would use the bare hardware.

The wide adoption of the commodity server machines in data centres has led to an increased interest in their virtualization. However, as full virtualization is not always possible in the popular Intel Pentium architecture [RI00], several techniques have been developed over the last 10 years, e.g. full virtualization with binary translation, para-virtualization [WSG02] and most recently hardware assisted virtualization. Different virtualization systems have been developed based on these techniques, such as VMware ESX server [esx08], the Xen hypervisor [BDF+03], and Microsoft Virtual Server [vir08].

Functionality

Independently of the virtualization technique used, there are three basic functional characteristics exported by most of the available virtualization systems: virtual machine control, resource management, and migration. These characteristics are described below independently of performance and implementation considerations.

Virtual Machine Control: The main functionality of operating system virtualization is the control of VMs. VMs can be created, paused, resumed, and deleted dynamically and on demand. Upon creation of a VM a new execution environment is created and a new operating system instance runs within it. In addition, a subset of physical resources available are allocated for the new VM. This is the equivalent of a new server machine being added to the infrastructure. The set-up of the applications running on the new VM can be either configured in advance or at run-time, exactly as it would be done as in a new server machine.

The execution of a running VM can be paused and thus all applications running within the VM are also paused. The VM no longer executes, but still has resources allocated to it. A paused or a running VM can be shutdown, and therefore, the execution of all running applications within the VM are stopped and all of its resources are freed. This is the equivalent of shutting down a physical server machine.

Resource Management: One of the key functionalities offered by virtualization systems is VM resource management. When creating a VM, the amount of resources that should be made available to it is specified; that is, disk and memory space, CPU share and network bandwidth. In this way, an initial execution environment is created. This is the equivalent of specifying and configuring a server machine with specific hardware characteristics.

The initial resource allocation can be changed during a VM's lifetime. A running VM can be configured online to a new memory allocation, CPU share policy, disk space and network allocation. Dynamic hardware configuration is different than with traditional

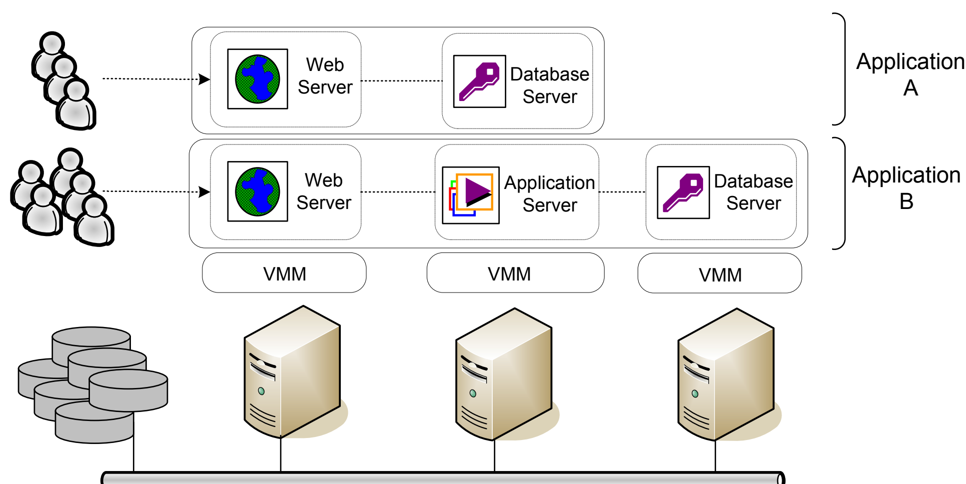


Figure 2.3: New Generation of Data Centres. Different applications are co-located on virtualized server machines. The Virtual Machine Monitor handles resource sharing and isolation among the running applications per physical server. Applications might be distributed across several machines.

server machines. For instance, when upgrading a machine's CPU, the machine has to be shutdown, therefore stopping all of its services.

Migration: A created VM can be migrated from one physical machine to another, assuming that the destination machine has the necessary free resources. The execution of the VM after the migration is resumed at the destination machine. During some part of the migration, all running services of the migrating VM will be temporarily paused. This is a key mechanism that allows VMs to run on different servers based on their resource needs and high-level operations for the data center such as server consolidation.

To summarise, virtualization provides a basic management interface and is now a key technology to create the next generation of data centres.

2.3.2 New Generation Data Centres

Using virtualization as a key feature, a new generation of data centres is now emerging. Based on the hardware to software decoupling offered by virtualization and using the VM as the basic computing unit, a drastically different image of the data centre is appearing (as shown in Figure 2.3) and described below.

The data centre consists of a set of virtualized servers, distributed on a local network. There is also shared storage to support the migration of virtual machines. All hosts are treated as a common unified set of resources, referred to as *resource pool*, that can be used by any VM. For instance, the total available memory in a resource pool is the sum of all physical memories from individual hosts. A server application is deployed on one or more VMs (depending on the structure of the application), a technique referred to as

server virtualization.

A high-level management layer is also needed to interact between the applications and every VMM at any physical machine, to manage the virtualization platform, and to provide functionality of advanced tasks such as load balancing and disaster recovery. This layer would be responsible for allocating the necessary resources for all VMs and for finding the most appropriate machines to host them according to a *utility function* or *global policy*, such as load balancing (e.g. all machines must operate at 80% of their CPU capacity) or power saving (e.g. use as few machines as possible). At run-time, VMs could be dynamically re-mapped to physical servers or re-assigned resources to adapt to changes such as the addition of new applications or machine failures.

2.3.3 Operations

Server virtualization has now been embraced by companies that aim to ease their data centre management and reduce costs, as shown by several surveys. In a 2007 report by the Aberdeen Group [abe07a], it was reported that in a survey of 140 companies, small companies have virtualized 27% of their server environments, medium ones 18% and large companies 14% — small companies are those with less than 50 employees, medium sized companies are those with between 51 and 1000 employees, while companies with more than 1001 are categorised as large companies. According to the same survey all companies, independent of their size, plan to increase their number of virtualized servers to almost 50% within the next three years. In another 2006 report conducted by Forrester Research, 40% of North American companies [GS06a, executive summary] and 26% of companies worldwide [GS06b, executive summary] surveyed by Forrester had implemented virtualization in their data centres.

The wide adoption of server virtualization is partly due to numerous benefits including server consolidation, ease in management operations, and debugging.

Server Consolidation

Server virtualization provides an abstract way for different servers to co-exist on the same server machine and share resources, while the underlying virtualization layer offers isolation and performance guarantees. The building block of this abstraction is the VM, which can accommodate a whole server application or parts of it. Multiple different server applications, even running on heterogeneous operating systems can be hosted by the same physical machine, as shown in Figure 2.2.

With average server utilisations as low as 15%, server consolidation can increase the utilisations in the average server machine. Current virtualization technologies report high utilisation rates; VMware selects to report that some of its customers exhibit a 60-80% utilization rate for server machines [VMw08f]. A key prerequisite to efficient server consolidation is dynamic resource allocation per VM along with isolation among the VMs. The resources for each VM are allocated upon its creation and are adjusted during its lifetime to accommodate changing workloads. In addition, performance isolation among

the running VMs guarantees that each VM does not compromise the resources allocated to others. An *adaptive* framework that "shrinks" and "expands" VMs according to their needs can be built to offer high utilisation rates per physical machine.

In addition, with server consolidation, fewer server machines are used. A 2007 IDC report estimates that the initial prediction of x86 shipments to increase by 61% has now dropped to 39% by 2010 due to multi-core technologies and server consolidation [IDC07]. By decreasing the number of machines, the total spending at data centres including buying new hardware as well as power and cooling expenditures are expected to drop. As the number of servers reduces and the utilisation per physical machine increases, less energy should be needed for data centres. Since in many cases power is reserved to meet peak loads, even though in general the machines are under-utilised, fewer more highly utilised machines can be used to achieve similar goals as before.

Management Operations

A key feature of virtualization is hardware-software decoupling. The VMM layer exports a generic interface to running VMs, which are no longer hardware specific. In this way, a number of administrative critical operations, such as hardware upgrade and system maintenance, can be performed without disrupting the running applications within the VMs. During these operations, VMs are migrated to other hosts; the same can occur when a hardware failure occurs [CLM+08].

Another benefit is fast server deployment. One of the major administrative operations in data centres is the deployment of a new server system. It is a rigorous process that involves a series of activities, such as defining the server specifications, purchasing the required hardware, configuring the new server machines and the application, and finally deploying and testing the new setup. This process can be significantly reduced with the use of virtualization. An existing host can be selected to host the VM while a VM template can be used to deploy the new server application. VMware claims that the time to deploy a new IT service can be reduced by as much as 50-70% [VMw08e].

Testing and Debugging

Virtualized servers can also be used to test and debug new applications before production. In such an environment, the new application can be rigorously tested against threats such as attacks, intense workloads, and malicious code [HFC+06]. It is also the case that a crashed VM can be more easily replaced than an operating system running on a dedicated machine. In addition, there are specialised tools to debug distributed applications running on VMs, that can pinpoint potential race conditions or performance problems [HH05, HSH05].

2.3.4 Challenges

Virtualization in data centres is being adopted rapidly due to its many applications as well as its potential to reduce cost. There are three key points to efficient data centre virtualization:

1. High performance and secure VMMs are essential to deploy virtualization in mission critical environments. Virtualization technologies continue to improve rapidly in this direction.
2. Flexible VMM functionality is necessary to build high-level management operations. As discussed before, the basic functionality provided by the most popular virtualization technologies can be used to support simple or more complicated application scenarios.
3. Automatic management tools are necessary to administer virtualized applications on large scale virtual servers, and to handle heterogeneous application demands.

Although, to date, much emphasis has been given by the community to the first two issues, the task of building management tools is evolving more slowly. In a survey conducted by Rackspace in August 2007 [rac07], involving 354 of their customers, it was reported that some of the main obstacles to the deployment of virtualization in their data centres is the lack of expertise, immature technologies, and management/administration. 71% of them prefer to host a production application on a virtualized platform managed by a hosting provider, since they possess the necessary expertise. In another Aberdeen Group report in July 2007 [abe07b], it was reported that a noteworthy percentage of companies — 22% of small ones (with less than $500M revenue) and 30% of large companies (with more than $500M revenue) — refuse to deploy virtualization in their data centres, mainly because of the lack of staff and domain knowledge of these new technologies. The two reports indicate that management and administration can be an obstacle to further adoption of virtualization.

Management tools are crucial to the efficient administration of virtual servers. There are two main categories of management tools. The first category involves essential tools that implement management operations, such as create a VM; allocate 500MB of memory to a VM; migrate a VM from host A to host B, and so on. The tools in the second category are built on those of the first to manage the virtualized servers for a specific high-level purpose such as server consolidation, load balancing, and disaster recovery. As data centres and subsequently virtualized ones accommodate hundreds and thousands of physical and virtual servers that serve complex distributed applications, the second category of tools are essential to deliver high availability and high performance.

High level management tools need to support many operations including disaster recovery and load balancing. Depending on the task, different operations are required. For instance, to achieve load balancing, different VMs need to be hosted on machines so that all hosts have roughly equal resource usage. To save power, all VMs need to be hosted by as few machines as possible in order to switch off the rest and save on energy. There is, however, a very important operation *integral* to the success of all high level tasks: *VM resource provisioning*.

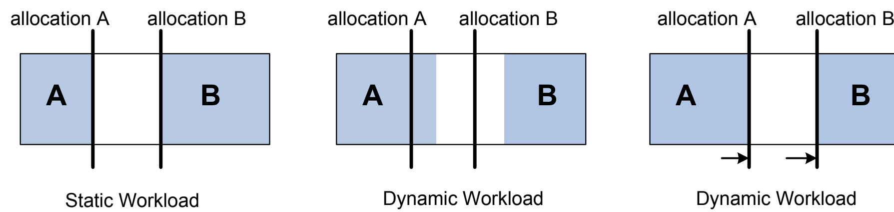


Figure 2.4: Resource Management in Server Consolidation. This figure illustrates three scenarios of resource provisioning of two virtualized server applications A and B co-located on a physical server. In the left diagram, the applications' resource utilisations are known in advance (shaded rectangles) and their allocations are adjusted accordingly (solid lines). In the middle diagram, the utilisations change due to workload fluctuations, however, the allocations remain the same. In this case, application A suffer from performance degradation, while there are unused resources allocated to application B. To address these limitations, allocations are adjusted to resource utilisations as shown in the right most diagram.

VM Resource Provisioning

Adequate provisioning for VMs' resources is crucial for a high performance data centre. On one hand, it is very important for the hosted application within the VM to always have the necessary resources to achieve its performance goals. On the other hand, as long as the VMs' resource requirements are met, any high level task can be planned and executed within the data centre. However, resource provisioning for virtualized server applications is a challenging task.

Consider a server consolidation example with two single-component server applications and one server machine. Assume that each application has a workload with known resource requirements and the sum of resources from both applications does not exceed the total available physical resources for the server machine. The left diagram in Figure 2.4 illustrates two VMs, each one hosting an application with resources allocated as required. In this way, both applications are served adequately and the total resource utilisation of the physical machine is now increased simply by augmenting the number of running servers.

Consider now the case where the workload in both applications changes, (middle diagram in Figure 2.4). In VM A it increases, therefore more resources are required, while in VM B it decreases so fewer resources are needed. In the case of VM A, the under-provisioning results in performance degradation, since the application does not have enough resources to serve its incoming requests. In the case of VM B, the over-provisioning does not affect the running application within the VM B. However it does reduce the free available resources for a third VM to be placed on the same machine. Therefore, in both cases, the resource allocation needs to adapt to the new resource demands (right most diagram in Figure 2.4).

Nevertheless, adapting to the new demands is a daunting task. Workload fluctuations make the problem of VM resource provisioning difficult. Furthermore, resource provisioning is more demanding due to the complexity of modern server applications as demonstrated by their multi-component nature. This dissertation is concerned with the development of automatic tools that dynamically adapt the CPU resource allocations of virtualized server applications in order to meet their performance goals. The next section explores the current state of available solutions and identifies challenges.

2.3.5 Commercial Solutions

There is a wide range of management products offered by the two most popular modern virtualization technologies — products from VMware [VMw08g] and from XenSource [Xen08c] — for managing server virtualization. Less emphasis has been given, however, to the creation of resource management products. The most related are presented below.

VMware Capacity Planner [vmw08c] offers long-term capacity planning assessment of virtualization and consolidation in a traditional data centre through scenario exploration and "what-if" modelling. This work focuses on dynamic, short-term resource provisioning of running virtualized servers.

VMware Distributed Resource Scheduler (DRS) [vmw08d] dynamically allocates cluster resources to running VMs. Each VM is configured with three attributes: (a) the reservation, which declares the minimum assigned resources to the VM, (b) the limit, which represents the maximum resources ever allocated to the VM, and (c) the shares attribute, which denotes the resource utilisation priority over other VMs. Similarly XenCenter [Xen08b] provides resource management capabilities by configuring priorities and limits for VM resources. These tools provide the mechanisms to ensure that the allocations for VMs' lie within certain limits, but do not deal with setting these limits to appropriate values for each application. Additionally, to the best of the author's knowledge, at the time of writing this dissertation, there is no published documentation describing the ways in which the above tools operate to maintain the resource allocations within the resource limits in the presence of dynamic workload demands.

VMware DRS also offers the capability to group VMs for multi-tier applications and collectively assign resources. Again, the online published documentation does not provide any further information.

In addition to tools for modern virtualization technologies, there are other traditional vendors, such as HP, that specialise on workload management. For instance, HP_UX Workload Manager (WLM) [hpW08b, hpW08c] is a tool that automatically manages the CPU resources of applications running on HP server machines with dynamic resource sharing capabilities. The allocation is based on user-defined Service Level Objectives (SLOs) and priorities. There are two modes when defining the SLO: a non-metric and a metric based mode. The non-metric allocation policy defines the application desirable CPU usage[4]. With the metric based SLO policy a user can define the portion of resources

[4]In this mode, a user can also choose a fixed amount of CPU resources.

assigned to a metric unit (e.g. *"five CPU shares for each active process"* [hpW08a, page 12]). The efficiency of this mode, however, depends on the correct mappings of resources to metric unit. In either case, the WLM allocates CPU resources to maintain the SLOs. In case of contention, different applications are assigned resources according to their priorities. As reported by the WLM's Overview Data Sheet [hpW08a, page 8], WLM is more suitable for CPU-intensive applications, while this dissertation targets multi-purpose multi-tier server applications.

2.3.6 Summary

The virtualization of commodity machines transforms the data centre into an agile environment for server application deployment. Applications are hosted within VMs which can be deployed on any physical server machine. Using the basic functionality offered by most modern virtualization technologies — VM control, VM resource management, and VM migration — high level operations such as server consolidation and power management can be planned to increase machine's resource utilisation and decrease power and cooling cost. To plan for high level management operations, there is, however, a very important task central to their success: adaptive VM resource provisioning. Adjusting the CPU shares of running virtualized server applications on demand in response to workload changes is challenging because of the diverse and fluctuating workload characteristics.

Feedback Control provides a flexible and reactive way to dynamically adjust the CPU resources as workload changes happen. The next section introduces the basic principles of feedback control and motivates its use towards the current problem.

2.4 Feedback Control

This section presents the basic concepts of feedback control systems and describes related terminology. The description presented here outlines those concepts related to this dissertation and which are directly applied to the current work. Therefore, it does not, in any way, constitute a thorough presentation of the Control Theory field. Finally, the section overviews the way control theory is applied to the problem of resource management of virtualized servers.

2.4.1 Overview

A control system (Figure 2.5) is composed from two main parts: the *plant* and the *controller*. The plant (or *target system*) is the system of interest which is built to perform a task/goal (e.g. room temperature regulator). The purpose of the controller is to determine the settings of the "knobs" that make the plant reach its user defined tasks despite the presence of *noise* in the operating environment. To this end, the controller monitors the plant at regular intervals and if any deviation from its goals is observed (*error*),

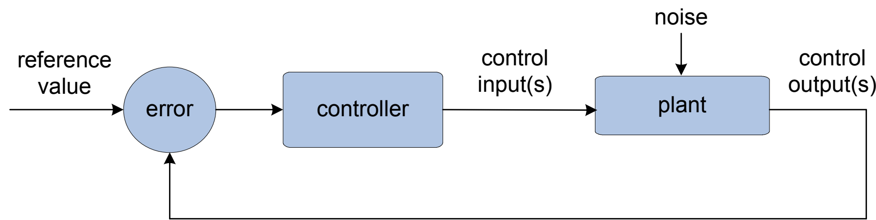


Figure 2.5: Feedback Control System. In a feedback control system the controller periodically gets updates of the controlled system, called plant, through the control output(s). Based on their values and the error from the reference value, it computes the next values of the control input(s). The goal of the controller is to maintain the plant's performance around the reference value despite the noise coming from its environment.

corrected actions are applied to it. The controller and the plant communicate through signals, named *control input(s)* and *control output(s)*. The control output(s) provide information regarding the latest state of the plant, while the control input(s) update the plant to correct its state towards its goal.

The control system operates in a *closed-loop* fashion, since data flows periodically between the controller and the plant, at regular *intervals*, and updated values of the control input(s) are based on measurements from the control output(s). This type of control is also referred to as *feedback control*. If there is one control input and one control output, then the system is referred to as a *Single Input Single Output* (SISO) system. If there are many inputs and many outputs the system is called a *Multiple Input Multiple Output* (MIMO) one.

The controller is the most important part of the control system. It uses a *model* of the target system and the control error and accordingly adjusts the input(s), so that the plant achieves its goals. The model captures the dynamics of the plant, and quantitatively associates the control input(s) to the control output(s). For example, consider a temperature control system of a room with an electric heater (part of this example is taken from [Oga90, page 10]). The purpose of the controller is to maintain the temperature of the room at a certain reference level. However, the temperature of the room fluctuates when for instance a door or a window is opened. To always maintain the same temperature in the room, the controller measures it at regular intervals. When a deviation/error from the reference value occurs, the controller based on the system model adjusts the heater to either increase or decrease its power and defines its magnitude. The process of discovering the system's model and in particular the combinations of input(s)/output(s) that best capture the dynamics of the target system for a specific goal and define their relationship is called *system identification* [Lju87, page 6].

There are four main properties of interest in a control system: *stability*, *accuracy*, *settling time*, and *overshoot*. These properties are also referred to collectively as SASO properties [HDPT04, page 8]. An informal definition of the properties is now given based

on [HDPT04, page 8].

- **Stability:** A system is Bounded-Input Bounded-Output (BIBO) stable if for any bounded input, the output is also bounded.[5] In a mathematic way this means that the poles of the transfer function of a discrete-time linear system have to be within the unit circle.
- **Accuracy:** A system is accurate when its output converges to its reference value. Rather than measuring accuracy, it is often the case that a system's inaccuracy is calculated. For a system in steady-state, its inaccuracy is measured as the *steady-state error*, the difference of the output from its reference value, usually denoted as e_{ss}.
- **Settling Time:** In addition, settling time (denoted as k_s) is defined as the time it takes for the system to converge, usually within 2%, to its steady-state value after a change in input or reference value.
- **Maximum Overshoot:** Finally, a system should converge to its steady-state without overshooting. Maximum overshoot (denoted as M_p) is defined as the largest amount by which the output exceeds the reference output scaled by the steady-state value: $(1 + M_p)y_{ss} = M_o$, where M_o is the maximum value of the output and y_{ss} the steady-state value.

Having presented the basic concepts of feedback control, this section continues by describing the way it is applied in the current work.

2.4.2 Feedback Control for Resource Management

There is a direct correlation between the problem addressed here and a feedback control system. The problem addressed in this dissertation is the provisioning of virtualized servers with resources in order for them to meet their performance goals in the presence of fluctuating workloads. The target system is any virtualized server with time-varying resource demands caused by diverse and fluctuating workloads. Despite the noise from the workload, the server should maintain its performance as indicated by the *reference input*. The controller is therefore responsible for maintaining the performance around the reference input (e.g. utilisation) by tuning certain parameters (e.g. CPU allocations). Feedback control for this problem is particularly attractive since (a) the model of the system is neither known in advance nor well defined and (b) the virtualized servers are under noisy workloads.

The focus of this dissertation is the design and implementation of controllers that manage the CPU allocation of virtualized servers. Chapter 3 presents the overall architecture and the implementation of the supporting control system. The system identification process is performed in Chapter 4 and Chapter 5 presents the different controllers.

Control theory has been used in computer systems in the past [HDPT04]. In fact it

[5] According to [HDPT04, Section 3.3.1]: A signal $t(k)$ is a *bounded* signal if there exists a positive constant L such that $|t(k)| \leq L$ for all k.

has also been used to address very similar problems to the current one; related work is discussed in Chapter 7.

2.5 Summary

There are two main approaches for hosting applications: dedicated and shared. The popular and widely adopted dedicated hosting provides performance isolation to running applications. However, it has resulted in resource under-utilisation mainly because of over-provisioning against highly fluctuating application workloads. Although shared hosting alleviates this problem, it has been difficult to implement due to lack of generic mechanisms in the popular Intel Pentium server machines. The resurgence of commodity machines virtualization is transforming the data centre into an agile pool of resources for secure and flexible sharing among the applications. Although modern virtualization technologies offer the basic functionality for high-level operations in data centre management, there is, however, a key prerequisite to their success: adaptive resource provisioning for virtualized applications. As long as each application is provisioned with enough resources to meet its performance goals in the presence of changing resource demands, further planning for tasks such as server consolidation and power management is possible.

This dissertation is concerned with the development of tools that automatically adjust the CPU shares of virtualized multi-component applications. To this end, a feedback control approach is adopted. Feedback control provides a flexible and intuitive approach to resource management, as allocations are updated to workload changes and there is no need for extensive *a priori* domain knowledge.

This dissertation proposes in Chapter 5 and evaluates in Chapter 6 different feedback controllers which perform adaptive resource management. However first Chapter 3 describes the evaluation platform used to deploy and assess the solution.

3 Architecture and Tools

This chapter presents the architecture of the resource provisioning process and its implementation in the prototype virtualized cluster that was built for evaluation.

This chapter begins (Section 3.1) by presenting (a) the main assumptions regarding the server application model, (b) the application deployment on VMs, and (c) the type of resource management considered in this dissertation. Next, Section 3.2 presents an overview of the evaluation platform used. The architecture of the resource management process is presented in Section 3.3. The rest of this chapter reviews the benchmark application (Section 3.4), presents the Xen VMM (Section 3.5) used for virtualizing the cluster, and finally, Section 3.6, discusses the resource management issues specific to the Xen platform.

3.1 Deployment Model

This section presents the assumptions made in this dissertation regarding the application model, the deployment on VMs, and the type of resource management. These assumptions provide the context for the resource management process.

Application Model

A server application is composed of one or more components/tiers. Incoming requests are processed by a subset of the components. The exact tier layout is defined before the application's deployment and remains the same throughout the resource provisioning

process. Each tier is a stand-alone server and relies on network connectivity to communicate with the other tiers. Tiers of the same functionality (e.g. replicated web servers) can exist, but each tier is considered unique for the resource provisioning process.

Deployment on VMs

Each tier is hosted by only one VM. A server application is therefore composed of one or more VMs. Components of the same server application can be deployed on the same or different physical machines.

Resource Management

Each VM is treated as a *black-box*. Under the same workload (mix of requests types and incoming requests' arrival rate), the application's performance depends on the resource allocations of the individual components. The resource usages for each VM include the utilisations caused by both the application's tier and the operating system running within the VM.

3.2 Evaluation Platform

Figure 3.1 shows the prototype cluster deployed to evaluate the application's performance and subsequently the controllers' performance. In this system, the 3-component Rubis server application [ACC+02] is deployed on three machines. Each machine runs the Xen VMM [BDF+03]. Each of the three Rubis server tiers, namely the Tomcat web server tier, the JBoss application server tier and the MySQL database (DB) server tier, is deployed on a separate VM running on a separate physical machine. A fourth machine hosts the Rubis Client Emulator used to generate the requests to the server. All machines are connected via Gigabit Ethernet. The `control` and the `manager` building blocks are also shown. There are three `manager` components, each one running within the Xen VM control domain of each physical machine. The Xen control domain is called `dom0`; the Xen architecture and terminology is explained in detail in Section 3.5.

The `manager` records CPU usage every 1 sec using the `xentop` command which is the equivalent of the `top` Linux command for the Xen system and periodically displays information regarding the Xen VMs. At the end of the controller interval, it calculates the mean over all data and submits the response to the `control`. The duration of the controller interval used in this dissertation is 5 seconds (s).

The prototype cluster is deployed on typical server machines used for commercial applications. All machines are x86-64, each equipped with 2 AMD Opteron Processors running at 2.4GHz, 4GB of main memory, 70GB of SCSI disk space and a NetXtreme Gigabit Ethernet Card. Each machine runs the popular Xen VMM, version 3.0.2 [BDF+03]. Finally, all VMs are similar and they run the commercial SUSE Linux Enterprise Server

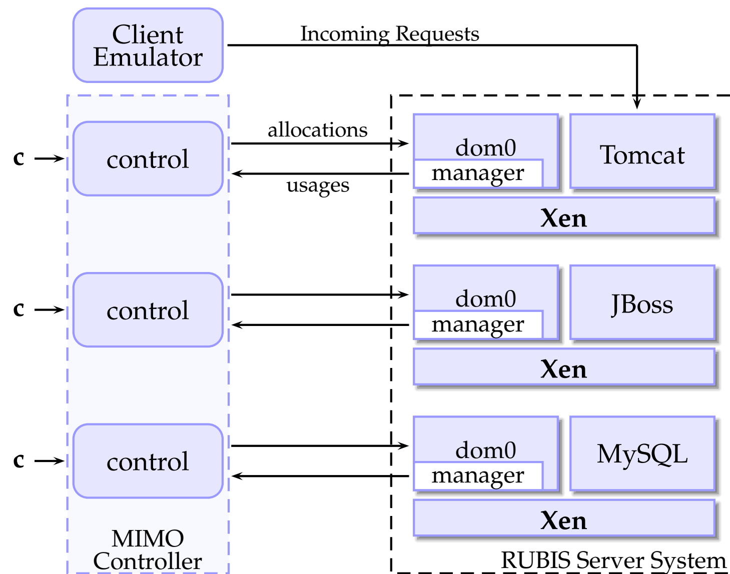


Figure 3.1: Virtualized Prototype and Control System Overview. Solid lines between the control modules and the Rubis Server System depict the connection of the three SISO controllers. The MIMO controller is shown by the dashed rectangle.

(SLES) 10 with Linux-xen 2.6.16, popular for server application deployment. The hardware and software setup of the server machines makes the cluster a realistic if small-scale implementation of a virtualized data centre.

The Client Emulator machine has the same hardware characteristics as the server machines and it runs the same SLES distribution with Linux 2.6.16.

The next section discusses in more detail the architecture of the `control` and the `manager` blocks.

3.3 Architecture

The controller is the most prominent part of the resource provisioning process. Based on the application performance model, it periodically calculates the required component resource allocations in order for the application to serve unknown fluctuating workloads and to meet its performance goals.

The resource provisioning process is based on control theory principles and has two main characteristics: (a) resource allocations for the application components are performed on-line at regular intervals while the application processes incoming requests and (b) allocations are made according to an application performance model that associates allocations and usages with performance metrics.

The architecture presented in this section provides the means to support the deployment of the controller in a virtualized data centre. The following functions are supported:

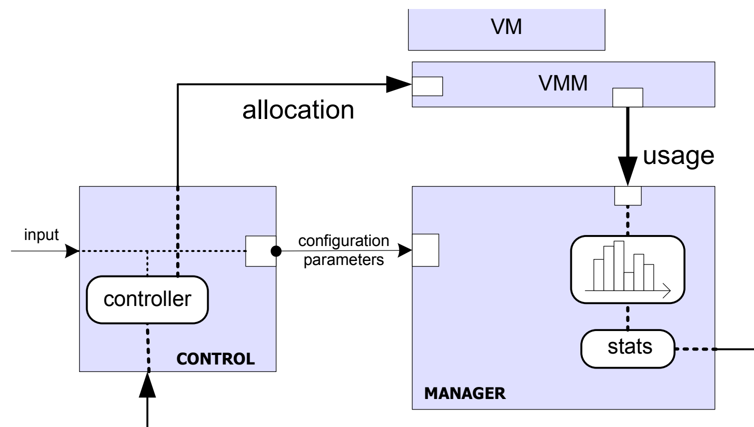


Figure 3.2: Resource Management Architecture. The architecture supports a feedback control loop for resource provisioning of VMs and consists of the `manager` and the `control` blocks. The `control` block adjusts the allocations of VMs based on information on past measured usages — as sent by the `manager` block — and the application model embedded in the controller.

(a) it provides remote resource monitoring and application of the controller's outputs at regular intervals at the VMs; (b) it enables deployment of different controller schemes on the same platform with minimal changes; and (c) it supports arbitrary combinations of server components deployed on physical machines. This section presents the architecture and elaborates on its operations.

A conceptual view of the architecture is shown in Figure 3.2. The resource management framework is composed of two software blocks, the `control` and the `manager` block. The `manager`, which runs on `dom0`, is responsible for the resource monitoring for each VM running on the same physical machine. It uses the interface provided by the VMM to measure the VM's resource utilisations, a summary of which (e.g. mean) is sent to the `control`. The `control`, which runs on an another machine, calculates the new allocations based on measurements from the `manager` and the performance model used for the server application. The new allocations are remotely applied to the VM by the `control` after performing any necessary transformations (e.g. checking that the new allocation does not exceed the total physical machine capacity). Finally, the `control` accepts input configuration parameters (e.g. controller interval) which are further sent if required to the `manager`. Both blocks are built using the Python programming language.

Measurement and allocation data flow between the two blocks at regular intervals. At the end of each interval, the `manager` sends its measurements to the `control` which responds back with the new allocations at the beginning of the new interval. During each interval, resource usages are measured at a time granularity as indicated by the `control`, (every t time units as presented in Figure 3.3). The shortest measurement update is restricted by the virtualization platform. Consider the interval between time instances k and $k+m$, referred to as interval k. The real time elapsed can be several seconds or minutes or any other time period as indicated by the `control`. During

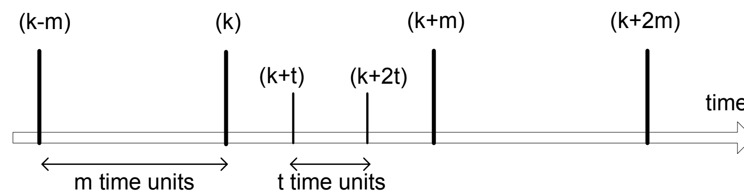


Figure 3.3: Controller and Measurement Intervals. The `control` periodically updates the allocations every m time units. The `manager` measures the utilisations every t time units during each controller interval.

interval k, the `manager` measures the resource usage every t time units. Every interval is m time units and $t \leq m$. Different time units for t and m can also be used. At the end of interval k, a summary (e.g mean) of all measured usages during the interval is sent to the controller. The time k and t, and the type of summary information are all input to the `control` block and are specific to the application and the control process.

The controller in the `control` block executes the most important operation of the resource provisioning process. Based on the application performance model, it periodically updates the allocations to meet the resource demands of incoming requests and to comply with the performance goals set for the application. The application performance model is derived off-line during the system identification process — a process that is described in Chapter 4 — and its parameters can be set in an off-line or an on-line fashion — both described in Chapter 5. The model associates the resource measurements to the allocations and the way they affect the application performance. Therefore, a predefined performance goal is achieved by adjusting the allocations at appropriate levels using the resource utilisations to measure the application performance. Different control schemes are easily supported by the current framework by deploying them at the controller. In this dissertation, different control schemes are considered and presented in Chapter 5.

Finally, the architecture supports remote resource allocation of arbitrary combinations of deployed components to physical machines. This is achieved by the clear separation of functions between the two building blocks, the `manager` and the `control`. The `manager` operates as a server that monitors the usage of all VMs on the same physical machine as requested by its client, the `control` block. A `control` block connects to it, remotely manages the resources of some or all the VMs, and receives the results. There can be one or more `control` blocks that manage the allocations of each, a subset of or all of the application's components. In this dissertation two different models are considered: (a) a one-to-one model, with one `control` block per application component and (b) a one-to-many model, with one `control` for all application components, both presented in Chapter 5.

Summary

This section presented the framework for remotely managing the resource allocations for server applications deployed on VMs. The architecture implements the basic prop-

erties of a control-based resource management process. Finally, it provides a flexible framework for deploying different controllers and supporting arbitrary combinations of multi-component applications deployed on physical machines.

The controllers discussed in this dissertation are evaluated against the Rubis multi-component benchmark application which is deployed on an industry-level prototype virtualized data centre built for this purpose. The rest of this chapter presents the implementation of the evaluation platform. More precisely it presents an overview of the Rubis benchmark server application and the Xen VMM used to virtualize the machines. Finally, it discusses Xen related resource management details.

3.4 Rubis Benchmark

This section describes the Rubis auction site benchmark used for the evaluation of the resource provisioning architecture, here Rubis version 1.4.3 with session beans is used.[1] Emphasis is given to its 3-tier layout description and the request processing path. An overview of the Client Emulator tool that is used to generate load on the server is also given. The section also outlines the reasons why the Rubis benchmark is an excellent candidate for the evaluation of the current approach.

3.4.1 Introduction

The Rice University Bidding System (Rubis) [ACC+02] is a prototype auction web site server application modelled after eBay.com. Rubis implements the basic operations of an auction site: selling, browsing, and bidding. Using the server, a client can perform 27 different types of requests including: browsing items from a category or a region; viewing an item; registering an item; bidding on an item; buying an item; and selling an item.

Rubis was originally designed for testing and benchmarking purposes. Initially, it was developed to study the performance of web applications with dynamic content and to compare different implementation methods such as PHP, Java servlets, and Enterprise Java Beans (EJB) [ACC+02]. It was also used to examine the performance and scalability of different J2EE application servers, namely JBoss and Jonas, as well as the application implementation [CMZ02, CCE+03]. Rubis has since been used for evaluation purposes in areas such as: fault detection [CAK+04], VM resource provisioning [PSZ+07], and component-based performance modelling [SS05].

This section gives an overview of the Rubis benchmark regarding the clients' operations on the server, its tier-layout, the backend data structure and volume, and the Client Emulator.

[1] Minor modifications to the official Rubis distribution were made. These include setting up the different configuration files for the application beans and recording the requests response times at the Client Emulator.

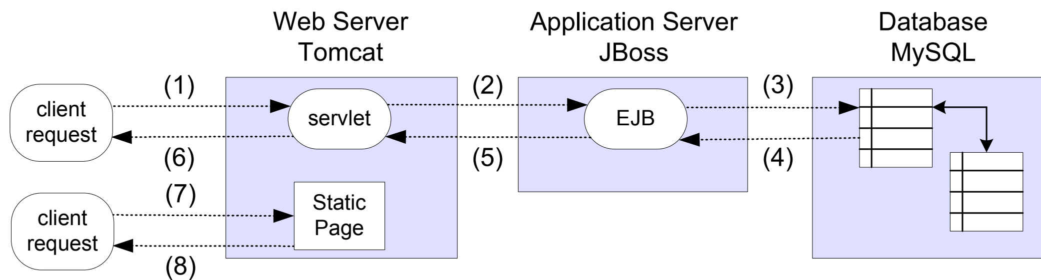


Figure 3.4: Rubis Tier Layout. The Rubis benchmark is composed of three tiers: the web, the application, and the database tier. The web server accepts clients' requests. Depending on whether the requests require access to the database, different action paths are followed. In the case of requesting dynamic content, actions (1)-(6) are invoked and all the tiers participate in serving the requests. In the case of static content, actions (7) and (8) serve the static HTML page back to the client.

3.4.2 Tier-Layout and Request Execution Path

Rubis is composed of three tiers: the web, the application, and the database tier, as shown in Figure 3.4.

The web tier is responsible for accepting clients' requests (actions (1) and (7)) and invoking the appropriate actions according to the type of request. If the client's request requires content from the database (e.g. browse regions, put comment on item), a new request to the database through the application server is invoked (action (2)). The majority of request types (22 out of 27) require the generation of dynamic content and trigger access to the database. Different servlet objects, depending on the request type, are launched to handle the internal operations necessary (between actions (1) and (2)). If the client's request does not require access to the database, then a shorter path is followed: the web server retrieves the static HTML page and sends it back to the client (action (8)).

The application server is responsible for establishing database connections to retrieve and/or store data as requested by the clients. It also maintains a consistent view of the database by updating the tables as necessary (actions (3) and (4)). Finally, it also performs additional application business logic operations whenever necessary such as user authentication (e.g. only registered users are allowed to put comments on items). Similar to the servlet architecture, different bean objects (EJB) are launched to handle the internal operations. The application server returns the results of the database access to the web server (action (5)). At the web server, the final HTML response page is formed and the result is sent back to the client, (action (6)).

The third tier hosts the Rubis database. An already populated database is used for the experiments. It contains around 34000 items for sale, which belong to 20 different categories and 62 different regions. The database dump is made using observations from the eBay.com web site [CCE+02].

The web container used is Tomcat [tom08], version 5.0.30. The JBoss [jbo08] version 4.0.2 application server executes the application logic. Finally, the database is stored on the MySQL [mys08] database server, version 5.0.1.

