Using Nonlinear Control of Resources to Achieve Differential Performance Objectives in Software Systems

Tharindu Patikirikorala

Submitted in fulfilment of the requirements of the degree of

Doctor of Philosophy

2013

Faculty of Information and Communication Technologies

Swinburne University of Technology

Abstract

In many competitive business domains, software systems have become vital to achieve the business objectives efficiently. In such software systems, maintaining performance properties such as response time and throughput at runtime is important to avoid customer dissatisfaction and violation of service level agreements. This is a challenging task as service providers typically need to share computing resources between service consumers in order to deliver those services efficiently under dynamic and unpredictable environmental conditions. Managing such systems using human-in-the-loop decision making methods at runtime is neither efficient nor cost-effective. As a result, runtime performance management tasks need to be automated.

Closed-loop approaches based on control engineering methodologies have been widely investigated, as a way to achieve relative and absolute performance management objectives at runtime, while sharing a limited amount of resources. These approaches are based on linear modelling and control methods. However, linear approaches neglect the prominent nonlinear dynamics of the relative and absolute performance management systems and provide effective control only in a limited operating range.

In this thesis, we classify the nonlinearities that exist in the relative and absolute performance management schemes. We then introduce two novel nonlinear feedback control methods to reduce the runtime impact of nonlinearities on the control system. In the first approach, compensators are integrated into the control system to reduce the impact of nonlinearities. In particular, a Hammerstein-Wiener block-oriented model is used for relative performance management while a MIMO Wiener model is used for absolute performance management. In the second approach, we represent the dynamics of the nonlinear system with multiple linear models. Multiple models and multiple linear controllers are implemented together with a switching scheme, to select the most suitable controller to provide control under the current operating conditions. In addition, we present a class library of control components, to facilitate the implementation of complex control systems for software systems.

The evaluations conducted using simulation studies and experimental real-world case studies indicate that the proposed nonlinear approaches can significantly improve the performance and resource management capabilities compared to other state-of-the-art approaches. We further demonstrate that the class library significantly improves the efficiency of the control system engineering process.

Acknowledgements

First and foremost, I would like to thank my supervisors Dr. Alan Colman, Prof. Liuping Wang and Prof. Jun Han for their support and guidance during my entire candidature. Their monitoring and advice on my experiments and publication drafts and the support extended to accrue required resources for my project enabled me to complete the planed work on time. I would also like to acknowledge Prof. Lars Grunske, Dr. Quoc Vo, Dr Edmonds Lau and Dr. Yoni Nazarathy for their helpful feedback related to my work.

I would like to express my profound gratitude to my mother, father, sister and other relatives for their unconditional love and encouragement. I would also like to appreciate the support of my colleagues Dr. I. Meedeniya, Dr. A. Aleti, Mr. M. Kapuruge, Mr. I. Kumara, Mr. A. Amin, Dr. T. Nguyen, Mr. A. Kabir, Mr. M Hussein, Dr. N. Bekmamedova and Mr. M Kayes during my candidature for helpful knowledge sharing and useful feedback. Further, I have to thank my friends Mr. T. Silva, Mr. A. Desilva, Mr. V. Randeniya, Mr. D. Pathirana, Mr. M. Coorey, Mr. I Avazpour and Mr. S. Fernando who helped me during my stay in Australia in numerous ways.

I would also like to thank Swinburne IT support staff, in particular Mr. Oai Dai and Mrs. Narinthorn Yooiam for supporting in setting up hardware and software resources for my case studies.

I would also like to acknowledge Swinburne University Postgraduate Research Award (SUPRA), which gave me the financial support throughout my candidature.

Declaration

Here by I certify that, this thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made in the text of the thesis. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

Tharindu Patikirikorala

Date

List of Publications

Journal publications

- Tharindu Patikirikorala, Liuping Wang, Alan Colman, Jun Han. Hammerstein-Wiener nonlinear model based predictive control for relative QoS performance and resource management of software systems, *Control Engineering Practice*, Volume 20, Issue 1, Pages 49-61, 2011 (work related to Chapter 4 and 5)

- Tharindu Patikirikorala, Alan Colman, Jun Han, Liuping Wang. An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems, *Journal of Systems and Software*, 2012 (work related to Chapter 6)

Conference publications

- Tharindu Patikirikorala, Alan Colman, Jun Han, Liuping Wang. A multi-model framework to implement self-managing control systems for QoS management, *International symposium on Software engineering for adaptive and self-managing systems*, Pages 218-227, ACM, 2011 (work related to Chapter 6)

- Tharindu Patikirikorala, Liuping Wang, Alan Colman. Towards Optimal Performance and Resource Management in Web Systems via Model Predictive Control, *Australian Control Conference*, Pages 469- 474 , IEEE, 2011 (work related to Chapter 5)

- Tharindu Patikirikorala, Alan Colman, Jun Han, Liuping Wang. A Systematic Survey on the Design of Self-Adaptive Software Systems using Control Engineering Approaches, *International symposium on Software engineering for adaptive and self-managing systems*, Pages 33-42, IEEE, 2012 (work related to Chapter 2)

- Tharindu Patikirikorala, Liuping Wang, Alan Colman. Managing Performance and Resources in Software Systems using Nonlinear Predictive Control, *American Control Conference*, IEEE, 2012 (work related to Chapter 4 and 5)

- Tharindu Patikirikorala, Indika Kumara, Alan Colman, Jun Han, Liuping Wang, Denis Weerasiri, Waruna Ranasinghe. Managing Performance and Resources in Software Systems using Nonlinear Predictive Control, *International Control Conference on Service Oriented Computing*, 2012 (work related to Chapter 8)

Workshop publications

- Tharindu Patikirikorala, Alan Colman. Feedback controllers in the cloud, *Asia-Pacific Software Engineering Conference, Cloud workshop*, 2010

List of Symbols and Abbreviations

Symbols

R_i	Response time of the ith class
S_i	Resource cap of the ith class
$R_{SLA,i}$	Desired response time by the ith class
S_{total}	Total number of resource units available in the system
$S_{min,i}$	Minimum number of resource units reserved for ith class
P_i	Differentiation or priority level of the ith class
n	Total number of classes sharing the resources in the system
a	Scaling factor for the discrete-time Laguerre functions
N	Number of terms in the discrete-time Laguerre functions expression
o_m	Zero vector with appropriate dimension

Abbreviations

VM	Virtual machine
CPU	Central processing unit
MIMO	Multiple-input, Multiple-output
SISO	Single-input, single-output
SID	System identification
MPC	Model predicative control
LQR	Linear quadratic regulator
STR	Self-tuning regulator
SLA	Service level agreement
ARX	Autoregressive exogenous input
PI	Proportional integral
MMST	Multi-model switching and tuning
Cap	Resource capacity
UML	Unified modeling language
DES	Discrete event simulation
BPS	Business process server