3.4.3 Client Emulator

The Rubis package contains the Client Emulator used to generate load for the Rubis server. The Client Emulator tool emulates a number of simultaneous clients that access the auction server via their web-browser. Each client opens a persistent HTTP connection to the auction server and creates a session with the server, during which the client generates a sequence of requests. After issuing a request, the client waits for the response. Upon receiving the response — an HTML web page — it waits for some "think time" and then issues the next request. The think time emulates the time it takes for a real user until he or she issues the next request. It is generated from a negative exponential distribution with a mean of 7 seconds [TPPC02, clause 5.3.1.1]. The next request type is determined by a state matrix which contains the probabilities of transit from one request type to another. The next request might use information from the last HTML response page (e.g. view an item with a specific ID). The session terminates when the maximum number of requests allowed by each client has been issued, or when such time has elapsed that the session has reached its predefined end.

The Client Emulator includes different transition tables, each corresponding to a different workload mix. There are two available mixes from the Client Emulator: the *browsing mix* (BR) with read-only requests and the *bidding mix* (BD) with 15% read-write requests. The experiments in this dissertation use the BR mix unless otherwise stated.

A number of parameters, such as the number of active clients and the type of workload mix, can be set at the beginning of the emulation using the interface provided by the Client Emulator. More information on the Rubis Emulator and the workload mixes can be found in [ACC+02].

Finally, the Rubis Emulator is altered to record all requests' response times. This is the time that elapses between the initiation of a client's request and the time that the response arrives at the client. In a real environment, recording the response times at the clients' side is unrealistic. Ideally, they must be recorded at the server side. However, due to the fast network connectivity (all machines are connected on a Gigabit Ethernet network) between the Emulator and the server machines, which results in negligible network delays between them, the response times can be recorded at the client side instead of at the server side without any performance implications to the resource control system.

3.4.4 Summary

The Rubis benchmark provides a realistic distributed implementation of a 3-tier web server auction site. It is designed according to industrial standards — servlets and EJB — and uses commercial middleware servers — Tomcat, JBoss, and MySQL. The tier-layout used in this dissertation is very close to the proposed layouts for Rubis server

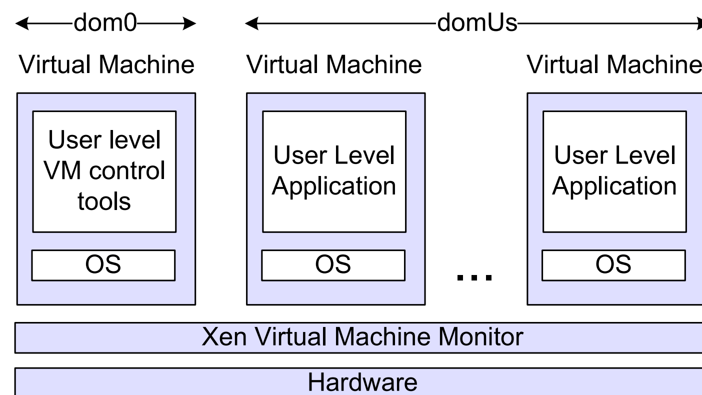


Figure 3.5: Xen Architecture Layout. The Xen Virtual Machine Monitor runs on top of the hardware and creates execution environments (Virtual Machines (VMs) or domains) where user-level applications run. In particular, there are two types of domains: (1) `dom0`, which is created automatically with Xen and provides the platform for user-level control tools that manage Xen operations and (2) `domUs`, which are used for running user-level applications.

deployment [CMZ02]. Its database structure and size is based on observations from the popular eBay.com auction site, at the time. In addition, the Client Emulator software distributed with the Rubis package is based on the well-known TPC-W [TPPC02] specification [ACC+02]. Therefore, the Rubis benchmark makes an excellent and realistic candidate for the evaluation of resource provisioning controllers. Finally, the 3-tier architecture of Rubis enables the evaluation of the controllers against diverse CPU resource conditions as seen by the highly noisy MySQL component utilisations to the least variable ones from the JBoss. This is shown in later in Chapter 6.

3.5 Xen

Xen is a Virtual Machine Monitor (VMM) designed for x86 commodity hardware machines. A Xen-virtualized system consists of three components as shown in Figure 3.5: (a) the Xen VMM, (b) the control domain (`dom0`), and (c) the guest VMs or *domains*.

The Xen VMM, also called the *hypervisor*, provides the basic layer of interaction between running operating systems and hardware resources. Based on x86 paravirtualization, it creates different execution environments the VMs or *domains* in Xen terminology. It provides a low-overhead virtualization platform, where running applications within VMs achieve almost native performance, and supports execution of heterogeneous operating systems with minimal modifications required within the operating system kernel.

The Xen VMM implements the basic mechanisms to ensure safe resource sharing and isolation for memory, I/O and CPU access. It also exports a generic interface for controlling the basic underlying mechanisms through the management tools of the control

domain, called dom0. Dom0 is a basic part of the virtualized platform and is created automatically with Xen's invocation. The management tools for controlling the Xen virtualization platform reside in dom0. Management of all other domains is achieved through a set of tools that use the appropriate interface exported by Xen. With this set of tools, basic control operations on other domains such as creation, deletion, migration and pausing, are possible. In addition, access to resources and permission settings on VMs are administered through dom0. Building on these basic mechanisms for resource management, more elaborate policies on the use of resources can be applied. Finally, a user-level application usually runs within a *guest domain* or domU.

In this dissertation, Xen is used as an example virtualization platform. The resource management tools are built on top of the basic management Xen tools. In the next section, the Xen CPU resource sharing scheme is presented.

3.6 CPU Sharing and Management in Xen

CPU resource sharing is performed at an operating system granularity. CPU time is partitioned among different operating systems; processes running on operating systems use the CPU share of their VMs as they would normally do in a non-virtualizable environment. Throughout this dissertation, the Simple Earliest Deadline First (SEDF) CPU scheduler is used. SEDF is the default CPU scheduler for the Xen 3.0.2 distribution, which is used for the evaluation cluster.

SEDF is a soft real-time scheduler that allocates CPU resources to VMs. CPU time is partitioned into fixed time periods. For every period, each VM is configured with the time it can use the CPU, called its slice. Another configuration parameter denotes whether VMs can use any free CPU time (the work-conserving mode (WC)), or not (the non-work-conserving mode (NWC)). For instance, for a 10ms period, two VMs can be configured with slices of 2ms and 5ms. In the NWC, VMs can use up to 20% and 50% of the CPU time respectively, even if either of the two requires more resources and the CPU is idle. In the WC mode however, any VM can use the free time as long as all VMs have used their share of CPU resources. In this dissertation, the NWC mode is used. This enables complete control over the allocation of CPU resources to VMs and, therefore, proper evaluation of the different controller schemes with respect to the allocations they make and the resulting application performance. Using the NWC, the resource provisioning tools can apply policies for performance guarantees and provide performance isolation among consolidated servers.

Each server machine has two CPUs and each domain is pinned on a separate CPU. Dom0 consumes CPU resources for handling I/O requests on behalf of the guest operating system (domU). To ensure best performance for the server application, I/O must be handled in a timely fashion. Therefore, assigning one CPU per dom0 ensures that the control domain has access to all CPU resources necessary to perform I/O for domU. DomU runs on the second CPU, which can use up to 100% of its resources according to the allocations made by the controller.

However this simple setup does not maximise resource utilisation per physical machine.

Due to SEDF's lack of automatic load balancing among CPUs,[2] dom0's CPU can still be under-utilised even if domU's CPU reaches its peak utilisation for some workloads. The focus of this dissertation, however, is the controller's ability to control VM allocations for multi-tier applications. At the time the work of this dissertation was started, SEDF was the stable scheduler. The now default Xen CPU scheduler, called the *credit* scheduler, enables automatic SMP load balancing and operates in both WC and NWC modes. The current system uses only the generic features of CPU schedulers, namely the NWC mode and the ability to set maximum resource utilisation per VM. Since both features exist in the new credit scheduler, the current architecture could easily be applied to the latest Xen version.

Two issues regarding the configuration of SEDF are now examined. The value of the period and slice SEDF scheduling parameters affect the performance of the server application. A simple experiment was performed to choose those that give the best performance for the server application. For this experiment the prototype cluster with the Rubis benchmark was used and the clients issued requests of the browsing mix to the server. The period values were varied between 10ms and 90ms in steps of 20ms. For each experiment, the response times of the client requests were recorded. For large period values, the response times increased to high values ($>$ 1sec) as the number of clients was increased, even though domU was not saturated. For small values such as 10ms, however, the response times depended only on the saturation of domU and were consistently low for increasing number of clients. When the period is set to large values, domU is not scheduled soon enough to accept the I/O requests. However, when the period is short, domU is scheduled more often therefore handling I/O more often. For all the experiments, a period of 10ms for domU is used. Dom0's period is set to 20ms — however, since it operates in the WC mode, this does not have any effect. Cherkasova *et al.* [LDV07] also study the same issue, and they make similar qualitative conclusions; they observe better server throughput as the period becomes smaller.

3.7 Summary

This chapter presented the architecture of the resource management process for virtualized server applications. In particular, it described the evaluation platform built for testing the control tools and algorithms (Section 3.2). The platform consists of three parts. First, there are four typical x86 server machines that host the Rubis benchmark application and the Client Emulator. The VMs run the commercial SLES operating system and are connected via a Gigabit Ethernet network. Second, the Rubis benchmark application, built based on industrial standards (Section 3.4), is used. Finally, the platform consists of the resource provisioning architecture that remotely controls — using a feedback loop — the CPU allocations of arbitrary combinations of applications and physical machines (Section 3.3). The architecture also supports flexible deployment of

[2] Some level of load distribution can still be achieved with the SEDF by issuing multiple virtual CPUs (vCPUs) per domain and deploying them on the physical machines as needed. Nevertheless, user-land tools are required to distribute the shares evenly across the vCPUs and additional care is needed to ensure that the running VMs do not get overwhelmed by constant switching between the CPUs.

different controllers. Finally, this chapter presented an overview of the Xen virtualization platform (Section 3.5) and discussed Xen-related CPU resource management issues (Section 3.6).

The evaluation platform provides a realistic albeit small-scale virtualized cluster suitable to deploy and test the different controllers presented in the following chapters. In particular, the next chapter presents the system identification process that exercises the application's dynamics in a variety of conditions. Based on this analysis, a number of controllers are derived and described in Chapter 5.

4
System Identification

Previous chapters highlighted the resource management of server components as an essential part of achieving high performance virtualization. The solution given in this dissertation is a control system that dynamically allocates CPU resources to server components. This chapter presents the *system identification* process that captures the model of the system based on which controllers in the next chapter are built. In particular, Section 4.1 introduces the related concepts and provides an outline of the work presented in this chapter. The QoS goal for the current server benchmark application is derived in Section 4.2. Section 4.3 identifies the control input/output (allocation, utilisation) pair. The system modelling procedure is presented in Section 4.4. Finally, Section 4.5 identifies the system model to include the resource utilisation coupling among components of multi-tier applications.

4.1 Introduction

Building a controller requires a system model that captures the dynamics of the system and associates the control input(s) to the control output(s). Using the model and the control error the controller adjusts the input(s) so that system achieves its goal. However, it is not always possible to know in advance the model of the system. The system identification process is a procedure which discovers the model of the system for a specific goal. The model describes the relationship between the control input(s) and the control output(s). The system identification process depends on the goals of the target system.

The goal of the current control system is to adapt the CPU resource allocations of server components in order for the application to maintain its performance at a reference QoS

level (mean response time $\leq$ 1s) in the presence of workload changes. In this way the application achieves reference performance and there are free resources to co-host other applications.

First, the performance goal for the application when deployed on the prototype cluster needs to be defined. Although, the goal of some systems is easily set — e.g. the goal for the temperature control system could be to maintain the room temperature at 18^oC — in this case, the performance of the server application depends on the deployed infrastructure; it is very likely that a different performance goal would be derived on a different prototype. In section 4.2 the QoS performance goal for the Rubis application is defined.

To maintain the reference performance, the control system adjusts the control input(s) based on the control output(s) and using the system model. However, it is difficult to know *a priori* the model of the system, especially when dealing with complex server applications. To address the complexity of server applications, this dissertation employs a *black-box* approach to system modelling. To this end, during the system identification process the server is subjected to variable workloads, its performance is measured and the pair of control input/output is identified for the current control task. At the same time the model between the input and the output signal is also derived.

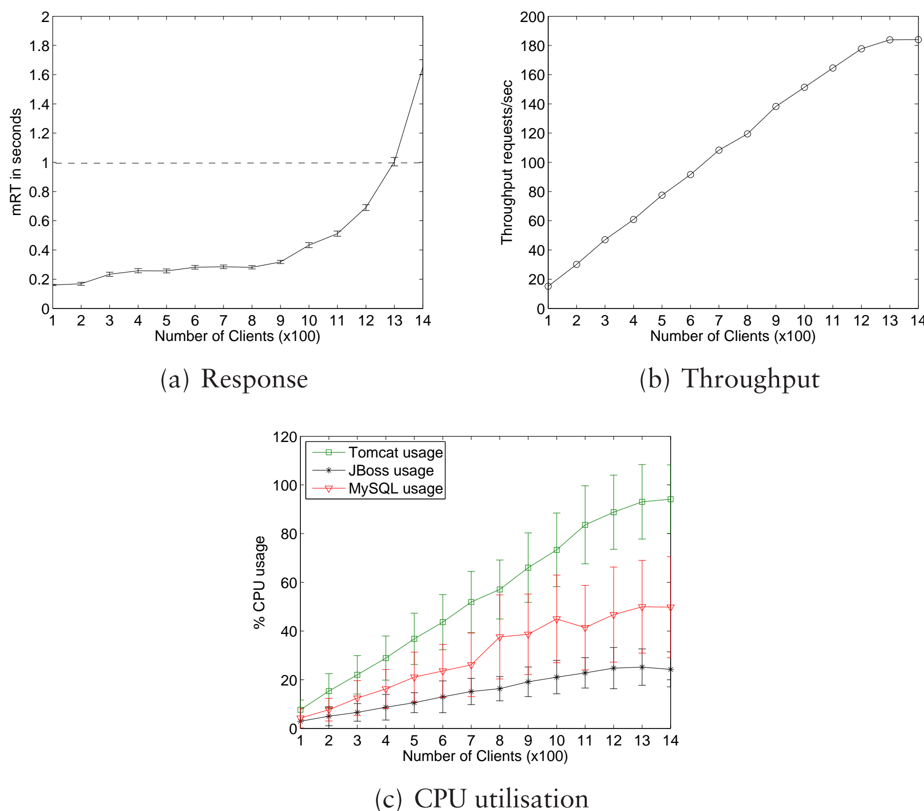
There are two workload parameters that affect the server's performance, namely the workload type mix and the number of clients simultaneously issuing requests to the server (hereafter referred to as the number of clients). Both parameters affect the components' CPU utilisation and consequently Rubis' performance. Analysis in this chapter studies the performance with respect to a changing number of clients and a single workload type mix, namely the browsing mix. A similar analysis can be done for different workload mixes.

The rest of this chapter presents the system identification process towards building the controllers. In particular, Section 4.3 presents the control input/output pair that captures the dynamics of the current system; Section 4.4 describes the model of the system; and Section 4.5 identifies and quantifies the utilisation resource coupling between the multiple tiers.

4.2 QoS Target

This section identifies the reference QoS performance of the Rubis server. The QoS target performance is defined as the number of clients the system can sustain effectively with respect to their response time.

The application performance is measured when each component is allocated 100% of the CPU capacity and the number of clients varies. Figures 4.1(a) shows the mean client response time (hereafter denoted as `mRT`) and Figure 4.1(b) illustrates the corresponding throughput (hereafter denoted as `Throughput`) as measured when the number of clients increases from 100 to 1400 in steps of 100. Each measurement is derived from an experiment where the corresponding number of clients issue requests to the server for 200 seconds (s) in total. The `mRT` is the mean response time over all completed requests. The



(a) Response

(b) Throughput

(c) CPU utilisation

Figure 4.1: System Identification. When the clients vary from 100 and 1200, all components are adequately provisioned for incoming requests. When the number of clients increases from 1300 to 1400, the Tomcat component reaches its maximum allocation, the server saturates, the `mRT` increases and the `Throughput` remains constant. The error bars in Figure 4.1(a) correspond to a 95% confidence interval (CI) around the mean and in Figure 4.1(c) they show $\pm$ one standard deviation (σ) around the mean.

`Throughput` is the number of completed requests divided by the experiment duration in seconds. This is very similar to the `Throughput` when calculated for every second which is therefore not presented here. In all cases, 200s is enough to capture the servers' dynamics.

As the number of clients increases up to 1200, the `mRT` stays well below 1s and the `Throughput` increases linearly with the number of clients. When the number of clients rises beyond 1200 the `mRT` grows beyond 1s, while the `Throughput` remains constant. As expected, the figures show that there is a point of saturation (in this case with respect to the number of clients) below which the server operates effectively and above which its performance is unpredictable. Here, the server saturates at around 1200 clients. If more clients issue requests, the `mRT` increases, as the requests are delayed in the server queues. This also results in each client issuing fewer requests on average (due to the closed-loop nature of the Client Emulator), and the `Throughput` remains constant despite the increasing number of clients.

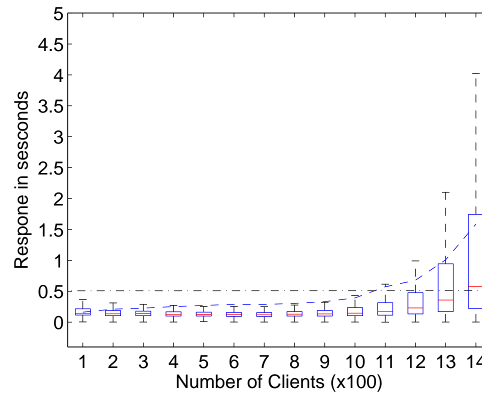


Figure 4.2: System Identification (Response Distributions Summary). For each number of clients, a summary of the clients response distribution is shown. Each boxplot shows the responses between Q_1 and Q_3. The median is also shown (red line within each box). Whiskers are extended to 1.5 the Inter Quantile Range (IQR) above Q_3 and below Q_1. The dashed blue line indicates the mean in each data set. As the number of clients increases (up to 1200) the distributions do not change significantly; only the mean is relatively more increased. When the number of clients exceeds 1200, the server saturates and this is shown by the larger variation of the two rightmost response distributions.

The components' CPU utilisation is also measured and shown in Figure 4.1(c). Again, each point in the graph is the mean of all utilisations (there is one measurement every second for the duration of the experiment). Each component uses more CPU resources as more clients issue requests to the server. When the number of client exceeds 1200 the Tomcat component reaches almost 100% of its allocation and it cannot serve more clients. It becomes the bottleneck component and as a result the `mRT` increases above 1s and the `Throughput` remains constant.

The reference QoS performance level is therefore summarised as: *The Rubis server can serve up to 1200 clients with a performance of* `mRT` $\leq$ 1*s*. This denotes the level of performance the server is expected to achieve, even when the controller dynamically allocates CPU resources to the components. This is also referred to as the *reference QoS performance* of the server or the *reference input* of the control system.

For the above analysis, the mean statistic is used to summarise the response time and CPU utilisation distributions. Further analysis, presented below, shows that the mean is enough to capture the dynamics of the system without loss of generality.

In Figure 4.2 a summary of the main statistics (first quartile Q_1 and third quartile Q_3 (box), median (line within each box)) of each of the 14 response data sets from Figure 4.1 is illustrated. In each boxplot, whiskers (single lines above and below each box) are extended to 1.5 the Inter Quartile Range (IQR) above the Q_3 and below the Q_1. The `mRT` for each data set is also shown by the dashed line across boxplots. Each response distribution is right skewed as the mean is larger than the median. Both the mean and the median remain relatively stable for the first 10 data sets (100 to 1000 clients). For the

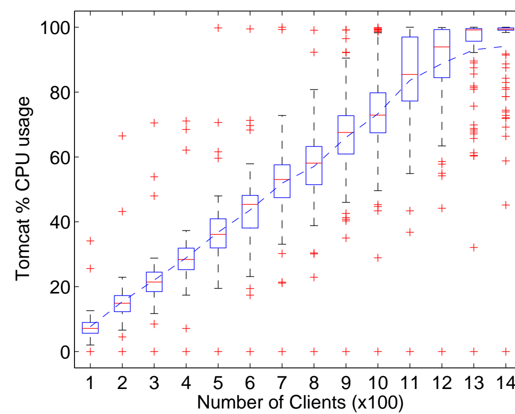


Figure 4.3: System Identification (Tomcat CPU usage Distributions Summary). For each number of clients, a summary of the usages for the Tomcat component is shown. Each boxplot shows the utilisations between Q_1 and Q_3. The median is also shown (red line within each box). Whiskers are extended to 1.5 the IQR above Q_3 and below Q_1. Outliers are plotted by red crosses. The dashed blue line across boxes indicates the mean in each data set. In general, the utilisations are normally distributed for small number of clients (less than 1000). When the number of clients increases the utilisation increases too and its distribution is upper bounded by the physical machine capacity.

last 4 sets (1100 to 1400 clients), the variance in the measured response times increases significantly and both the mean and the median are affected. For the current analysis, both the mean and the median exhibit similar behaviour and can be used as an application performance metric. In the case of the median, the reference performance level of the server is: *The Rubis server can serve up to 1300 clients with a performance of median response time* $< 0.5s$.[1] The dashed-dot line shows the 0.5s cut-off point.

In addition, Figure 4.3 shows a summary of some of the main statistics (Q_1, $Q_{3,}$, median, mean) for each of the 14 utilisation data sets for the Tomcat component. In most cases, the mean (dashed line across boxplots) is very close to the median (solid line within each boxplot), indicating that the CPU usages in each data set are normally distributed. In the rightmost four data sets however, the mean is below the median as the usages are left skewed and most of them close to the 100% of the CPU usage. In the case of the JBoss and MySQL components, the utilisations in all the cases of different clients are (or are very close) to being normally distributed since these components do not saturate. Therefore, in the usage distributions case, either the mean or the median can be used to measure the server's performance.

This section defined the reference QoS performance for Rubis server. The above analysis showed that similar conclusions can be drawn irrespectively of whether the mean or the median of the response and utilisations distributions are used. For the rest of this

[1]The server's performance in the presence of the controllers was also measured in several cases using the median response time and the above statement and similar conclusions to the ones presented in this thesis (Chapter 6) were derived.

dissertation the mean is used as the centrality index.

4.3 Control Signals

In a control system, the selection of the control input(s)/output(s) signals depends on the task assigned to the system. Here, the control system dynamically allocates CPU resources for server components. Therefore, the control input(s) are the parameters that change the CPU allocation of the components. As described in Chapter 3, this is achieved by using the interface exported by the SEDF CPU scheduler and assigning a proportion of the machine's CPU capacity to the running VMs.

The control output(s) is the component's CPU utilisation. The problem addressed here is a CPU allocation one, and intuitively the utilisation provides a very good indication of the allocation itself. In addition, the utilisation indirectly relates to the server's performance; if a server is CPU saturated, it is very likely that its performance is degraded. A component's utilisation indicates its required allocation and to maintain the reference performance, the controller should *follow* the components' utilisations. A change in the usage observed over one period of time can be used to set the allocation for the next one. There are three advantages of using the utilisation: (a) it is easily measured at the server side; (b) it does not require any application domain knowledge; and (c) it has negligible overhead over the control process. Thus, utilisation is a suitable control output signal.

The next section presents the model between the control input/output pair: allocation and utilisation.

4.4 System Modelling

Previous analyses have identified the allocation/utilisation as the control input/output pair and suggested that to achieve reference performance the allocations should *follow* the utilisations. This section identifies a simple type of relationship between the allocation and the utilisation that: (a) satisfies the above statement and (b) handles the mismatch between the metrics (utilisation and `mRT`) that accounts for the control error.

A simple way to model the statement that the allocations *follow* the utilisations is to always assign the allocations to the mean utilisations as computed over a time interval. Although the mean statistic provides a simple summary of the utilisation, it does not however capture the utilisation variability. For instance, the allocations for each component in the case of 800 clients could be set to the corresponding mean utilisations as shown in Figure 4.1(c). However, the error bars in the same figure show that the components' utilisations vary around the mean even for stable workloads. To better assess the use of the mean utilisation and the effect of a component's utilisation variability to its allocation and subsequently to the server's performance the next three experiments are performed.

For a stable workload (e.g. 800 clients of the browsing mix) the allocation of one component is varied in the following way: if u is a component's mean utilisation and r denotes an additional amount of CPU resources, hereafter denoted as `extra` allocation, then its allocation a is assigned by:

$$a = u + r. \tag{4.1}$$

The `extra` allocation increases from 0 up to 40 in steps of 5. The allocation for the other two components is set to 100% of their CPU capacity.

Figures 4.4(a) and 4.4(b) illustrate the `mRT` and the `Throughput` respectively when the Tomcat component is subject to varying allocation. As the `extra` allocation increases, the `mRT` decreases and the `Throughput` increases. Both the `mRT` and `Throughput` stabilise when the `extra` allocation is 15. Increasing the allocation beyond this value does not improve the performance significantly. Similar experiments are performed for the other two components and the results are shown in Figures 4.4(c) and 4.4(d) for the JBoss component, and in Figures 4.4(e) and 4.4(f) for the MySQL component. A similar analysis indicates that the `extra` allocation should be set to 10 for the other two components.[2]

Results indicated that to maintain the reference server performance the allocation can be assigned to the mean utilisation plus a value of the parameter r which should be above a certain threshold. The parameter r captures the utilisation variability. Note that the reference server performance is achieved for various r values above the threshold. To estimate the minimum such value a much larger number of experiments (e.g. varying number of clients, changing workload mixes, combinations of components and varying allocations) is required. However, the current analysis aims to identify the system model between the control input and control output. Assigning the parameter r to its *best* value is part of the tuning process in a live system. Results in Chapter 6 examine how the values of this parameter affect the server's performance.

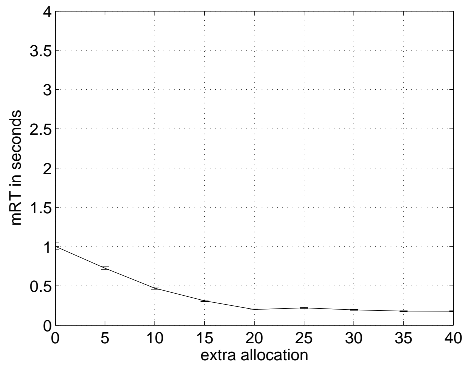
Maintaining the allocation above the utilisation is a common practice and has also been used elsewhere. In data centres there is usually a *headroom* of CPU resources above the utilisation to enable applications to cope with workload fluctuations and variable utilisations. In this case, the allocation is expressed as a multiple of the utilisation and takes the form:

$$a = x * u, \tag{4.2}$$

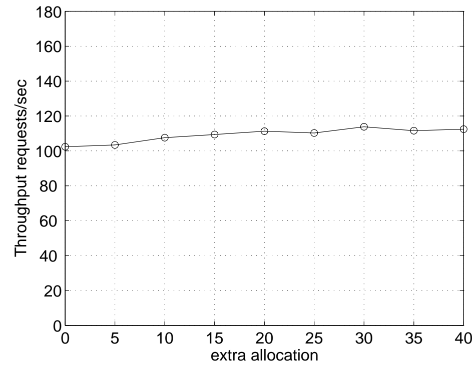
where x should be > 1. In [PSZ$^+$07] an analysis that used the later model for a 2-components virtualized Rubis server showed that as long as the allocation was at least equal to a utilisation proportion (well above 1) the application achieved good performance. Further increasing the allocation above this threshold did not improve the application performance significantly.

This section has identified the type of relationship between the allocation and the utilisation. In general, to sustain the reference performance, the allocation should *follow* the

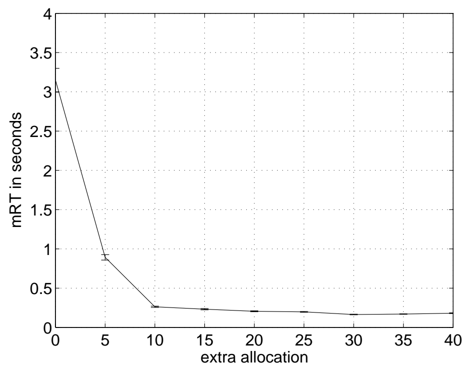
[2] The current experiments provide an approximation of the `extra` allocation values.



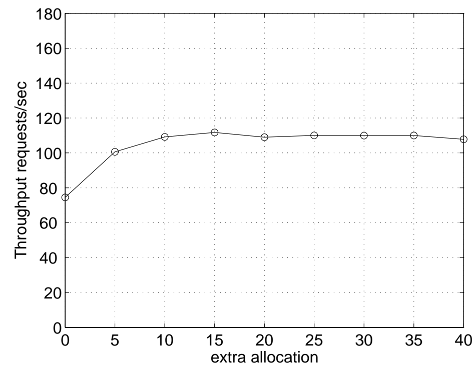
(a) mRT for Tomcat allocations



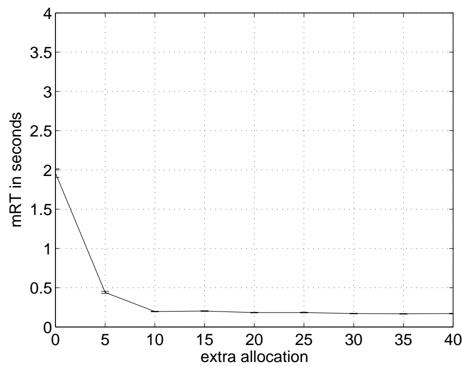
(b) Throughput for Tomcat allocations



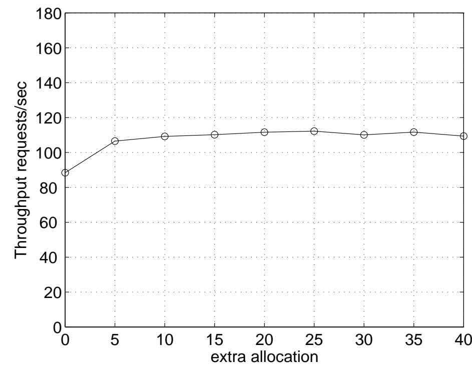
(c) mRT for JBoss allocations



(d) Throughput for JBoss allocations



(e) mRT for MySQL allocations



(f) Throughput for MySQL allocations

Figure 4.4: Extra Allocation. These figures illustrate the server's performance when the allocation of each component is changing for a fixed number of 800 clients. Each point comes from an experiment of 100s duration. In all three components, there is an extra allocation after which the server's performance stabilises. The error bars in the mRT figures correspond to a 95% CI around the mean as calculated over .

mean utilisation and it should also provide some additional resources to the mean. Additional resources can be expressed either (a) as an *additive* term to the mean utilisation (Equation (4.1) is referred hereafter to as the *additive model*), or (b) as a *multiplicative*

factor of the mean utilisation (Equation (4.2) is hereafter denoted as the *multiplicative* model). These terms are subject to each application and the reference QoS input. The controllers presented in the next chapter are built based on these models.

4.5 Inter-Component Resource Coupling

The system modelling process in the previous section analysed the server's performance with respect to the behaviour of each component. In multi-tier applications, there is, however, a resource coupling between the components. This section starts with an example to illustrate the drawbacks when not considering the resource coupling when allocating resources to virtualized components. It then proceeds to model the utilisation resource coupling for the components of the Rubis benchmark application.

Figure 4.5 illustrates a system that controls the allocations of each tier of a 3-component application independently. For each component, the dashed line indicates its required usage for the current workload,[3] and the solid line shows its current allocation. The shaded part of each component shows the actual usage. The figure illustrates two snapshots of the allocation procedure. In the top illustration component B is the *bottleneck* tier, since its allocation is less than its required usage. This results in under-utilisation of the other two tiers, despite those having been allocated with enough resources (solid lines) for the current workload. Later, the controller for component B adjusts the allocation of the bottleneck tier. The allocations of the other tiers remain unchanged, as they have not reached saturation point. Depending on the allocation of each of the other tiers, the bottleneck point can then be moved to the tier(s) with the smallest difference between the allocation and the usage, e.g. component C, as shown in the bottom illustration. This scheme, where each tier's allocation is controlled independently, would result in slower overall response to workload changes, as the bottleneck point could move from component to component.

In a multi-tier application, each component uses a different amount of resources to process incoming requests (even when tiers are running on machines of the same physical capacity), since they perform different sets of operations for each input. When dynamically allocating component resources, the controller should adjust them in a manner that meets each component's distinct demands. In fact, in multi-tier systems the resource usages of the different components are closely related and a control system that considers this behaviour is appropriate.

Recall that the Rubis components consume different amounts of CPU resources, with Tomcat consuming the most and JBoss the least (Figure 4.1(c)). The components' CPU usages are coupled since the workload on each component is affected by the workload on the rest, as long as there are adequate resources. If one of the components does not have enough resources to process all the incoming requests (bottleneck component), then the rest of the components cannot process the requests of more clients.

[3] Assuming for this example that the components' usage for the current workload is measured and known in advance.

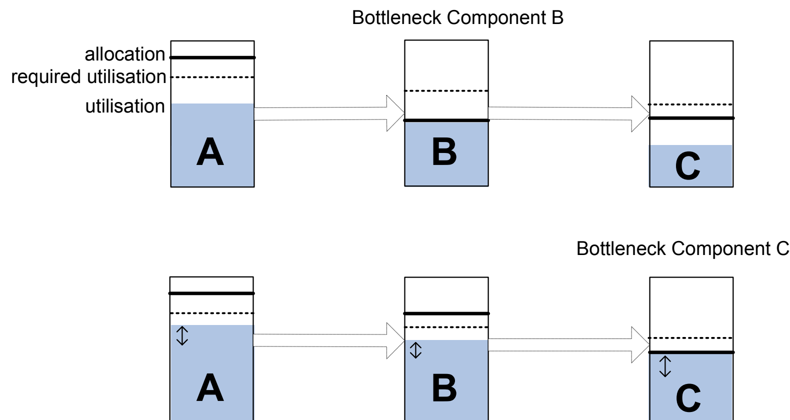
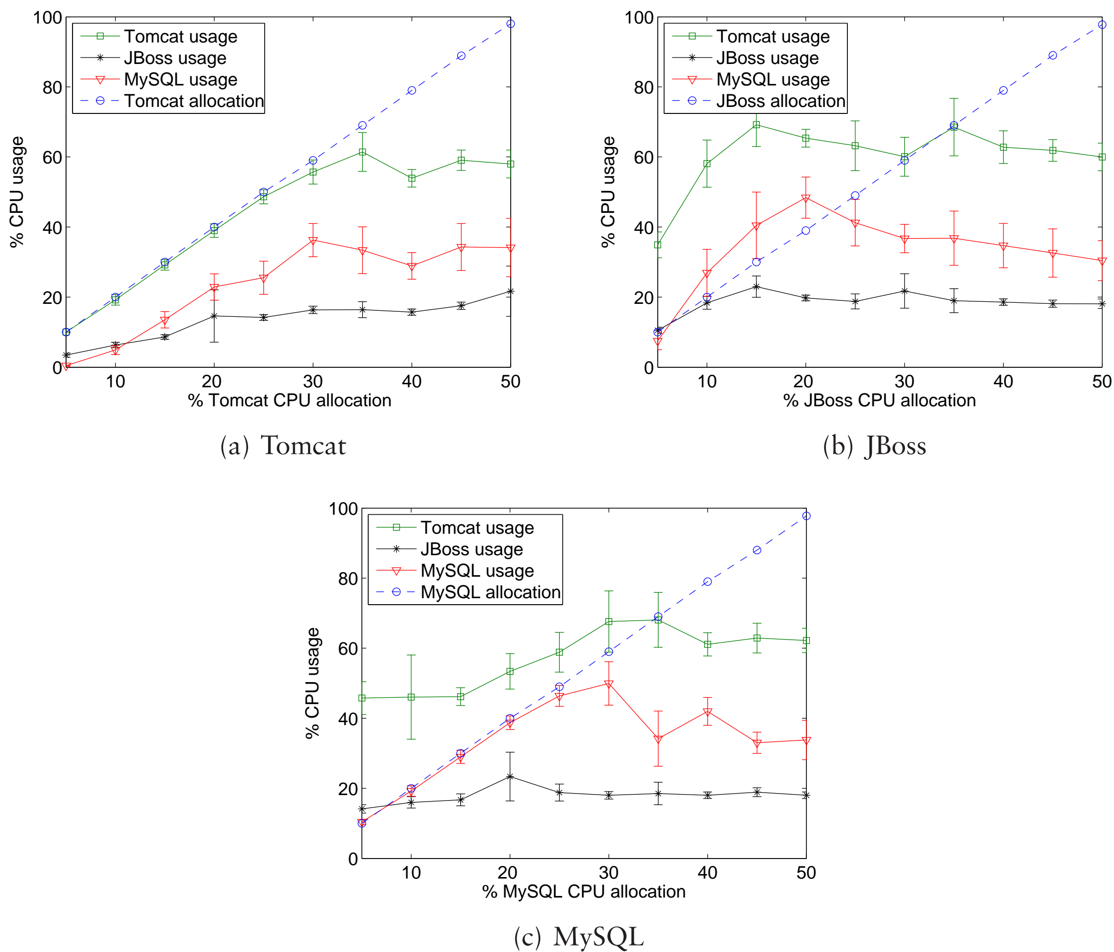


Figure 4.5: Inter-Component Resource Coupling Example. The figures illustrate two snapshots of the allocation procedure in the case of a 3-component application, where each tier is provisioned individually. The solid lines indicate the components' resource allocation, the dashed lines show how much resources are required for the current workload, and the shaded part illustrates the actual usage. As shown in the topmost diagrams, the component B lacks of resources to process the incoming workload and is therefore, the bottleneck component. When more resources are allocated to it (bottommost diagrams), the saturation point moves to component C, whose allocation remains the same and not enough for the current workload.

This is further illustrated by the following experiment. The CPU allocation of one of the three components is varied from 10 to 100 in increments of 10, the number of clients is kept constant at 800, and each of the other two components is allocated 100% of its CPU capacity. Initially, the allocation of the Tomcat component is varied. As shown in Figure 4.6(a) its usage follows the allocation until the allocation exceeds the required one for the current workload. The usage for the other two components increases slowly, despite their having the necessary resources to serve 800 clients. In this case the bottleneck component is Tomcat, and since it does not have adequate resources to cope with the current workload, the other components' usages are affected as well. Similar behaviour is observed when either the JBoss or the MySQL components are the bottlenecks as shown in Figures 4.6(b) and 4.6(c).

Overall, in the case of a bottleneck, an increase of its allocation eventually leads to the increase in the CPU usage of the other components, suggesting that their allocations should be increased as well. A controller that takes into account the CPU usage of all the components and assigns the CPU allocation to each of them will clearly do better than one that does not.

The provisioning of multi-tier applications based on a model of all tiers' resource demands was also proposed in [UC05]. The authors proposed the use of an analytical model to compute the resource demands of all tiers and then allocated servers for each tier accordingly.



(a) Tomcat

(b) JBoss

(c) MySQL

Figure 4.6: Inter-Component Resource Coupling Experiments. These figures illustrate the components' utilisations when one of the three is subject to variable allocation. In all cases, if one of the components is not adequately provisioned to serve incoming requests (800 clients), the rest also consume less resources, and the server's performance is affected. The error bars correspond to a 95% CI around the mean utilisation.

coefficients name	coefficients values	R^2
γ^1, δ^1	3.77, -8.23	0.98
γ^2, δ^2	0.47, 1.68	0.98
γ^3, δ^3	0.55, 1.21	0.97

Table 4.1: Parameters of the Models of Components' Utilisation Coupling.

This section continues to quantify the resource coupling among the utilisations of the different components using a black-box approach. This relationship will later be used to built the Multi-Input Multi-Output (MIMO) Usage Based (MIMO-UB) controller. First, the relationships between the different components' usage are extracted. Data is collected (10 sets of CPU usages over 100s duration each, for all three components running with clients in the range of [100,1200]) and then processed with the aid of the MATLAB Curve Fitting Toolbox [MAT]. The CPU usage for all components (denoted by u^1, u^2 and u^3) are found to be related by the following formulae:

$$u^1 = \gamma^1 u^2 + \delta^1, \quad (4.3)$$
$$u^2 = \gamma^2 u^3 + \delta^2, \quad (4.4)$$
$$u^3 = \gamma^3 u^1 + \delta^3, \quad (4.5)$$

where γ^i, δ^i are the coefficients found.[4] The coefficients found are shown in Table 4.1. In general $R^2 \geq 0.8$ provides a very good fit to the data.

4.6 Summary

This chapter presented the system identification analysis for the Rubis server system. Through experimental analysis the following emerged: (a) the reference QoS input that the controller system should maintain was identified (Section 4.2); (b) the (allocation, utilisation) signal pair was presented to control and monitor the server applications (Section 4.3); (c) a linear model between the two signals was derived (Section 4.4), and (d) the resource coupling model between component utilisations was also given (Section 4.5).

The next chapter presents the controllers design based on the above conclusions from the system identification analysis. Each controller aims to maintain the reference performance despite workload changes. It uses the utilisation control output to monitor the server and, based on the system model, it updates the allocations.

[4]Two of the equations are adequate to describe the relationships between all three components, but, they are all retained here for notational simplicity.

5
Controllers Design

This chapter presents five controllers that dynamically adjust the CPU allocation of virtualized applications. The controllers are designed based on the system identification analysis from previous chapter. In particular, they make use of the observation that the VM allocation should *follow* the utilisation of the hosted application. There are two models that describe this relationship: the additive (Equation (4.1)) and the multiplicative model (Equation (4.2)). Some also make use of the fact that there is a utilisation resource coupling among the components in a multi-tier application.

Based on these observations, this chapter presents five novel controllers: (a) the SISO Usage-Based (SISO-UB) controller (Section 5.1.1) controls the CPU allocation of individual tiers based on their utilisation; (b) the Kalman Basic Controller (KBC) controller (Section 5.1.4) also adjusts the CPU allocation of individual tiers and is based on the Kalman filtering technique; (c) the MIMO Usage-Based (MIMO-UB) controller (Section 5.2.1) extends the SISO-UB controller to collectively allocate resources for multi-component applications; (d) the Process Noise Covariance Controller (PNCC) controller (Section 5.2.2) expands the KBC design for multi-tier applications; and, (e), the Adaptive PNCC (APNCC) (Section 5.3) controller which further extends the PNCC design with online estimation of the model parameters. Table 5.1 below presents all controllers notation used in this dissertation:

5.1 Single-Tier Controllers

This section presents two Single-Input Single-Output controllers that adjust the CPU allocations for each application component separately. Figure 3.1 of the evaluation plat-

symbol	description
n	number of application components
i	component index
a_k^i	allocation of component i at interval k
u_k^i	measured usage of component i at interval k
p^i	minimum proportion of utilisation assigned to the allocation of component i
e_k^i	controller error for component i at interval k
λ	tunable parameter to multiple the control error
r^i	`extra` resources for component i
v_k^i	real usage of component i at interval k
t_k	process noise of real utilisation v at interval k
z_k	process noise of allocation a at interval k
w_k	measurement noise at interval k
c	fraction of the utilisation that accounts for the final allocation
Q	process noise variance
S	measurement noise variance
K_k	Kalman gain at interval k
$\widetilde{a}_k$	*a priori* allocation estimation at interval k
$\widehat{a}_k$	*a posteriori* allocation estimation at interval k
$\widetilde{P}_k$	*a priori* estimation error variance at interval k
$\widehat{P}_k$	*a posteriori* estimation error variance at interval k
$\mathbf{a}_k$	allocation for all components at interval k, $\mathbf{a}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{u}_k$	measured utilisation for all components at interval k, $\mathbf{u}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{P}$	diagonal matrix with the p^i values along its diagonal, $\mathbf{P} \in \mathbb{R}^{n\times n}$
$\mathbf{M}$	array with the coefficients of the usage coupling models, $\mathbf{M} \in \mathbb{R}^{n\times n}$
$\mathbf{r}$	reference values for all components, $\mathbf{r} \in \mathbb{R}^{n\times 1}$
$\mathbf{e}_k$	control errors from all components , $\mathbf{e}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{e}_k(i)$	controller error for component i at interval k, i^{th} element of $\mathbf{e}_k$
$\widetilde{\mathbf{a}}_k$	*a priori* allocation estimations for all components at interval k, $\widetilde{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$
$\widehat{\mathbf{a}}_k$	*a posteriori* allocation estimations for all components at interval k, $\widetilde{\mathbf{a}}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{W}_k$	process noise for all components at interval k, $\mathbf{W}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{V}_k$	measurement noise for all components at interval k, $\mathbf{V}_k \in \mathbb{R}^{n\times 1}$
$\mathbf{C}$	array with the c values for all components along its diagonal, $\mathbf{C} \in \mathbb{R}^{n\times n}$
$\mathbf{Q}$	process noise covariance matrix, $\mathbf{Q} \in \mathbb{R}^{n\times n}$
$\mathbf{S}$	measurement noise covariance matrix, $\mathbf{S} \in \mathbb{R}^{n\times n}$
$\mathbf{K}$	Kalman gains for all components, $\mathbf{K} \in \mathbb{R}^{n\times n}$
$\mathbf{Q}_k$	process noise covariance matrix at interval k, $\mathbf{Q}_k \in \mathbb{R}^{n\times n}$
$\mathbf{R}_k$	measurement noise covariance matrix at interval k, $\mathbf{R} \in \mathbb{R}^{n\times n}$
$\mathbf{K}_k$	Kalman gains for all components at interval k, $\mathbf{K} \in \mathbb{R}^{n\times n}$

Table 5.1: Controllers Notation.

form illustrates the way three SISO controllers adjust the allocations for the Rubis components. Both SISO controllers are built based on the observation from the system identification process, that the allocation *follows* the utilisation. In particular, Section 5.1.1 presents the SISO Usage Based controller (SISO-UB). This is a simple controller which adjusts the allocation based on the last interval mean utilisation and uses both system models (Equations (4.1) and, (4.2)). The second SISO Kalman Basic controller (KBC) uses the Kalman filter to *track* the utilisation and update the allocation accordingly. This is a more advanced approach where a filtering technique is used to eliminate the noise of the CPU utilisation signal and still discovers its main fluctuations. To better explain the KBC controller, this section also briefly presents the Kalman filter (Sections 5.1.2 and 5.1.3). Finally, the two controllers also differ in the way their control errors are defined. The control error in the SISO-UB controller uses the additive model (Equation (4.1)), while the KBC controller uses the multiplicative model (Equation (4.2)).

5.1.1 SISO Usage-Based Controller

This section presents the SISO-UB controller (Figure 5.1). The SISO-UB notation is given first. If i denotes the application component, then: a_k^i is defined as the proportion of the total CPU capacity of a physical machine allocated to a running VM for interval k; u_k^i denotes a component's CPU usage or utilisation as the proportion of the total CPU capacity of a physical machine measured to be used by that component for interval k; and, p^i is a tunable parameter which indicates the lowest proportion of the utilisation that the allocation is assigned to and its values are > 1. The SISO-UB control law is given by:

$$a_{k+1}^i = p^i u_k^i + \lambda e_k^i, \tag{5.1}$$

where e_k^i, its control error, is calculated as:

$$e_k^i = |r^i - (a_k^i - u_k^i)|, \tag{5.2}$$

where λ in (5.1) is a tunable parameter which shows the portion of the control error that is considered towards the final allocation; and r^i in (5.2) denotes the `extra` allocation required for this component. Recall that the `extra` allocation is the additional amount of CPU resources added to the mean utilisation in order for the component to be adequately provisioned for incoming requests; it is required to capture the CPU utilisation variability around the mean.

The SISO-UB controller aims to allocate enough resources for each component to serve incoming requests based on the previous interval's utilisation. System identification analysis showed that there are two ways to model required CPU resources as a function of the utilisation: the additive model (Equation (4.1)) and the multiplicative one (Equation (4.2)). The SISO-UB control law draws on both models.

First, according to the additive model, the difference between the allocation and the utilisation should be sustained around a reference value. This is incorporated in the control error (5.2). The allocation a_{k+1}^i depends on the control error λe_k^i, which shows the dif-

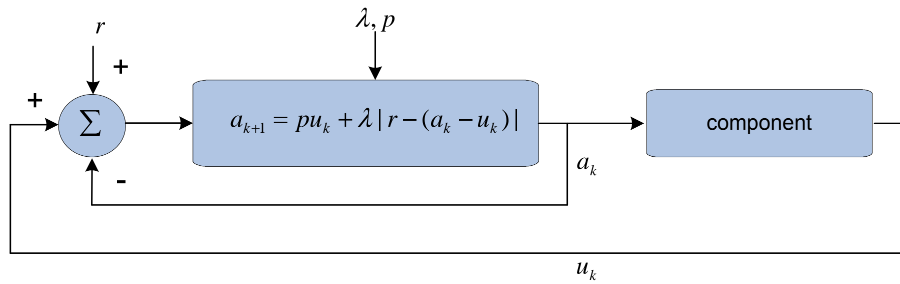


Figure 5.1: SISO-UB Controller Layout. The allocation for the next interval a_{k+1} is calculated based on the utilisation u_k of the previous interval and the control error e_k, which denotes the difference between the allocation and the utilisation over the previous interval from the reference value r. The controller takes as input the parameters: r, λ and p. For simplicity the superscript component index i is omitted.

ference between the allocation and the utilisation, and the reference value r^i. To always allocate more resources than the CPU utilisation, the absolute error is used. Different λ values can be used to build controllers with different reactions to the control error. However, if the difference between the allocation a_k^i and the utilisation u_k^i equals the reference value r^i, then the control error e_k^i becomes 0. To always allocate more CPU resources than the previous utilisation, p^i is also introduced. The allocation a_{k+1}^i is also proportional to the utilisation $p^i u_k^i$ (use of the multiplicative model (4.2)) and since $p^i > 1$, the controller always allocates more resources than the mean utilisation u_k^i, providing therefore, some minimum resources.

The SISO-UB control law comes from a combination of the two system models. The final allocation is a sum of the utilisation of the previous interval plus some additional resources, which come from both the utilisation proportion and the control error. The advantages of using the control error and the utilisation proportion are two-fold. First, when a high control error occurs, the controller allocates more resources according to the error, thereby acting faster to considerable changes than when just using the utilisation proportion. Second, the control error enables the controller to provide adequate resources when the utilisation is low and its variability high. For example, consider a system with low utilisation $u_k^i = 20$, high variance (values of u^i during interval $k \in [20 - 15, 20 + 15]$, and $p^i = 1.25$. If just the multiplicative model is used, then $a_{k+1}^i = 25$. However, this allocation is lower than several possible utilisation values, and therefore, the application could be inadequately provisioned. Now add the control error. If $a_k^i = 23, \lambda = 0.8$ and $r^i = 15$, then $a_{k+1}^i = 34.6$. The final allocation has now increased, and the application has more resources to serve incoming requests with variable utilisation demands. The combination of the two models provides a flexible scheme for applications with diverse characteristics.

Stability

The SISO-UB is stable when $|\lambda| < 1$, as shown below. For $(r^i - (a_k^i - u_k^i)) > 0$ the allocation signal is:

$$a_{k+1} = pu_k + \lambda(r - a_k + u_k),$$

where the superscript component index i is omitted for simplicity and without loss of generality. The Z-transform of the allocation signal is:

$$\begin{aligned} zA(z) - za(0) &= pU(z) + \lambda * (\frac{r}{1 - z^{-1}} - A(z) + U(z)) \Leftrightarrow \\ zA(z) - za(0) &= pU(z) + \frac{\lambda r}{1 - z^{-1}} - \lambda A(z) + \lambda U(z) \Leftrightarrow \\ (z + \lambda)A(z) &= (p + \lambda)U(z) + za(0) + \frac{\lambda r}{1 - z^{-1}}. \end{aligned}$$

So, the transfer function is:

$$T(z) = \frac{A(z)}{U(z)} = \frac{p + \lambda}{z + \lambda},$$

where the denominator is the characteristic function. The pole of the characteristic equation is:

$$z + \lambda = 0 \Leftrightarrow z = -\lambda$$

and for stability it suffices for the pole to be within the unit circle:

$$|z| < 1 \Leftrightarrow \vdash \lambda| < 1. \tag{5.3}$$

When $(r^i - (a_k^i - u_k^i)) < 0$ a similar analysis shows that the transfer function is:

$$T(z) = \frac{p - \lambda}{z - \lambda}.$$

The pole of the characteristic equation is:

$$z - \lambda = 0 \Leftrightarrow z = \lambda$$

and for stability it suffices:

$$|z| < 1 \Leftrightarrow |\lambda| < 1. \tag{5.4}$$

To summarise, from Equations (5.3) and (5.4) the SISO-UB controller is stable when:

$$|\lambda| < 1. \tag{5.5}$$

Discussion

This section has presented the SISO-UB controller. This is a simple controller that uses the mean utilisation of the last interval and the two system models (the additive and the multiplicative) to update the allocation for the next interval. The next sections present the integration of the Kalman filtering technique into a feedback controller. The advantage of using a filter to track the utilisation over the SISO-UB approach is the ability of filters to "clean" a signal from noise and discover its main fluctuations. This is particularly attractive in the case of noisy CPU utilisations. In this dissertation, the Kalman filter is used because it is the optimal recursive estimator when certain conditions hold. It also provides good results even when these conditions are relaxed and is a very well researched technique.

5.1.2 The Kalman Filter

Since first presented by R.E. Kalman in his seminal 1960 paper [Kal60], the Kalman filter has been used in a large number of areas including autonomous or assisted navigation, interactive computer graphics, motion prediction, and so on. It is a data filtering method that estimates the state of a linear stochastic system in a recursive manner based on noisy measurements. The Kalman filter is optimal in the sum squared error sense under the following assumptions: (a) the system is described by a *linear* model and (b) the process and measurement noise are *white* and *Gaussian*. It is also computationally attractive, due to its recursive computation, since the production of the next estimate only requires the updated measurements and the previous predictions.

To briefly illustrate the basics of the Kalman filter, a very simple example is now discussed [SGLB99]. Assume that via measurements the value of a quantity, let's say `length`, is to be calculated. Every new measurement (same or different equipment can be used, such as a mechanical ruler or a laser system) provides an observation of the true value with some error. Assume that N measurements are taken. An estimate of the `length` that minimises the distances from all the measurements can easily be calculated using the normalised Euclidean distance. The result minimises the sum of the distances to all measurements, weighted by their standard deviations. Consider now the case where, after each new measurement, a new best estimation of the `length` is to be calculated. A simple but expensive approach would be to calculate the new best `length` given all the previous measurements at hand plus the new one. With the Kalman filter estimator, the new best `length` is calculated using the current best estimate so far and the next measurement. Note that this is an iterative process and, at any point, no more than two measurement states are involved.

In what follows, an introduction to the basics of the Kalman filter is provided. Emphasis is given to presenting those that are needed in later sections to approach the problem of the CPU allocation for virtualized server applications. There are numerous papers and books that provide a thorough and more comprehensive analysis of the Kalman filter (e.g. [Sim06, May79, WB95]).

5.1.3 Kalman Filter Formulation

In this subsection the basics of the discrete Kalman filter are presented.[1] The Kalman filter estimates the next state of the system. The states usually correspond to the set of the system's variables that are of interest. The evolution of the system's states is described by the following *linear* stochastic difference equation:

$$x_{k+1} = Ax_k + Bb_k + w_k, \tag{5.6}$$

where x is a $n \times 1$ vector representing the system states; A is a $n \times n$ matrix that represents the way the system transits between successive states in the absence of noise and input; b is the optional $m \times 1$ vector of system inputs; the $n \times m$ B matrix relates the inputs to the states; w is the $n \times 1$ process noise vector; and the subscript k denotes the interval.

The states x are *linearly* related to the measurements z:

$$z_k = Hx_k + v_k, \tag{5.7}$$

where the $n \times n$ matrix H represents the transition between the states and the measurements; and v is the $n \times 1$ measurement noise vector. A and H might change between time steps, but here they are assumed to be constant.

The measurement and process noise are independent of each other, stationary over time, *white*, and *normally distributed*:

$$p(w) \sim N(0, Q),$$
$$p(v) \sim N(0, S).$$

The state x_{k+1} of the system at any time point $k+1$ is unknown. The purpose of the Kalman filter is to derive an estimate of x_{k+1}, given the measurements (5.7) and the way the system evolves (5.6), while minimising the estimation error. This process is described below.

The Kalman filter always iterates between two estimates of the x_k state; the *a priori* estimate denoted as $\widetilde{x}_k$ and the *a posteriori* estimate given by $\widehat{x}_k$, as shown in Figure 5.2. It first estimates the *a priori* state $\widetilde{x}_k$ during the *Predict* or *Time Update* phase. This is a prediction of the state x_k given the measurements so far over the previous intervals and using the system model (5.6). Since this is only a prediction of the true state x_k, the Kalman filter later further adjusts this estimation closer to the true value by incorporating the updated knowledge of the system coming from the new measurements. This latter phase is called the *Correct* or *Measurement Update* phase and it results in the *a posteriori* estimate $\widehat{x}_k$. The is the closest and final estimation to the real value of x_k.

The exact relationship between the two estimates is given by:

$$\widehat{x}_k = \widetilde{x}_k + K_k(z_k - H\widetilde{x}_k). \tag{5.8}$$

[1]Notation in this section is independent of the notation used elsewhere in this dissertation and is not contained in Table 5.1.

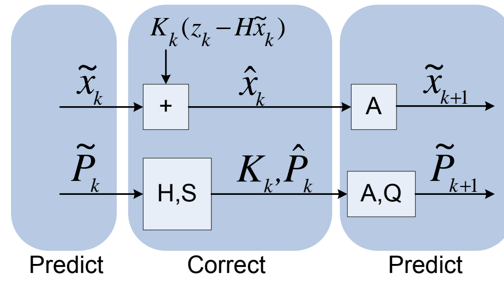


Figure 5.2: Kalman Filter Overview. The Kalman filter operates between two phases, the *Predict* and the *Correct*. During the predict phase, the filter estimates the next state of the system $\widetilde{x}_k$ based on the observations so far. During the correct phase, the filter adjusts its prediction using the latest observation to $\widehat{x}_k$. This is also the prediction for the next phase $\widetilde{x}_{k+1}$. All predictions are made using the system's dynamics through the Kalman gain K_k which depends, among other variables, on the process noise variance Q and measurement noise variance S.

In this equation the *a priori* estimate is adjusted as given by the residual $z_k - H\widetilde{x}_k$, also known as the *innovation*. The factor K_k is called the *Kalman gain*. The Kalman gain is calculated by minimising the mean squared error of the estimation. The purpose of the filter is to make the best estimate of the next state. The error in the estimate is calculated by:

$$\widehat{P}_k = E[(x_k - \widehat{x}_k)(x_k - \widehat{x}_k)^T], \tag{5.9}$$

where $\widehat{P}_k$ is the *a posteriori* covariance error $n \times n$ matrix and E is the expected value.

The Kalman gain K_k that minimises the covariance error is derived by substituting the $\widehat{x}_k$ from Equation (5.8) in Equation (5.9) and performing the necessary calculations. The resulting minimum gain is:

$$K_k = \widetilde{P}_k H^T (H\widetilde{P}_k H^T + S)^{-1} \tag{5.10}$$

where $\widetilde{P}_k$ is the *a priori* estimate of the $\widehat{P}_k$. The updated error covariance is thus given by:

$$\widehat{P}_k = (I - K_k H)\widetilde{P}_k. \tag{5.11}$$

Finally, using the updated estimates the new predicted values for the next state and the covariance matrix are:

$$\widetilde{x}_{k+1} = A\widehat{x}_k + Bb_k, \tag{5.12}$$

$$\widetilde{P}_{k+1} = A\widehat{P}_k A^T + Q. \tag{5.13}$$

Simply put, the Kalman filter takes as input the predictions of the new state and the covariance error matrix and, using the new measurements, it produces the new adjusted estimations. The error covariance matrix is also used to evaluate the predictions.

The Kalman filter has been widely applied, although not all systems can be described

through linear processes. There are cases involving an *assumption* that a linear model describes a system and captures its main dynamics where it is enough to apply the Kalman filter. This is the direction taken in this dissertation; results in Chapter 6 show that the current approach is sufficient.

5.1.4 Kalman Basic Controller

This section presents the SISO Kalman Basic Controller (KBC) (Figure 5.3). It is a utilisation tracking controller based on the Kalman filtering technique. Rather than using Kalman filters to estimate the parameters of an application performance model [ZYW$^+$05], Kalman filters are here used both as a tracking method *and* to build a feedback controller. The Kalman filter is particularly attractive since it is the optimal linear filtering technique when certain conditions hold and has good performance even when those conditions are relaxed. All metrics presented in this subsection are scalar and refer to a single component.

The time-varying CPU usage is modelled as a one-dimensional random walk. The system is thus governed by the following linear stochastic difference equation:

$$v_{k+1} = v_k + t_k, \tag{5.14}$$

where v_k is the proportion of the total CPU capacity of a physical machine actually used by a component and the independent random variable t_k represents the process noise and is assumed to be normally distributed.

Intuitively, in a server system the CPU usage in interval v_{k+1} will generally depend on the usage of the previous interval v_k as modified by changes, t_k, caused by request processing, e.g. processes being added to or leaving the system, additional computation by existing clients, lack of computation due to I/O waiting, and so on.[2] Knowing the process noise and the usage over the previous interval v_k, one can predict the usage for the next interval v_{k+1}.

To achieve reference performance the KBC controller uses the multiplicative system model Equation (4.2). To this end, the allocation should be maintained at a certain level $\frac{1}{c}$ of the usage, where c is customised for each server application or VM. c parameter corresponds to the transition matrix H from Equation (5.7) between the states, in this case the allocation, and the measurements. The allocation signal is described by:

$$a_{k+1} = a_k + z_k, \tag{5.15}$$

and the utilisation measurement u_k relates to the allocation a_k, as:

$$u_k = ca_k + w_k. \tag{5.16}$$

The independent random variables z_k and w_k represent the process and measurement

[2] In the current context of virtualized servers, v_k also models the utilisation "noise" coming from the operating system that runs in the VM.

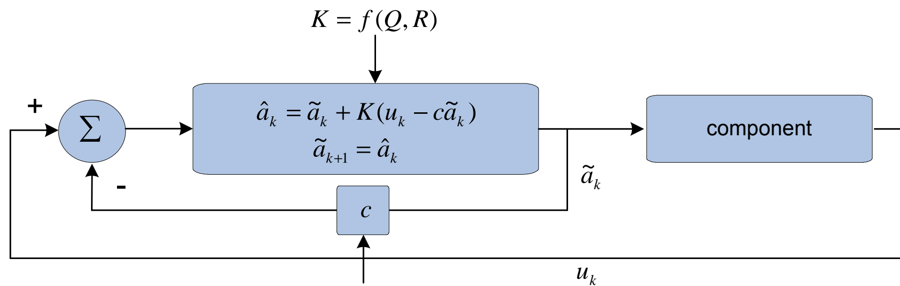


Figure 5.3: KBC Controller Layout. The KBC controller is based on the Kalman filter to adjust the CPU allocation of individual components. The controller uses the *a priori* estimation of the allocation $\widetilde{a}_k$ and the new measurement u_k to compute the allocation for the next interval $\widetilde{a}_{k+1}$ using the Kalman gain K_k. The Kalman gain is a function of the input parameters Q and S which are computed offline.

noise respectively, and are assumed to be normally distributed:

$$p(z) \sim N(0, Q),$$
$$p(w) \sim N(0, S).$$

The measurement noise variance S might change with each time step or measurement. Also, the process noise variance Q might change in order to adjust to different dynamics; however, for the rest of this section they are assumed to be stationary during the filter operation. Later, another approach, which considers non-stationary noise, is presented. Given that the equations (5.15) and (5.16) describe the system dynamics, the required allocation for the next interval is a direct application of the Kalman filter theory, presented below.

$\widetilde{a}_k$ is defined as the *a priori* estimation of the CPU allocation, that is the predicted estimation of the allocation for the interval k based on previous measurements. $\widehat{a}_k$ is the *a posteriori* estimation of the CPU allocation, that is the corrected estimation of the allocation based on measurements. Similarly, the *a priori* estimation error variance is $\widetilde{P}_k$ and the *a posteriori* estimation is $\widehat{P}_k$. The predicted *a priori* allocation for the next interval $k+1$ is given by:

$$\widetilde{a}_{k+1} = \widehat{a}_k, \tag{5.17}$$

where the corrected *a posteriori* estimation over the previous interval is:

$$\widehat{a}_k = \widetilde{a}_k + K_k(u_k - c\widetilde{a}_k). \tag{5.18}$$

At the beginning of the $k+1$ interval the controller applies the *a priori* $\widetilde{a}_{k+1}$ allocation. If the $\widetilde{a}_{k+1}$ estimation exceeds the available physical resources, the controller allocates the maximum available. In the region where the allocation is saturated, the Kalman filter is basically inactive. Thus, the filter is active only in the underloaded situation where the dynamics of the system are linear. The correction Kalman gain between the actual and

the predicted measurements is:

$$K_k = c\widetilde{P}_k(c^2\widetilde{P}_k + S)^{-1}. \tag{5.19}$$

The Kalman gain K_k stabilises after several k iterations (Appendix A). The *a posteriori* and *a priori* estimations of the error variance are respectively:

$$\widehat{P}_k = (1 - cK_k)\widetilde{P}_k, \tag{5.20}$$

$$\widetilde{P}_{k+1} = \widehat{P}_k + Q. \tag{5.21}$$

Kalman Gain

The Kalman gain is important when computing the allocation for the next interval $\widetilde{a}_{k+1}$. It is a function of the variables Q and S which describe the dynamics of the system. In general, K_k monotonically increases with Q and decreases with S. This is also explained intuitively: Consider a system with large process noise Q. Its states experience large variation, and this is shown by the measurements as well. The filter should then increase its confidence in the new error (the difference between the predicted state and the measurement), rather than the current prediction, in order to keep up with the highly variable measurements. Therefore the Kalman gain is relatively large. On the other hand, when the measurement noise variation S increases, the new measurements are biased by the included measurement error. The filter should then decrease its confidence in the new error as indicated by the smaller values of the Kalman gain. In fact the Kalman gain depends on the ratio $\frac{S}{Q}$ (Appendix A). In addition, the original Kalman gain values as computed from Q and S can be tuned to make the filter more or less reactive to workload changes; as will be demonstrated by the results shown in Chapter 6.

Modelling Variances

To obtain a good estimation of the allocation process noise variance Q, since it is considered to be proportional to the usage, it is enough to estimate the usage variance and then evaluate it via the following formula (var denotes variance):

$$var(a) \simeq var(\frac{u}{c}) = \frac{1}{c^2}var(u). \tag{5.22}$$

The usage process noise corresponds to the evolution of the usage signal in successive time frames. Estimating its variance is difficult, since the usage signal itself is an unknown signal, which does not correspond to any physical process well described by a mathematical law. The usage variance is calculated from measurements of the CPU utilisation. When the KBC controller is used, the stationary process variance Q is computed offline before the control process and remains the same throughout.

Finally, the measurement noise variance S corresponds to the confidence that the measured value is very close to the real one. Once more it is difficult to compute the *exact* amount of CPU usage. However, given the existence of relatively accurate measurement

tools, a small value (such as $S = 1.0$, which is used throughout this dissertation) acts as a good approximation of possible measurement errors.

Stability

The KBC controller is stable for all the values of the Kalman gain, as shown below. The KBC control law is:

$$a_{k+1} = a_k + K(u_k - ca_k). \tag{5.23}$$

The Z-transform of the allocation signal is:

$$zA(z) - za(0) = A(z) + KU(z) - cKA(z) \Leftrightarrow \tag{5.24}$$
$$(z - 1 + cK)A(z) = KU(z) + za(0). \tag{5.25}$$

The transfer function is:

$$T(z) = \frac{A(z)}{U(z)} = \frac{K}{z - 1 + cK}, \tag{5.26}$$

where the denominator is the characteristic function. The pole of the characteristic equation is:

$$z - 1 + cK = 0 \Leftrightarrow z = 1 - cK \tag{5.27}$$

and for stability it suffices for the pole to be within the unit circle:

$$|z| < 1 \Leftrightarrow |1 - cK| < 1 \Leftrightarrow 0 < K < \frac{2}{c}. \tag{5.28}$$

Equation (5.28) is true for all Kalman gain values, shown below. From (5.28):

$$
\begin{aligned}
K &< \frac{2}{c} && \Leftrightarrow \text{from (A.4)} \\
\frac{c + \sqrt{c^2 + 4\frac{S}{Q}}}{c^2 + c\sqrt{c^2 + 4\frac{S}{Q}} + 2\frac{S}{Q}} &< \frac{2}{c} && \Leftrightarrow \\
c + \sqrt{c^2 + 4\frac{S}{Q}} &< 2c + 2\sqrt{c^2 + 4\frac{S}{Q}} + \frac{4\frac{S}{Q}}{c} && \Leftrightarrow \\
0 &< \sqrt{c^2 + 4\frac{S}{Q}} + c + \frac{4\frac{S}{Q}}{c},
\end{aligned}
$$

is always true. Therefore, the KBC controller is stable for all values of the Kalman gain.

5.1.5 Discussion

This section presented two SISO controllers: the SISO-UB and the KBC controller. Both controllers track the utilisation of the previous interval to update the allocation for the next. The SISO-UB controller implements a simple tracking approach, while the KBC integrates the Kalman filtering technique into its design and creates a more elaborate approach, where the system's dynamics are incorporated into the tracking process through the Kalman gain.

There are several input parameters in both controllers' cases that need to be defined before the control process. In the case of the SISO-UB, the parameters are: p^i, λ and r^i. There is also the parameter c of the KBC controller. All of them relate to the additional CPU resources required for the application to achieve the reference QoS performance. However, setting these parameters to appropriate values might require some offline analysis and, in some cases a "trial by error" approach. The controllers presented in this dissertation do not aim to find the *best* such values that would allocate the *minimum* resources for the reference performance. Rather they provide a framework where the parameters can be set to different values for different applications and control objectives. The evaluation of controllers in Chapter 6 show the effects of parameter values on controllers' allocations and server's performance.

In addition, the KBC controller has another two input parameters, Q and S, which denote the system's noise variance and are particularly important to the control process. Section 5.3 will present a mechanism that estimates these parameters online, adapts to systems' dynamics at run-time and therefore makes the deployment of the Kalman filtering technique for any application even more attractive in practice.

Finally, all controllers adjust the allocations of individual components. System analysis showed that there is a resource coupling between tiers. This observation is used to build controllers that adjust the allocations for all application tiers collectively, therefore, reacting faster to workload changes. The next section presents two such multi-tier controllers.

5.2 Multi-Tier Controllers

This section presents two MIMO controllers that control the CPU resources of all application tiers. Figure 3.1 of the evaluation platform illustrates the way a MIMO controller adjusts the allocations for the Rubis components. The MIMO type of controllers make use of the resource coupling observation from the system analysis (Section 4.5). The MIMO-UB controller (Section 5.2.1) extends the SISO-UB to dynamically allocate resources based on the offline-derived utilisation coupling models of component pairs (Equations (4.3), (4.4), and (4.5)). The PNCC controller (Section 5.2.2) extends the KBC to consider the process covariance noises between component pairs.

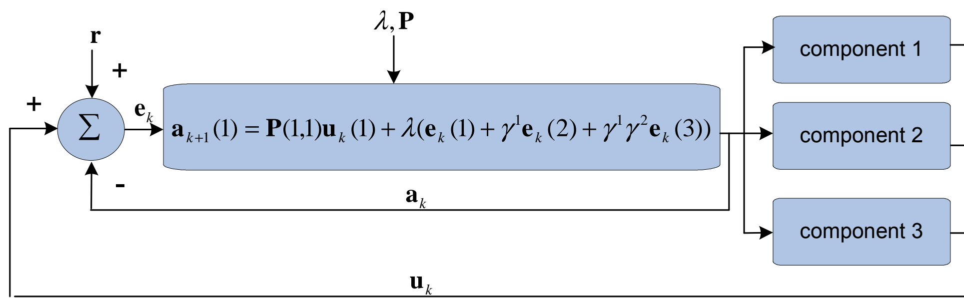


Figure 5.4: MIMO-UB Controller Layout. The MIMO-UB controller allocates CPU resources collectively to all components. The allocation for each component is adjusted based on its utilisation over the previous interval plus a fraction (λ) from all components' errors. This figure illustrates the allocation for the first component in a 3-tier application.

5.2.1 MIMO Usage-Based Controller

This section presents the MIMO Usage Based (MIMO-UB) controller which dynamically allocates resources for multi-component server applications (Figure 5.4). The MIMO-UB controller is based on the SISO-UB controller and considers the individual resource demands of each tier; it also takes into account the resource usage coupling between the different tiers. By considering the coupling between the tiers, the controller adjusts the allocation for all components accordingly. In the example from Figure 4.5 the controller would allocate new resources not only to the bottleneck component B, but also to the component C. In this way, faster overall response to workload changes could be achieved.

First, the MIMO-UB notation is given. If n denotes the number of application components, then: $\mathbf{a}_k \in \mathbb{R}^{n\times 1}$ and $\mathbf{u}_k \in \mathbb{R}^{n\times 1}$ are the allocation and usage vectors respectively, and each row corresponds to a component; $\mathbf{P} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with the p^i values which denote the minimum proportion of utilisation allocated to each component and must be set to values > 1 for each component along the diagonal; and, λ shows the proportion of each component's error accounted towards the allocation. The MIMO-UB controller (Figure 5.4) assigns the new allocations by using the following control law:

$$\mathbf{a}_{k+1} = \mathbf{P}\mathbf{u}_k + \lambda\mathbf{M}\mathbf{e}_k, \tag{5.29}$$

where $\mathbf{e}_k \in \mathbb{R}^{n\times 1}$ is the control error vector and each row corresponds to a component's error (e.g. error for component i is $\mathbf{e}_k(i)$). If $|x|$ is the element-wise absolute value of the vector (i.e. $|x| \triangleq [|x_i|]$), then the controller error vector is defined as:

$$\mathbf{e}_k = |\mathbf{r} - (\mathbf{a}_k - \mathbf{u}_k)|, \tag{5.30}$$

where the $\mathbf{r} \in \mathbb{R}^{n\times 1}$ vector contains the reference values that the difference between the CPU usages $\mathbf{u}$ and the CPU allocations $\mathbf{a}$ is to be maintained from; again each row cor-

responds to a component. The absolute error is used to always provide more resources than the previous utilisations. Finally, to consider the resource coupling between the components in a 3-tier server application, $\mathbf{M}$ is introduced and multiplied with $\mathbf{e}$:

$$\mathbf{M} = \begin{pmatrix} 1 & \gamma^1 & ... & \gamma^{n-2}\gamma^{n-1} \\ \gamma^{n-1}\gamma^n & 1 & ... & \gamma^{n-1} \\ \vdots & \vdots & \vdots & \vdots \\ \gamma^n & \gamma^1\gamma^n & .. & 1 \end{pmatrix},$$

where $\gamma^1, \gamma^2, ..., \gamma^n$ are the coefficients from the linear utilisation coupling models between components. By using $\mathbf{M}$, the errors from all components are included when calculating the error for every other component.

For example, consider the case where the first component's error in a 3-component application ($n = 3$) is calculated. The total error for the first component is given by: $\mathbf{e}(1) + \gamma^1\mathbf{e}(2) + \gamma^1\gamma^2\mathbf{e}(3)$, which is the sum of all components' errors as seen from the point of view of the first component. $\mathbf{e}(1)$, $\mathbf{e}(2)$, and $\mathbf{e}(3)$ are calculated by using equation (5.30). Formulae (4.3), (4.4), (4.5) are used to relate the errors between the different components. Therefore, any new allocation is affected by all components' errors. Finally, only a portion of the final error is considered by introducing the tunable parameter λ. λ does not depend on $\gamma^1, \gamma^2, ..., \gamma^n$ and the controller is globally stable when $|\lambda| < 1/3$ as shown below in the case of a 3-tier application.

Stability

The stability for the MIMO-UB in the case of a 3-tier application is as follows. For the stability proof the following notation, Theorem, and relationship of the coefficients of the utilisation coupling models are required:

Notation: $|A|$ is the element-wise absolute value of the matrix (i.e. $|A| \triangleq [|A_{ij}|]$), $A \leq B$ is the element-wise inequality between matrices A and B and $A < B$ is the strict element-wise inequality between A and B. A nonnegative matrix (i.e. a matrix whose elements are nonnegative) is denoted by $A \geq 0$ and a positive matrix is denoted by $A > 0$. $det(A)$ denotes the determinant of matrix A. $\rho(A)$ denotes the spectral radius of matrix A, i.e. the eigenvalue with the maximum magnitude.