Contents

List of Figures

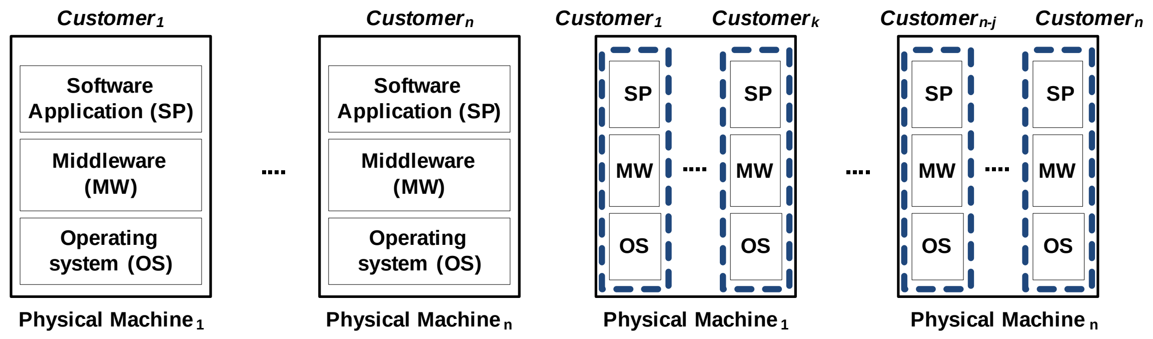
List of Tables

Chapter 1

Introduction

Software systems and services have become an essential part of business operations today to achieve the business objectives in a more efficient and cost-effective manner. For instance, banking, online trading and medical services are some of these businesses. In such competitive business domains, it is essential to provide the correct functional services, while maintaining the performance properties such as response time and throughput at acceptable levels throughout the entire operations. This is because poor performance management can lead to customer dissatisfaction and violation of the legal requirements, which would ultimately cause significant financial loses. Due to the complex business requirements, the increasing customer base and the dynamic, unpredictable nature of the operating environments, managing the performance properties of these software systems during operations is a complex task. As a consequence, the runtime management decisions are still made by the IT staff of many business organizations. However, this human intensive management approach have proven to be error-prone, inefficient and costly [47, 80]. In order to reduce the human involvement, the automation of runtime performance management of software systems is a promising alternative, which subsequently led to many research challenges in the past decade.

Among the challenges, one is to maintain the performance properties at runtime while efficiently and effectively sharing the limited amount of resources. This has become an active research area with the popularity of the shared resource software environments operating under the utility computing model. In traditional resource environments, a dedicated physical machine or machines are allocated for customers to deploy their software stack (see, Figure 1.1a). Consequently, there is no sharing of resources between different customers leading to significant resource under-utilization, high operating costs and scalability issues [129, 177]. In contrast, the vision of the emerging utility computing model is to deliver hardware, middleware and software services as a utility similar to the delivery of the electricity and water supplies in the modern world [12, 14]. The cloud computing data centers and multi-tenanted software platforms are realizations of the utility computing model, which serves multiple customers using and sharing common pools of resources at the hardware, middleware and application levels of the system stack as shown in Figure 1.1b. Utility computing is increasing rapidly [31] because there are many other advantages to all the stakeholders and addresses issues that exist in the traditional resource environments as described below.



(a) Dedicated resources for each customer (b) Resources shared between customers

Figure 1.1: Resource settings in software environments serving multiple customer classes

A data center provider can reduce the operational costs by maintaining a shared resource environment and by allocating the hardware resources required by a particular customer for a given time during which the customer pays only for the hardware resources they have used. Therefore, consolidation of multiple customers into one machine reduces the number of machines that have to be operated to provide services to n customers (see, Figure 1.1b), thereby reducing the operating costs and improving resource utilization and sharing. However, the data center provider has to maintain the performance properties of customer applications by dynamically managing resources in physical machines. This dynamic performance and resource management problem does not exist in the traditional method (Figure 1.1a), which maintains and calculates the fee for a fixed amount of hardware resources for the entire period regardless of whether the resources have been used or not.

An application service provider on the other hand, can use a multi-tenanted shared resource software architecture to provide services to multiple customers by maintaining a single infrastructure. This approach significantly reduces the source code, infrastructure management costs and scalability issues in comparison with maintaining separate code-bases and infrastructures. In this situation as well, one of the main objectives is to deliver agreed or acceptable performance properties at runtime for all the customers sharing the same resources.

In order to deliver the required performance properties in a shared resource environment serving multiple customer classes, efficient and timely resource allocations have to be performed at runtime (so-called dynamic resource management). There are many factors and characteristics of such an environment that inhibit the design and implementation of a runtime performance and resource management system.

1.1 Key Characteristics of Shared Resource Software Environments

As illustrated in Figure 1.1b, a shared resource software system serves the service requests of n number of customer classes simultaneously. In this work, we call such a software environment a *multi-class shared resource software system* as defined below:

If a software or hardware service provider maintains a single environment to achieve the business objectives of multiple customers or organizations, then the environment given by the service provider is called a multi-class shared resource software system. At the same time, each customer/organization perceives the system as they have their own dedicated environment.

In order to design and implement a runtime performance and resource management system for a multi-class shared resource software environment, the relevant key characteristics are discussed as follows.

1. The shared resource software system needs to provide services to multiple client classes and achieve the service level agreements (SLAs) associated with these classes. These SLAs may specify the required values for the performance properties or priority levels of the classes, which need to be maintained by the shared resource software system.

2. The workloads (request rates) from these different client classes may vary overtime in a dynamic and unpredictable fashion. These variations could happen in a short or a long period of time with different magnitudes. The stochastic nature of the workload makes it hard to predict or create a model of the incoming workload.

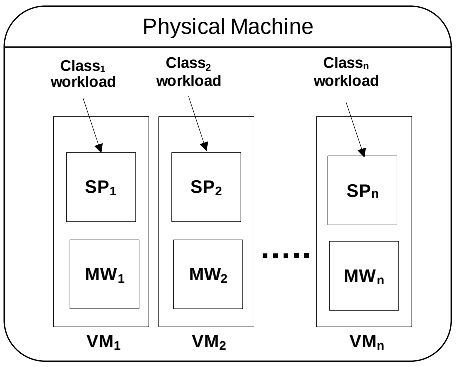
3. When multiple clients are sharing the resources of a single system, a resource allocation decision for one client class ideally should not affect the performance attributes of the other classes (so-called performance isolation) or should provide differentiated service levels depending on the priority of the client class [108, 145, 150]. For instance, an aggressive/malicious client class could overload the system, leading to degradation of the performance for the other client classes. Such situations should be minimized or avoided.