Theorem from [HJ85]: Let $A \in \mathbb{R}^{N\times N}$ and $B \in \mathbb{R}^{N\times N}$, with $B \geq 0$. If $|A| \leq B$, then

$$\rho(A) \leq \rho(|A|) \leq \rho(B). \tag{5.31}$$

Relationship among the coefficients of the utilisation coupling models: From (4.5):

$$\begin{aligned}
u^3 &= \gamma^3 u^1 + \delta^3 \\
&= \gamma^3(\gamma^1 u^2 + \delta^1) + \delta^3 && \text{from (4.3)} \\
&= \gamma^1\gamma^3 u^2 + \gamma^3\delta^1 + \delta^3 \\
&= \gamma^1\gamma^3(\gamma^2 u^3 + \delta^2) + \gamma^3\delta^1 + \delta^3 && \text{from (4.4)} \\
&= \underbrace{\gamma^1\gamma^2\gamma^3}_{1} u^3 + \underbrace{\gamma^1\gamma^3\delta^2 + \gamma^3\delta^1 + \delta^3}_{\text{equal to } 0},
\end{aligned}$$

and so,

$$\gamma^1\gamma^2\gamma^3 = 1. \tag{5.32}$$

For $(\mathbf{r} - (\mathbf{a}_k - \mathbf{u}_k)) < 0$, the allocation signal is:

$$\mathbf{a}_{k+1} = \mathbf{P}\mathbf{u}_k - \lambda\mathbf{M}(\mathbf{r} - (\mathbf{a}_k - \mathbf{u}_k)),$$

and its Z-transform is:

$$\begin{aligned}
z\mathbf{A}(z) - z\mathbf{a}(0) &= \mathbf{P}\mathbf{U}(z) - \lambda * \mathbf{M}\left(\frac{1}{1-z^{-1}}\mathbf{r} - (\mathbf{A}(z) - \mathbf{U}(z))\right) \Leftrightarrow \\
z\mathbf{A}(z) - z\mathbf{a}(0) &= \mathbf{P}\mathbf{U}(z) - \frac{\lambda\mathbf{M}\mathbf{r}}{1-z^{-1}} + \lambda\mathbf{M}\mathbf{A}(z) - \lambda\mathbf{M}\mathbf{U}(z) \Leftrightarrow \\
(z\mathbf{I} - \lambda\mathbf{M})\mathbf{A}(z) &= (\mathbf{P} - \lambda\mathbf{M})\mathbf{U}(z) - \frac{\lambda\mathbf{M}\mathbf{r}}{1-z^{-1}} + z\mathbf{a}(0),
\end{aligned}$$

where $\mathbf{I}$ is the identity matrix. The transfer function is:

$$T(z) = \frac{\mathbf{A}(z)}{\mathbf{U}(z)} = \frac{\mathbf{P} - \lambda\mathbf{M}}{z\mathbf{I} - \lambda\mathbf{M}},$$

where the denominator is the characteristic function. The poles of the characteristic function are the z values that make its determinant equal to 0:

$$\begin{aligned}
det(z\mathbf{I} - \lambda\mathbf{M}) = 0 \Leftrightarrow \\
det\begin{pmatrix} z-\lambda & -\lambda\gamma^1 & -\lambda\gamma^1\gamma^2 \\ -\lambda\gamma^2\gamma^3 & z-\lambda & -\lambda\gamma^2 \\ -\lambda\gamma^3 & -\lambda\gamma^1\gamma^3 & z-\lambda \end{pmatrix} = 0.
\end{aligned}$$

By expansion and using equation (5.32), the above equation becomes:

$$\begin{aligned}
(z-\lambda)((z-\lambda)^2 - \lambda^2) - \lambda^2(z-\lambda) - \lambda^3 - \lambda^3 - \lambda^2(z-\lambda) = 0 \Leftrightarrow \\
(z-\lambda)(z^2 - 2z\lambda) - 2z\lambda^2 = 0 \Leftrightarrow \\
z^3 - \lambda z^2 - 2\lambda z^2 = 0 \Leftrightarrow \\
z^2(z - 3\lambda) = 0.
\end{aligned}$$

So there are three poles, the double pole $z = 0$ and the pole $z = 3\lambda$. $z = 3\lambda$ is the spectral radius of the matrix $\lambda\mathbf{M}$:

$$\rho(\lambda\mathbf{M}) = 3\lambda. \tag{5.33}$$

For stability, the poles have to be within the unit circle, $|z| < 1$, hence:

$$|3\lambda| < 1 \Leftrightarrow |\lambda| < \frac{1}{3}. \tag{5.34}$$

For different combinations of either positive or negative components' errors e_k^i in $\mathbf{e}_k$, the Z-transform of the allocation signal is:

$$\begin{aligned} z\mathbf{A}(z) - z\mathbf{a}(0) &= \mathbf{P}\mathbf{U}(z) + \ \lambda * \mathbf{M}(\frac{1}{1-z^{-1}}\mathbf{r} - (\mathbf{A}(z) - \mathbf{U}(z))) \Leftrightarrow \\ z\mathbf{A}(z) - z\mathbf{a}(0) &= \mathbf{P}\mathbf{U}(z) + \frac{\lambda\mathbf{M_1}\mathbf{r}}{1-z^{-1}} + \ \lambda\mathbf{M_2}\mathbf{A}(z) + \lambda\mathbf{M_3}\mathbf{U}(z) \Leftrightarrow \\ (z\mathbf{I} - \lambda\mathbf{M_2})\mathbf{A}(z) &= (\mathbf{P} + \lambda\mathbf{M_3})\mathbf{U}(z) + \ \frac{\lambda\mathbf{M_1}\mathbf{r}}{1-z^{-1}} + z\mathbf{a}(0), \end{aligned}$$

where $\mathbf{M_1}, \mathbf{M_2}$ and $\mathbf{M_3} \in \mathbb{R}^{3\times3}$ are matrices whose element-wise absolute values equal the corresponding values of $\mathbf{M}$ elements ($|\mathbf{M_1}|, |\mathbf{M_2}|, |\mathbf{M_3}| = \mathbf{M}$), but their elements are either positive or negative depending on whether the errors e_k^i are negative or positive.

The transfer function is:

$$T(z) = \frac{\mathbf{A}(z)}{\mathbf{U}(z)} = \frac{\mathbf{P} + \lambda\mathbf{M_3}}{z\mathbf{I} - \lambda\mathbf{M_2}},$$

where the denominator is the characteristic function. To prove MIMO-UB stability for all combinations of negative or positive $\mathbf{e}_k(i)$ errors, the poles of the characteristic function (or the eigenvalues of $\lambda\mathbf{M_2}$) have to be within the unit circle. Note that matrix $\mathbf{M}$ has all its entries positive and in the case where the error is negative, it was shown that the spectral radius of $\lambda\mathbf{M}$ is less than one and hence within the unit circle when $|\lambda| < \frac{1}{3}$ (Equations (5.33) and (5.34)). Therefore, using the Theorem from [HJ85] and $\lambda \geq 0$, since $|\mathbf{M_2}| \leq \mathbf{M} \Leftrightarrow |\lambda\mathbf{M_2}| \leq \lambda\mathbf{M}$ and then $\rho(\lambda\mathbf{M}_2) \leq \rho(|\lambda\mathbf{M}_2|) \leq \rho(\lambda\mathbf{M}) < 1$. So, the eigenvalue with the maximum magnitude of $\lambda\mathbf{M}_2$ is within the unit circle and, therefore, the MIMO-UB system is stable for any combination of errors when:

$$|\lambda| < \frac{1}{3}. \tag{5.35}$$

Discussion

The MIMO-UB uses the offline-derived linear resource coupling models of components' utilisations. However, not all applications' resource coupling can be linearly modelled. In addition, the MIMO-UB relies on offline system identification to derive matrix M. To tackle both problems, an online version of MIMO-UB that derives M every several controller intervals using utilisation measurements can be proposed. The resource coup-

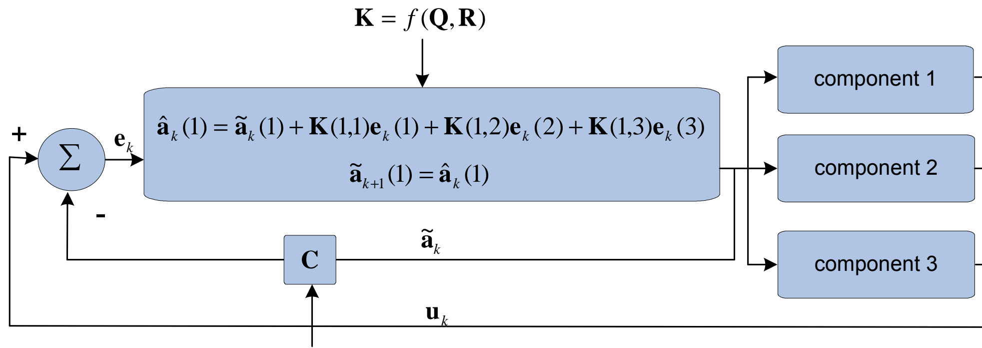


Figure 5.5: PNCC Controller Layout. The PNCC controller allocates resources to all application tiers collectively. It considers their resource coupling by using their utilisation covariances. This figure illustrates the allocation for the first component in a 3-tier application.

ling could still be modelled through linear equations for shorter periods of time and, therefore, the current M form can be retained as it is.

5.2.2 Process Noise Covariance Controller

This section presents the MIMO Process Noise Covariance Controller (PNCC) (Figure 5.5) which extends the KBC to consider the resource coupling between multi-tier applications. The allocation for each component is adjusted based on the errors of the current component in addition to the errors caused by the other components, through the covariance process noises. If n is the number of application components, then the PNCC Kalman filter equations for stationary process and measurement noise take the form:

$$\mathbf{a}_{k+1} = \mathbf{a}_k + \mathbf{W}_k, \tag{5.36}$$

$$\mathbf{u}_k = \mathbf{C}\mathbf{a}_k + \mathbf{V}_k, \tag{5.37}$$

$$\widehat{\mathbf{a}}_k = \widetilde{\mathbf{a}}_k + \mathbf{K}_k(\mathbf{u}_k - \mathbf{C}\widetilde{\mathbf{a}}_k), \tag{5.38}$$

$$\mathbf{K}_k = \mathbf{C}\widetilde{\mathbf{P}}_k(\mathbf{C}\widetilde{\mathbf{P}}_k\mathbf{C}^T + \mathbf{S})^{-1}, \tag{5.39}$$

$$\widehat{\mathbf{P}}_k = (\mathbf{I} - \mathbf{C}\mathbf{K}_k)\widetilde{\mathbf{P}}_k, \tag{5.40}$$

$$\widetilde{\mathbf{a}}_{k+1} = \widehat{\mathbf{a}}_k, \tag{5.41}$$

$$\widetilde{\mathbf{P}}_{k+1} = \widehat{\mathbf{P}}_k + \mathbf{Q}, \tag{5.42}$$

where $\mathbf{a}_k \in \mathbb{R}^{n\times 1}$ and $\mathbf{u}_k \in \mathbb{R}^{n\times 1}$ are the allocation and usage vectors respectively and each row corresponds to a component; $\mathbf{W}_k \in \mathbb{R}^{n\times 1}$ is the process noise matrix; $\mathbf{V}_k \in \mathbb{R}^{n\times 1}$ is the measurement noise matrix; $\mathbf{C} \in \mathbb{R}^{n\times n}$ is a diagonal matrix with the target value c for each component along the diagonal; $\widetilde{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ and $\widehat{\mathbf{P}}_k \in \mathbb{R}^{n\times n}$ are the *a priori* and *a posteriori* error covariance matrices; $\mathbf{K}_k \in \mathbb{R}^{n\times n}$ is the Kalman gain matrix and $\mathbf{S} \in \mathbb{R}^{n\times n}$ and $\mathbf{Q} \in \mathbb{R}^{n\times n}$ are the measurement and process noise matrices respectively.

For matrices $\mathbf{Q}$ and $\mathbf{S}$, the diagonal elements correspond to the process and measurement noise for each component. The non-diagonal elements of the matrix $\mathbf{Q}$ correspond to the process noise covariance between different components. Similarly, the non-diagonal elements of the $\mathbf{K}_k$ matrix correspond to the gain between different components. For a 3-tier application, for example, the *a posteriori* $\widehat{\mathbf{a}}_k(1)$ estimation of the allocation of the first component at interval k is the result of the *a priori* estimation $\widetilde{\mathbf{a}}_k(1)$ of the allocation plus the corrections from all components' innovations, given by:

$$\begin{aligned}\widehat{\mathbf{a}}_k(1) = \widetilde{\mathbf{a}}_k(1) &+ \mathbf{K}_k(1,1)(\mathbf{u}_k(1) - \mathbf{C}(1,1)\widetilde{\mathbf{a}}_k(1)) \\ &+ \mathbf{K}_k(1,2)(\mathbf{u}_k(2) - \mathbf{C}(2,2)\widetilde{\mathbf{a}}_k(2)) \\ &+ \mathbf{K}_k(1,3)(\mathbf{u}_k(3) - \mathbf{C}(3,3)\widetilde{\mathbf{a}}_k(3)).\end{aligned}$$

The covariances between variables show how much each variable is changing if the other one is changing as well. In this case the covariances indicate the coupling of the utilisation changes between components.

Modelling Covariances

Like the computation of the allocation variances, the covariances between the components' allocations are computed offline based on the usage covariances. If u^i and u^j are the measured usages between components i and j, then the covariance between their allocations a^i and a^j is computed as (cov denotes the covariance):

$$cov(a^i, a^j) \simeq cov(\frac{u^i}{c}, \frac{u^j}{c}) = \frac{1}{c^2} cov(u^i, u^j). \tag{5.43}$$

Stability

As described in Section 2.4.1 a system is stable if, for any bounded input, the output is also bounded. The PNCC is stable because both of its inputs, the CPU utilisations, and its output, the CPU allocations, are bounded by the physical machine's capacity: a component's utilisation and allocation cannot exceed the 100% of the machine's capacity.[3]

5.2.3 Summary

This section presented two MIMO controllers, the MIMO-UB and the PNCC. Both controllers consider the resource coupling between the components and incorporate it into their design. In the case of the MIMO-UB, this is achieved through the $\mathbf{M}$ array. In the case of the PNCC, it is done by using the covariance matrices. Results in the next chapter show that the MIMO controllers offer better performance than the SISO ones.

There are input configuration parameters in both controllers: $\mathbf{P}, \mathbf{r}$, and λ for the MIMO-UB and $\mathbf{C}$ for the PNCC. In addition, the MIMO-UB controller uses offline derived

[3]This approach to determine stability can be applied to all the controllers of this dissertation. However, in cases where the analysis using the poles of the transfer function was feasible it is also given.

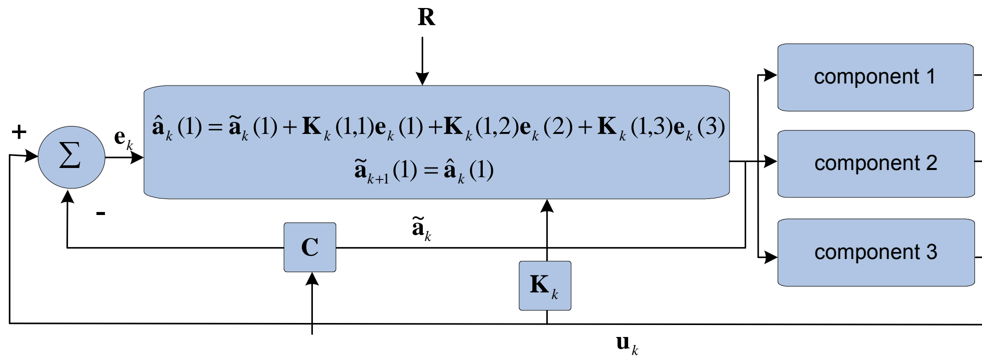


Figure 5.6: APNCC Controller Layout. The APNCC controller extends the PNCC design to online estimate the system's process noise covariance matrix and update the Kalman gain at regular intervals. This figure illustrates the allocation for the first component in a 3-tier application.

utilisation models incorporated in the $\mathbf{M}$ array and the PNCC uses the system dynamics in the $\mathbf{Q}$ and $\mathbf{S}$ arrays. The advantage of the PNCC over the MIMO-UB is that its parameters (apart from $\mathbf{C}$) are related to the system's dynamics and therefore easier to set than those in the MIMO-UB. However, it would be even more useful to automate the process of setting the parameters and therefore eliminate any offline analysis. The next section presents an online parameter estimation mechanism for the Kalman based controller.

5.3 Process Noise Adaptation

So far only stationary process and measurement noises have been considered. Both controllers can be easily extended to adapt to operating conditions by considering non-stationary noises. For example in the case of the PNCC controller, all formulae are as before but instead of the stationary $\mathbf{Q}$, the dynamic $\mathbf{Q}_k$ is now used. In this case, $\mathbf{Q}_k$ is updated every several intervals with the latest computations of variances and covariances from CPU utilisation measurements over the last iterations. For simplicity, the measurement noise variance is considered to always be stationary, i.e. $\mathbf{S}_k = \mathbf{S}$.

The next chapter evaluates the adaptation mechanism in the case of the PNCC controller. The new controller is called Adaptive PNCC (hereafter denoted as APNCC) and its layout is given in Figure 5.6.

Stability

As in the case of the PNCC, the APNCC is stable, because for any bounded input, CPU utilisations, its outputs, CPU allocations, are also bounded by the machines' physical capacities.

5.4 Discussion

This chapter presented five feedback controllers that allocate CPU resources to virtualized server applications. Controllers are based on the system identification analysis. First, the controllers are based on the additive and multiplicative models to track the utilisations and adjust the allocations. Second, the MIMO designs incorporate the resource coupling between the components' utilisation to allocate resources more quickly to application tiers. This section categorises the controllers according to four different characteristics (Table 5.2).

Kalman Filter

All controllers adjust the allocations to follow the components' utilisation. However, three of them, the KBC, the PNCC, and the APNCC, are based on the Kalman filtering technique. These controllers use Kalman filters to track the utilisation and subsequently the allocation itself using a linear model of the system. The advantage of this technique over a simple tracking method is as follows. The Kalman filter uses the system dynamics to adjust the allocations. This is achieved through the Kalman gain which is a function of the process and measurement noise of the system. The process noise depicts the evolution of the utilisation between intervals and, therefore, contains important information about the system itself. The measurement noise corresponds to the belief of the measurement tools and hence provides information regarding the tools used in the control system.

Inter-VM Coupling

The two SISO controllers (SISO-UB and KBC) allocate resources individually to application tiers. System analysis showed that there is a resource coupling between component utilisations. The three MIMO controllers (MIMO-UB, PNCC, and APNCC) use this observation to collectively update the components' allocations based on errors from all tiers. The MIMO controllers are designed to react more quickly to workload changes than the SISO designs. The MIMO-UB uses an offline derived model that depicts the utilisation correlations. The PNCC and APNCC use the utilisation covariance between components.

Allocation Tuning

Another advantage that comes with integrating Kalman filters into feedback controllers is that the Kalman gain can be tuned to different values that make the final allocations react in different ways to resource fluctuations. Depending on the gain, the controllers might not be strongly affected by transient resource fluctuations while still adapting to important workload changes. In a shared cluster environment where the allocation of one application affects the available resources for another, this feature can be particularly useful.

	Kalman filter		inter-VM coupling		allocation tuning		parameter adaptation	
controller	no	yes	yes	no	adaptive	constant	adaptive	constant
SISO-UB	✓			✓		✓		✓
MIMO-UB	✓		✓			✓		✓
KBC		✓		✓	✓			✓
PNCC		✓	✓		✓			✓
APNCC		✓	✓		✓		✓	

Table 5.2: Classification of Controllers based on the following four criteria: Kalman filter design method, inter-VM coupling, allocation tuning, and parameter adaptation.

Parameter Adaptation

The final characterisation of the controllers is whether they are able to estimate their parameters online. All controllers have input configuration parameters, the values of which affect the server's performance. In a control-based resource provisioning scheme, however, it might be more practical to update these parameters online and adjust them to applications and operating conditions. The last section of this chapter provided an adaptation mechanism for the Kalman based controllers, and, in particular, it suggested the use of the APNCC controller, an adaptive version of the PNCC.

The next chapter evaluates the performance of all five controllers. In addition, it evaluates whether (a) the MIMO controllers improve the performance of the SISO controllers; (b) the Kalman gains can be tuned to provide adjustable allocations, and (c) the adaptation mechanism captures the system dynamics.

6 Experimental Evaluation

The previous chapter presented different controllers that dynamically provision virtualized servers with CPU resources. The SISO-UB and the KBC controllers provision each component separately, while the MIMO-UB and the PNCC manage the allocation collectively for all components. In addition, the KBC and the PNCC are based on the Kalman filter. Finally, an adaptive mechanism for estimating the parameters for the Kalman based controllers was introduced.

This chapter evaluates each controller individually — SISO-UB in Section 6.2.1, MIMO-UB in Section 6.2.2, KBC in Section 6.3.1, PNCC in Section 6.3.2, and APNCC in Section 6.3.4 — and compares them — MIMO-UB and SISO-UB in Section 6.2.3, KBC and PNCC in Section 6.3.3, and PNCC and APNCC in Section 6.3.5. The chapter starts by introducing the evaluation methodology (Section 6.1). The term "usage based controllers" is only used in this chapter to distinguish the SISO-UB and MIMO-UB controllers from the Kalman based (KBC, PNCC, and APNCC), although all controllers in this dissertation are based on the components' utilisations.

6.1 Preliminaries

This section discusses the evaluation procedure of the controllers. There are three types of experiments used for the evaluation which are summarised in Table 6.1. Two different workload mixes, available from the Rubis distribution are used: the *browsing mix* (BR) contains read-only requests and the *bidding mix* (BD) that includes 15% read-write requests. BR is mostly used unless otherwise stated.

SYMBOL	DESCRIPTION
E0(t1, t2, t3)	**Varying number of clients.** 300 clients issue requests to the server for t3 intervals in total. At the $t1^{th}$ interval, another 300 clients are added until the $t2^{th}$ interval.
E1(n,d)	**Static number of clients.** n number of clients issue requests to the server for d intervals in total.
E2	**Big workload change in the number of clients.** 200 clients issue requests to the server for 60 intervals in total. At the 30th interval, another 600 are added for the next 30 intervals.

Table 6.1: Performance Evaluation Experiments. Intervals are set to 5s.

An E0(t1, t2, t3) experiment is used for the basic evaluation of each controller. The purpose of this type of experiment is two-fold. It tests the controller's ability to (a) adjust the allocations to follow the utilisations while (b) maintaining the reference Rubis server performance (mRT $\leq$ 1s) under diverse workload conditions as the number of clients increase (their number doubles at t1 interval) and decrease (their number halves at t2 interval). The total experiment duration is t3 intervals. There are six different types of graphs used to illustrate the results. Three of them show the average component utilisation and the average corresponding controller's allocation for each controller interval (denoted as *sample point* in the graphs) for each different Rubis tier. The other three graphs present the server's performance: (a) mean response times (mRT) (in seconds) for each controller interval, (b) Throughput (requests/sec) for each controller interval, and (c) cumulative distribution function (CDF) of response times for the duration of the experiment. All together these graphs provide a complete and detailed view of the server's performance, from both the point of view of the resource allocations and the requests performance. Finally, recall that the mRT data are *not* used to control the allocations, rather, are captured to provide a graphical representation of the server's performance and are used to assess the server's performance.

E1(n,d) experiments are used to evaluate the values of controllers' parameters. In this experiment n clients issue requests to the server for d intervals in total. This simple experiment with stable workload[1] tests the controllers when changing their parameters and compares the different configurations with the least implications from workload variations.

Finally, to compare the different SISO and MIMO controller designs, E2 experiments are used. As the MIMO is designed to allocate resources faster in workload changes,

[1]The term stable is used in this chapter to characterise a workload with static number of clients.

SYMBOL	DESCRIPTION
CR	number of completed requests
NR	proportion of requests with response time $\leq$ 1s over CR
RMS	Root Mean Squared error over parameter `l` $$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\frac{l - predicted(l)}{predicted(l)})^2},$$ where N is the total size of l observations
`additional` allocation	sum of the differences of CPU resources between the allocations and the utilisations of a component
COV	coefficient of variation $$COV = \frac{s}{\bar{x}},$$ where s is the sample standard deviation and $\bar{x}$ is the sample mean of a statistic

Table 6.2: Performance Evaluation Metrics. All metrics are calculated over a duration of several intervals and given in the text when used.

when components are most likely to saturate, E2 experiments are designed to stress the server under these conditions. To this end, in an E2 experiment, a sudden and large change in the number of clients occurs; clients increase from 200 to 800. The controllers' performance is evaluated for the duration of the change between the intervals 30 up to 50 (i.e. 100s in total). That is from the interval the number of clients increases up to the one where the server settles down to the new increased number of clients. In this way, emphasis is given only to the actual workload change.

For the rest of this chapter the numbers reported for E1 and E2 experiments come from multiple runs of the same experiment (with respect to input configuration parameters) for statistically confident results. The number of repetitions, given in each case, is defined as follows. Initially, a small number of repetitions (usually 2-3 times) is used; then, the same experiment is repeated until the new data do not significantly change the results. The results reported here usually come from all the repetitions.

Table 6.2 contains all metrics used for evaluation in this chapter. All metrics are calculated over a period of several intervals; their number is given throughout. Metrics belong to two categories: (a) metrics that evaluate the server's performance and (b) metrics that evaluate the controllers' resource allocations.

The metrics that evaluate the performance of the server are described now. CR denotes the number of completed requests. NR gives the proportion of requests with response time $\leq$ 1s over CR. The server's performance improves when the values of either metric increases between similar types of experiments of the same duration.

Both CR and NR metrics provide aggregate numbers that describe request characteristics over some duration. To provide a more detailed view of the request response times, the Root Mean Squared (RMS) error metric is also used. The RMS metric is widely used to evaluate the performance of prediction models as it provides a measure of the accuracy of the models for all predictions combined. Here the RMS is used to provide a more detailed view of the response times throughout the experiments. To calculate the RMS in the current system, the predicted value is the `mRT` for a specific number of clients. Since the current system does not predict an individual request response time, the RMS uses the `mRT` to evaluate the controllers' performance. Therefore, an error will always be present between the model predictions (`mRT`) and the measured response times. For example, the `mRT` in the case of 600 clients was measured in the system identification analysis and equals 0.282s (Figure 4.1(a)). Using this `mRT`, the RMS in the case of an E1(600,20) experiment is measured and is 2.17. In general, the smaller the RMS values the closer the response times to the `mRT`. A combination of the three metrics CR, NR, and RMS provide enough information to compare the different controllers.

The metrics `additional` allocation and COV evaluate the controllers' resource allocations. The `additional` allocation denotes the sum of resources given for each component on top of its mean utilisation over some intervals. This metric enables comparisons between controllers with respect to the amount of resources they occupy and their performance. If similar performances (according to CR, NR, and RMS) are achieved for different `additional` allocations, then the smallest values are preferred. In this way, more resources are available for other applications to run.

The COV metric is used to measure the variability of the allocation and the utilisation signals. Different degrees of allocation variability might be appropriate depending on the control system operation and the types of applications sharing a virtualized cluster. For instance, if two controllers adjust the allocations for two applications with strict QoS performance guarantees, it might be more practical for the two controllers to be robust to transient utilisation fluctuations and adjust the allocations only when important workload changes occur. On the other hand, consider a server application which shares CPU resources with a batch-processing workload that does not have real-time QoS guarantees. In this case, if the server's allocations vary with every utilisation fluctuation, this does not affect the overall performance for the batch-processing workload. Therefore, depending on the types of virtualized applications, different allocation variability might be desired.

Controller Interval

The controller interval is important to the resource allocation process since it characterises the frequency of the new allocations and the time period over which the component usages are averaged and used by the controller to make new allocations. The new allocations are calculated based on both the average component usages and the error over a time interval. With a small interval the controller reacts fast to workload changes but is prone to transient workload fluctuations. A better approximation is achieved with a larger interval, as the number of sample values increases. However, the controller's responses can be slower. Depending on the workload characteristics, the interval can be

set to smaller values for frequently changing workloads and larger ones for more stable workloads.

For the current evaluation intervals of 5s and 10s were examined. Utilisations averaged over both durations were close to the mean utilisation over long runs (e.g. 100s) and hence, both intervals are suitable to summarise usages. In addition, preliminary experiments (three SISO-UB controllers allocating resources during E0(20,40,60) experiments) using either interval duration showed that with a shorter interval the server had better responses to workload increases. Therefore, a 5s controller interval is selected for the evaluation in this chapter.

Single Application Evaluation

As discussed earlier, the controllers' evaluation is based on a single Rubis instance partitioned in three components each hosted by one VM running on a different machine. One might argue that this is not a representative case where resource management is needed. The particular setting is chosen so as to focuss on the evaluation of the controllers: this evaluation examines how do the controllers perform assuming there are free physical resources when needed. The current work is assumed that would be part of a larger data center management scheme, where there is need to isolate the performance of applications by adjusting their resources according to their needs and be able to account the available free resources for further application placement.

6.2 Usage Based Controllers

The usage-based controllers, SISO-UB and MIMO-UB, are evaluated in this section. The SISO-UB controller allocates CPU resources to individual components, while the MIMO-UB uses the offline-derived utilisation model between components to collectively allocate resources to all tiers. The performance of each controller is initially evaluated using an E0(20,40,60) experiment (SISO-UB controller in Section 6.2.1 and MIMO-UB in Section 6.2.2). Both controllers also have a number of input configuration parameters whose values affect the server's performance. The parameter values are evaluated through a number of E1(600,60) experiments. Finally, in previous chapters, it was advocated that the MIMO controller which considers the resource coupling between tiers act faster to workload changes and therefore provide better performance than the SISO designs. Section 6.2.3 experimentally evaluates this claim and shows that the MIMO-UB controller has better performance than the SISO-UB ones.

6.2.1 SISO-UB

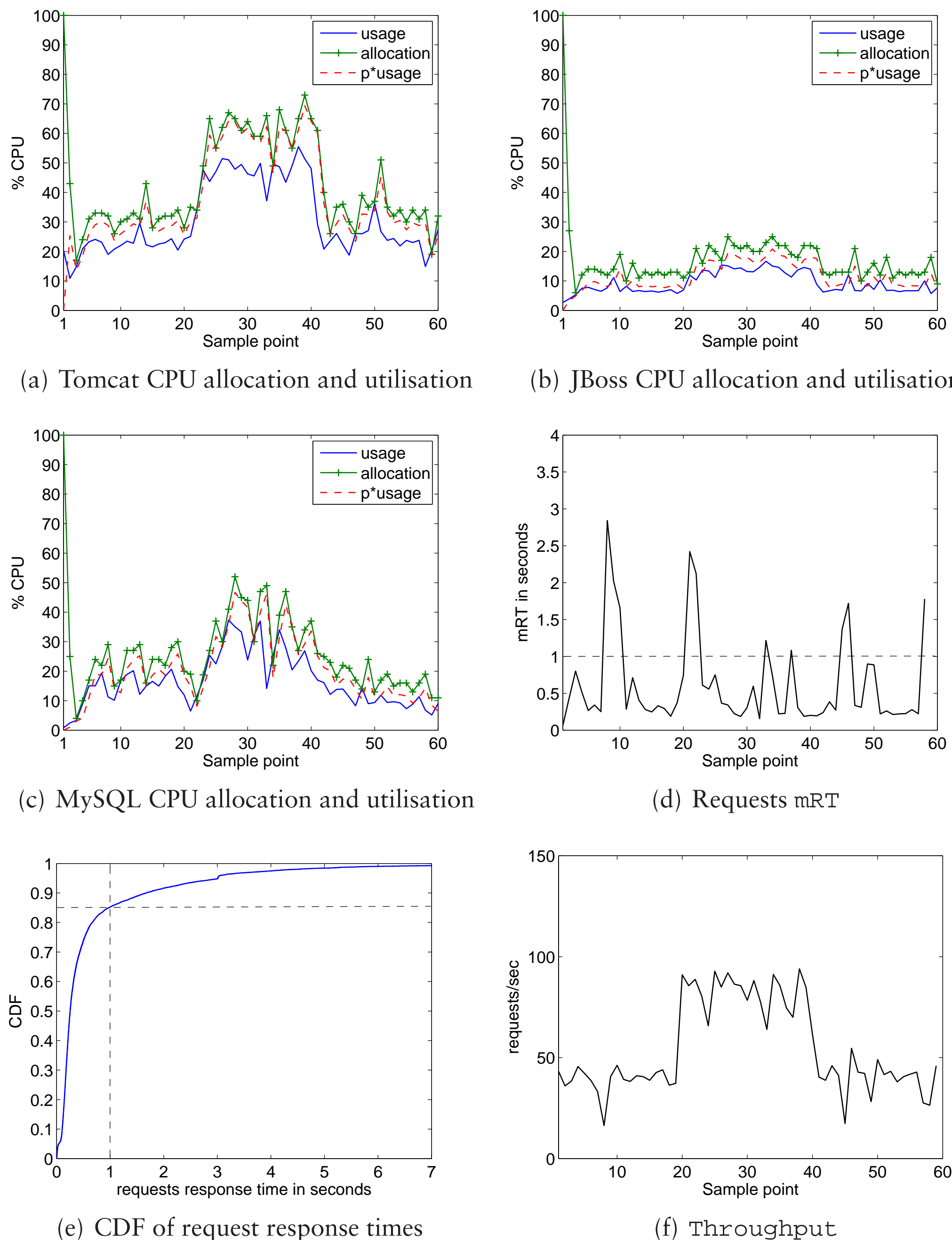
The SISO-UB controller allocates resources based on the mean utilisation of the previous interval and the control error. In particular, the allocation is proportional to the utilisation and proportional to the control error (the SISO-UB control law from Equations (5.1) and (5.2) is: $a^i_{k+1} = p^i u^i_k + \lambda |r^i - (a^i_k - u^i_k)|$).

Initially, the SISO-UB performance is evaluated using an E0(20,40,60) experiment (Figure 6.1). There are three input configuration parameters for each SISO-UB controller; their values for the E0(20,40,60) experiment are set to: $p^i = 1.25, r^i = 20$ and $\lambda = 0.3$. A justification of the parameter values is given later in this section. Figures 6.1(a), 6.1(b), and 6.1(c) illustrate the CPU allocations and utilisations for the three Rubis components for the duration of the experiment. At the beginning, each controller drops the allocation from the initial 100% to approach the usage in just one interval and then the allocations are adjusted as utilisations change. Each SISO-UB controller is able to adjust the allocations to follow the CPU utilisations as small resource fluctuations occur throughout the experiment and large workload changes happen around the 20th and 40th sample points. Therefore, the first goal of each SISO-UB controller, which is to update the allocations to follow the usages, is achieved.

Section 5.1.1 described how the SISO-UB controller uses both system models and in this way components with low utilisation and high variance are allocated more resources than when just using the multiplicative model. This is shown in Figure 6.1. The dashed lines in Figures 6.1(a), 6.1(b), and 6.1(c) depict hypothetical allocations if each SISO-UB controller used just the multiplicative system model and its law was: $a_{k+1} = p^i * u_k$. It is shown that in the case of the JBoss and MySQL components when their utilisations are low and their variance considerable, the SISO-UB controller (allocation line) allocates more resources than just the simple controller that uses the multiplicative model (p*usage line). When the utilisation is high — for instance in the Tomcat component during 20 and 40 intervals — both lines are very close. The combination of both system models results in a flexible controller whose allocations are adjustable to different utilisations. In this way, the SISO-UB parameters do not need to be configured differently for different levels of utilisation.

Figures 6.1(d), 6.1(e), and 6.1(f) show the server's performance. As shown in Figure 6.1(d), for most of the experiment the requests' `mRT` is close to its reference value ($\leq$ 1s). There are however, a few spikes above and close to the reference value; this is happening when one or more components are saturated for that interval. To better evaluate the `mRT` spikes, Figure 6.1(e) shows the cumulative distribution function (CDF) of the request response times. This figure shows that 85.25% of the requests have response times $\leq$ 1s. The spikes in Figure 6.1(d) are caused by a small percentage of requests (14.75%) with large response times. Finally, Figure 6.1(f) depicts the server's `Throughput`. The `Throughput` changes as the workload varies for the duration of the experiment. Its values approach those values that were measured before-hand for different numbers of incoming clients (Figure 4.1(b)) when all machine CPU resources were allocated to components; approximately 90 req/sec for 600 clients between 20 and 40 intervals and 48 req/sec for 300 clients for all other intervals. As in the case of the `mRT`, when any of the components saturates, the `Throughput` drops from its corresponding value for the given number of clients. Overall, the second goal of the SISO-UB controller, which is to maintain the reference performance of `mRT` $\leq$ 1s is achieved for most of the intervals and the vast majority of the requests have response times $\leq$ 1s.

Note, however, that the SISO-UB allocations follow all utilisation changes even subtle ones, which causes the allocation signal to be as noisy as the utilisation. This behaviour might not be suitable in a server consolidation scenario, where the allocation of one



(a) Tomcat CPU allocation and utilisation

(b) JBoss CPU allocation and utilisation

(c) MySQL CPU allocation and utilisation

(d) Requests mRT

(e) CDF of request response times

(f) Throughput

Figure 6.1: SISO-UB Controllers Performance. Figures illustrate the performance of the SISO-UB controllers for variable number of clients. All controllers adjust the allocations to follow the utilisation changes. The server's performance is sustained close to its reference value, with a few spikes in mRT and drops in Throughput when one or more components are saturated.

application might affect the available resources for the other co-hosted applications. A controller that adjusts the allocations but is not so strongly affected by the transient resource fluctuations might be more appropriate. The KBC controller, presented later, addresses this issue.

Parameter Configuration

It was shown previously (Figure 4.4) that the server's performance depends on the amount of `extra` resources allocated to the components which in turn depends on the parameter values. The rest of this section evaluates the way the parameter values affect the server's performance through a number of E1(600,60) experiments.

There are three input configuration parameters for each SISO-UB controller. If i corresponds to a component then, (a) p^i denotes the minimum percentage of the utilisation that the allocation is assigned to; (b) r^i corresponds to the reference value of the `extra` allocation; and (c) λ^i denotes the percentage of the control error accounted for the final allocation. For the current analysis, each parameter is assigned to a value that is either empirically set or guided from the system identification analysis and correspond to worse-, medium-, and best-case allocation policy.

The p^i have to be ≥ 1 and in practice each p^i is set to values from $\{1.11, 1.25, 1.43\}$. These values correspond to a target CPU utilisation rate within each VM of $\{90\%, 80\%, 70\%\}$ and are used to model "tight" (worse-case), "moderate" (medium-case), and "loose" (easy-case) allocation policies. These values are used in data centres. r^i values are set from $\{10, 20, 30\}$. In this case, system identification analysis showed that to achieve reference performance, the r^i values for the Tomcat, JBoss, and MySQL components were $\{15, 10, 10\}$ respectively (Figure 4.4). In these experiments only one component was subject to varying allocation each time, and so the above values ($\{15, 10, 10\}$) correspond to "tight" allocation policies. Finally, λ values are set to $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ as the SISO-UB controller is stable when $|\lambda| < 1$ (Equation (5.5)). Here only positive λ values are considered, since there are combinations of p^i and negative λ values that would result in smaller allocations than utilisations, and therefore, they would compromise the controller's performance.

The parameters for the E0(20,40,60) experiment (in Figure 6.1) were set to values that correspond to a middle-case allocation policy (for all i, $\{p^i, r^i, \lambda\} = \{1.25, 20, 0.3\}$). Although λ could have also been set to 0.5, the value of 0.3 also provides a middle-case allocation and enables comparisons with the MIMO-UB E0(20,40,60) experiment in the next section. The rest of this section provides an empirical evaluation of the effects of the configuration parameters on the server's performance across the range of values given above. The purpose of this evaluation is to provide guidelines on the relative effects of the different parameters.

The server's performance is evaluated against different sets of parameter values using four metrics. Parameters are varied as shown in Table 6.3 where each row corresponds to a different set of values; for reasons of simplicity all three components are assigned to the same value for the same parameter. In each row only one parameter changes each time, while the rest are assigned to values that correspond to middle-case allocations. For each set of values an E1(600,60) experiment is performed 5 times. Server's performance is measured using two metrics: CR and NR (defined in Table 6.2). Another two metrics are used to study the resource allocations at the server components: (a) `additional` allocation which corresponds to the sum of the differences between the allocation and the utilisation for all intervals and (b) COV which shows the coefficient of variance of

index	parameters			application performance metrics				
	p^i	λ	r^i	CR	NR	`additional` allocation	COV alloc.	COV usage
1	1.25	**0.1**	20	24354 ± 269	0.80 ± 0.02	788 ± 11	0.15	0.11
2	1.25	**0.3**	20	24934 ± 248	0.85 ± 0.01	892 ± 5	0.14	0.10
3	1.25	**0.5**	20	25276 ± 116	0.89 ± 0.01	987 ± 19	0.14	0.11
4	1.25	**0.7**	20	25381 ± 229	0.89 ± 0.01	1067 ± 10	0.14	0.09
5	1.25	**0.9**	20	25385 ± 206	0.90 ± 0.01	1198 ± 32	0.15	0.11
6	**1.11**	0.5	20	24359 ± 436	0.82 ± 0.02	670 ± 7	0.16	0.11
7	**1.25**	0.5	20	25276 ± 116	0.89 ± 0.01	987 ± 19	0.14	0.11
8	**1.43**	0.5	20	25619 ± 78	0.92 ± 0.01	1491 ± 13	0.12	0.11
9	1.25	0.5	**10**	24482 ± 236	0.85 ± 0.02	960 ± 24	0.12	0.11
10	1.25	0.5	**20**	25276 ± 116	0.89 ± 0.01	987 ± 19	0.14	0.11
11	1.25	0.5	**30**	25521 ± 192	0.90 ± 0.01	1134 ± 5	0.14	0.11

Table 6.3: Input Parameter Configuration for SISO-UB Controllers. Each row corresponds to a different set of parameter values. Performance metrics are computed over an E1(600,60) experiment repeated 5 times. Results are averaged and mean values are shown with a 95% CI. The COV and the `additional` allocation values are shown only for the Tomcat component. Similar observations hold for the other two components.

the allocation and the utilisation.