4. The available resources are limited, so that these resources have to be efficiently allocated at runtime by taking into account the total available resources in order to ensure a viable service delivery to each individual client class. Furthermore, one of the objectives is to improve the resource utilization and minimize the required resources as much as possible.

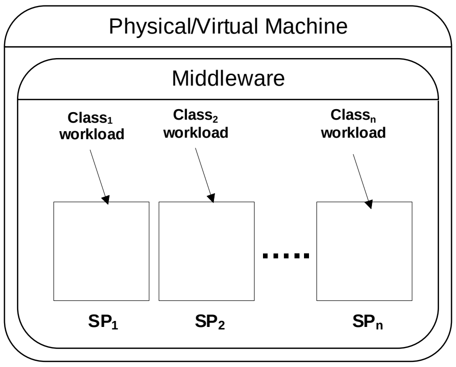
5. The resources of a software system could be shared at multiple levels. Typically, a software system stack is composed of hardware, middleware and software entities. The resource sharing could happen at any one of these levels depending on the deployment strategy used. Figure 1.2 illustrates the possible deployment options.

   Hardware level: With the significant improvements in the virtualization technology, the modern data centers or hardware providers offer virtual machines (VMs), which compartmentalize a single physical machine into multiple separate hosting environments. These VMs could be used to deploy software stacks of different customers as shown in Figure 1.2a. As a consequence, the CPU, memory and disk of a physical machine are shared between multiple customers, making such environments, multi-class shared resource systems at the hardware level.

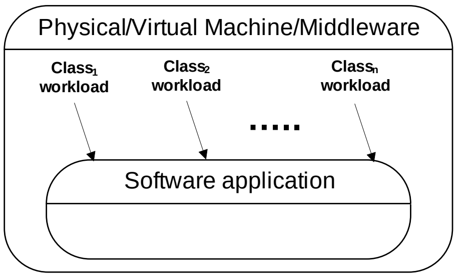
   Middleware level: Middleware is a software environment that lie between the software applications and the operating system or hardware. It provides development and deployment support for software applications. For instance, web servers, appli-



(a) Hardware level



(b) Middleware level



(c) Software application level

Figure 1.2: Deployment options of the multi-class shared resource environments at different levels of the system stack

cation servers and enterprise service buses are such middleware. A single middleware deployed in virtual or physical machines can also be used to host software applications and serve several customer classes as shown in Figure 1.2b. With this option therefore, the environment can be further consolidated compared to the shared infrastructure option, because a single virtual machine can be used to run application instances of multiple customers. Unlike at the hardware level, it is hard to allocate hardware resources (e.g., CPU, memory) at the middleware level because the underlying operating system is not aware of the existence of the multiple customer applications. This makes it difficult to control hardware resources at the granularity of the customer applications. Due to this fact, the resources that could be dynamically allocated are soft-resources such as worker threads, communication connections, message queues and middleware-level cache.

Software application level: In this option, any software application instance can handle the workload of any or many customers. As shown in Figure 1.2c, the same application instance created out of a single codebase serves multiple customer workloads. Therefore, this option delivers further consolidation of customer environments compared to the previous two options when the software application instances are deployed on middleware, virtual or physical machines. However, in this option as well, allocating the hardware resources between customer classes is impractical because customer classes are an abstraction at the software application level. A possibility is to share bottlenecked soft-resources at the application level. Such resources may be worker-threads provided by the middleware, database connections or other business domain specific resources.

It is evident that depending on the level of the stack at which the multi-class shared resource system is deployed, the resources that need to be dynamically managed vary.

6. Unlike other engineering disciplines, there are no underlying physical laws (such as mass balance and electrical laws) to describe the behavior or the performance of a software system and the interactions between the layers it is composed of [81]. In addition, as shown by the existing works [179, 266] the performance properties are nonlinearly related to the shared resource allocation. This inherent nonlinear characteristic and dynamic behavior of the software system impose difficulties to model the behavior of the system either at the design time or runtime [81, 108, 238].

1.2 An Overview of Existing Solutions in a Nutshell

Many techniques have been proposed in the literature to address some of the problems characterised in Section 1.1. In this section, an overview of the existing solutions is presented. In general, there are two mainstream approaches: open-loop and closed-loop, which also have their own sub-streams. A detailed analysis of the literature is given in Chapter 2.

Open-loop approaches

In open-loop approaches, the feedback of performance properties is not considered at runtime. The simplest approach is to maintain dedicated, fixed resources or to provide best effort performance to each client. For instance, many of the shared hardware resource environments such as data centers still allocate fixed and dedicated hardware resources to deploy the software applications of each customer. The fixed resource partitioning strategy is also implemented by many state-of-the-art virtualization platforms, namely Xen and VMware to allocate hardware resources such as CPU, network bandwidth and memory between virtual machines (VMs) in order to maintain performance isolation [19]. Although some of the scheduling mechanisms are implemented to adjust the resource allocations at runtime, the feedback of performance properties of the customer applications are not considered. Therefore, the required performance levels by each customer may not be maintained. This management strategy provides isolation among the customers, however, at the expense of resource utilization, sharing and scalability [12, 177]. In addition, most of the middleware platforms (e.g., web servers, database servers, business process engines), which can be used to host software applications of multiple customers or organizations, still provide best-effort performance by treating all service requests equivalently [101, 241, 256, 257]. Consequently, there is less emphasis on performance management of different customers while sharing the resources in an efficient way.

Closed-loop approaches

In closed-loop approaches, the feedback signals of the performance properties are considered in the design and implementation of the performance and resource management system. There are many such techniques proposed for shared resource environments. These approaches can be classified as *admission control* and *dynamic resource management* strategies. The admission control strategies reject or abort requests, when the workload of the system exceeds a certain threshold, in order to maintain the performance properties such as response times below certain bounds (e.g., [51, 101, 137, 163, 241, 256]). These approaches also require detailed knowledge about the workload models (including service time, request arrival rates) and have assumptions on the workload distributions [81, 177]. In contrast to the admission control and fixed resource partitioning, the dynamic resource management adjusts the resource partitions of client classes in order to maintain the performance properties such as response time and improves the resource sharing under changing workloads and resource demands (e.g., [4, 107, 108, 145, 177, 178, 180, 257]). However, it sacrifices the performance isolation in comparison with the fixed resource partitioning techniques. Even though performance isolation and resource utilization/sharing are competing aspects in deciding the management scheme, a hybrid resource management approach of fixed and dynamic resource partitioning may lead to a much more effective management scheme as mentioned in [55].