Parameter λ

λ is varied as shown in the first 5 lines of Table 6.3. As λ increases, the application serves more requests (increasing CR), and the percentage of requests with response times $\leq$ 1s also increases (increasing NR). This is because with larger λ the controller allocates more resources (increasing `additional` allocations) and therefore it serves more requests. λ does not have a significant affect on the allocation COV.

Parameter p^i and r^i

Changing the values of parameters p^i (lines $6 - 8$ in Table 6.3) and r^i (lines $9 - 11$ in Table 6.3) also affect the controller's performance. As p^i or r^i increase, more resources are allocated to the server components (increasing `additional` allocations) and the server achieves better performance (increasing CR and NR). In addition, the COV of the allocation decreases as p^i increases. This is because more resources are allocated and the two signals, allocation and utilisation, are not that close to each other and therefore do not affect each other. r^i values do not significantly affect the COV of the allocation signal.

To conclude, increasing the parameters improves the server's performance as more resources are allocated to the components. However, there are cases where further increasing the `additional` allocations do not significantly improve the performance (e.g. lines 7 and 8 in Table 6.3). This is aligned with the system identification analysis that defined the `extra` allocation parameter. A balance between the desired application performance and the `additional` allocations on a shared cluster can be achieved with further system identification analysis.

Finally, one of the most challenging tasks in applying any control-theoretic tool to resource management is the configuration of its parameters. This section followed a step-by-step procedure to setting the values of the SISO-UB input parameters which can be used as guidelines when deploying a new application. The procedure is the following: (a) define the r^i values with simple system identification experiments (as in the case of Figure 4.4), (b) set the p^i values from empirical analysis; (c) set λ values according to the controller's stability analysis. The server's performance can be improved by increasing any of the p^i, r^i, or λ, at the cost of additional resources.

Properties

In Chapter 2 it was discussed (Section 2.4.1) that the SASO properties (stability, accuracy, short settling times, and small overshoot) are desirable in a control system. The SISO-UB stability analysis was given in Section 5.1.1. The analysis for the other three properties is given here. Recall, that a controller's properties are studied with respect to the reference output and the steady-state value. However, in the case of the SISO-UB controller, there is no reference output; this controller maintains the allocation within boundaries (between $p^i u_k^i$ and $p^i u_k^i + \lambda e_k^i$) which change with u_k^i, p^i, and λ. The analysis is based on Matlab simulations where the utilisation signal is kept constant and without fluctuations in order for the allocation to reach a steady-state value.

Figure 6.2 shows the SISO-UB allocations for three different λ values. The utilisation is kept constant at 50%, the allocation starts at 70%, $r^i = 30$, and $p^i = 1.25$. Although, the output does not converge to a pre-defined value, it converges to some value, as shown in the figure. The properties are discussed with respect to these values for different λs. As the λ increases, the allocations converge to certain values, therefore, the steady-state error (e_{ss}), with respect to these values, is zero. The controller practically does not overshoot; for instance, in the worse case of allocations fluctuations when $\lambda = 0.9$, M_p is calculated and equals 0.008. Finally, the allocations converge to a steady-state value immediately; the allocations are all within 2% of the steady-state utilisations.

Summary

This section evaluated the SISO-UB controller performance. An E0(20,40,60) experiment showed that the controller achieves both of its goals: (a) the allocations are adjusted according to utilisation and (b) the server's performance remains close to its reference value throughout the experiment; the majority of the intervals had `mRT` $\leq$ 1s. In addition, an evaluation of the parameter values on the server's performance was given. Parameters

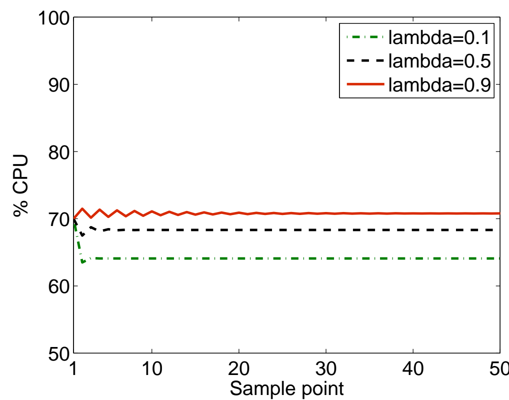


Figure 6.2: SISO-UB Allocations for Stable Input.

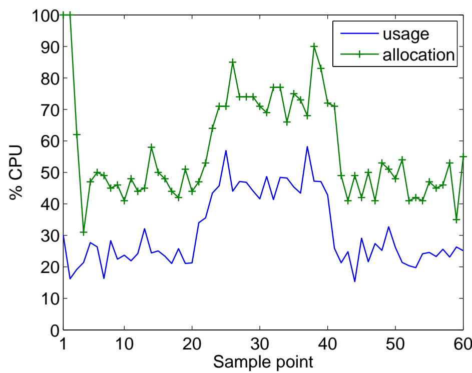
p^i have the most effects on the allocation COV. Finally, this section provided guidelines to choose and tune the SISO-UB configuration parameters when deploying a new application.

6.2.2 MIMO-UB

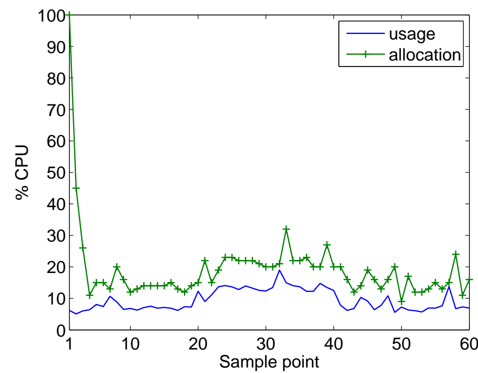
The MIMO-UB controller (presented in Section 5.2.1) extends the SISO-UB design to collectively allocate CPU resources to all application tiers based on the offline derived utilisation model between components. Its control law is given by: $\mathbf{a}_{k+1} = \mathbf{P}\mathbf{u}_k + \lambda\mathbf{M}\mathbf{e}_k$, from Equation (5.29). The allocation of each component is proportional (by p^i) to its utilisation of the previous interval and also proportional (by λ) to its control error and the other components' errors from its own point of view. Finally, the control error for each component uses the additive system model and, as in the SISO-UB case, equals the difference of the allocation less the utilisation from the reference value r^i.

Initially, the MIMO-UB performance is evaluated using an E0(20,40,60) experiment (Figure 6.3). In this experiment, the controller's configuration parameters are set to: $p^i = 1.25, \lambda = 0.3$ and $r^i = 20$, where the same parameters for all components are set to the same values for simplicity. Figures 6.3(a), 6.3(b), and 6.3(c) show the controller's CPU allocations and components' utilisations. At first, the controller decreases the CPU allocations from the initial 100% to an amount close to each component's CPU utilisation. The CPU allocations however, follow the changes in CPU usage for the duration of the experiment, even when the number of clients increases or decreases substantially as can be seen at around sample points 20 and 40. The allocations follow all utilisation changes, even small ones. This was also observed in the case of the SISO-UB controller. The Kalman based controllers address this behaviour and allocate resources that are not so strongly affected by noisy fluctuations.

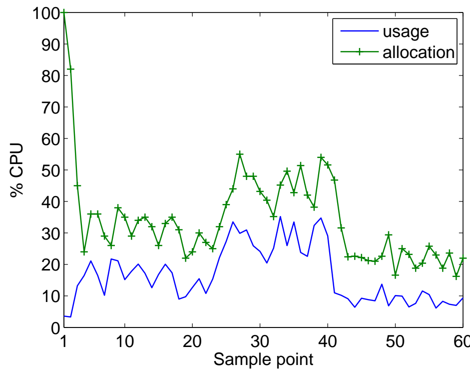
The server's performance is shown in Figures 6.3(d), 6.3(e), and 6.3(f). For most of the experiment, the `mRT` stays very close to its target performance ($\leq$ 1s). There are a few `mRT` spikes that approach 1s; this is happening when any of the component's utilisation gets very close to its allocation for that interval. To better evaluate the `mRT` spikes,



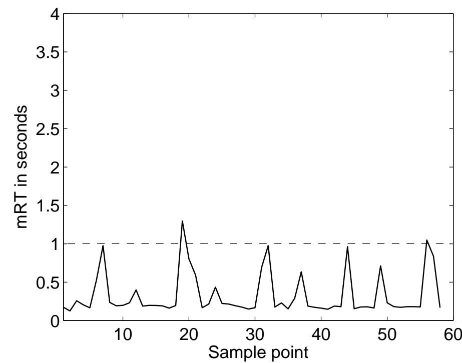
(a) Tomcat CPU allocation and utilisation



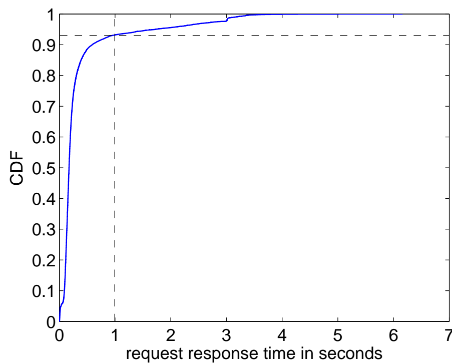
(b) JBoss CPU allocation and utilisation



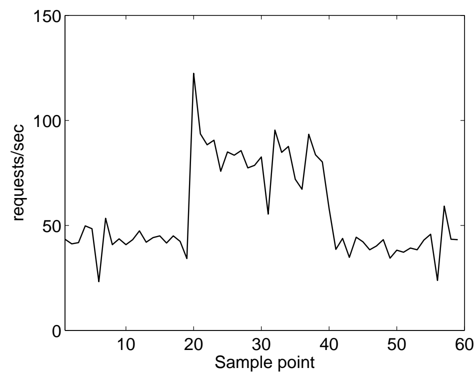
(c) MySQL CPU allocation and utilisation



(d) Requests `mRT`



(e) CDF of request response times



(f) `Throughput`

Figure 6.3: MIMO-UB Controller Performance. Figures illustrate the performance of the MIMO-UB controller for variable number of clients. The controller adjusts the allocations to follow the utilisations. The server's performance approaches very close to its reference value, with a few short spikes in `mRT` and drops in `Throughput`, when the utilisation of one or more components are very close to their allocations.

Figure 6.3(e) depicts the CDF of request responses. The majority of requests (93.18%) have response times ($\leq$ 1s). Finally, the `Throughput` increases or decreases substantially with workload changes. Overall, the MIMO-UB controller achieves both its goals: (a) its allocations follow the components' utilisations and (b) the server's performance is very close to its target performance goal.

When comparing Figure 6.3 (MIMO-UB) and Figure 6.1 (SISO-UB) it is clear that the MIMO-UB controller performs better because it allocates more resources to components aligned to all components' errors and, therefore, the requests have overall better response times. The two controllers are thoroughly compared in Section 6.2.3 where they are also configured to provide similar `additional` allocations under stable workloads.

Parameter Configuration

There are three categories of MIMO-UB configuration parameters: (a) the diagonal $\mathbf{P}$ matrix contains the p^i values. If i denotes the component, then p^i denotes the minimum proportion of utilisation that the allocation is assigned to; (b) the matrix $\mathbf{r}$ contains the r^i values which correspond to the `extra` allocation used to compute the control error for each component i; and, (c) the λ parameter shows the proportion of the control error accounted for the final allocation. This section continues to evaluate the effects of the configuration parameter values on the server's performance. For simplicity, and without loss of generality, the same category parameters are set to the same values for all components for the current evaluation.

Similarly to the SISO-UB parameter analysis, the MIMO-UB parameters are set to values that are either empirically assigned or come from the system identification analysis and correspond to worse-, medium-, and best-case allocation policies. p^i have to be > 1 and are set to values from $\{1.11, 1.25, 1.43\}$. r^i are set to values from $\{10, 20, 30\}$. Finally, λ values are set to $\{0.1, 0.2, 0.3\}$ because the MIMO-UB controller is stable when $|\lambda| < \frac{1}{3}$ (Equation (5.35)). As in the SISO-UB case, only positive λ values are considered. To evaluate the effects of each parameter in turn, only its values change while the values of the other two parameters remain the same and are assigned to the corresponding values of the middle-case scenario.

The evaluation is performed across different sets of values of the configuration parameters. The server's performance is measured over E1(600,60) experiments repeated five times for each set of parameter values and evaluated using four different metrics: CR, NR, `additional` allocation, and COV. Results are shown in Table 6.4 where λ values are varied in lines (1-3), p^i are varied in lines (4-6), and r^i in lines (7-9). In general, increasing any of the input parameters increases the allocated resources at the server's components (increasing `additional` allocation). This causes the application to serve more requests in total (increasing CR) and to increase the proportion of requests with response time $\leq$ 1s (increasing NR).

index	parameters			application performance metrics				
	p^i	λ	r^i	CR	NR	`additional` allocation	COV alloc.	COV usage
1	1.25	**0.1**	20	25000 ± 270	0.86 ± 0.02	1130 ± 12	0.14	0.15
2	1.25	**0.2**	20	25539 ± 138	0.90 ± 0.01	1441 ± 12	0.12	0.12
3	1.25	**0.3**	20	25835 ± 201	0.93 ± 0.00	1724 ± 15	0.10	0.11
4	**1.11**	0.2	20	25270 ± 118	0.89 ± 0.01	1108 ± 8	0.13	0.11
5	**1.25**	0.2	20	25539 ± 138	0.90 ± 0.01	1441 ± 12	0.12	0.12
6	**1.43**	0.5	20	25722 ± 219	0.92 ± 0.00	1855 ± 16	0.10	0.12
7	1.25	0.2	**10**	25024 ± 273	0.86 ± 0.01	1111 ± 13	0.13	0.10
8	1.25	0.2	**20**	25539 ± 138	0.90 ± 0.01	1441 ± 12	0.12	0.12
9	1.25	0.2	**30**	25906 ± 100	0.93 ± 0.01	1760 ± 16	0.11	0.12

Table 6.4: Input Parameter Configuration for MIMO-UB Controller. Each row corresponds to a different set of parameter values. Performance metrics are computed over an E1(600,60) experiment repeated 5 times. Results are averaged and mean values are shown with a 95% CI. The COV and the `additional` allocation values are shown only for the Tomcat component. Similar observations hold for the other two components.

Properties

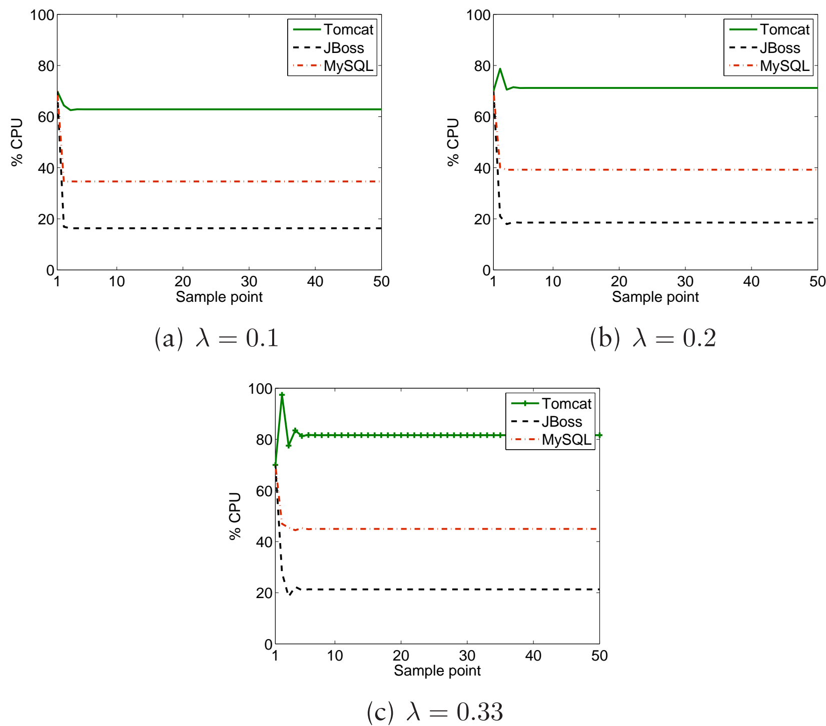
As in the case of the SISO-UB controllers, the MIMO-UB controller does not maintain its output to a reference value. However, when the utilisations are kept constant, its allocations converge to some values based on which the properties of accuracy, overshoot, and settling times are now discussed.

The analysis is based on Matlab simulations where the utilisation signal is kept constant in order to avoid noise fluctuations. Figure 6.4 shows the MIMO-UB allocations for all components and for three different λ values. The utilisation is kept constant at 40%, the allocation starts at 70%, $r^i = 30$, and $p^i = 1.25$.

In all cases, the allocations converge to certain values, therefore, the steady-state error, e_{ss}, is zero. The controller generally does not overshoot, unless the utilisation is relatively high; for instance, when $\lambda = 0.33$ for the Tomcat component, M_p is calculated and equals 0.2. Finally, the allocations converge to a steady-state value in just a few intervals; for example in the case of $\lambda = 0.33$ and the Tomcat component, the allocations are within 2% of the steady-state value in 5 intervals.

Summary

This section initially evaluated the MIMO-UB controller's performance using an E0(20,40,60) experiment. Results showed that the controller achieved both its goals: (a) allocations follow utilisation and (b) the server's performance stays very close to its target performance. This section also evaluated the effects of the controller parameter



(a) $\lambda = 0.1$ (b) $\lambda = 0.2$

(c) $\lambda = 0.33$

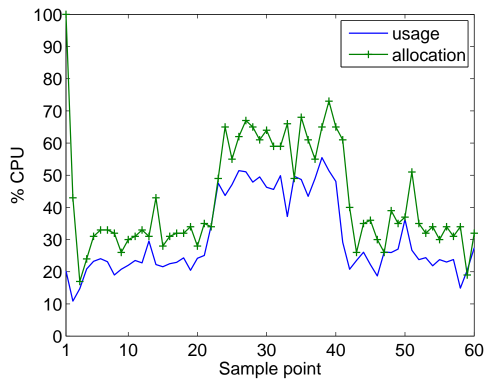
Figure 6.4: MIMO-UB Allocations for Stable Input.

values on the server's performance. Increasing the values of any parameter improves the server's performance. In a production environment, an economic model based on further parameter analysis can be adopted to choose the most appropriate values. The procedure used to choose the parameter values and to further analyse their effects on server performance can be used as a guideline when deploying a new application.

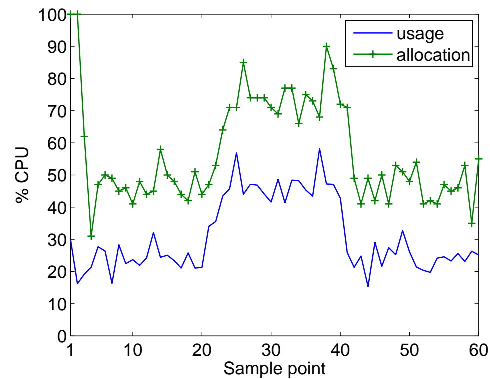
6.2.3 SISO-UB and MIMO-UB Comparison

The MIMO-UB controller is designed to incorporate the resource coupling between tiers and provide faster responses to workload changes for multi-tier applications by taking into account the control error from all components when allocating resources to any of the tiers. This section compares the performance of the MIMO-UB controller against the SISO-UB to evaluate the above. The roadmap of the evaluation is as follows. Firstly, the behaviour of the controllers are studied when configured with same parameter values. Secondly, the controllers are examined when they are configured to make similar `additional` allocations.

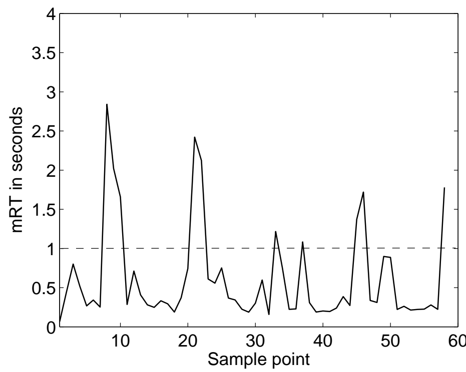
Initially, the two controllers are compared using the two E0(20,40,60) experiments from the last two sections. Their configuration parameters are set to the same values. Key results from the two experiments are placed side-by-side in Figure 6.5 for comparison reasons. Figures 6.5(a) and 6.5(b) illustrate the allocations and utilisations for the Tomcat component. While both controllers follow the changes of the utilisations throughout the experiment, they differ in the amount of resources that they allocate for similar util-



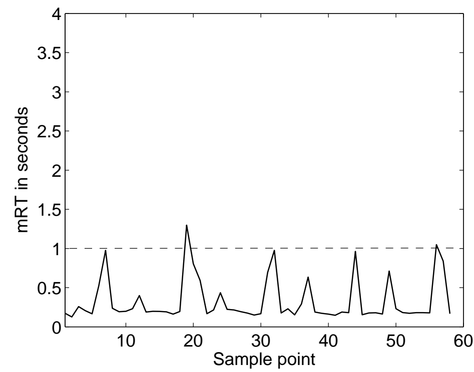
(a) SISO-UB Tomcat CPU allocation and utilisation



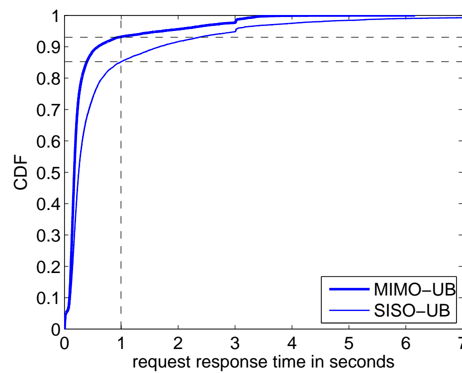
(b) MIMO-UB Tomcat CPU allocation and usage



(c) SISO-UB Requests mRT



(d) MIMO-UB Requests mRT



(e) SISO-UB, MIMO-UB CDF of response times

Figure 6.5: SISO-UB and MIMO-UB Comparison. The MIMO-UB controller allocates more resources to components. The application is better provisioned to serve incoming requests as can be seen by the fewer spikes in the mRT and the higher proportion of requests with response times $\leq$ 1s.

isations. The MIMO-UB controller clearly allocates more resources since it accounts for the aggregate control errors from all components. As a result, the application is better provisioned to serve incoming requests. The `mRT` in the case of the MIMO-UB controller (Figure 6.5(d)) has fewer and shorter spikes and stays closer to its target performance for more intervals than the SISO-UB controller (Figure 6.5(c)). Finally, Figure 6.5(e) depicts the CDF of the request response times from both controllers. Again, in the MIMO-UB case the proportion of requests with response times $\leq$ 1s is higher than in the SISO-UB case.

This section next compares the two controllers across different configuration parameters. To this end, an E2 experiment, where a large workload increase occurs, is performed 20 times for each different set of parameter values for the SISO-UB and MIMO-UB controllers. Table 6.5 shows the values of the parameters which are chosen to exercise the spectrum of their possible values. Results are shown in Figure 6.6 where the performance of the server is compared using the CR, NR, RMS, and `additional` metrics. Each figure also shows the absolute percentage of the metric difference between the MIMO-UB controller over the SISO-UBs with a 95% CI. The CI is calculated after a T-test is applied to compare the results from the different experiment runs.

Figures show that according to all metrics (CR, NR, and RMS), the MIMO-UB controller performs better than the SISO-UBs. There are more completed requests (CR) during the workload change, more requests with response times $\leq$ 1s, and more requests with response times closer to their mean value for that workload (RMS). The MIMO-UB controller outperforms the SISO-UBs because it allocates more resources to components as aligned to their errors (Figure 6.6(d)).

To better evaluate how much it *costs* for the MIMO-UB to improve the SISO-UB with respect to the increased additional resources it allocates, the following analysis is performed. The aim is to roughly estimate how many more resources the MIMO-UB allocates than the SISO-UB for a given server performance. To this end, the data from Tables 6.3 and 6.4, which come from E1(600,60) experiments with different parameter configurations for both controllers, are used. The mean `additional` resources per controller interval (which is 5s) each controller allocates for every 400 completed requests[2] across configurations for the Tomcat component is 16.25 CPU units[3] for the SISO-UB and 22.65 CPU units for the MIMO-UB. So, the MIMO-UB allocates on average 6.5 more CPU units than the SISO-UB controllers. To put this number under perspective with respect to the mean utilisation in the case of the E1(600,60) experiments, which is around 46, the following sentence gives an indication of the cost of the increased allocations by the MIMO-UB: for every 400 completed requests the MIMO-UB allocates more resources than the SISO-UB which equal in the case of the Tomcat component to $6.4/46 = 13.91\%$ of its utilisation.

So far the comparison showed that the MIMO-UB outperforms the SISO-UB controllers simply because the former controller allocates more resources when both are configured

[2]The mean `additional` resources is calculated for 400 requests because, this is very close to the average number of completed requests per controller interval over which mean utilisation and mean allocation measurements are taken.

[3]CPU units is the % of CPU resources.

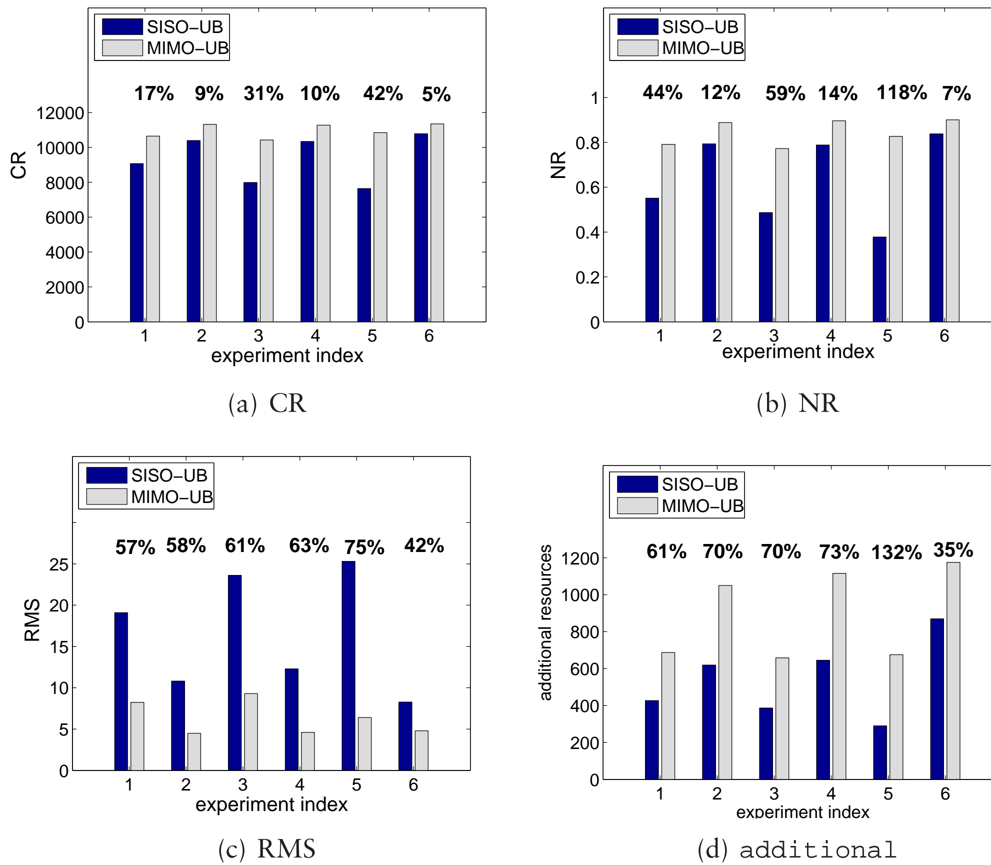


Figure 6.6: SISO-UB and MIMO-UB Comparison for Different Parameter Values. Percentages in each case show the absolute metric difference of the MIMO-UB controller over the SISO-UBs within a 95% CI.

experiment index	p^i	r^i	λ
1	1.25	20	0.1
2	1.25	20	0.3
3	1.25	10	0.2
4	1.25	30	0.2
5	1.11	20	0.2
6	1.43	20	0.2

Table 6.5: Experiment Description for the MIMO-UB and SISO-UB Comparison.

to same parameter values. It can also be noted that there are cases between different sets of configuration parameters, that the performance of the server is similar between the MIMO-UB and the SISO-UB controllers (e.g. in Figure 6.6(a) the MIMO-UB CR in the 1st experiment is almost equal to the SISO-UB CR in the 4th experiment). This is because the parameters of the SISO-UB controllers caused the allocations to be high enough and compared with the MIMO-UB allocations.

The evaluation continues to compare the controllers when they are particularly con-

	SISO 0.1	SISO 0.2	SISO 0.3	SISO 0.4	SISO 0.5	SISO 0.6	SISO 0.7	SISO 0.8	SISO 0.9
MIMO 0.1	MIMO 56%	MIMO 27%	MIMO 9.6%	SISO 2.3%	SISO 11%	SISO 19%	SISO 25%	SISO 31%	SISO 37%
MIMO 0.2	MIMO 116%	MIMO 77%	MIMO 51%	MIMO 35%	MIMO 22%	MIMO 12%	MIMO 3.6%	SISO 5.5%	SISO 13%
MIMO 0.3	MIMO 170%	MIMO 120%	MIMO 89%	MIMO 68%	MIMO 52%	MIMO 39%	MIMO 29%	MIMO 17%	MIMO 8.2%

Table 6.6: `Additional` Resources Comparison. This table compares the `additional` resources between the SISO-UB (short SISO) and the MIMO-UB (short MIMO) controllers as measured for stable workloads and different λ configurations. For each different λ and for each controller design an E1(600,60) experiment is repeated 10 times. The table shows the results from the pairwise comparisons, using a T-test. Each comparison shows the controller with the highest amount of `additional` resources along with its increased percentage.

figured to make similar `additional` allocations. This enables to focuss on the MIMO-UB mechanism for faster workload adaptation against the SISO-UB controllers. This comparison is performed in two steps. Firstly, the values of the λ parameters that make the two controllers to allocate similar `additional` resources are identified. Secondly, using these values the controllers are compared when a large workload increase occurs through an E2 experiment.

Initially, the next experiment identifies the λ values of the SISO-UB and MIMO-UB which under stable workload conditions cause the controllers to have similar `additional` allocations. To this end, the `additional` allocations for each controller and for different λ values are measured using an E1(600,60) experiment repeated 10 times. The range of λ values comes in the case of the SISO-UB controllers from Equation (5.5) and in the case of the MIMO-UB from Equation (5.35). Pairwise comparisons of the `additional` allocations between the two controllers for different λ values are performed and results are shown in Table 6.6. For each combination of λ, the table shows which controller allocates more resources and also indicates the percentage of its increased `additional` resources. For instance, when comparing the SISO-UB with $\lambda = 0.3$ and the MIMO-UB with $\lambda = 0.1$, the table shows that the MIMO-UB controller allocates 9.6% more resources.

This table indicates that the `additional` allocations for each controller also depend on its parameter values; this is aligned with previous results (Figure 6.6). For example, as the SISO-UB λ increases for same MIMO-UB λ, the SISO-UB controllers allocate more resources. This table is used to narrow the search range for λ values which cause similar allocations between the controllers; two regions, among many, are identified: (SISO-UB $\lambda \in (0.4, 0.5)$ and MIMO-UB $\lambda \in (0.1, 0.2)$) and (SISO-UB $\lambda \in (0.7, 0.8)$ and MIMO-UB $\lambda \in (0.1, 0.2)$). Additional E1(600,60) experiments repeated 20 times across λ values within the above ranges were performed. Two combinations of parameters that give similar `additional` allocations are identified: (SISO-UB $\lambda = 0.45$ and MIMO-UB $\lambda = 0.12$) and (SISO-UB $\lambda = 0.75$ and MIMO-UB $\lambda = 0.2$).

For each of the two combinations of parameters the controllers are now compared

intervals	additional	CR	NR
1-29	-0.2%	0.4%	1.2%
30-40	6.2%	2.8%	2.6%

Table 6.7: UB Controllers Comparison, SISO-UB $\lambda = 0.45$, MIMO-UB $\lambda = 0.12$.

intervals	additional	CR	NR
1-29	-0.6%	-0.67%	0.7%
30-40	5.1%	1.6%	3.2%

Table 6.8: UB Controllers Comparison, SISO-UB $\lambda = 0.75$, MIMO-UB $\lambda = 0.12$.

against large workload increases. In each case, an E2 type of experiment is performed 20 times and results are shown in Tables 6.7 and 6.8. Each table compares the controllers across two regions of intervals. The first region (intervals 1-29) involves the period before the workload increase and is used to examine that the two controllers allocate similar `additional` resources. The second region (intervals 30-40) is the time during the workload increase. Percentages in each case show the metric difference between the two controllers over the SISO-UB with a 95% CI after a T-test is applied. A positive number indicates that the metric in the MIMO-UB case is larger than in the SISO-UB.

Results show that during the first 29 intervals the controllers have very similar performances (first row in both tables). Their `additional` resources are very close and so is their performance, shown by CR and NR. However, during the intervals of the workload change (second row in both tables) the MIMO-UB controller allocates more resources despite being configured to make similar allocations to the SISO-UB. The increased allocations cause the MIMO-UB controller to have better performance as shown by the increased number of completed requests (CR) and the proportion of requests with `mRT` $\leq$ 1s (NR).

The MIMO-UB controller is able to act faster to workload increases by allocating more resources than the SISO-UB controllers because it accounts for all components' errors according to their resource coupling model derived offline. In this case, the error from any component is "translated" to an error for any other component. In this way, the MIMO-UB adjusts the allocations so that all components are settled down to the workload change as soon as possible.

Summary

This section compared the two usage based controllers, the SISO-UB and the MIMO-UB. Comparison showed that under workload increases and the same parameter values between the two controller designs the MIMO-UB controller outperforms the SISO-UBs because it allocates more resources since it accounts for the aggregate error from all components. However, even when the two controllers are configured to make similar allocates, the MIMO-UB responds faster to workload increases because it allocates more resources according to the resource coupling model.

6.2.4 Discussion

The last section compared the usage based controllers, the SISO-UB and the MIMO-UB. Analysis showed that both controllers achieve their goals. Further parameter tuning showed that increasing their values results in improving the server's performance since more resources are allocated to components. In addition, the MIMO-UB controller improves the SISO-UB controllers, since it considers the control errors from all components according to their resource coupling model.

Before proceeding to the evaluation of the Kalman based controllers, there are three issues related to the usage-based controllers that need to be addressed. First, there are a number of input configuration parameters for both controllers, the values of which affect the server's performance. This section also suggested a procedure towards setting their values and provided guidelines for their tuning based on experimental analysis. However, setting their values for a new application is a challenging task. Second, the allocations made by either controller are affected by every resource fluctuation, even the subtle ones. This behaviour might not be appropriate in several consolidation cases where changes in the resource allocations should reflect important workload changes. Third, the MIMO-UB controller uses the resource coupling model derived offline for one workload type. Since this might be a drawback for this controller, an online version was also suggested (Section 5.2.1).

The Kalman based controllers address all of these issues. The KBC and PNCC have fewer configuration parameters than the usage based ones and the APNCC further contains an adaptation mechanism to set all but one of its parameters. In addition, with any of the Kalman based controllers the allocations can be tuned to different degrees of variability. The next section evaluates the Kalman based controllers and discusses these issues.

6.3 Kalman Based Controllers

This section evaluates the performance of the three Kalman-based controllers. The SISO Kalman Basic Controller (KBC) (presented in Section 5.1.4) allocates CPU resources separately to each application tier. Based on the Kalman filtering technique it uses past utilisation measurements to predict the utilisation for the next interval and hence the allocation itself. The KBC controller is based on the multiplicative system model (Equation (4.2)) to achieve reference performance; the controller aims to maintain the utilisation rate within each VM at a certain $\frac{1}{c}$ of its allocation (Equation (5.16)). The integration of the Kalman filtering technique into a feedback controller is particularly useful for the current problem, since: (a) the controller filters out the transient changes from the utilisation signal and discovers the important underlying resource variations; and (b) the responsiveness of the controller to the control error (through the Kalman gain) depends on the system's variance itself (via the process noise variance computed offline). Different degrees of noise filtering are achieved with parameter tuning.

The second Kalman-based controller is the MIMO Process Noise Covariance Controller (PNCC) controller (Section 5.2.2). The PNCC controller extends the KBC design to

collectively allocate resources to all application tiers. It captures the resource correlation between components' utilisations through the use of the covariance process noises. The computation of the covariance matrix **Q**, which contains all the components' variances and covariances, is performed offline.

The final controller evaluated is the Adaptive Process Noise Covariance Controller (APNCC) controller (Section 5.3). This controller is similar to the PNCC design, but, in addition, it updates the estimate of the covariance matrix **Q** online by using the utilisation measurements.

The purpose of the experimental procedure is to evaluate each controller's ability to achieve its goals: the allocations should (a) follow the utilisations, and (b) maintain the server's performance to its reference value under workload changes. In addition, this section evaluates specific features such as: (a) the effects of the parameter tuning to the variability of the allocation signal and (b) the ability of the APNCC to adapt to workload changes and compute the system variances and covariances online. Finally, comparisons between pairs of controllers are performed. The PNCC is compared against the KBC in order to evaluate the use of the covariance noise between components. Finally, the APNCC is compared against the PNCC to evaluate the effectiveness of the adaptation mechanism to compute the system variances. The rest of this section discusses the settings of some parameter values that are common to all experiments that follow.

Parameter c

The parameter c, which denotes the level of the CPU utilisation rate to the allocation, is set to 60%. Although 60% might seem low and that resources are "wasted" and hence less resources are left for another application to run, this value enables the study of the controllers' performance with few implications from saturated components. Previous analysis showed that when the allocation approaches the utilisation, the server's performance degrades. To study the Kalman based controller, a low utilisation rate is chosen, so as to avoid transient component saturation as much as possible.

Variances and Covariances

Both the KBC and the PNCC controllers use offline computed parameters. Each KBC controller uses the allocation process variance Q and the PNCC controller uses the process covariance matrix $\mathbf{Q}$. The diagonal elements contain the variances of the components (e.g. $\mathbf{Q}(1,1)$ denotes the process noise variance of the 1st component). The non-diagonal elements of the symmetric $\mathbf{Q}$ matrix contain the covariances between components, (e.g. $\mathbf{Q}(1,2)$ denotes the process noise covariance between the 1st and the 2nd components). These parameters are essential to the computation of the Kalman gains and to the allocations for the next interval.

To compute their values, it was shown that it suffices to compute the corresponding values for the utilisations and then perform the necessary transformations using the parameter c (Equations (5.22) and (5.43)). The variances and covariances of the components'

component	usage variance
Tomcat	28.44
JBoss	4.75
MySQL	47.43

component pair	usage covariance
(Tomcat, JBoss)	2.36
(Tomcat, MySQL)	5.06
(JBoss, MySQL)	1.80

Figure 6.7: Values of Utilisation Variances and Covariances. These values are computed from utilisation measurements under static workload of 600 clients. The set of the offline computed variance values is referred to as Q_0 values. The covariance matrix with the offline computed variances and covariances is referred to as $\mathbf{Q}_0$ matrix.

utilisations are computed using an E1(600,40) experiment repeated ten times for confident results and where each component is allocated 100% of its CPU. Although the variances and covariances might change with the number of clients, for simplicity the offline computations are performed for a fixed number of clients (600). The current offline computed numbers for 600 clients are used throughout the evaluation as an approximation for other workloads as well. The left hand-side table in Figure 6.7 shows the utilisation variances of the three Rubis components. Hereafter, these set of values are referred to as Q_0. Finally, the right hand-side table in Figure 6.7 depicts the utilisation covariances between components. The covariance matrix with the offline computed variances and covariances is referred to as $\mathbf{Q}_0$.