Furthermore, in the shared resource environments, closed-loop approaches are used to achieve mainly the absolute and relative performance management objectives by adjusting the resource partitions. The objective of the absolute management scheme is to maintain the performance properties of each client class at or around the specified value (for example, the works [41, 105, 108, 118, 145, 177, 178, 180, 231]). In contrast, the relative management scheme maintains the ratios of the performance properties of two or more client classes at or around a specified ratio (see works in [41, 145, 150, 177, 180]). The main challenge in realizing these schemes is to achieve the prescribed management objective while dynamically adjusting the resource partitions of each client class with the characteristics listed in Section 1.1 [145].

The existing works that propose approaches to achieve the absolute and relative performance management objectives while dynamic resource allocation are based on either *control engineering* and *non-control engineering* methodologies. Two of the dominant methods in non-control engineering approaches involve either simple rule-based techniques (e.g., [17, 20, 35, 86, 110, 138, 160]) or complex optimization techniques (e.g., [109, 127, 170, 220]). These are useful techniques because of their ability to handle complex policies/constraints and are relatively easy to design and implement. However, they suffer from a lack of well-founded design processes in selection of important design parameters (for example, thresholds and weights in utility functions) and a lack of systematic processes to guarantee system stability. In contrast, the control engineering methods provide the formal systematic design process and the ability to achieve the diversified management objectives under highly unpredictable operating conditions using the feedback principle, which is also tolerant to a certain degree of model uncertainty [29, 81, 143]. As a result, feedback control engineering techniques have been identified in recent years as one of the major enabling techniques to automate runtime management of software systems, in particular to realize the runtime resource management in shared resource environments [4, 29, 37, 121, 145, 177, 178, 203, 231, 233].

The control system design generally consists of two main steps. The first step is to establish the dynamics of the system as a mathematical model of inputs and outputs of the system (so-called system model). The model of the system is then utilized in the second step, which includes controller design, simulation, analysis, implementation and validation. The characteristics of software systems mentioned in Section 1.1 present many challenges to the general control system design methodology. As mentioned before, one of the characteristics of the software system is the lack of first principle models. Consequently, significant research efforts have been put to investigate black-box models of the software performance management system with respect to certain operating conditions [81, 189] using model estimation techniques. Typically, in such model estimation techniques, the input-output data gathered from an experiment is used to derive the system model. These efforts have focused only on linear black-box models and estimation techniques, neglecting the inherent nonlinear dynamics of the software systems. In the case of a shared resource environment and with respect to absolute and relative management objectives, the relationship between the performance properties such as response time and throughput (the performance variable controlled) of a single client class and the resource allocation (the manipulated variable) is known to be nonlinear [41, 81, 106, 150]. In addition, the characteristics of the nonlinearity depend on the workload intensity as well as the level of the system stack the resource being shared (for instance, hardware and middleware) [177]. Consequently, the aforementioned linear control engineering approaches that have neglected the nonlinear dynamics become inadequate when the system operation condition changes, which frequently occurs in a software performance management system. In order to design and implement a performance management system for shared resource software systems that works in a spectrum of operating regions with unpredictable workload conditions and operates with different resources at different levels, the nonlinearity in the system dynamics must be addressed adequately in the process of modelling and in the process of control system design.

1.3 Contributions

The motivation of this thesis is to *investigate novel nonlinear modelling and feedback control system design techniques in managing performance and resources of multi-class shared resource software systems or environments at runtime.* More specifically, the management system will possess the following attributes:

- to achieve the required absolute or relative performance management objectives of multiple customers or organizations served by the shared resource environment self-adaptively under changing unpredictable operating conditions with less or no human interventions.

- to adjust the resource partitions of each client class efficiently under dynamic unpredictable resource demands honouring any constraints on the resources,
- to provide performance isolation between client classes,
- to provide mechanisms to change the control objectives and requirements at runtime with less or no overhead of redesign or reimplementation of the management system,
- to provide a systematic design approach and tool support to implement the management system.

The major contributions in this thesis, associated with the design and development of this software performance management system are listed as follows.

I. Modelling the nonlinear dynamics Due to difference in the management objectives of the absolute and relative performance management schemes, the nonlinear characteristics illustrated are significantly different from each other. In the case of relative performance management objectives, consideration of ratios of performance properties (system output) and resource allocations (system input) between the client classes creates severe nonlinear dynamics at the system input and output. In the case of the absolute performance management scheme, unlike the relative management scheme, the performance property of each individual client class and the corresponding resource allocation shows a nonlinear relationship. Furthermore, when all the client classes are considered together in a single shared resource system there are multiple objectives that have to be achieved under resource constraints.

In this thesis, we have presented two novel model identification techniques to estimate the nonlinear dynamics of software systems, namely, non-linear block-oriented estimation and multiple linear model estimation. We have applied these techniques to characterize the dynamic relationship between resource inputs and performance outputs in both the relative and absolute performance management schemes. We give an overview of these techniques below.

1) Nonlinear block-oriented model based estimation

Firstly, the nonlinearities that exist in the relative performance and resource management scheme are characterized into separate blocks of input, output nonlinearities and the rest of the dynamics. Then, each block is estimated by a mathematical model. This block-oriented model is known as the Hammerstein-Wiener model in the control literature. A systematic identification process is presented in Chapter 4 to enable the modelling of the relative performance management system as the Hammerstein-Wiener model.

Secondly, the nonlinearities that exist in the absolute performance management scheme are also characterized as output nonlinearities and represented as a multiple-input and multiple-output Wiener model structure. A new model identification procedure is presented for this estimation in Chapter 4.

This thesis shows for the first time that the above nonlinear block-oriented modelling techniques can be successfully used to model the dynamics of multi-class shared resource software systems.

2) Multiple linear model based estimation

Instead of using a single linear model to represent a wide range of dynamics as in the existing work, we present a multiple linear model based technique to estimate the dynamics of the relative and absolute performance management systems. In this work, we have presented effective methods to divide the operating region into manageable sub-regions depending on the management requirements and nonlinearities. Then, the dynamics of each of these sub-regions are identified separately to represent the dynamics of the system with multiple models.

II. Nonlinear control system design

After the estimation of nonlinear models, the next step is to design the suitable control systems to provide performance and resource management decisions at runtime. For the two types of modelling techniques, we present two nonlinear control system design methodologies as follows:

1) Control system with compensators

With the estimated input and output nonlinear components of the Hammerstein-Wiener model, pre-input and post-output compensators are designed and integrated to the control system in order to reduce the impact of the nonlinearities on the relative performance management system at runtime. The management system is composed of a linear controller designed using well-established control system design techniques.