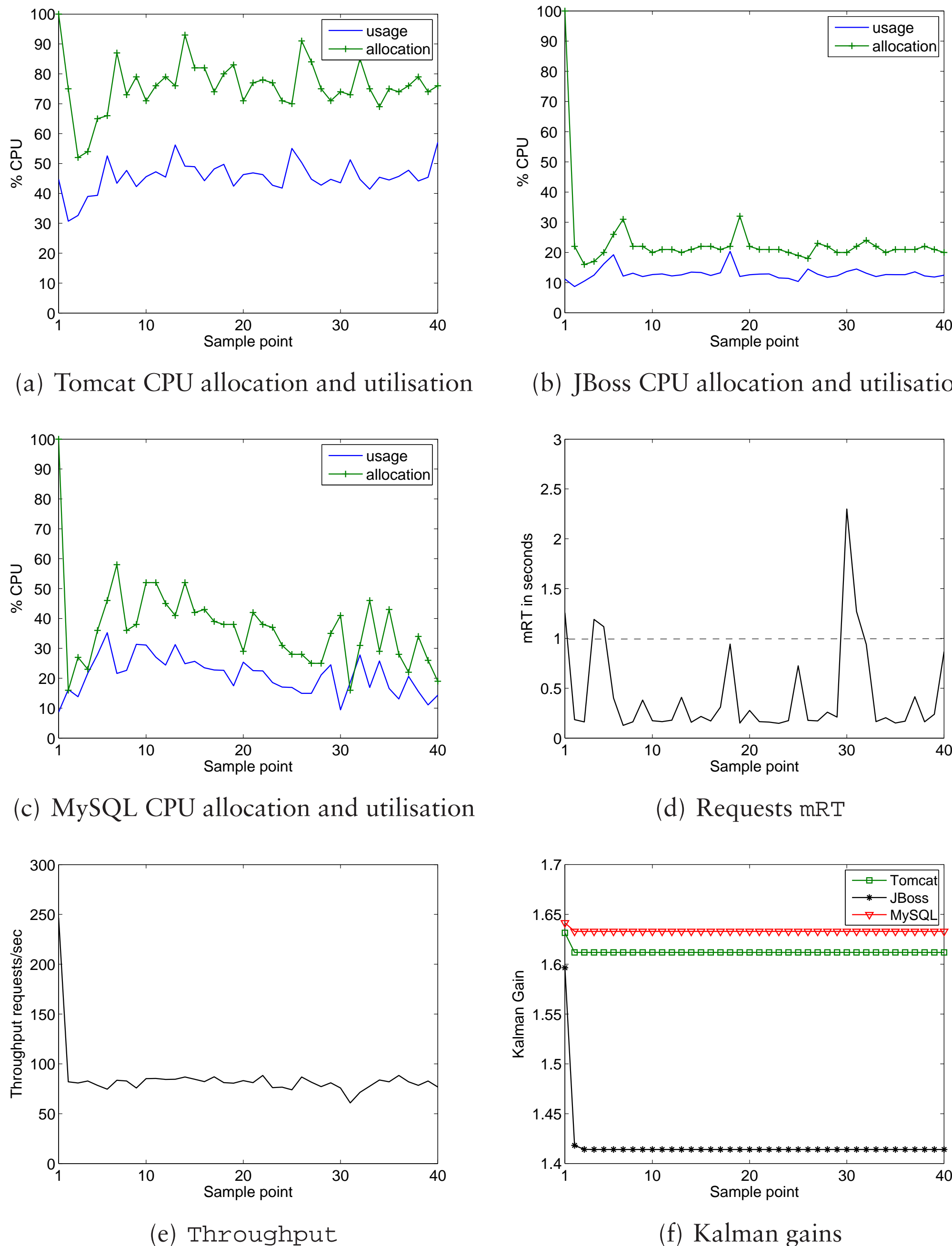
Finally, the measurement noise variance is set to a small value (e.g. $S = 1.0$) given the existence of relatively accurate measurement tools. This value acts as a good approximation of possible measurement errors.

6.3.1 KBC

This section evaluates the performance of the KBC controllers first in the case of a stable workload and then for a variable number of clients. In each case the controllers are evaluated using: (a) the process noise variances Q_0 as computed offline and (b) using a portion of Q_0 for each component which affects the allocation variability and server's performance. The experiments that use stable workloads emphasise the effects of Q_0 values on the allocation variability. The experiments with variable number of clients study the effects of Q_0 values on server performance during a workload change.

Stable Workload

Figure 6.8 illustrates the results of an E1(600,40) experiment with a constant number of clients. Figures 6.8(a), 6.8(b), and 6.8(c) show that the CPU allocations track the utilisations for each component. The server's performance (Figures 6.8(d) and 6.8(e)) is sustained at very good levels. The `mRT` is below its target value (1s) with a few spikes and the `Throughput` remains stable with small variations. The `mRT` spikes are caused when one or more components is close to saturation. The saturation is caused because each



(a) Tomcat CPU allocation and utilisation

(b) JBoss CPU allocation and utilisation

(c) MySQL CPU allocation and utilisation

(d) Requests `mRT`

(e) `Throughput`

(f) Kalman gains

Figure 6.8: KBC Performance for Stable Workload and Q_0 Values. The KBC controllers allocate resource to follow utilisation changes, while the server's performance maintains its target performance (`mRT` $\leq$ 1s) with only a few spikes.

KBC controller corrects its allocation to match even the more subtle resource utilisation changes. For example, if the utilisation drops in one interval but increases the next, the allocation drops as well in the next interval and the component saturates (e.g. sample points 30 – 32 in Figures 6.8(c) and 6.8(d)). This behaviour, which was also observed in the SISO-UB and MIMO-UB controllers, is because of the large values of the Kalman

gains (Figure 6.8(f)) which make each controller to have an increased confidence in the new measurements; therefore, the allocations are corrected after each interval to match the usage. Although in the case of the SISO-UB and MIMO-UB controllers this issue was not addressed, the controllers in this section can be used to filter out small signal fluctuations.

Filters in general and the Kalman filter in particular are used to eliminate the noise from measured signals, while discovering the underlying patterns. In the above figures, the allocations correctly "follow" the usage. However it might be more useful to construct a controller that adjusts to the underlying trends of the CPU signal while not being strongly affected by its variance. In the case of the Kalman filter, this is achieved through the tuning of the parameters Q and S that affect the Kalman gain K (Section 5.1.4). The Kalman gain depends on the relative values of Q and S. If the process noise variance Q is larger than the measurement noise variance S, then the system states are noisy and the Kalman gain is relatively large in order to follow the variable system. If, however, S is larger than Q then the measurements of the system states are noisy, and therefore, the Kalman gain is relatively small and the system is less confident in the new observations of the states.

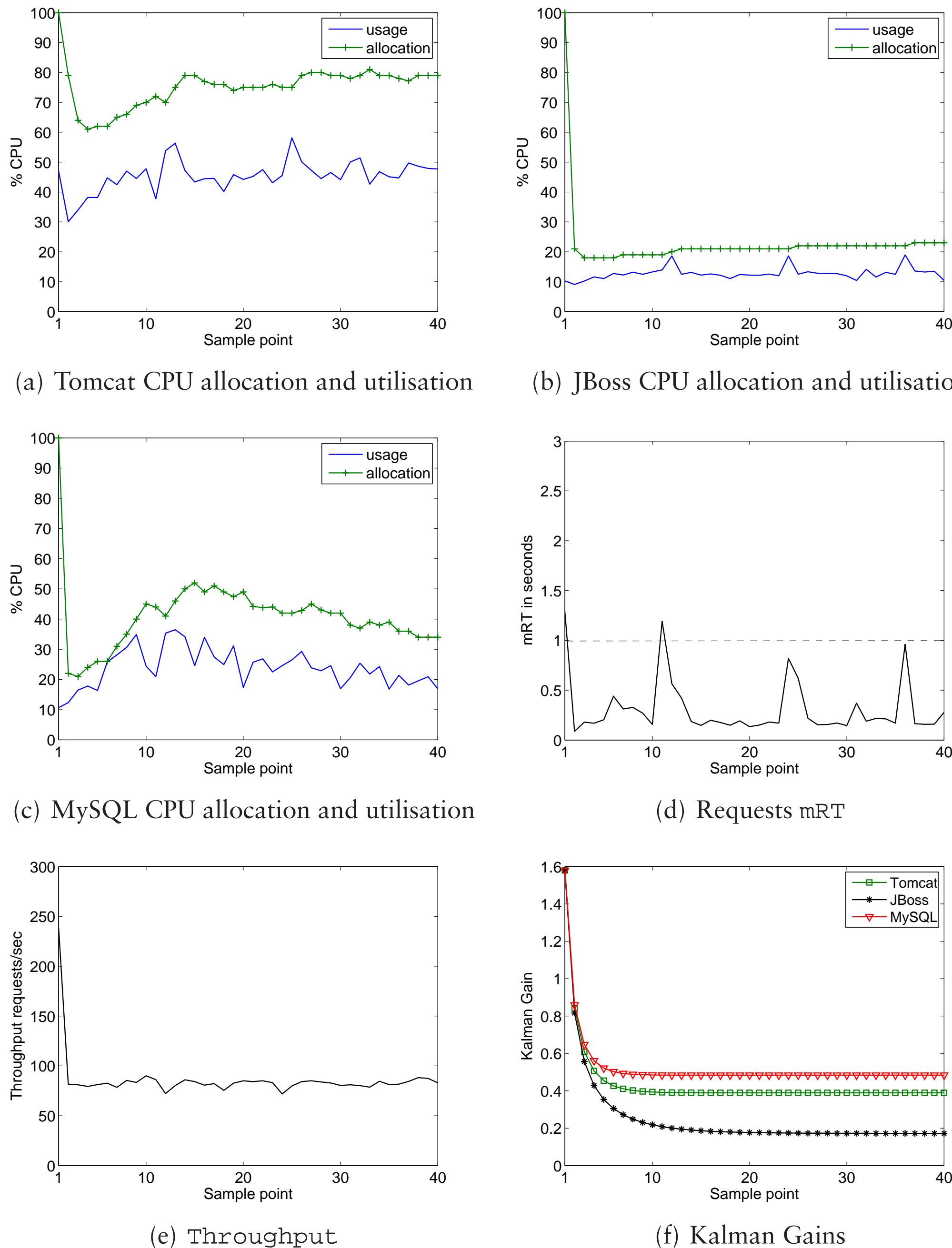
To better illustrate this behaviour a similar experiment of stable workload E1(600,40) is now performed for a different value of K. To this end, only a portion of the offline computed Q values is now considered (Q is divided by 400[4]); for simplicity reasons S is kept the same. Results are shown in Figure 6.9. The allocations are smoother than the utilisations for all the three components (Figures 6.9(a), 6.9(b), and 6.9(c)) and smoother than before (e.g. Tomcat allocation in Figure 6.8(a) is more noisy than the corresponding allocation in Figure 6.9(a)). In the second case, the values of the Kalman gains (Figure 6.9(f)) are smaller than previously (Figure 6.8(f)) and therefore the filter has more confidence in the predicted values than in the measured ones. The server's performance is again maintained below its target performance, with fewer `mRT` spikes than before (Figure 6.9(d)) and stable `Throughput` (Figure 6.9(e)).

To properly evaluate the effects of the different Q values on the Kalman gains and the server's performance, Qo variances are divided by different x values drawn from $Y = \{1, 4, 8, 10, 40, 80, 100, 400, 800, 1000\}$. Hereafter x is denoted as the *damping factor*. These values are chosen in order to cover a wide range of Kalman gains K values.[5] For each x value an experiment E1(600,200) is performed, and results are shown in Figure 6.10 and Table 6.9 where the following metrics are computed: (a) the components' Kalman gains (Figure 6.10(a)), (b) the COV for each component (Figures 6.10(b), 6.10(c) and 6.10(d)), (c) the CR, NR, RMS, and the `additional` resources (Table 6.9).

As the process noise Q decreases (when the damping factor x increases), the Kalman gain for each component drops (Figure 6.10(a)), indicating that the predicted value

[4]Other values are also considered later in this chapter.

[5]It is common practice when studying the Kalman filter in different applications to retain one of the Q or S stable and vary the other.



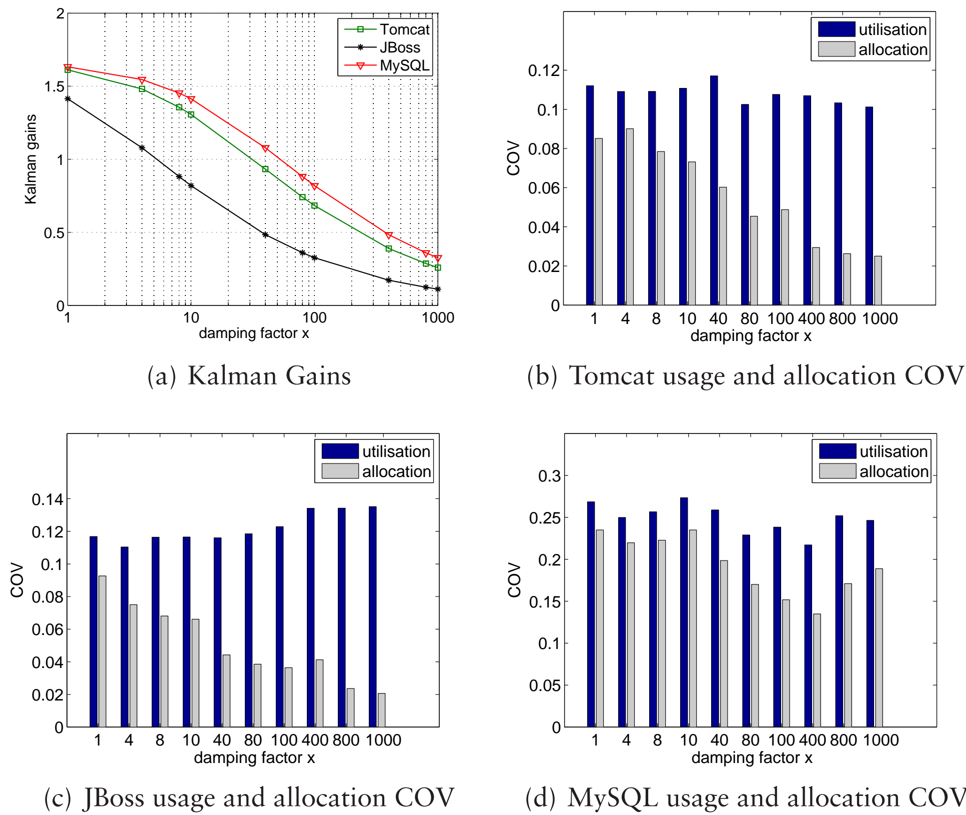
(a) Tomcat CPU allocation and utilisation (b) JBoss CPU allocation and utilisation

(c) MySQL CPU allocation and utilisation (d) Requests `mRT`

(e) `Throughput` (f) Kalman Gains

Figure 6.9: KBC Performance for Stable Workload and $Q_0/400$ Values. The allocations from the KBC controllers follow the utilisations without adapting to every subtle resource change due to their small Kalman gains. The server maintains its target performance (`mRT` $\leq$ 1s) with just a few short spikes.

of the allocation becomes more important than the new measurement. This further affects the variability of the allocation signal. Figure 6.10(b) shows the COV of the allocation and the utilisation signals for the Tomcat component. As Q decreases the allocation COV decreases too while the utilisation COV remains almost the same; the allocations are smoother due to the small Kalman gains. The same observations hold



(a) Kalman Gains (b) Tomcat usage and allocation COV

(c) JBoss usage and allocation COV (d) MySQL usage and allocation COV

Figure 6.10: KBC Performance for Stable Workload and Different Q Values. As Q values decrease (when the x damping factor increases) the Kalman gains decrease, the allocation signal becomes less noisy than the utilisation one and the server's performance, shown in Table 6.9, improves.

x values	CR	NR	RMS	additional
1	82840	0.9164	2.3433	10145
4	82949	0.9304	2.0628	10477
8	82940	0.9324	1.8298	10350
10	83512	0.9365	1.7725	10374
40	83554	0.9350	1.8826	10530
80	83700	0.9429	1.5800	10530
100	83564	0.9425	1.7291	10587
400	83860	0.9462	1.5017	10972
800	83639	0.9470	1.3871	11091
1000	83962	0.9504	1.4417	11085

Table 6.9: KBC Server Performance for Stable Workload and Different x.

for the JBoss (Figure 6.10(c)) and MySQL components (Figure 6.10(d)). In the case of MySQL, the COV increases for the last two x values. This is because there is a small utilisation fluctuation that changes the mean utilisation depicted in the allocation as well.

The server's performance is also affected by the choice of Q. Table 6.9 shows the CR, NR, RMS, and `additional` resources for the duration of the experiment. In the case of the RMS, the `predicted`(response time) is the `mRT` as measured offline for the 600 clients. As x increases the server's performance improves: it serves more clients (increasing CR), the proportion of requests with response times $\leq$ 1s increases too (increasing NR), and all requests are closer to the `mRT` (decreasing RMS). However, note, that the improvement in server's performance with increasing x is achieved because the workload is stable. As the Kalman gains become smaller (increasing x), the controllers are less affected by transient fluctuations, and therefore, they avoid transient component saturations. This is also shown by the increasing `additional` resources allocated to components. However, in a workload increase, a small Kalman gain would stall the controller to adapt to a resource change. Later this section evaluates the effects of Q values on workload changes.

For the rest of this chapter, the evaluation of the Kalman controllers is based on different Q values as divided by x drawn from $X = (x \in \{8, 10, 40, 80, 100, 400\})$. As can be seen from Figure 6.10(a) the Kalman gains for $x \in \{1, 4\}$ are close to the gains for $x = 8$. Similarly, the gains for $x \in \{800, 1000\}$ are close to the gains for $x = 400$. Further evaluation based on these $x \in \{1, 4, 800, 1000\}$ does not significantly contribute to the analysis.

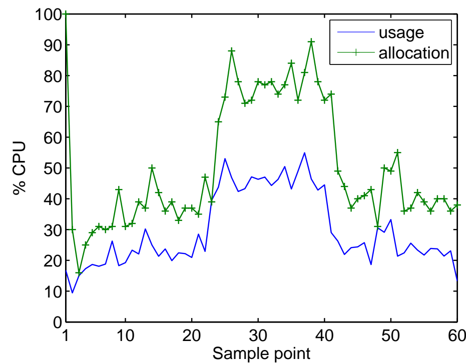
To conclude, by computing the process noise variance offline, the KBC controllers correctly allocate resources as needed and sustain the server's performance very close to its target values. Tuning the Q parameters causes different degrees of allocation variability. With decreasing Q values the controllers' allocations are smoother and do not get so affected by transient fluctuations. For stable workloads this also results in improved server's performance. However, as the allocations are slower to act to resource changes, this might cause slower reactions to workload changes. This issue is considered next.

Workload Changes

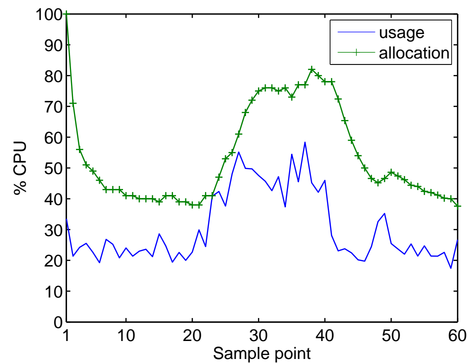
The evaluation of the KBC controllers has been so far on a stable workload. The rest of this section studies performance with variable workloads. Figures 6.11 and 6.12 illustrate the KBC allocations and server's performance respectively during two E0(20,40,60) experiments. The graphs on the left hand-side in both Figures come from an E0(20,40,60) experiment which considers the initial Q_0 process noise variances (hereafter denoted 1st experiment). The graphs on the right hand-side show the results from another E0(20,40,60) experiment (hereafter denoted 2nd experiment) which however, accounts only for a fraction ($Q_0/400$) of the initial process noise variances.

In both experiments, the controllers track the usage fluctuations and the allocations are adjusted accordingly (Figure 6.11). However, it is obvious that the allocations on the right hand-side graphs are smoother than those on the left hand-side and slower to follow the resource changes. This is due to smaller Kalman gains in the 2nd experiment (Figure 6.12(d)) when compared to the 1st (Figure 6.12(c)). This causes the controllers in the 2nd experiment to have more confidence in their predicted values than in the new measurements.

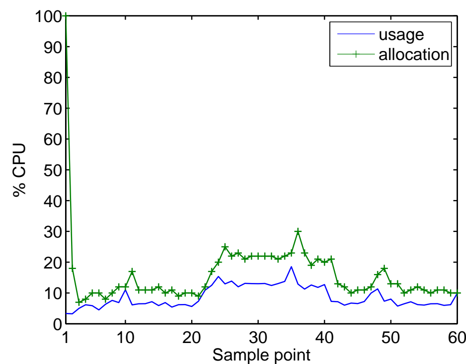
The two Q configurations also affect the servers' performances in different ways. At the



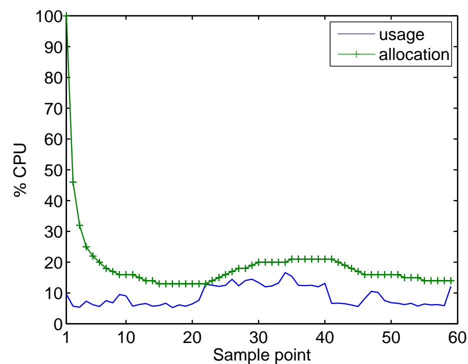
(a) Tomcat CPU allocation, usage (Q_0)



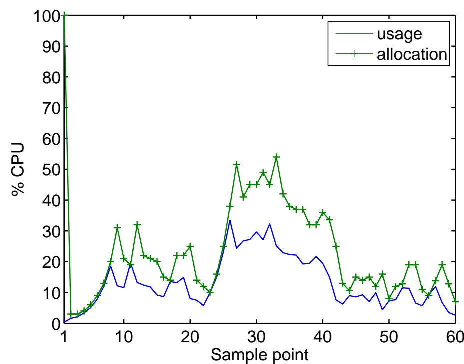
(b) Tomcat CPU allocation, usage ($Q_0/400$)



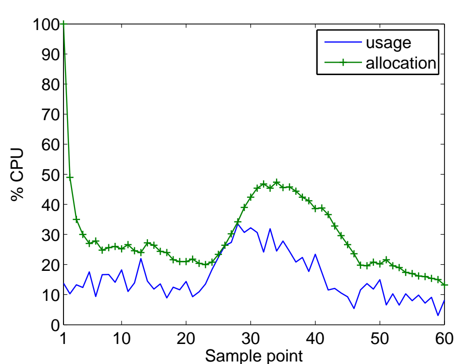
(c) JBoss CPU allocation, usage (Q_0)



(d) JBoss CPU allocation, usage ($Q_0/400$)

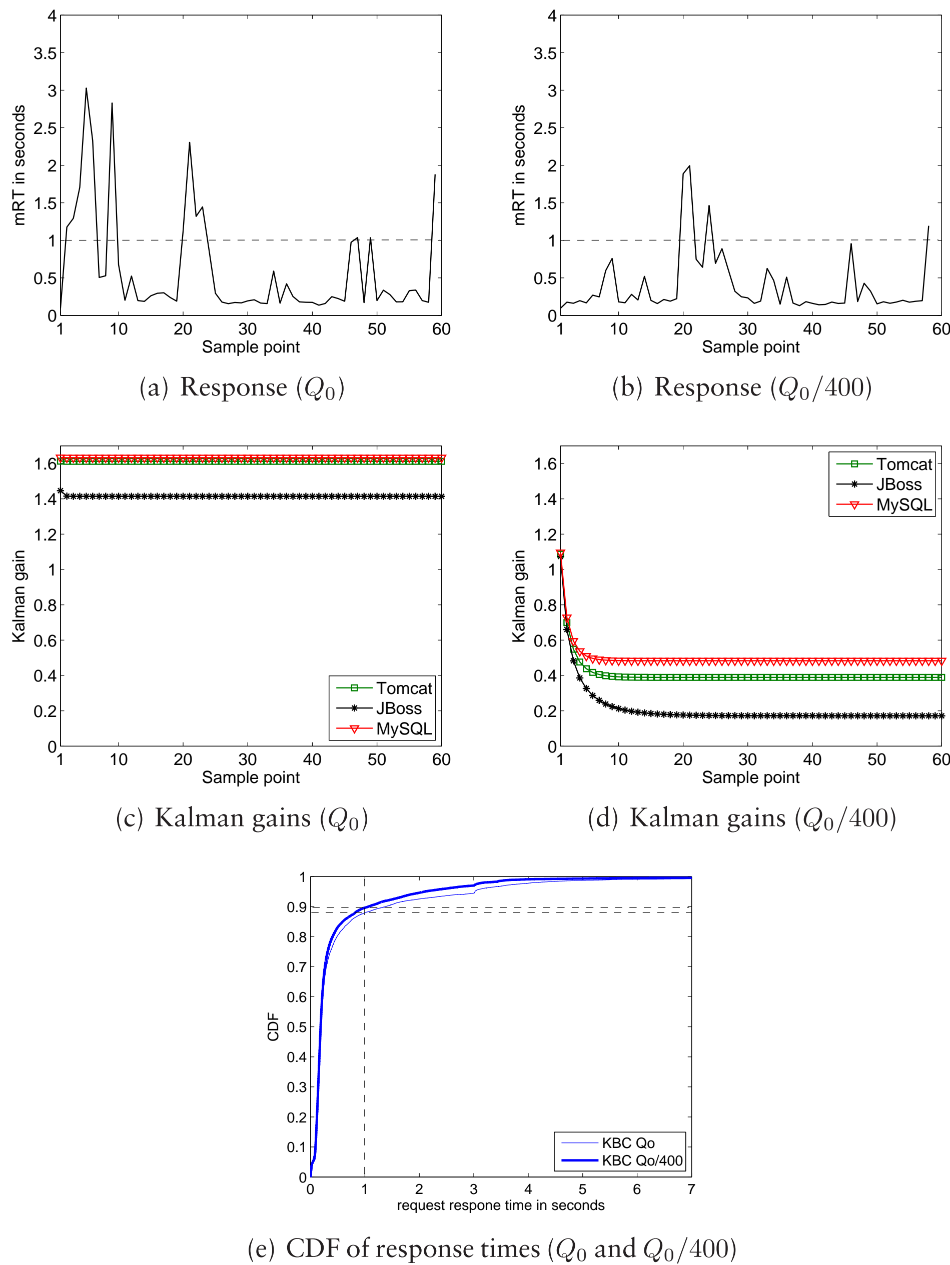


(e) MySQL CPU allocation, usage (Q_0)



(f) MySQL CPU allocation, usage ($Q_0/400$)

Figure 6.11: KBC Allocations for Variable Workload and two of Q Values. In the left hand-side graphs, the KBC controllers are configured with Q_0 values. In the right hand-side graphs the controllers are configured with $Q_0/400$ values, which makes the allocations to be less affected by small workload changes.



(a) Response (Q_0)

(b) Response ($Q_0/400$)

(c) Kalman gains (Q_0)

(d) Kalman gains ($Q_0/400$)

(e) CDF of response times (Q_0 and $Q_0/400$)

Figure 6.12: KBC Server Performance for Variable Workload and two Q Values. When the controllers are configured with Q_0 values (left hand-side graphs) their Kalman gains are higher than the KBC controllers when configured with $Q_0/400$ values (right hand-side graphs). With smaller gains the server's performance has fewer `mRT` spikes since the allocations are less affected by transient fluctuations.

beginning of the 1st experiment the controllers adjust the allocations to the utilisations in just one interval. This behaviour in combination with the fact that during the first few intervals the workload increases gradually, results in components' saturation and performance degradation. In this case for the duration of almost the first 10 intervals

(or approximately 50s) the MySQL component in the 1st experiment is saturated (Figure 6.11(e)) and the mRT stays above its reference value of 1s (Figure 6.12(a)). However, in the 2nd experiment for the same duration, the allocations follow more slowly the utilisations leaving the components with enough resources to adapt to the new increased workload and the mRT stays below its target value (Figure 6.12(b)). However, the slower allocation adaptation in the case of workload increases causes server degradation. For example, when the number of clients doubles around the 20th sample point, it causes the requests' mRT in the 2nd experiment to stay above their target value of 1s for more intervals (see mRT spikes between sample points 20 to 30 in 6.12(b)) than in the 1st experiment (mRT spikes between sample points 20 to 30 in 6.12(a)). Overall for these experiments as shown in Figure 6.12(e), there are more requests (89.6%) in the 2nd experiment with response times $\leq$ 1s than in the 1st (87.92%). Although smoother allocations are potentially more useful since they ignore transient resource fluctuations, they might cause server performance degradation when workload increases occur.

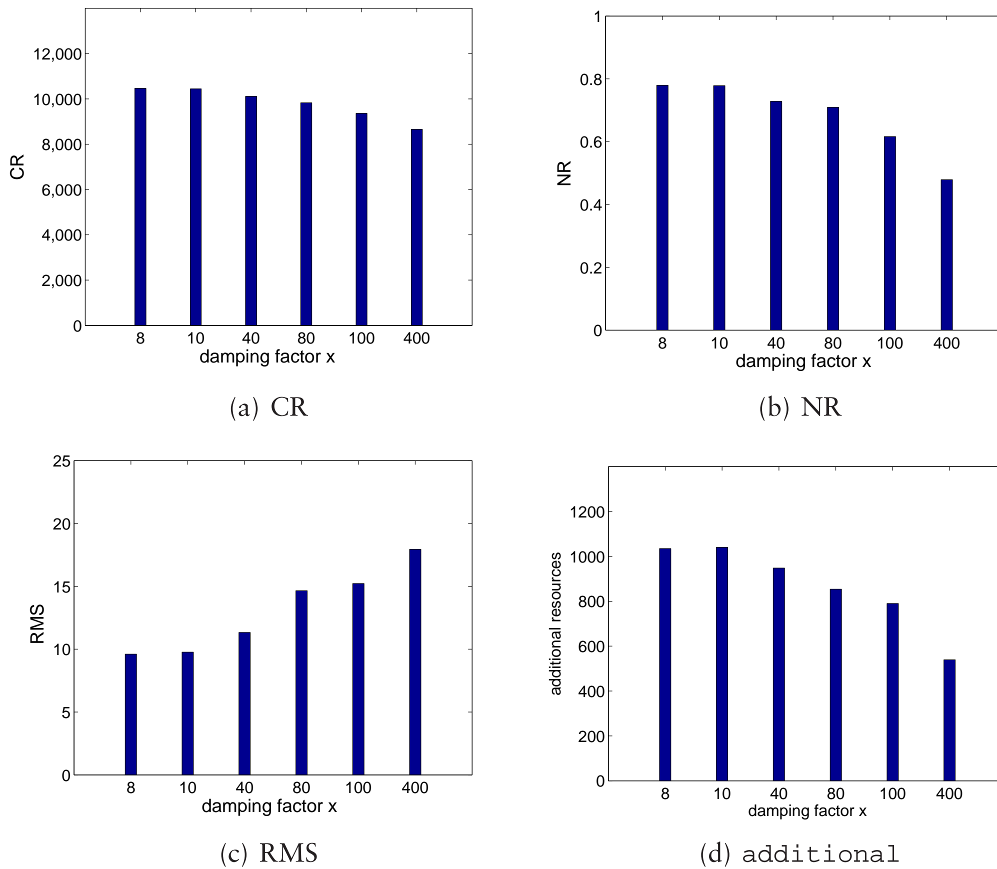
The effects of Q values on server's performance under workload increases is evaluated using E2 experiments repeated 20 to 40 times for each $x \in X$. Recall that during this experiment the number of clients quadruples (from 200 to 800) and hence the server is under a sharp workload increase. The server's performance is measured using the CR, NR, RMS, and additional metrics and results are shown in Figure 6.13. CI are very small and therefore not plotted.

As Q decreases (when the x damping factor increases) both CR (Figure 6.13(a)) and NR (Figure 6.13(b)) decrease too. This is because with decreasing Q, the Kalman gains decrease too and the KBC controllers become less confident in their predictions than the measured values and hence they are slower to adapt to the increasing resource demands, also shown by the decreasing additional resources in Figure 6.13(d). For the same reasons, the RMS (Figure 6.13(c)), which depicts the difference of request response times from the mean response time in the case of 800 clients, increases while Q decreases.

Properties

Three of the SASO properties: settling times, maximum overshoot, and zero steady-state error for the KBC controller are now examined using experimental results from this section. The analysis uses Figures 6.14(a) and 6.14(b) which are produced using the data from Figures 6.11(a) and 6.11(b) to illustrate allocations based on two Kalman gains which cause diverse controller behaviour. Each figure shows the measured allocation (line allocation) and the hypothetical allocation as calculated from the utilisation signal divided by the c parameter (line utilisation/c); the latter corresponds to the steady-state output of the controller according to the reference input. In this case, there is a reference output and the KBC controllers should converge to it because they are integral controllers and have zero steady-state error.

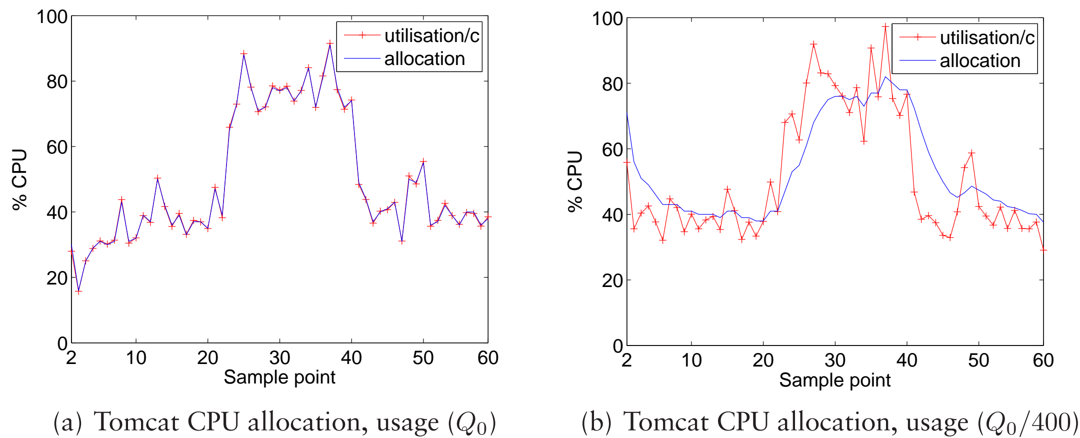
The settling times depend on the values of the Kalman gains. In Figure 6.14(a), where the Kalman gain is relatively large, the allocation signal is identical to the signal that corresponds to the steady-state values, therefore, the controller converges in just one interval. In Figure 6.14(a), where the Kalman gain is smaller, the controller takes a few



(a) CR (b) NR

(c) RMS (d) `additional`

Figure 6.13: KBC Performance for Workload Increases and Different Q Values. As Q decreases the server's performance drops since the KBC controllers are slower to adapt to workload demands.



(a) Tomcat CPU allocation, usage (Q_0) (b) Tomcat CPU allocation, usage ($Q_0/400$)

Figure 6.14: Settling Times and Overshoot for KBC Controllers.

intervals to converge to values very close to the steady-state, despite the noisy utilisations. For example, at the beginning of the experiment, the controller takes a few intervals (around 10) to reach to values very close to the utilisation/c signal despite the utilisation

fluctuations. Additionally, the KBC controllers do not overshoot. In both cases they approach the steady-state without exceeding its value (e.g. intervals 20-40).

Summary

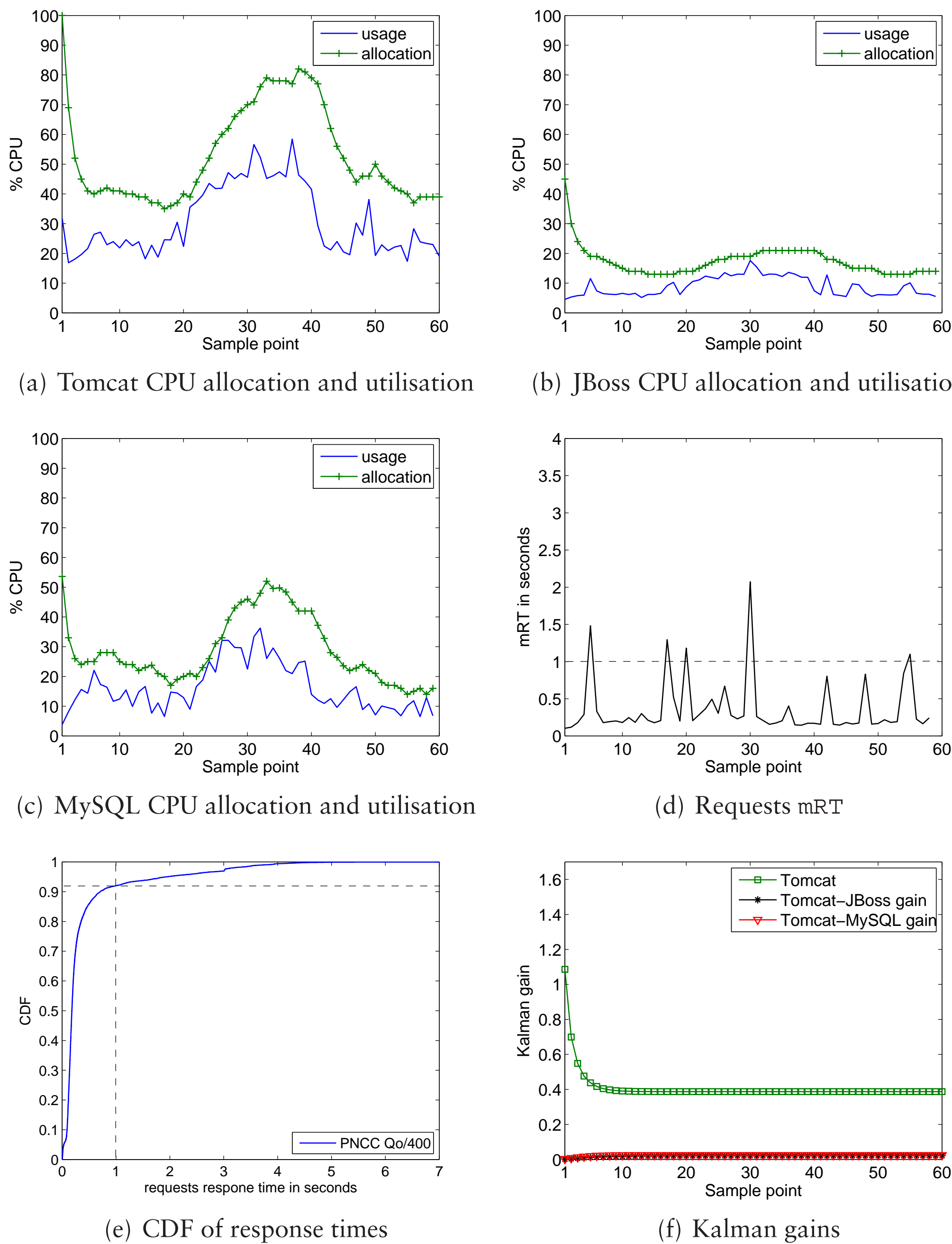
This section evaluated the performance of the SISO KBC controllers in stable and variable workload conditions. In all cases the controllers achieved their goals by following the resource fluctuations and maintaining the reference server performance. Depending on the values of the process noise variances Q, the controllers achieve different variations in the allocation and subsequently exhibit different server performance. As Q decreases the server's performance under workload increase decreases too. Depending on the applications, different damping factor x values can be chosen.