Using the estimated multi-input multi-output Wiener model, compensators are designed and connected at the each output of the absolute performance management system in order to mitigate the issues of nonlinearity. The absolute performance management scheme also necessitates complex multiple objectives to be achieved at runtime. We formulate a multi-input multi-output control and constraint optimization problem and then solve it using a model predictive controller equipped with a quadratic programming solver.

The above described control systems based on the compensator frameworks are novel control system architectures in the area of software system management. Furthermore, these nonlinear control systems have shown significant improvements in the performance management of relative and absolute schemes compared to the linear control approaches.

2) Control system with switching capabilities

As mentioned above, the system dynamics can also be represented by multiple linear models. Then, multiple controllers can be developed from these models, where each individual controller is most suitable to operate in certain operating conditions. Although there are multiple suitable controllers, only a single controller can be connected to the control system to make the decisions at a given time instance. Thus, the most appropriate model and controller have to be selected and then connected to the system autonomously, without any human intervention. Therefore, switching decisions are paramount to achieve the control objectives with multiple models under changing operating conditions. Furthermore, when these controllers are switched back and forth, most suitable control algorithms have to be selected and implemented in order to reduce the transient responses and overhead of porting the runtime state data in-between the controllers. In this thesis, we have successfully developed effective switching schemes and control algorithms in order to integrate multiple models and controllers into the control system of relative and absolute performance management schemes.

III. Implementation and evaluation of control system

Following the systematic control system design process presented in this thesis, we have implemented control systems to achieve relative and absolute performance management objectives of simulation and several real-world shared resource software environments, sharing resources at different levels of the system stack (e.g., application and middleware level). In addition, the evaluations of the implemented nonlinear control systems presented in this thesis are conducted in both simulation and real-world shared resource environments serving multiple customers or organizations under different settings, which include

1. unpredictable workload variations (based on time varying, step like, ramp like and real-world workload traces),
2. changing agreements or priorities of the customer requirements,
3. number of customer classes deployed in the shared-resource environment, and
4. changes to the tuning parameters of the proposed management system.

Further to evaluate that the simulation parameters have no impact on the results and conclusions of the experiments, we also conduct Monte-Carlo simulations.

In these evaluations, we use the existing linear control strategies as a bench-mark to compare the improvements delivered by each proposed nonlinear control approach. These evaluations conducted on both simulation and real-world systems illustrate the external and internal validity of the presented approaches. From the outcomes of these evaluations, we can conclude that the nonlinear approaches presented in this thesis have significantly outperformed the existing state-of-the-art management approaches in many cases.

IV. Tool support

The proposed management system architectures rely on the formal and rigorous control system design and implementation methodologies. However, the development of such management systems requires specialized knowledge and substantial design, development and testing efforts by the software engineers. To aid this design and implementation process, we have implemented a configurable, extendible and cost-effective off-the-shelf software class library, which provides implementations of different control components. This off-the-shelf class library will reduce the development efforts and knowledge requirements in the design and deployment of runtime management systems for shared resource software environments.

An empirical study conducted with a group of software engineers has shown that this class library has facilitated the control system implementation process significantly compared to the implementations from scratch.

1.4 Thesis Outline

Figure 1.3 shows the high-level structure of the thesis using a flow chart.

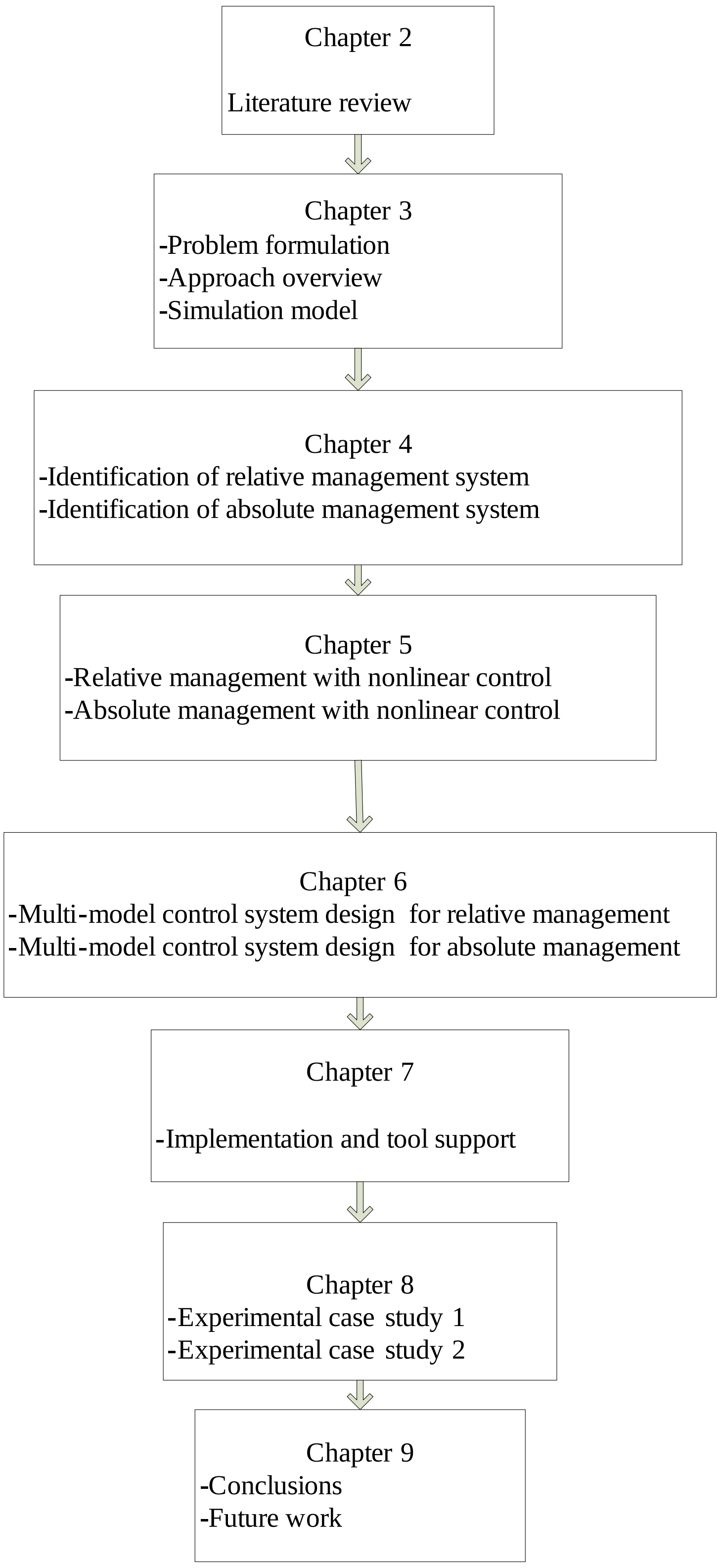

Chapter 2
Literature review
Chapter 3
-Problem formulation
-Approach overview
-Simulation model
Chapter 4
-Identification of relative management system
-Identification of absolute management system
Chapter 5
-Relative management with nonlinear control
-Absolute management with nonlinear control
Chapter 6
-Multi-model control system design for relative management
-Multi-model control system design for absolute management
Chapter 7
-Implementation and tool support
Chapter 8
-Experimental case study 1
-Experimental case study 2
Chapter 9
-Conclusions
-Future work

Figure 1.3: Thesis outline

Chapter 2 overviews the existing literature related to resource management of shared resource software environments.

Chapter 3 formulates the research problem followed by the approach taken by this thesis. Chapter 3 concludes with the details of the simulation environment and experimental case studies.

Chapter 4 presents the block-oriented nonlinear modeling approaches for relative and absolute performance management systems.

Chapter 5 is divided into two segments. The first segment presents and evaluates the new control architecture design methodology for the case of relative performance management. In the second segment nonlinear control system design techniques proposed for absolute performance management will be presented and evaluated.

Chapter 6 proposes and evaluates the techniques to integrate multiple controllers and switching schemes to design management schemes based on multiple models for both relative and absolute performance management systems.

Chapter 7 covers the design and implementation details of the off-the-shelf class library built to assist the development of control systems for software systems. The details of an empirical study, which evaluates this class library will be presented at the end of the Chapter 7.

Chapter 8 gives the details of two experimental case studies investigated in this thesis. The first case study focuses on a production multi-class business process server called WSO2 Stratos, while the second case study focuses on a real-world travel reservation system. Chapter 8 covers experiment setup and results of these case studies.

Chapter 9 presents possible directions for future research and concluding remarks.

Chapter 2

Literature Review

This chapter overviews the literature related to the work of this thesis. The general background and work related to multi-class shared resource systems will firstly be discussed before going into the details of the existing performance and resource management methodologies. We then present a taxonomy of the literature, which also highlights the focus areas of this thesis.

2.1 Multi-class Shared Resource Systems

Multi-class shared resource systems, particularly at the middleware and software levels are still new areas of study. There are many challenges that have to be faced in order to design, develop and deploy such systems. These challenges include

1. selecting a maturity level,
2. enabling configurability and customizability of the customer applications,
3. designing the database,
4. placing customer application in the environment,
5. maintaining (security and performance) isolation properties,
6. managing performance and resource at runtime.

This section overviews the work proposed to address the first five challenges, while work on the performance management will be reviewed in Section 2.2.

Four maturity levels are proposed in [43] to implement multi-class shared resource environments. A suitable maturity level could be selected by the shared resource service providers in order to realize their environment. The selected maturity level impacts on the way the system is designed, deployed and how the above described challenges can be addressed. In maturity *Level-1* there is a separate customized source codebase and dedicated hosting instance for each customer class. The *Level-2* has a single source codebase, which is configured and deployed in a dedicated hosting instance for each customer class. In the third level again a single codebase is maintained, but only a single hosting instance exists for all the customer classes. In the final level a single codebase deployed in multiple shared hosting instances provides services to all the customer classes. It is evident that levels 1 and 2 show limitations of high operation costs, maintenance costs and resource underutilization issues that existed in the traditional methods. In contrast, maturity levels 3 and 4 address these issues by consolidating the environment and improving the resource sharing and utilization. However, the main challenge in such implementations is providing customized business functionalities to each customer, maintaining isolation and management of performance properties. It follows that if maturity levels 3 and 4 are to be achieved, then techniques such as those described in this thesis will need to be developed.

Several approaches have been proposed to address other challenges described above in recent years. The works in [162, 187, 213, 244] have proposed techniques to enable configurability and customizability in shared resource environments. In order to design the database of multi-class shared resource systems, a set of patterns are given in [239] and successful real-world implementation details can be found in [187, 244]. The techniques to enable security isolation is investigated in [75, 187]. In the aforementioned maturity level 4, when there are multiple hosting instances that can be used to deploy customer applications, the tenant placement problem occurs. The optimization solutions to address this problem have been proposed in [55, 122].

Although there are many important challenges on the different aspects of multi-class shared resource environments, the focus of this thesis is to investigate the other vital issue of managing performance and resources of multi-class shared resource environments at runtime.

2.2 Performance and Resource Management

In this section, the work related to performance and resource management is reviewed in detail. The two mainstream approaches: open-loop and closed-loop are covered in

Sections 2.2.1 and 2.2.2 respectively.

2.2.1 Open-loop Approaches

In the open-loop approaches the feedback of the performance properties are not considered in the decision making. The sub-streams of this mainstream approach are capacity planning, fixed resource allocation and scheduling.

2.2.1.1 Capacity Planning

Capacity planning involves estimation of the hardware or software resources required for a future period of time. Traditionally, the estimation is performed by the historical resource demands extracted from the workload traces of customer software applications. In addition, the expert knowledge could be taken into account to further optimize the estimation. However, these estimations are done offline or infrequently due to the costs and manual work required for conducting such approximation procedures [102, 202]. After this estimation, the next step is to solve a complex multi-variable optimization problem to place these different applications into minimal number of physical servers [122, 202]. The outcome of these approaches is to maintain a fixed capacity till the next capacity planning is carried out, which may lead to significant under or over utilization of resources when the capacity is over- or under-estimated respectively.

In multi-class shared resource systems, capacity planning approaches are impractical to use as a dynamic resource allocation technique because the workload patterns of a class may vary significantly in short time periods. Consequently, fixing resource limits for each class depending on the peak workload demand may lead to resource wastage. However, capacity planning is still required in the current IT infrastructures to specify a limit on the maximum amount of resources, because there are costs involved on the resource usage and there is no possibility to accrue unlimited amount of resources. However, dynamic distribution of the resources among client classes is hard to achieve using capacity planning under significant variations in workloads.

2.2.1.2 Fixed Resource Reservation

The simplest approach is to maintain a fixed amount of resources in a hosting instance for each customer class irrespective of the changing resource demands. This is still the dominant approach of resource allocation in the multi-class shared resource systems at the hardware level. The major scheduling algorithms used in the state-of-the-art virtualization platforms such as Xen and VMware allocate hardware resources in fixed amounts between VMs in order to maintain the isolation properties [19]. This approach has the shortcoming of resource underutilization because there is no dynamic resource management.