The slow responsiveness of the controller to workload changes is mainly the result of two factors. Firstly, when the controller is configured to make smooth allocations it responds slowly to sudden workload changes. Secondly, the KBC controller controls the allocations for each component separately, ignoring any resource coupling between them. It was shown previously (Section 6.2.3) that merging the SISO-UB controllers into the MIMO-UB (with the utilisation resource coupling between components) gives better server performance. The next section presents the results of the PNCC controller which comes from merging the KBC controllers into one MIMO controller and incorporating the resource correlation of the components.

6.3.2 PNCC

The MIMO PNCC controller combines the SISO KBC controllers to collectively allocate resources based on their resource utilisations. Components' utilisations are used in two ways. First, they guide the allocations through the use of the control errors. Second, they are used to compute the process noise variances and covariances. Similar to the computation of the allocation variances, the covariances between the components' allocations are computed offline based on the usage covariances (Figure 6.7). The utilisation variances are used to compute the allocation variances along the diagonal in the covariance matrix $\mathbf{Q}$. The utilisation covariances are used to compute the allocation covariances in the non-diagonal elements in $\mathbf{Q}$. This matrix is referred to as $\mathbf{Q_0}$.

The PNCC controller is now evaluated using an E0(20,40,60) experiment with workload fluctuations; results are shown in Figure 6.15. The process covariance $\mathbf{Q}$ matrix is set to the offline computed values $\mathbf{Q_0}$ divided by 400. This $\mathbf{Q}$ value enables direct comparisons with the 2nd E0(20,40,60) experiment with KBC controllers from Figures 6.11 and 6.12. The PNCC allocations adapt to the utilisation fluctuations (Figures 6.15(a), 6.15(b), and 6.15(c)). The server's performance is very close to the reference value since the `mRT` (Figure 6.15(d)) stays below 1s for most of the experiment's duration. When compared to the previous E0(20,40,60) experiment with the three KBC controllers, the PNCC which allocates resources collectively, adjusts to the increased demand faster than in the KBC case. This is shown by the fewer `mRT` spikes at the 20th interval comparing the PNCC



(a) Tomcat CPU allocation and utilisation

(b) JBoss CPU allocation and utilisation

(c) MySQL CPU allocation and utilisation

(d) Requests mRT

(e) CDF of response times

(f) Kalman gains

Figure 6.15: PNCC Performance for Variable Workload and $Q_0/400$ Values. The PNCC controller adapts to workload changes and the requests mRT is at very good levels. The PNCC controller considers the control error from all components by a small fraction as shown by the Kalman gains between components.

controller (Figure 6.15(d)) to the KBC (Figure 6.12(b)) and the increased proportion of requests with response times $\leq$ 1s (92.1% as shown in Figure 6.15(e)).

The PNCC's ability to adapt faster to workload changes than the KBC controllers comes from the utilisation resource covariance between components integrated into its design.

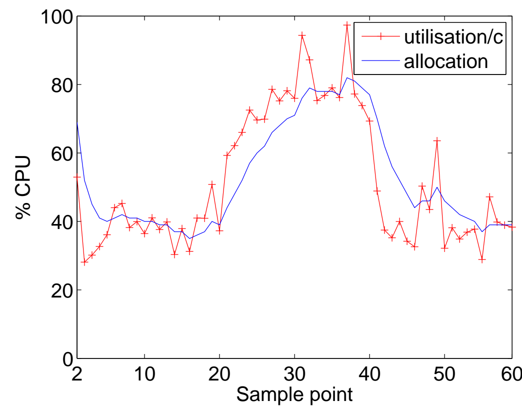


Figure 6.16: Settling Times and Overshoot for the PNCC Controller.

The PNCC controller uses the covariance matrix **Q** to account for the resource coupling between components. The final allocation of each component is the result of its own error plus the errors from the other components. The non-diagonal elements of the Kalman gain K matrix indicate these fractions. For instance, Figure 6.15(f) shows the Kalman gains for each of the three control errors accounted for the allocation of the Tomcat component. This figure shows that Tomcat's allocations are the result of its own error (multiplied by the Tomcat gain) in addition to the control errors from JBoss (multiplied by the Tomcat-JBoss gain) and MySQL (multiplied by the Tomcat-MySQL) components. Section 6.3.3 will compare the PNCC and KBC controllers for different values of the covariance **Q** matrices.

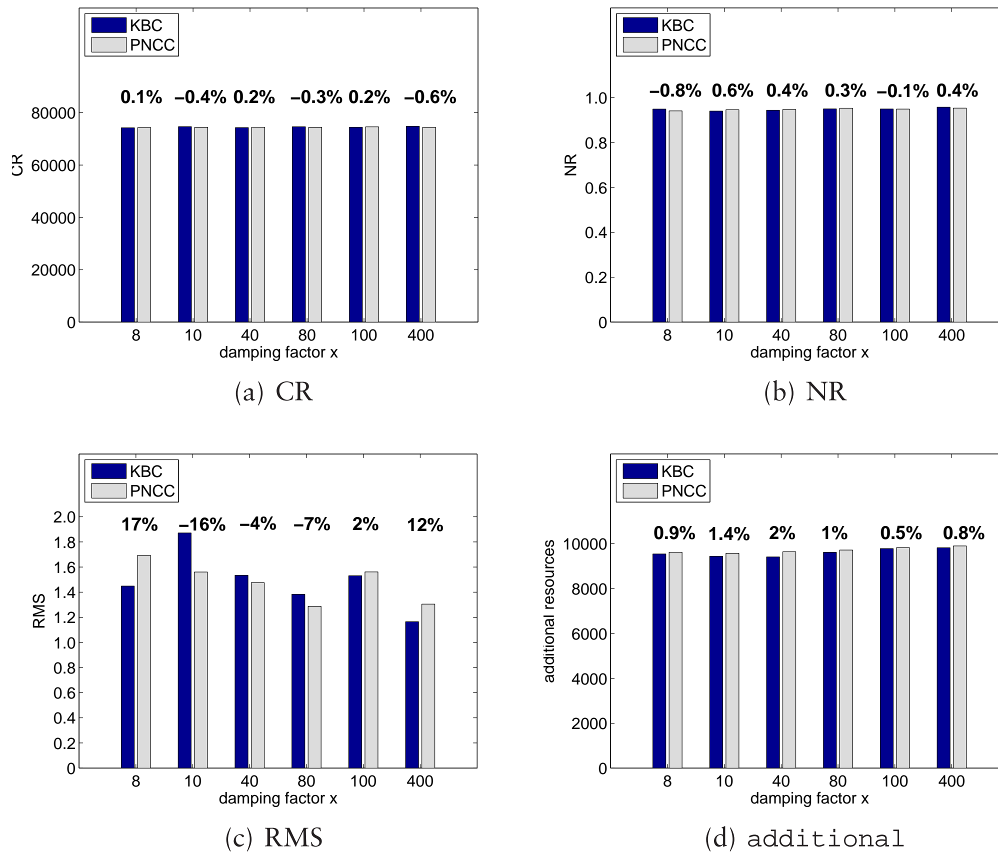
Properties

The following PNCC properties are now examined using Figure 6.16 (the data of this figure come from Figure 6.15(a)): settling times, maximum overshoot, and zero steady-state error. The Figure shows the measured allocation (line allocation) and the hypothetical allocation as calculated from the utilisation signal divided by the c parameter (line utilisation/c); the latter corresponds to the steady-state output of the controller according to the reference input. In this case, there is a reference input and the PNCC should converge to it because it is an integral controller and has zero steady-state error.

It has short settling times despite being configured for smooth allocations ($\mathbf{Q_o}/400$). As shown in the Figure (first 15 intervals), the PNCC approaches to steady-state values in just a few intervals. Finally, the PNCC does not overshoot. Its allocations do not exceed the steady-state values (e.g. intervals 20-40).

6.3.3 KBC and PNCC Comparison

This section compares the two Kalman based controllers, the KBC and the PNCC. The PNCC differs from the KBC controllers since it considers all components' control errors when computing the final allocation of each tier. Since the fraction of each control error



(a) CR (b) NR (c) RMS (d) `additional`

Figure 6.17: PNCC and KBC Comparison for E1(600,200) Experiments and Different x. Percentages shown in each case represent the percentage of the metric difference of the PNCC controller over the KBC.

considered to the components' allocations depends on the values of the covariance matrix $\mathbf{Q}$ the evaluation is performed for different $\mathbf{Q}$ values ($\mathbf{Q}$ is divided by $x \in X$). To better evaluate the two controllers the comparison is performed in two steps. First the controllers are compared under stable workloads and then using a series of E2 experiments that stress the controllers for sharp workload increases.

Comparison against stable workload conditions

Figure 6.17 shows the results from the KBC and PNCC comparison for a stable workload; for each x and controller an E1(600,200) experiment is performed. Results show that the two controllers have very similar performances: they allocate similar `additional` resources (Figure 6.17(d)); the server performance is similar in both cases of CR (Figure 6.17(a)) and NR (Figure 6.17(b)); and, there are small differences in the case of RMS (Figure 6.17(c)). In the case of stable workloads, the PNCC mechanism does not provide any significant improvement over the KBC controllers. The control errors and the coupling from the different components do not affect the final allocation for each component more than the KBC controllers.

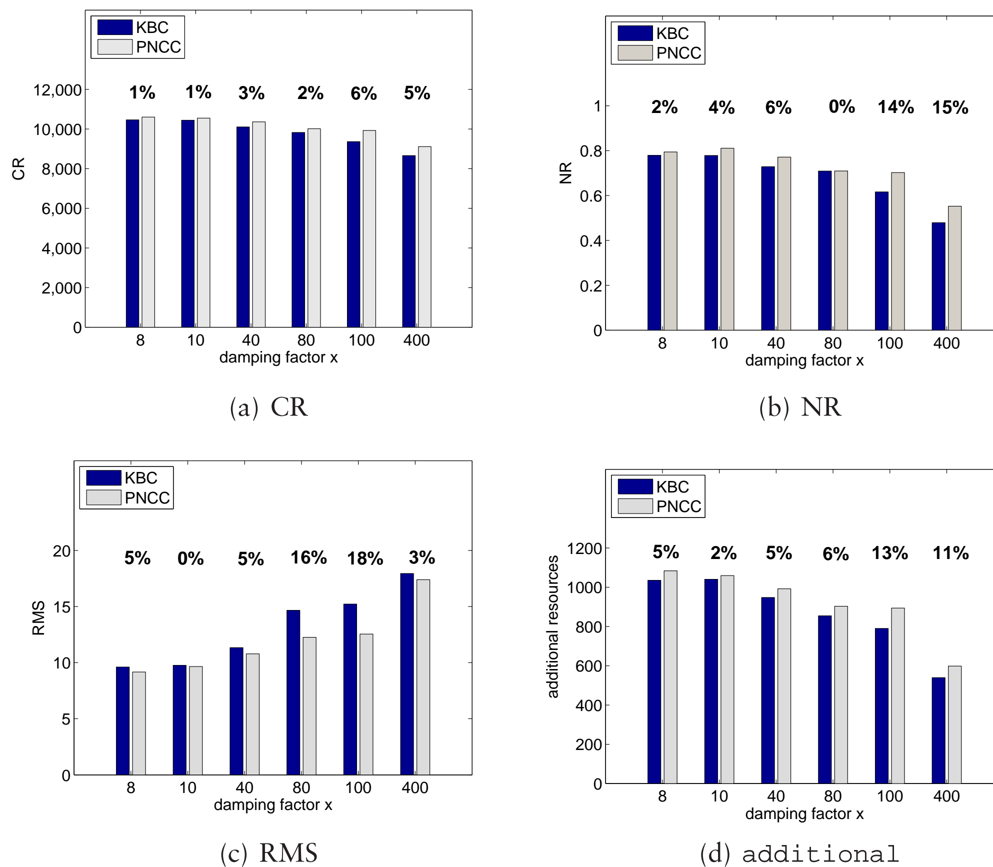


Figure 6.18: PNCC and KBC Comparison for E2 Experiments and Different x. Percentages shown in each case represent the absolute percentage of the metric difference of the PNCC controller over the KBC with a 90% CI.

Comparison against a large workload increase

The comparison is now focussed on sharp workload increases using an E2 experiment where 200 clients issue requests to the server for 60 intervals and at the 30th interval another 600 are added. Figure 6.18 shows the performance differences between the PNCC and the KBC as measured using the metrics NR, CR, RMS, and `additional` resources. In all figures the percentages shown in each case represent the absolute metric difference of the PNCC controller over the KBC with a 90% CI. To obtain this CI a T-test over all metrics across data sets in each case is applied. If after applying the T-test the means of the two data sets show no difference at the 90% CI, a 0% difference is shown (e.g. $\mathbf{Q} = \mathbf{Q_0}/10$, Figure 6.18(c)).

The PNCC controller has equal or better performance over the KBC controllers when looking at CR, NR, and RMS metrics combined for each x value. As the damping factor x values increase both controllers are becoming slower to adjust their allocations to the workload increase. However, the performance improvement of the PNCC over the KBC is increased as the x values increase. In these cases, where the small Kalman gains make the allocations slow to react to workload changes, the improvement of the

PNCC over the KBC is more apparent. The PNCC is able to react faster to important workload changes — increased `additional` resources (Figure 6.18(d)) — than the KBC controllers because it incorporates all components' errors.

Discussion

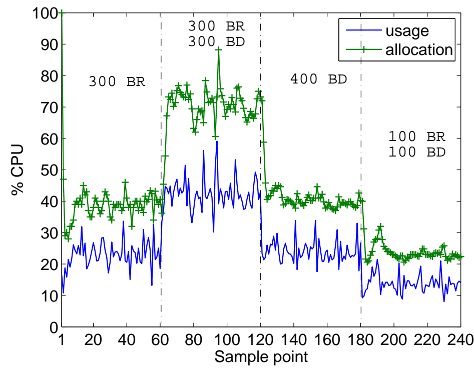
The section compared the KBC and the PNCC controllers. Both controllers make smooth allocations as the damping factor increases and, hence, the Kalman gains decrease. The PNCC provides better response to significant workload changes in all cases of the damping factor, although the two controllers perform similarly under stable workloads. The PNCC provides better response to workload changes because it considers the control errors from all components through the utilisation covariances. Under stable workloads, taking into account the control errors from the different components does not affect its performance when compared to the KBC. Overall, the PNCC controller provides a timely resource adaptation for significant workload changes. When the server operates under stable workloads the PNCC maintains similar server performance to the simpler KBCs.

6.3.4 APNCC

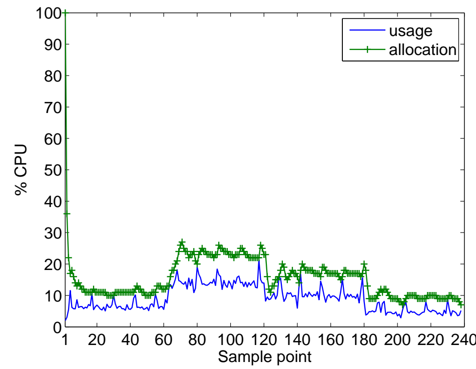
This section evaluates the Adaptive PNCC controller (APNCC). APNCC uses the adaptation mechanism that estimates the covariance matrix $\mathbf{Q}$ online from utilisation measurements and updates its values every few intervals (Section 5.3). The advantage of this controller over all other controllers is that it does not require offline parameter estimation and that the adaptation mechanism captures the system's dynamics as they happen.

The performance of the APNCC is evaluated using an experiment with fluctuating workloads coming from different mixes, the browsing (BR) and the bidding (BD) mix, as shown in Figure 6.19(a). The covariance matrix $\mathbf{Q}$ is updated using a sliding window mechanism which calculates the gains every controller interval using the measurements from the last 10 intervals and the controller always uses $\mathbf{Q}/40$ values. Results are shown in Figure 6.19; Figures 6.19(a), 6.19(b), and 6.19(c) show the controller's allocations and components' utilisations. The controller adjusts the allocations to track the utilisations of unknown and diverse workloads for the duration of the experiment. The server's performance (shown in Figures 6.19(d), 6.19(e), and 6.19(f)) is very close to its target performance. The `mRT` (Figure 6.19(d)) is below 1s for the duration of the experiment. There are a few prolonged spikes when the workload increases, but 89.58% of the requests have response times ≤ 1s (Figure 6.19(e)). The spikes in the response times (Figure 6.19(d)) are due to momentarily JBoss CPU increases caused by the Rubis benchmark (Figure 6.19(d)) and make the JBoss component to saturate. Finally, the `Throughput` changes according to the number of clients with some fluctuations (Figure 6.19(f)).

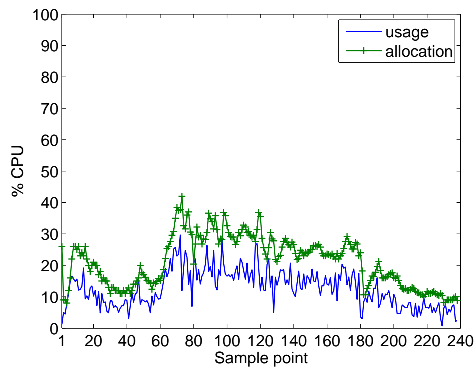
Figure 6.20 illustrates the Kalman gain values as computed throughout the experiment. In each case the gains adapt to the different utilisation fluctuations. The adaptation mechanism captures the workload changes and the controller's parameters are updated accordingly, e.g. the Tomcat gains increase around the interval 60 when the workload



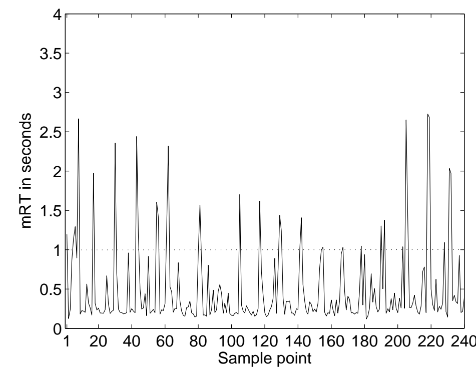
(a) Tomcat CPU allocation and utilisation



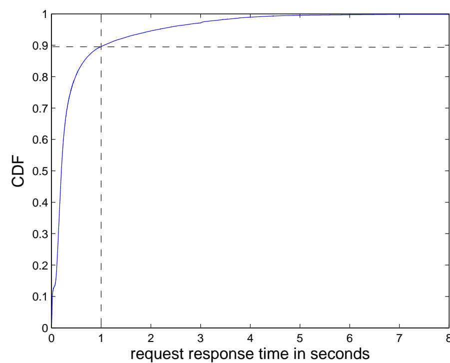
(b) JBoss CPU allocation and utilisation



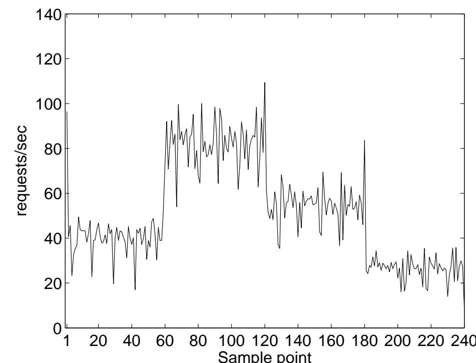
(c) MySQL CPU allocation and utilisation



(d) Requests mRT

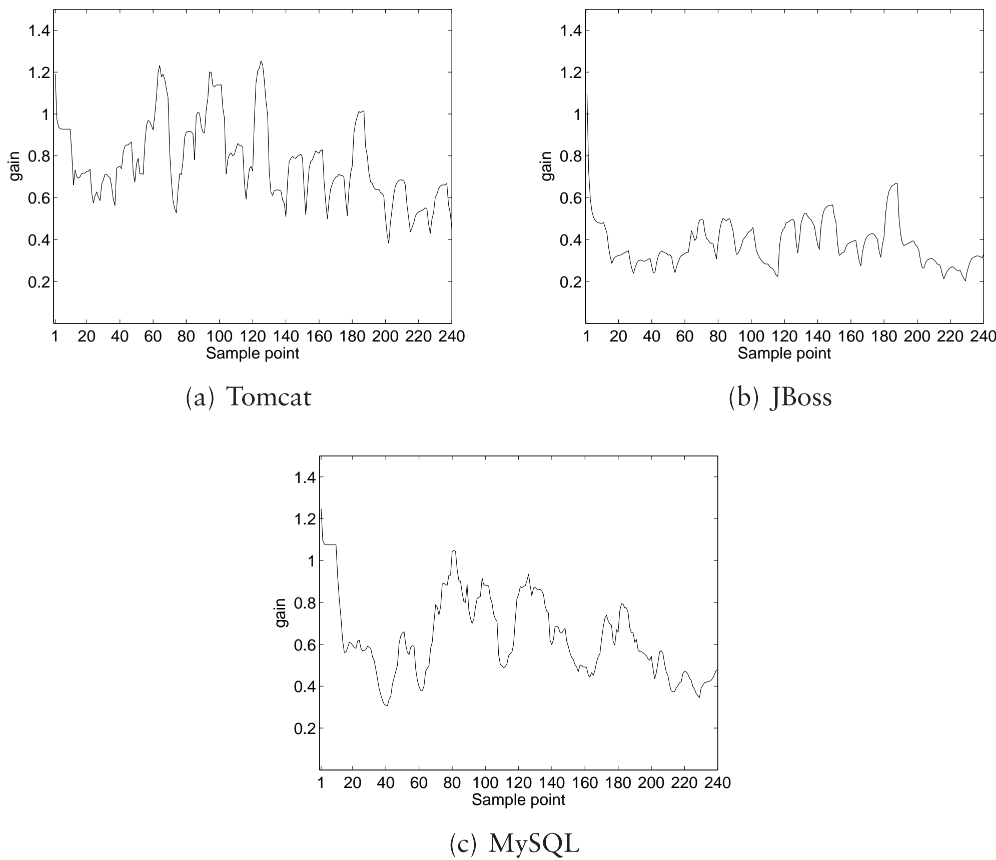


(e) CDF of request response times



(f) Throughput

Figure 6.19: APNCC Performance for Variable Workload and **Q**/40 Values. The APNCC controller adapts to diverse workload mixes (browsing (BR) and bidding (BD) mix) and fluctuating number of clients.



(a) Tomcat

(b) JBoss

(c) MySQL

Figure 6.20: APNCC Gains for Variable Workload and $\mathbf{Q}/40$ Values.

changes. While it is difficult for the offline parameter estimation to capture all the dynamics of the system, the APNCC controller follows all utilisation changes without *a priori* knowledge of the workload characteristics, namely the workload mixes and the number of clients.

6.3.5 PNCC and APNCC Comparison

To better evaluate the adaptive APNCC controller this section compares this controller against the PNCC in two situations. In the first case, the two controllers are evaluated using E0(40,80,120) experiments. These experiments simulate a relatively "normal" situation where the workload remains stable for several intervals and only two workload changes happen throughout. In the second case, the two controllers are evaluated using E2 experiments. In this case, the comparison is focussed around a large workload increase.

Comparison against "normal" workload conditions

The APNCC is now compared against the PNCC using E0(40,80,120) experiments for different x values. For each $x \in X$ and for each controller, an E0(40,80,120) is repeated

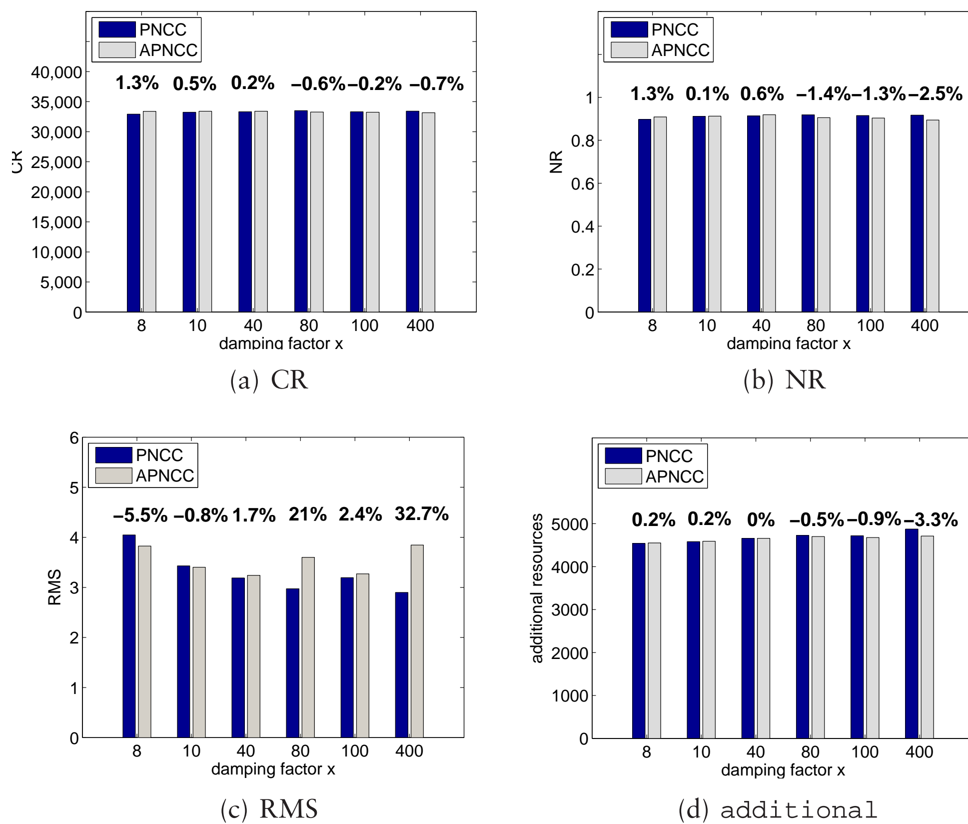


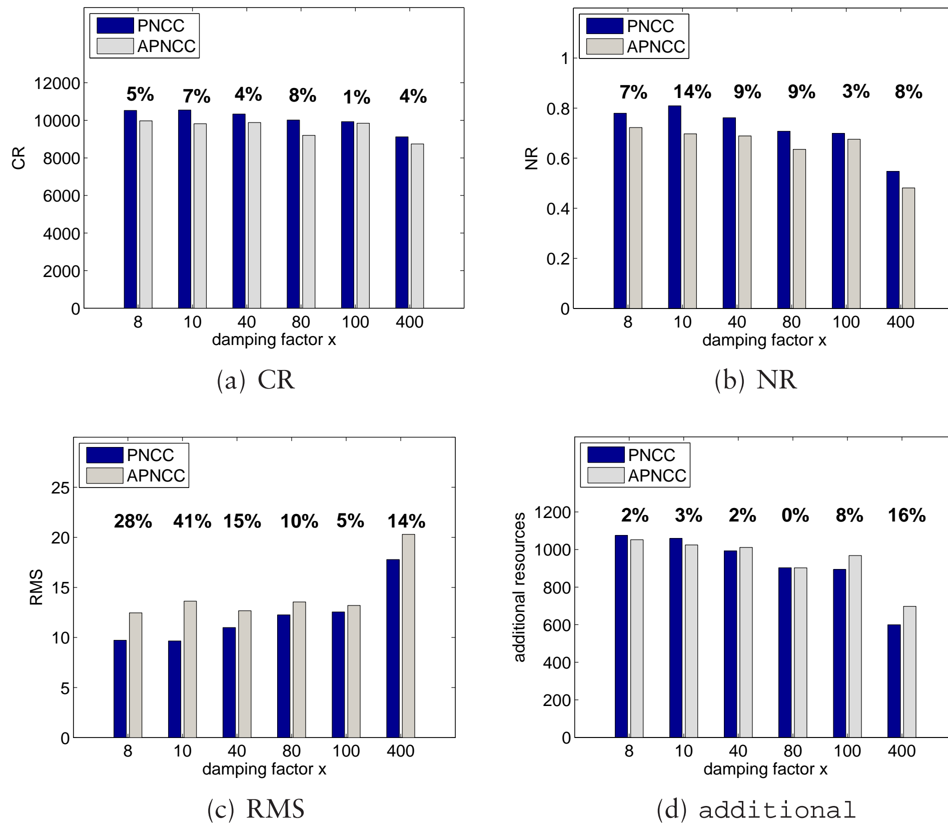
Figure 6.21: PNCC and APNCC Comparison for E0(40,80,120) Experiments. Percentages in each case show the metric difference of the APNCC controller over the PNCC with a 95% CI after a T-test is applied. A positive sign in the percentage indicates that the metric in the APNCC case has a higher value than in the case of the PNCC. A negative sign shows the opposite.

five times. The covariance matrix **Q** is estimated every 10 controller intervals from the utilisation measurements. Results are shown in Figure 6.21. According to all three metrics calculated for the duration of the experiment — CR in Figure 6.21(a), NR in Figure 6.21(b), RMS in Figure 6.21(c), and `additional` resources in Figure 6.21(d) — the two controllers perform almost the same.

The APNCC controller performs equally well to the PNCC with offline computed parameters while it eliminates the need for any offline computations, since its parameter **Q** is updated online.

Comparison against a large workload increase

The two controllers are also compared based on E2 experiments that simulate a large workload increase; the number of clients quadruples (from 200 to 800) half-way through the experiment. **Q** is updated based on a sliding window mechanism which calculates the gains every controller interval using the measurements from the last 10 intervals.



(a) CR (b) NR

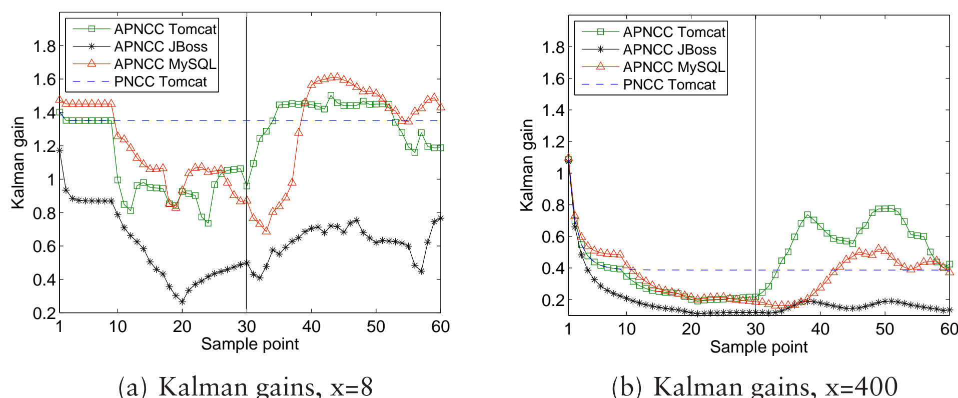
(c) RMS (d) `additional`

Figure 6.22: PNCC and APNCC Comparison for E2 experiments. Percentages in each case show the absolute metric difference of the APNCC controller over the PNCC with a 95% CI after a T-test is applied.

For every value $x \in X$ each experiment is performed 20 times. Results are shown in Figure 6.22. The APNCC controller performs almost as well as the PNCC as shown by the CR in Figure 6.22(a), NR in Figure 6.22(b), and RMS in Figure 6.22(c).

To better explain these last results Figure 6.23 depicts the APNCC Kalman gains as computed online in the case of two different x values. Figure 6.23(a) shows the gains for a small value of the damping factor $x = 8$, which makes their values relatively large. Figure 6.23(b) illustrates the gains for a larger value $x = 400$, which causes their values to decrease and therefore the allocations are slower to detect the utilisation changes. Both figures also show the offline computed gains in the case of the PNCC controller for the Tomcat component (dashed line). It is apparent that the absolute gain values are affected by the damping factor x. For example, for small x values the gains (Figure 6.23(a)) are larger than for large x values (Figure 6.23(b)). However, their behaviour with respect to detecting the workload increase is similar and discussed below.

Initially, the gain values are assigned to the offline computed ones and after the first ten intervals they are adapted for every sample point. At first, their values decrease from the offline computed ones which were calculated based on 600 clients and in this case the



(a) Kalman gains, x=8

(b) Kalman gains, x=400

Figure 6.23: APNCC and PNCC Kalman Gains for $x = 8$ and $x = 400$. Figures show the PNCC offline estimated gains only for the Tomcat component.

number of clients is 200 until the 30th sample point[6]. When the workload increase occurs at the 30th point, the adaptation mechanism detects the utilisation change and all gain values are adapted from there on. The absolute gain values increase beyond the values of the offline computed values as the clients have now increased to 800.

The PNCC controller in the E2 experiments is slightly better than the APNCC when comparing same x values (Figure 6.22). This is because at the time the workload change happens, the PNCC offline computed gain values are larger than the online APNCC estimated ones. This causes the PNCC to operate with a larger gain than the APNCC when the workload increases, at the 30th sample point. In this case, the PNCC controller reacts faster to the workload change for the next few intervals. However, although the APNCC gain is small at the time of the workload increase starts, the controller detects the change and adapts its gain according to the new utilisations. At the end, its performance is only slightly worse than the PNCC.

Now consider the hypothetical scenario where the offline computed gain values are smaller than the online computed ones before the workload increase occurs. For instance, this occurs when APNCC is configured for $x = 8$ and the PNCC for $x = 400$ (APNCC gains are shown in Figure 6.23(a) and the Tomcat PNCC gain is shown in Figure 6.23(b) with the dashed line). In this case the APNCC performs better than the PNCC as illustrated in Figure 6.22 (APNCC performance for $x = 8$ and PNCC for $x = 400$). In cases where the PNCC Kalman gains over-estimate the online APNCC computed ones, the PNCC performs better than the APNCC. However, if their values under-estimate the real ones, the APNCC is better than the PNCC. The APNCC adapts to any workload by calculating the process noise variance as it happens and, therefore, its gains correspond to the current workload variability.

Although the PNCC has an increased performance in all configurations, the APNCC allocates more resources for some values of x (Figure 6.22(d)). The APNCC allocates more `additional` resources because its Kalman gains for the Tomcat and JBoss components

[6]Figures show only the Tomcat offline computed gains, but similar observations hold for the other two components.

are higher than in the PNCC case after the workload increase (e.g. Figure 6.23). In addition, further analysis showed that the increased `additional` allocations are also caused by the relatively high values of the APNCC inter-component Kalman gains with respect to the Kalman gain in the Tomcat and JBoss cases. In other words, the errors from the other components significantly contribute to the final allocation for these two components. However, although the APNCC allocates more resources in total, the PNCC performs better. This is because the APNCC increased `additional` allocations come after a few intervals of the workload increase and, therefore, the PNCC has already served more requests.

Discussion

This section evaluated the adaptive APNCC controller and compared its performance with the PNCC which uses Kalman gains based on offline workload measurements. The APNCC is able to capture the utilisation variability across workloads in a server application and efficiently adapts its Kalman gains without using any a priori workload information as in the case of the PNCC.

The comparison between the two controllers showed that the APNCC performs as well as the PNCC under "normal" workload conditions where a roughly estimated PNCC gain is enough to capture the small workload variability (Figure 6.21). The APNCC is also able to adapt to substantial workload increases resulting in a consistent good behaviour (Figure 6.22). However, there are cases where the APNCC allocates more resources when compared to PNCC because it retrospectively detects and adapts to important workload changes. Nevertheless, it offers great workload adaptability potentials when the PNCC's performance depends on how close its offline estimated gains are to the current workload. In addition, the APNCC does not need any *special* mechanism to detect or predict a large workload increase. This controller, which treats all workload changes the same, automatically adapts its gain to values that depict the importance or not of the workload change. Finally, when tuning the APNCC controller, the x damping factor values do not seem to be that important when it comes to workload increases. One can essentially choose any x value and the controller would still adjust its gain to capture a substantial workload increase.

6.4 Discussion

This chapter thoroughly evaluated the performance of the five different controllers presented in this dissertation across different workload conditions and different evaluation metrics. Basic E0 type of experiments showed that each controller achieved its goals towards tracking the utilisations and maintaining the `mRT` close to the reference value (`mRT` $\leq$ 1s) for the majority of the intervals.

Additional E1 experiments evaluated the usage based controllers' performance with different parameter values. Analysis for the SISO-UB and MIMO-UB controllers showed that increasing any of their values improves the server's performance since more resources

are allocated to the components. It was also shown that the performance does not significantly improve after a certain amount of `additional` allocations. This is aligned with the system identification results, where it was pointed out that there is a threshold for the `extra` allocation beyond which the performance of the server stabilises. Based on the experimental analysis, guidelines were given to set the parameter values. When deploying a new application, further system experimentation similar to the one followed for evaluating the parameter values can be used to find those values that provide acceptable server performance based on its QoS performance goals and the policy on how to share resources on the virtualized cluster.

The MIMO controllers are designed to adjust the allocations faster to saturated components and the SISO-UB and MIMO-UB controllers were compared using E2 experiments that simulate a large workload increase. Analysis showed that the MIMO-UB controller outperformed the SISO-UBs for the same parameter values, at a cost of allocating more resources than the SISO-UBs (Figure 6.6). To better evaluate the MIMO-UB resource coupling mechanism another comparison based on E2 experiments was performed where the two controllers were particularly configured to make similar `additional` allocations. In this case, the MIMO-UB performed better during the workload increase, because it allocated more resources using the errors from all components (Table 6.7 and Table 6.8).

There are two issues concerning the usage based controllers: (a) they use a number of input configuration parameters and (b) their allocations are prone to every workload fluctuation. The Kalman based controllers address these issues. Based on experiments, it was illustrated that the COV of the allocations can be decreased when using a fraction (based on the damping factor) of the initial variances, in the case of the KBC, or covariance matrix, in the case of the PNCC. As the damping factor increases the COV decreases and the controllers are not affected by transient fluctuations. However, very smooth allocations were shown to be slow to respond to sudden large workload changes. In all cases, the MIMO PNCC controller, which takes into account the resource coupling between components, improved the performance of the KBC controllers during the workload change (Figure 6.18), although the two controller made similar `additional` allocation under stable workload conditions (Figure 6.17).

Although the KBC and the PNCC use fewer configuration parameters than the usage based controllers, their values are also based on offline system identification analysis. The APNCC uses a parameter adaptation mechanism to compute the parameter values online. Experiments showed that the APNCC adjusts the Kalman gains to different operating conditions. In fact it is able to detect large workload increases and different workload mixes and adapts its gain, even in cases where it is configured for smooth allocations (large damping factor) where it avoids transient utilisation fluctuations. When compared to the PNCC under large workload increases, its performance is only slightly worse since it detects and responds to the workload change retrospectively. Nevertheless, it updates and configures its Kalman gains to always reflect to the latest utilisation patterns.

The APNCC controller provides the best choice for multi-tier applications among controllers because: (a) it uses an online adaptation mechanism to estimate its parameters

and hence, reduces offline system identification analysis; (b) it captures workload variability as it happens, and (c) it can be configured to allocate with different degrees of variability depending on the virtualized applications and the way they share resources. In the case of a single-tier application and for similar reasons the KBC controller is the best choice. Its adaptive version is expected to perform equally well.