2.2.1.3 Scheduling Methods

The existing production middleware platforms (e.g., web servers, database servers and business process engines) do not provide all the necessary features required to be a multi-class shared resource environment. In particular, in the case of performance and resource management, the popular web servers such as Apache provide best-effort performance by implementing simple First-In-First-Out (FIFO) scheduling, treating all requests equivalently [101, 241, 256, 257]. Other types of scheduling such as strict priority, earliest deadline first and shortest remaining time first are also proposed in works [24, 72, 77, 204, 256] for web servers. These techniques are also open-loop approaches because they do not consider the performance properties in scheduling. Instead they assume that the service times and deadlines are known when a request arrives and the requested contents are static. These assumptions are not valid in the current distributed software environments that provide versatile and dynamic business functionalities.

2.2.2 Closed-loop Approaches

In closed-loop approaches, the feedback of the performance properties is directly considered in the performance management at runtime. As a consequence, compared to open-loop approaches, the performance properties requested by the different customer classes can be achieved using a closed-loop approach. Therefore, the management objective has to be specified using the performance properties, so that the responsibility of the designed management system is to achieve that objective under changing conditions. We provide details and the brief history of the major performance management objectives investigated in the literature below.

With the development and popularity of the internet, providing quality of service (QoS) in the communication over networks and routers was a vital issue in 1990's. The openness and dynamicity of the internet hindered such delivery of QoS, which led to demand for different levels of QoS for different user classes [263]. The best-effort service model implemented with resource reservation in the network equipment and software systems was not able to deliver differentiated levels of QoS, consequently new techniques were required for QoS provisioning.

Two major QoS differentiation architectures were proposed by Internet Engineering Task Force (IETF[1]), namely Integrated service in 1994 (*IntServe*) [28] and Differentiated service in 1996 (*DiffServe*) [25]. In IntServe, the network resources are reserved along the path for a particular user who needs QoS guarantees. This approach faced many issues, in particular the scalability and requirement of all the network providers along the path should agree to reserve the resources. As a result, this architecture did not become popular [263]. The DiffServe proposed in 1996 provided the features of maintaining different QoS levels depending on the priority of the user class, which was implemented in the communication resources.

The DiffServe standard distinguished two types of performance guarantees: *Absolute DiffServ*, where minimum service rate is guaranteed based on the workload and *Relative DiffServ*, which provides better performance to the higher priority user class compared to the lower priority classes. These developments in network communication or network layer were not sufficient to provide end-to-end differentiated services to the different user classes because a significant amount of communication delay is also induced by the software system or the service. Therefore, effective management methodologies were required for the software systems and services as well. There have been many approaches that attempted to apply the above differentiation schemes to software layers. A survey can be found in [263]. However, application of such schemes is difficult at the software layer because of the complex business requirements and versatility of software applications or functionalities compared to the network layer which has a standard way of communication. This fact makes the management at the software layer difficult and different to the network layer.

In a nutshell, from the existing work the major performance management objectives of

[1]www.ietf.org/

multi-class shared resource software environments are commonly specified using absolute and relative performance management schemes [145].

The following subsections review the closed-loop approaches, which are classified as admission control approaches (Section 2.2.2.1) and dynamic resource management approaches (Section 2.2.2.2).

2.2.2.1 Admission Control Approaches

The idea behind admission control is to adjust the admission rates of requests to the system when the current workload is above the system capacity (so-called overloaded situation). Firstly, based on the content or customer class, the requests are classified. Secondly, a decision is taken whether to accept or reject the request based on the management objectives and workload conditions. Under the overloaded conditions requests will be rejected in order to maintain the performance properties and the integrity of the software environment based on the priority of the classes. Such admission control techniques have been widely investigated to manage performance properties of software systems in the last decade. The details on these approaches can be found in [7, 64, 263]. Many of these works rely on the assumptions related to the workload arrival distributions (typically, Poisson distribution) and service rate distributions (e.g., [7]). Such assumptions rarely hold in the real-world workloads [81, 177]. Several other approaches are based on the analytical queuing models or networks (e.g., [101, 218, 219]). These analytical models require structural details of the software architecture and approximations of various parameters such as request arrival and service rates in order to be effective [53].

In multi-class shared resource environments, the total workload is a composition of different classes. As a consequence, the arrival and service rate monitoring has to be done for each class, which makes it a computationally expensive approach. In addition, representing the dynamics of the shared resource environments using a queuing model have shown low accuracy under changing dynamics [30]. Furthermore, the queuing models are not fine grained enough to capture the transient behavior of the system due to unpredictable operating conditions in short time periods [81, 177].

The admission control is a vital methodology for the multi-class shared resource environments, because under persistently overloaded conditions, the incoming requests have to be rejected in order to maintain the system stability and integrity. Otherwise, performance properties such as response time may increase significantly, while the software system may end up crashing. In this work, we implement simple policy based admission control. However, we adjust the resource capacities of each class so that the provisioning of resources is performed to cater the demand of the workload of each class. Consequently, the above assumptions on the arrival request and service rates are not required. In addition, we treat the system as a black-box, thus the structural details of the system are abstracted away compared to the analytical models.

2.2.2.2 Dynamic Resource Management Approaches

The idea behind this approach is to allocate the required amount of resources depending on the workload variations in short time periods (in minutes if not seconds), thereby maintaining the required performance objectives throughout the operations. As a consequence, in contrast to the fixed resource allocation techniques, the dynamic resource allocation improves the resource utilization and sharing. However, under sudden workload bursts, performance isolation many not be achieved during the transient period because the time taken to reallocate the resources. In [75], a hybrid mechanism of fixed and dynamic resource allocation is recommended due to these issues of both schemes to achieve the required performance management objectives effectively.

The related works that propose approaches to achieve the performance management objectives using the dynamic resource allocation techniques can be further divided in to *control engineering* and *non-control engineering* methodologies.

Non-control engineering approaches

The dominant methods under this approach are either based on heuristics, simple rules (e.g., [17, 20, 35, 86, 110, 138, 160]) or complex optimization techniques (e.g., [109, 127, 170, 220]). These are useful techniques because of their ability to handle complex policies/constraints and are relatively easy to design and implement. However, the design parameters (for example, thresholds and weights in the utility functions) of these approaches have to be decided based on the trial and error procedure. These approaches consequently suffer from a lack of well-founded design processes in deciding important design parameters and require assumptions to be made regarding system variables (e.g.

workload distributions and arrival rates) [231]. In some approaches, complex policies can result in computationally expensive optimization problems that need to be solved in every sampling instance [109].

Such heuristic and simple rule based approaches are hard to apply in multi-class systems, which demand multi-objective control and constraint optimization at runtime. This is because implementing if-then rules or approximating threshold levels in the context of multi-variant competing demands, requirements and operating conditions is a challenging task. Furthermore, there is no way to test, validate and analyze the performance and guarantee stability of such techniques due to lack of formal grounding.