7 Related Work

This chapter presents work related to this dissertation. Section 7.1 discusses other control-based solutions to resource management in virtualized server applications and compares the proposed schemes to the controllers of this dissertation. Section 7.2 presents two filtering methods developed to estimate the CPU utilisation in shared non-virtualized clusters. The next section briefly discusses recent resource management based on machine learning techniques. Finally, Section 7.4 discusses resource management on the resource sharing platform of Grids.

7.1 Control-Based Resource Provisioning

Some related work leverages on the dynamic resource capabilities of modern virtual containers and exploits control theory to build robust resource provisioning controllers for highly stochastic server applications. This section reviews related work starting from schemes that control the allocation of single-tier server applications and continues to related work on provisioning of multi-tier applications.

7.1.1 Single-Tier Feedback Control

This section discusses papers published in 2005 and 2006 that control the CPU resources of a single-tier Apache server running on the HP-UX PRM resource container [LZSA05, WZS05, ZWS06]. Different controllers are discussed that aim to regulate QoS metrics such as mean response time `mRT` across overload (i.e. a tier has insufficient resources for

its current workload) and underload (i.e. a tier has more resources than needed). Liu *et al.* [LZSA05] present an adaptive Proportional-Integral (PI) controller which regulates the clients' `mRT` at desired levels by controlling the CPU allocation. The controller builds on a linear relationship between the inverse `mRT` and the CPU allocation, as shown by system identification measurements on the overload region. The controller performs well and adapts to different workload rates by dynamically updating its model's parameters and subsequently calculating its gain. This controller is based on a linear input-output relationship existing for the application under control and for only the overload region.

Wang *et al.* [WZS05] identify the bimodal behaviour of the `mRT` under overload and underload conditions and study the operation of three different controllers across all conditions. The first adaptive PI controller, also presented in [LZSA05], regulates the inverse `mRT` based on its linear relationship to the CPU allocation, which exists however only in the overload region. When stepping from the overload to the underload region this controller fails to adjust the allocations, as the inverse `mRT` is no longer linear to the CPU allocations. The second gain-adaptive Integral (I) nonlinear controller regulates the relative CPU utilisation to a reference value. Its adaptive gain operates so as, when in the overload region, the controller allocates resources more quickly to avoid performance degradation, while responding more slowly when the system steps into the underload region. This controller in general offers slow overall response time and good resource utilisation. However, this controller does not provide any QoS metric guarantees and, therefore, the authors propose a third adaptive controller that regulates the `mRT` based on an ARX model of both the inverse `mRT` and the relative utilisation. This controller aims to compensate for the lack of controllability of the inverse `mRT` in the underload region by using the relative utilisation. This controller offers better stability only when a clear boundary of the two operating regions can be detected.

Finally, in [ZWS06], a nested control design is presented. An inner control loop uses the gain-adaptive I controller from [WZS05] to regulate the relative utilisation. The outer loop implements another I controller which regulates the reference relative utilisation to a value so that the `mRT` is maintained within a user-specified range. The output of the outer loop controller is given as the input to the inner loop. The authors provide no justification for the outer loop controller design. When comparing to a single loop controller with either the relative utilisation controller or the `mRT` controller from [WZS05], this nested-loop controller meets more effectively the QoS guarantees and provides better resource utilisation.

The controllers of this dissertation do not provide fine-grain QoS output metric guarantees (which are mostly application specific). The `mRT` is used in the broad context of controller evaluation. They build on the usage output metric which is intuitively indicative of the application's resource requirements and they operate across all regions. The related work on `mRT`-based controllers are application and workload dependent. They are based on offline system identification analysis and although they provide reasonable performance to the systems under test, they offer no guarantees when applied to a different system.

The relative utilisation adaptive controller presented in [WZS05] is conceptually similar to the SISO-UB and the KBC controllers presented in this dissertation as they both track

the relative utilisation. However, the SISO-UB and the KBC controllers are linear and respond equally quickly (subject to tuning in the case of the Kalman filter) to the transitions between different operating conditions. This might be important in a server consolidation environment where resources are important to other running applications. The SISO-UB controller frees resources much more quickly than other controllers without needing to distinguish between overload and underload regions.

7.1.2 Multi-Tier Feedback Control

The control of multi-tier server applications has also recently gained attention. In [PSZ+07, WLZ+07, LZP+07], different controllers that regulate the relative utilisation of multi-tier Rubis server applications deployed on Xen VMs are studied and discussed below.

In [PSZ+07] the authors present a two-layered controller that regulates the relative utilisation of two instances of two-tier virtualized Rubis servers co-hosted on two physical servers. The first layer controller regulates the relative utilisation for each server tier. The second layer controller adjusts the allocations of the first layer according to a QoS differentiation metric in cases where the sum of the requested CPU allocations from the first-layer controllers are above the physical machines' capacity. Offline analysis identifies the same bimodal behaviour as in [WZS05], and so the same gain-adaptive I controller is used to control the allocations for each tier separately without considering any resource coupling between the components.

Wang *et al.* [WLZ+07] present a 3-layer nested control design to control the CPU allocations of a 3-tier Rubis application. The two inner loops are similar to [ZWS06]. The outer loop is used to provide with a better approximation of the corresponding utilisation per tier in respect to the reference `mRT`. It uses a model-based approach, presented in [SKZ07], which computes the `mRT` based on the utilisation per tier and the transaction mix. The outer loop uses the above model to find the reference utilisation per tier based on the reference `mRT` and the transaction mix as recorded at the client emulator during the last interval. This combination of controllers uses application specific information to compute the transaction mixes however, which might not be available to any application.

The MIMO-UB and the PNCC one-layer controllers use a simple model of the resource coupling between the components. It is either computed off-line for stable workloads or on-line for more general workloads as in the case of the APNCC and updated every some controller interval. The MIMO controllers solely depend on the easily available and non-intrusive relative utilisation per tier. Detailed transaction mixes are not always available at the server-side due to privacy issues and additional computational costs.

Finally, Liu *et al.* [LZP+07] present an optimal controller that computes the resource allocations of two 2-tier virtualized Rubis server applications providing QoS differentiation when the shared resources are overloaded. A first-order ARX model that captures the resource coupling across tiers of the same and different applications is used. Based on this model, in which parameters are estimated online during the last interval, the controller allocates resource by minimising a linear quadratic function of the normalised QoS

ratio for each application and the relative utilisation for each tier. The controller performs well and manages to keep the ratio of mean response times between applications close to its reference value. However, it does not provide any guarantees as to individual application performance.

This controller is similar to the MIMO-UB and PNCC controllers as they both use the tiers' resource coupling. However the controller in [LZP+07] has a limited scope of providing QoS differentiation of co-hosted applications only in the overload region where the utilisation equals the allocation and therefore, a resource coupling observation based on the allocations is present. The MIMO-UB and PNCC controllers, however, operate across all regions as they are based on the tiers' relative utilisation which is different to the allocation in the underload region. Finally, they provide a more general framework in which resource coupling is not attached to any QoS differentiation metric.

7.1.3 Predictive Control

In contrast to feedback control, where reactive actions are taking place to correct the measured error, there is also predictive control, where a controller proactively tunes the system to cope with future estimated load.

Xu *et al.* [XZSW06] present a predictive controller that regulates the relative utilisation of a single-tier virtualized server based on three time-series prediction algorithms, namely the AR auto-regressive model, the ANOVA decomposition, and the MP multipulse model. Any of the above methods predicts the utilisation for the next interval based on past measurements; this is used directly by the controller to set the allocation of the next interval. Results show that once the predictive model is properly trained either via on-line or off-line analysis, the predictive controllers react quickly to repeated CPU usage trends. However, predictive controllers perform poorly against newly seen trends. The predictive controllers outperform the relative-utilisation feedback controller of [WZS05] when the utilisation exhibits regular patterns that can be captured and accurately modelled by the prediction controller. However, *unseen* utilisation patterns cause the predictive controller to fail to provide adequate resources for the server application.

Time-series prediction is a powerful tool for resource provisioning if the system's parameters exhibit regular patterns. Either off-line or on-line training can be used to capture the system's dynamics and provide accurate predictions. However, feedback control is more general and does not depend on system patterns. The Kalman filter is a very powerful method that predicts future demands based on past measurements and directly relates measurements to system states. It is a one-step prediction method; the allocation for the next interval is the best estimate given all measurements so far. By tuning the different parameters, the controller achieves different degrees of responsiveness to sudden workload changes. Finally, when adaptive, it captures the changing dynamics of the system.

7.2 Filtering methods

Before commodity hardware virtualization became widely adopted, other systems were developed to manage resource multiplexing and server consolidation. Section 2.2.3 discussed these approaches. The next two paragraphs present their methods for resource estimation.

In Sharc [US04], an exponentially-weighted moving average (EWMA) filter is used to estimate future CPU and network bandwidth resources based on past observations. The Sharc system uses the estimates to redistribute resources among co-located applications. EWMA filters provide a flexible framework to estimate the values of noisy signals. Depending on the values of the filter's smoothing parameter, the filter can be set to work in a range of modes from being aggressively adaptive to changes of the observed signal (agile filter), or to being smooth on transient fluctuations (stable filter) and, therefore, slower to any changes. However, a filter with a statically assigned parameter like this works only in one mode and therefore it is not adaptive to different operating conditions.

To address the above limitations, Chase *et al.* [CAT+01] use a *flop-flip* filter, which is based on flip-flop filters [KN01]. The flop-flip filter uses the moving average of the estimations over a 30s window and, if that estimation fails outside one standard deviation then, it switches to the new moving average. The authors use this filter to smooth particularly bursty signals as they describe them: "*Even during steady load, this bursty signal varies by as much as 40% between samples.*". The adaptivity of the Kalman filter-based controllers in diverse workload conditions presented in this dissertation was demonstrated in two ways: (a) through the estimation of the process noise variance from measured values and (b) by dividing the noise variance by different values. Results showed a wide range of different operations of the Kalman filters that act either in an adaptive manner to all workload fluctuations or in a more stable way to transient changes while still being adaptive to notable workload changes.

Finally, simple time-series analysis techniques have also been used to predict future resource demands in modern virtualized shared clusters. A first order AR predictor is used by the Sandpiper system [WSVY07] to estimate future resource demands in virtualized servers.

7.3 Machine Learning in Resource Management

Machine learning approaches to performance modelling in virtualized environments have also been developed. Xu *et al.* [XZF+07] present a 2-layer resource management system that aims to minimise the resources consumed for single-tier applications to meet their SLAs, while maximising the profit of a utility function over the shared resources' revenue. They use fuzzy modelling to learn the relationship, modelled as a group of states, between workload and required resource usages to meet SLAs. Their models employ a black-box approach to the applications' internal structure, although in order to update the CPU resource allocations it requires information on the current workload. Controllers in this

dissertation do not consider the current workload but rather incrementally adjust the allocations based on the relative utilisation.

Tesauro *et al.* [TJDB07] apply Reinforcement Learning (RL) to data centre server allocation. The authors employ a 2-layer resource management scheme. For each application at the first layer, an application manager provides a utility curve of its expected value based on the number of allocated servers. At the second layer, a resource arbiter decides how to allocate servers among all applications so as to maximise some global utility function. RL is a trial-and-error method, where the system learns to make good decisions through a series of interactions with its environment. Tesauro *et al.*, use queueing models initially to bootstrap the resource management process as well as to train the system. The final model learnt by the arbiter outperformed the initial estimations based on the queueing model.

7.4 Resource Management in Grids

This last section briefly discusses resource management in the Grid environment.

In Grid Computing a set of heterogeneous machines distributed world-wide and interconnected through the Internet are sharing their resources to support advanced scientific applications. Similar to the virtualized cluster paradigm, Grid applications share a pool of resources distributed over a set of machines. In both cases, resource management of the applications is important.

The Grid middleware is a collection of software tools that manage the Grid, such as the very popular, open-source Globus Toolkit [glo08]. It handles all operations from low-level tasks such as remote control of the Grid machines to high-level functions like resource management. Grids use a Resource Management System (RMS) [KBM02] to (a) coordinate resource allocation between applications and Grid machines; (b) handle diverse policies and constraints imposed by the contributing peers; (c) manage scalability and heterogeneity issues, and so on. In particular in the case of the Globus toolkit, the Globus Resource Allocation Manager (GRAM) [CFK+98] accepts application requests for resources, discovers resources that meet the application demands, and allocates them. Application requests define the limits of the resources the application will need for execution such as minimum and maximum amount of memory [gra08].

This dissertation is concerned with applications with diverse and unpredictable workloads. Their resource demands fluctuate over time and hence adaptive resource allocations are key for the applications to continuously meet their QoS performance goals. In contrast, the applications deployed on Grids have pre-defined resource requirements and Grid resource managers are mostly concerned with the discovery and allocation of resources among the machines.

8 Conclusion

This dissertation is concerned with the resource management of virtualized server applications. It presented feedback controllers for adjusting the CPU resource allocations based on tracking the utilisations over previous intervals. The controllers maintain the allocations within a certain distance of the utilisations in spite of fluctuating workloads. In this way, the application maintained its performance very close to its goals. Finally, when considering the resource coupling between components, the controllers adjusted the allocations more quickly in response to workload changes.

8.1 Summary

Chapter 2 presented the motivation for the problem of resource management of virtualized server applications and the reasons for using a control-theoretic based approach to this problem. The chapter started by describing server applications in Section 2.1 and their demanding workload characteristics in Section 2.1.3. It then discussed how virtualization is changing the nature of data centres and explained why adaptive resource management is integral to creating a truly agile and efficient environment for hosting virtualized server applications (Section 2.3). Finally, Section 2.4 motivated the use of feedback control to create a reactive adaptive allocation scheme and introduced its basic concepts.

Chapter 3 presented all the necessary means for the realisation of this dissertation. It presented the prototype virtualized cluster used for the evaluation of this work (Section 3.2) and related issues such as the Xen virtualization technology (Section 3.5) and CPU resource management considerations (Section 3.6). It also presented the Rubis

benchmark server application (Section 3.4), which was deployed on the prototype cluster, and used throughout for experimental evaluation. Section 3.3 presented the architecture used to realise the resource management control framework.

The system identification process was described in Chapter 4. The application was subjected to a number of experiments that aimed to capture the behaviour of the system. First, components' utilisations were identified to be indicative of the application's CPU resource requirements and the pair (allocation, utilisation) was identified as the control input and control output signals (Section 4.3). Second, the system model was identified and results showed that for the application to maintain its reference performance goals, allocations should be sustained above a certain limit of the utilisations; this relationship was expressed using either the additive or the multiplicative model (Section 4.4). Finally, this chapter showed that there is a resource coupling between components' CPU utilisations (Section 4.5). The relationships between utilisations were expressed using linear models that provided a very good fit over a large collection of utilisation data across components and diverse workloads.

Chapter 5 formulated all five controllers of this dissertation. The two SISO controllers, the SISO-UB (Section 5.1.1) and the KBC (Section 5.1.4), adjust the CPU allocations for individual application components. The two MIMO controllers, the MIMO-UB (Section 5.2.1) and the PNCC (Section 5.2.2), allocate resources collectively to all components based on their resource coupling. The MIMO-UB controller uses the offline-derived resource coupling model between components, while the PNCC is based on the components' utilisation covariances. The KBC and the PNCC controllers incorporate the Kalman filtering technique into their design to track the components' utilisations and update the allocations. Section 5.3 presented a mechanism to dynamically update the estimated variances and covariances of the KBC and PNCC controllers and introduced the adaptive version of the PNCC controller, the APNCC.

Chapter 6 presented the experimental evaluation results for all five controllers. Each controller managed to maintain the application's performance very close to its reference value in spite of workload fluctuations. Extensive analysis also showed the effects of input configurations parameters on the controllers' allocation in the case of the SISO-UB (Section 6.2.1) and the MIMO-UB (Section 6.2.2) controllers. The Kalman based controllers were shown to make smooth allocations by changing the values of their configuration parameters (Section 6.3.1). Comparisons between controllers were also performed. The MIMO-UB controller outperformed the SISO-UB ones when their input configuration parameters were set to same values (Section 6.2.3). In addition, the MIMO-UB responded better to large workload increases even when the two usage-based controllers were configured to make similar allocations for same workloads. The MIMO PNCC controller also performed better than the KBC controllers when faced large workload increases (Section 6.3.3). This chapter also evaluated the APNCC controller (Section 6.3.4) which adjusts its Kalman gain online in response to workload variations. The APNCC controller was compared against the PNCC in Section 6.3.5. For small workload variations, its performance was comparable to the PNCC. For large workload increases the APNCC was able to automatically detect the change by adapting its gain accordingly, retaining relatively high performance levels.

Chapter 7 discussed other approaches to resource management similar to the context of this dissertation. It described work on resource management for virtualized servers using feedback control (Section 7.1). The main differences of the work presented in this dissertation and other related work is that it uses the Kalman filter to track the utilisation and it builds MIMO controllers based on the resource coupling between components which work in any operating region. In addition, this chapter presented resource estimation using simple filters (Section 7.2), related work on server performance modelling based on machine learning (Section 7.3), and resource management on Grids (Section 7.4).

8.2 Future Work

Resource management of virtualized servers has become a prominent area of work with increasing interest from the research and industrial community. This dissertation tackled the problem of CPU resource provisioning for virtualized server applications and provided several efficient solutions. During the course of this work, the author identified extensions of the current dissertation and future directions of research as outlined below.

Enhanced Evaluation

The evaluation of the controllers used the Rubis benchmark application deployed on three machines. Each Rubis tier was deployed on a separate machine. Each machine was dual core, and each of `dom0` and `domU` were pinned on one of the two CPUs. This setup enabled an extensive system identification and experimental evaluation for the controllers without implications from other co-hosted applications and scheduling artifacts. To enable deployment of the controllers in an industrial data centre, a more detailed evaluation is required. The current work can be further evaluated under a server consolidation example scenario where multiple benchmark applications (e.g. multiple copies of Rubis or a mixture of server applications) are hosted by the various machines in the cluster.

Enhanced Functionality

The current work can be further extended to provide a more general solution to the resource management of virtualized server applications. The system model could be extended to incorporate `dom0`'s utilisations and the controllers should then manage the CPU allocations accordingly for `dom0` as well; Cherkasova *et al.* [CG05] have performed some initial evaluation on the `dom0` CPU overhead for I/O and disk processing. In addition, the controllers could be expanded to provision resources for memory space, network and disk bandwidth in addition to CPU. Both enhancements would provide data centre administrators more complete solutions to manage their applications' resources.

Solving an Optimisation Problem for `extra` Resources

The amount of `extra` resources that allocations should maintain above the mean utilisations play an important role in the management of resources in virtualized server machines and in the application's performance. All controllers in this dissertation maintain this difference at reference values as indicated by their input parameters, which are set based on either system identification or empirical analysis. However, setting these parameters to appropriate values is challenging. On one hand, if `extra` resources are less than required, the application‘ performance degrades. On the other hand if more resources than needed are given to components, machine resources are underutilised. In fact, there is a minimum amount of `extra` resources above which the application performance stabilised as shown in the simple case where the allocations of only one component was subjected to varying resources (Figure 4.4).

An interesting direction of research is to formulate the allocation problem as an optimisation one. In this case, a controller would have to find the *minimum* amount of resources required by components to maintain a reference application performance. This problem becomes particularly challenging under fluctuating workloads.

High Level Application Management Tasks

It was mentioned before that CPU resource provisioning for applications is an integral part of any high-level data centre operation, such as load balancing, power saving, etc. The controllers of this dissertation can be used in conjunction with other controllers for advanced data centre management.

For instance, consider the demanding job of handling hundreds of multi-tier applications in a large virtualized data centre consisting of tens or hundreds of machines. Assume that the administrator aims to place applications on as few machines as possible and switch off the rest to save on power.

To tackle this problem one can use two levels of controllers. At the bottom level, the resource management of individual applications is handled by any of the controllers of this dissertation as chosen using the criteria mentioned throughout. At the high level, another set of controllers can be built to handle application placement on physical machines based on their resource requirements. The two controllers can operate at different granularities. The controllers of the bottom level operate at short intervals and provide resources for small scale fluctuations. The controller(s) at the high-level operate when workload fluctuations cause such changes in applications' demands that component migration is necessary to ensure maximum power saving and/or maintain QoS.

8.3 Conclusions

This dissertation makes the following three main conclusions:

1. Reactive resource provisioning for virtualized server applications is feasible using feedback control.

This dissertation proposed and evaluated five novel feedback controllers — namely the SISO-UB, the MIMO-UB, the KBC, the PNCC, and the APNCC — for CPU resource provisioning. Thorough evaluation showed that each of these controllers allocated CPU resources to application components while maintaining the server's performance very close to its reference performance in the presence of fluctuating and unknown workloads.

2. Resource provisioning based on the resource coupling found in multi-tier applications is more efficient than its per single-tier allocation counterpart.

Modern virtualized applications employ the multi-tier model and span multiple components deployed on separate VMs. This dissertation identified and modelled the utilisation resource coupling between application components. Evaluation showed that the MIMO controllers, which consider this coupling in making their decisions regarding resource allocation to all application components improve the performance of the SISO controllers when facing large workload increases.

3. The Kalman filtering technique integrates efficiently into resource allocation feedback controllers.

Three of the controllers presented here — namely the KBC, the PNCC and the APNCC — use the Kalman filtering technique to track utilisation changes based on which they allocate CPU resources. The Kalman filter estimates the next state of a system based on past observations. The key point of this technique which makes it attractive for resource management is that it uses the dynamics of the system itself to estimate the validity of the next observation and choose whether it is an important change to follow or just a small variation. The adaptive APNCC controller encompasses exactly this behaviour. Experimental evaluation showed that the APNCC dynamically adapts to workload conditions and distinguishes between small variations and important workload changes.

A
Steady-State Kalman Gain

The Kalman gain depends on the ratio $\frac{S}{Q}$. According to [Sim06, section 5.4.2] the steady-state Kalman gain is given by:

$$K_k = \frac{P_k^- H_k^T}{H_k P_k^- H_k^T + S_k}, \tag{A.1}$$

P_k^- is the steady-state covariance. The steady-state covariance for a scalar, time-invariant Kalman filter is given by:

$$\lim_{k \to \infty} P_k^- = \frac{\tau_1}{2H^2}, \tag{A.2}$$

where τ_1 is:

$$\tau_1 = \sqrt{H^2 Q + S(F+1)^2}\sqrt{H^2 Q + S(F-1)^2}, \tag{A.3}$$

where H represents the transition between the states and the measurements; F represents the transition between states in the absence of noise; Q is the process noise variance; and, S the measurement noise variance.

In the case of the KBC controllers: $H = c$, $F = 1$, and Q and S are the process noise and measurement variance respectively and are measured offline. From Equations (A.3), (A.2), and (A.1) and using the parameter values for the current system, the steady-state

Kalman gain for the KBC controllers is given by:

$$K_k = \frac{c + \sqrt{c^2 + 4\frac{S}{Q}}}{c^2 + c\sqrt{c^2 + 4\frac{S}{Q}} + 2\frac{S}{Q}} \tag{A.4}$$

Bibliography

[abe07a] Business Continuity: The Next Phase in Server and Storage Virtualization (Research Brief). Aberdeen Group, July 2007.

[abe07b] Vendors Invest to Simplify Virtualization: Will Small/Medium Businesses (SMBs) Bite? Aberdeen Group, July 2007.

[ACC+02] Cristiana Amza, Anupam Chandra, Alan L. Cox, Sameh Elnikety, Romer Gil, Karthick Rajamani, Willy Zwaenepoel, Emmanuel Cecchet, and Julie Marguerite. Specification and Implementation of Dynamic Web Site Benchmarks. In *Proceedings of the 5th Annual IEEE International Workshop on Workload Characterization (WWC-5)*, pages 3–13, 2002.

[ACZ03] Cristiana Amza, Alan L. Cox, and Willy Zwaenepoel. Conflict-Aware Scheduling for Dynamic Content Applications. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[ADZ00] Mohit Aron, Peter Druschel, and Willy Zwaenepoel. Cluster Reserves: a Mechanism for Resource Management in Cluster-Based Network Servers. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 90–101, 2000.

[AFF+01] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D.P. Pazel, J. Pershing, and B. Rochwerger. Océano - SLA Based Management of a Computing Utility. In *Proceedings of the 2001 IEEE/IFIP International Symposium on Integrated Network Management*, pages 855–868, 2001.

[AJ00] Martin Arlitt and Tai Jin. A Workload Characterization Study of the 1998 World Cup Web Site. *IEEE Network*, 14(3):30–37, May/June 2000.

[ASDZ00] Mohit Aron, Darren Sanders, Peter Druschel, and Willy Zwaenepoel. Scalable Content-aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the Annual USENIX Technical Conference (USENIX)*, pages 323–336, 2000.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 164–177, 2003.

[BDM99] Gaurav Banga, Peter Druschel, and Jeffrey C. Mogul. Resource Containers: A New Facility for Resource Management in Server Systems. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 45–58, 1999.

[BVW+02] Andy Bavier, Thiemo Voigt, Mike Wawrzoniak, Larry Peterson, and Per Gunningberg. SILK: Scout Paths in the Linux Kernel. Technical Report 2002-009, Department of Information Technology, Uppsala University, Uppsala Sweden, 2002.

[CAK+04] Mike Y. Chen, Anthony Accardi, Emre Kiciman, , Dave A. Patterson, Armando Fox, and Eric A. Brewer. Path-Based Failure and Evolution Management. In *Proceedings of the 1st USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04)*, pages 309–322, 2004.

[CAT+01] Jeffrey S. Chase, Darrell C. Anderson, Prachi N. Thakar, Amin Vahdat, and Ronald P. Doyle. Managing Energy and Server Resources in Hosting Centres. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 103–116, 2001.

[CCE+02] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. A Comparison of Software Architectures for E-business Applications. Technical Report TR02-389, Rice University Computer Science, 2002.

[CCE+03] Emmanuel Cecchet, Anupam Chanda, Sameh Elnikety, Julie Marguerite, and Willy Zwaenepoel. Performance Comparison of Middleware Architectures for Generating Dynamic Web Content. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware)*, pages 16–20, June 2003.

[CFK+98] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, 1998.

[CG05] Ludmila Cherkasova and Rob Gardner. Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor. In *Proceedings of the Annual USENIX Technical Conference (USENIX)*, pages 387–390, 2005.

[CLM+08] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High Availability via Asynchronous Virtual Machine Replication. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 161–174, 2008.

[CMH08] Lancelot Castillo, Walter Montes, and Stephen Hochstetler. Virtualizing an Infrastructure with System p and Linux. `http://www.redbooks.ibm.com/`, 2008.

[CMZ02] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepoel. Perform-

ance and Scalability of EJB Applications. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 246–261, 2002.

[CRS99] Ariel Cohen, Sampath Rangarajan, and Hamilton Slye. On the Performance of TCP Splicing for URL-Aware Redirection. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 117–126, 1999.

[ENTZ04] Sameh Elnikety, Erich Nahum, John Tracey, and Willy Zwaenepoel. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. In *Proceedings of the International Conference on World Wide Web (WWW)*, pages 276–286, 2004.

[esx08] VMware ESX Server. `http://www.vmware.com/products/vi/esx/`, 2008.

[glo08] Globus Home Page. `http://www.globus.org/`, 2008.

[gra08] GRAM Job Manager Reference Manual, RSL Attributes. `http://www.globus.org/api/c-globus-4.2.0/globus_gram_job_manager/html/globus_job_manager_rsl.html`, 2008.

[GS06a] Frank E. Gillett and Galen Schreck. Pragmatic Approaches To Server Virtualization. Forrester Research, June 2006.

[GS06b] Frank E. Gillett and Galen Schreck. Server Virtualization Goes Mainstream. Forrester Research, February 2006.

[GVC96] Pawan Goyal, Harrick M. Vin, and Haichen Cheng. Start-time Fair Queuing: A Scheduling Algorithm for Integrated Services Packet Switching Networks. In *Proceedings of ACM SIGCOMM'96*, pages 157–168, 1996.

[HDPT04] Joseph L. Hellerstein, Yixin Diao, Sujay Parekh, and Dawin M. Tilbury. *Feedback Control of Computing Systems*. IEEE Press/Wiley Interscience, 2004.

[HFC+06] Alex Ho, Michael Fetterman, Christopher Clark, Andrew Warfield, and Steven Hand. Practical Taint-based Protection Using Demand Emulation. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 29–41, 2006.

[HH05] Alex Ho and Steven Hand. On the Design of a Pervasive Debugger. In *Proceedings of the 6th Int. Symposium on Automated Analysis-Driven Debugging (AADEBUG'05)*, pages 117–122, 2005.

[HJ85] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.

[hpW08a] HP_UX Workload Manager overview. `http://h71036.www7.hp.com/hpux11i/downloads/wlm.overview.pdf`, 2008.

[hpW08b] HP_UX Workload Manager (WLM) Home Page. http://h71036.www7.hp.com/hpux11i/cache/328328-0-0-0-121.html, 2008.

[hpW08c] Using HP_UX Workload Manager: A quick reference. http://h71036.www7.hp.com/hpux11i/downloads/wlm.quickref.pdf, 2008.

[HSH05] Alex Ho, Steven Smith, and Steven Hand. On Deadlock, Livelock, and Forward Progress. Technical Report UCAM-CL-TR-633, University of Cambridge, Computer Laboratory, 2005.

[ICDD00] Arun Iyengar, Jim Challenger, Daniel Dias, and Paul Dantzig. High-Performance Web Site Design Techniques. *IEEE Internet Computing*, 4(2):17–26, Mar/Apr 2000.

[IDC07] Virtualization and Multicore Innovations Disrupt the Worldwide Server Market. IDC, March 2007.

[jbo08] JBoss, Java Middleware Platform. http://www.jboss.org/, 2008.

[JLDJSB95] Michael B. Jones, Paul J. Leach, Richard P. Draves, and III Joseph S. Barrera. Modular Real-Time Resource Management in the Rialto Operating System. In *Proceedings of the 5th Workshop on Hot Topics in Operating Systems (HotOS-V)*, pages 12–17, 1995.

[jon08] JOnAS, Java Open Application Server. http://jonas.objectweb.org/, 2008.

[JRR97] Michael B. Jones, Daniela Rosu, and Marcel-Catalin Rosu. CPU Reservations and Time Constraints: Efficient, Predictable Scheduling of Independent Activities. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, pages 198–211, 1997.

[Kal60] Rudolph E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transaction of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[KBM02] Klaus Kraute, Rajkumar Buyya, and Muthucumaru Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software – Practice & Experience*, 32(22):135–164, 2002.

[KMN04] Abhinav Kamra, Vishal Misra, and Erich M. Nahum. Yaksha: A Self-Tuning Controller for Managing the Performance of 3-Tiered Web Sites. In *Proceedings of the International Workshop on Quality of Service (IWQoS)*, pages 47–56, 2004.

[KN01] Minkyong Kim and Brian Noble. Mobile Network Estimation. In *Proceedings of the ACM Annual Conference on Mobile Computing and Networking (MobiCom)*, pages 298–309, 2001.

[LDV07] Cherkasova Ludmila, Gupta Diwaker, and Amin Vahdat. When Virtual is Harder than Real: Resource Allocation Challenges in Virtual Machine Based IT Environments. Technical Report HPL-2007-25, HP Laboratories, 2007.

[Lju87] Lennart Ljung. *System Identification: Theory for the User*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[LMB+96] Ian M. Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul T. Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1280–1297, 1996.

[LZP+07] Xue Liu, Xiaoyun Zhu, Pradeep Padala, Zhikui Wang, and Sharad Singhal. Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform. In *Proceedings of the IEEE Conference on Decision and Control*, pages 3792–3799, 2007.

[LZSA05] Xue Liu, Xiaoyun Zhu, Sharad Singhal, and Martin Arlitt. Adaptive Entitlement Control of Resource Containers on Shared Servers. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management*, pages 163–176, 2005.

[MA98] Daniel A. Menascé and Virgilio A.F. Almeida. *Capacity Planning for Web Performance: Metrics, Models, & Methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[MAT] MATLAB Curve Fitting Toolbox. `http://www.mathworks.com/products/curvefitting`.

[May79] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 141 of *Mathematics in Science and Engineering*. New York: Academic Press, INC., 1979.

[MIG+02] Justin Moore, David Irwin, Laura Grit, Sara Sprenkle, and Jeff Chase. Managing Mixed-Use Clusters with Cluster-on-Demand. Technical Report CS-2002-07, Duke University, 2002.

[MP96] David Mosberger and Larry L. Peterson. Making Paths Explicit in the Scout Operating System. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 153–167, 1996.

[MS70] Richard A. Meyer and Love H. Seawright. A Virtual Machine Time-Sharing System. *IBM Systems Journal*, 9(3):199–218, 1970.

[mys08] MySQL, Open Source Database. `http://www.mysql.com/`, 2008.

[net08] Netcracft, March 2008 Web Server Survey. `http://news.netcraft.com/archives/web_server_survey.html`, 2008.

[Oga90] Katsuhiko Ogata. *Modern Control Engineering*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990.

[ora08] Oracle Database. `http://www.oracle.com/database/index.html`, 2008.

[PAB+98] Vivek S. Pai, Mohit Aron, Gaurav Banga, Michael Svendsen, Peter Druschel, Willy Zwaenepoel, and Erich M. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems Conference (ASPLOS)*, pages 205–216, 1998.

[PSZ+07] Pradeep Padala, Kang Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 289–302, 2007.

[rac07] Rackspace Virtualization Survey. Rackspace, August 2007.

[RI00] John Scott Robin and Cynthia E. Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. In *Proceedings of the 9th USENIX Security Symposium*, pages 129–144, 2000.

[SE07] Jed Scaramella and Matthew Eastwood. Solutions for the Datacenter's Thermal Challenges. White Paper, 2007.

[ser08] Server Sprawl Definition from SearchDataCenter.com. `http://searchdatacenter.techtarget.com/sDefinition/0,,sid80_gci1070280,00.html`, 2008.

[SGLB99] Joris De Schutter, Jan De Geeter, Tine Lefebvre, and Herman Bruyninckx. Kalman Filters: A Tutorial. *Journal A*, 40(4):52–59, December 1999.

[Sim06] Dan Simon. *Optimal State Estimation*. John Wiley & Sons, Inc., 2006.

[SKZ07] Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting Nonstationarity for Performance Prediction. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, pages 31–44, 2007.

[SS05] Christopher Stewart and Kai Shen. Performance Modeling and System Management for Multi-component Online Services. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 71–84, 2005.

[SV98] Prashant Shenoy and Harrick M. Vin. Cello: A Disk Scheduling Framework for Next Generation Operating Systems. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 44–55, 1998.

[sys08] System z9: The Value of Mainframe. `http://www-03.ibm.com/systems/z/about/charter/value_utilization.html`, 2008.

[TJDB07] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. On the Use of Hybrid Reinforcement Learning for Autonomic Resource Allocation. *Cluster Computing*, 10(3):287–299, 2007.

[tom08] Apache Tomcat. `http://tomcat.apache.org/`, 2008.

[TPPC02] TPC Transaction Processing Performance Council. TPC Benchmark, Web Commerce, Specification, Version 1.8. `http://www.tpc.org/tpcw/default.asp`, February 2002.

[UC05] Bhuvan Urgaonkar and Abhishek Chandra. Dynamic Provisioning of Multi-tier Internet Applications. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, pages 217–228, 2005.

[US04] Bhuvan Urgaonkar and Prashant Shenoy. Sharc: Managing CPU and Network Bandwidth in Shared Clusters. *IEEE Transactions on Parallel and Distributed Systems*, 15(1):2–17, 2004.

[USR02] Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. Resource Overbooking and Application Profiling in Shared Hosting Platforms. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 239–254, 2002.

[VGR98] Ben Verghese, Anoop Gupta, and Mendel Rosenblum. Performance Isolation: Sharing and Isolation in Shared-Memory Multiprocessors. In *Proceedings of the Architectural Support for Programming Languages and Operating Systems Conference (ASPLOS)*, pages 181–192, 1998.

[Vir02] Virgilio Almeida and Martin Arlitt and Jerry Rolia. Analyzing a Web-Based System's Performance Measures at Multiple Time Scales. *SIGMETRICS Performance Evaluation Review*, 30(2):3–9, 2002.

[vir08] Microsoft Virtual Server. `http://www.microsoft.com/windowsserversystem/virtualserver/default.aspx`, 2008.

[vmw08a] Server Consolidation and Containment, With Virtual Infrastructure. `http://www.vmware.com/`, 2008.

[vmw08b] VMware. `http://www.vmware.com`, 2008.

[vmw08c] VMware Capacity Planner. `"http://www.vmware.com/products/capacity_planner/`, 2008.

[vmw08d] VMware Distributed Resource Scheduler (DRS). `http://www.vmware.com/products/vi/vc/drs.html`, 2008.

[VMw08e] VMware. Server Consolidation and Containment. `http://www.vmware.com/pdf/server_consolidation.pdf/`, 2008.

[VMw08f] VMware. Virtualization Basics. `http://www.vmware.com/vinfrastructure/`, 2008.

[VMw08g] VMware. VMware Products. `http://www.vmware.com/products/`, 2008.

[Wal02] Carl A. Waldspurger. Memory Resource Management in VMware ESX Server. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 181–194, 2002.

[WB95] Greg Welch and Gary Bishop. An Introduction to the Kalman Filter. Technical Report 95-041, University of North Carolina at Chapel Hill, Department of Computer Science, February 1995.

[WC03] Matt Welsh and David Culler. Adaptive Overload Control for Busy Internet Servers. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, page 4, 2003.

[WLZ+07] Zhikui Wang, Xue Liu, Alex Zhang, Christopher Stewart, Xiaoyun Zhu, Terence Kelly, and Sharad Singhal. AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments. In *Proceedings of the IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, 2007.

[WSG02] Andrew Whitaker, Marianne Shaw, and Steven Gribble. Denali: Lightweight Virtual Machines for Distributed and Networked Applications. Technical Report 02-02-01, University of Washington Department of Computer Science and Engineering, February 2002.

[WSVY07] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 229–242, 2007.

[WW94] Carl A. Waldspurger and William E. Weihl. Lottery Scheduling: Flexible Proportional-Share Resource Management. In *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–11, 1994.

[WZS05] Zhikui Wang, Xiaoyun Zhu, and Sharad Singhal. Utilization and SLO-Based Control for Dynamic Sizing of Resource Partitions. In *Proceedings of the IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, pages 133–144, October 2005.

[xen08a] XenSource. `http://www.xensource.com/`, 2008.

[Xen08b] XenSource XenCenter. `http://www.citrixxenserver.com/Documents/xensourcev4_datasheet.pdf`, 2008.

[Xen08c] XenSource. Virtual Data Center. `http://www.xensource.com/solutions/Pages/Virdata.aspx`, 2008.

[XZF+07] Jing Xu, Ming Zhao, José Fortes, Robert Carpenter, and Mazin Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*, page 25, 2007.

[XZSW06] Wei Xu, Xiaoyun Zhu, Sharad Singhal, and Zhikui Wang. Predictive Control for Dynamic Resource Allocation in Enterprise Data Centers. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 115–126, 2006.

[ZBCS99] Xiaolan Zhang, Michael Barrientos, J. Bradley Chen, and Margo Seltzer. HACC: An Architecture for Cluster-Based Web Servers. In *Proceedings of the USENIX Windows NT Symposium*, pages 155–164, 1999.

[ZWS06] Xiaoyun Zhu, Zhikui Wang, and Sharad Singhal. Utility-Driven Workload Management using Nested Control Design. In *Proceedings of the American Control Conference*, 2006.

[ZYW+05] Tao Zheng, Jinmei Yang, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. Tracking Time-Varying Parameters in Software Systems with Extended Kalman Filters. In *Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON)*, pages 334–335, 2005.