Control engineering approaches

In contrast to non-control engineering approaches, the control engineering methods provide a formal systematic design process and the ability to achieve the diversified management objectives under unpredictable operating conditions. These feedback based approaches are also tolerant to a certain degree of model uncertainty [29, 81, 143]. As a result, feedback control engineering techniques have been identified in recent years as one of the major enabling techniques to automate runtime management of software systems, in particular to realize the runtime resource management in shared resource environments [4, 29, 37, 121, 145, 177, 178, 203, 231, 233].

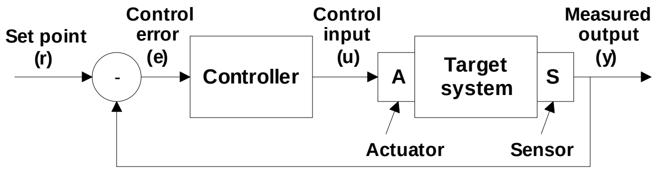


Figure 2.1: Block diagram of a control system

Figure 2.1 shows a block diagram of a feedback control system. The software system controlled by the controller is referred to as the *target system.* The target system provides a set of performance metrics as properties of interest (e.g. response time) referred to as *measured outputs* or simply outputs. *Sensor* monitors the outputs of the target system, while the *control inputs* are send to the actuator to adjust the system inputs (e.g. resource allocation) to change the behavior of the system. The controller is the autonomous decision making unit of the control system. The main objective of the controller is to maintain the outputs of the system sufficiently close to the desired values, by adjusting the inputs in response to disturbances. The desired values are translated into the control system terms as *set point signals*. These set point signals give the option for the control system designer to specify the goals or values of the outputs that have to be maintained at runtime. In each sample instance k, the controller calculates the difference between the measured outputs and the set points, which is called as the control error ($e(k)$). Because of the unpredictable disturbance and un-modelled dynamics, the absolute value of $|e(k)|$ maybe greater than zero. This means that the controller has not achieved the control objective. Consequently, a formal algorithm implemented in the controller will take into account the control error and come up with the input to be applied in the system at the current sample instance. This process continues in each sample instance in order to achieve the desired control objectives.

Depending on the type of system and control problem at hand, the design of the control system could be a single-input-single-output (SISO) or multi-input-multi-output (MIMO). That is if the system has a single input, output and set point, a SISO control system has to be designed. In contrast, if a system has multiple inputs, outputs and set points, a MIMO control system has to be designed.

To design a control system in a systematic way, two main steps have to be carried out. First, a sufficiently accurate model to represent the dynamics of the system has to be constructed. Second, a controller has to be selected, tuned and tested. More details about these steps are given in Appendix A. There are different types of control algorithms and control systems that can be implemented to achieve the required control objectives. These include fixed gain (PID), model predictive (MPC), adaptive, gain scheduling and reconfiguring control systems. An overview of these control systems can be found in Appendix B.

2.3 Control Engineering Approaches to Manage Software Systems

This section, firstly reviews the control engineering approaches proposed for software systems in general. Secondly, the control engineering approaches applied to manage performance and resources in multi-class shared resource environments will be reviewed.

2.3.1 General Applications of Control Engineering Methods

Although control engineering approaches provide useful and systematic design mechanisms, the application of such control approaches in software environments is still an emerging field of study [177]. Our systematic survey [191] comprehensively analyses many research efforts that have been made in the last decade. In [191], we also present a taxonomy to classify the applications of control engineering approaches based on the different software application areas and managed performance variables. The application areas include the data centers, virtual machine environments, middleware platforms, data storage and real-time systems. In addition, the managed performance variables are response time, throughput, power utilization, processor utilization and so on. Furthermore, the taxonomy also covers the characteristics of the control engineering solutions proposed in the literature, including the model (black-box or other), dimension of the control system (SISO, multiple-SISO or MIMO) and control scheme (e.g., fixed (PID), adaptive and model predictive) utilized. A detailed quantitative analysis based on this taxonomy could be found in [191].

Tables 2.1, 2.2 and 2.3 present the clustering patterns of existing works based on some of the major subcategories of control engineering approaches proposed to manage different software systems.

From Table 2.1, it is evident that the control engineering approaches have been applied in many different application domains[2]. An interesting observation is that depending on the application domain, the performance variable has to be selected carefully to design the control system. With respect to the performance variables, the response time is one of the major performance properties considered in the existing work. The reasons for this could be that the response time is (1) the user perceived performance attribute of the system (2) one of the attributes specified in agreements and (3) useful to formulate a set point tracking control problem [191]. The processor and power utilization are the other performance variables looked at by a large number of papers, in particular in data center environments.

Table 2.2 provides an interesting classification based on the modelling mechanism used. The black-box modelling mechanism is the dominant modelling mechanism used in most

[2]Some of these works relate to multi-class shared resource systems, which will be covered in Section 2.3.2

Table 2.1: Classification of paper references according to the application domain and performance variable

	Data center	VM	Data storage	Middleware
Response time	[39, 63, 83, 94, 118, 119, 120, 121, 178, 179, 208, 224, 231, 233, 235, 236, 238, 246, 266, 267]	[84, 120, 121, 126, 140, 178, 179, 231, 232, 233, 234, 235, 236, 248, 267]	[36, 48, 52, 74, 95, 107, 144, 181, 217]	[9, 22, 38, 50, 57, 82, 92, 93, 103, 104, 106, 108, 112, 113, 114, 134, 139, 141, 145, 150, 158, 180, 189, 193, 194, 197, 198, 199, 206, 211, 216, 240, 242, 249, 253, 261, 262]
Throughput	[128, 178]	[71, 128, 142, 178]	[161]	[34, 103, 106, 108, 130, 158, 250]
Progress/Miss ratio		[184, 185, 186]		
Power Utilization	[39, 118, 119, 120, 121, 124, 200, 226, 227, 231, 237]	[71, 120, 121, 142, 169, 231, 232]	[37]	[71, 93]
Processor Utilization	[8, 132, 133, 179, 200, 235, 236, 237, 238, 251, 266, 267]	[84, 179, 235, 236, 251, 267]	[132, 176]	[9, 38, 46, 61, 88, 214]
Hit rate/ratio			[62, 115, 151, 152, 153, 247, 261]	

of the existing work compared to the analytical and queuing models. Black-box models (typically, linear time invariant models) are useful because there are no first principle models to describe the versatile behaviors of software systems. Analytical and queuing models, which can be classified as first principle models, make assumptions with regard to the workload distributions and behavior of the underlying software system, while their complexity imposes practical difficulties in the application of classical control engineering techniques.

Table 2.2: Classification of paper references according to the modeling mechanism

Queuing	Black-box	Analytical
[1, 22, 23, 50, 78, 92, 93, 94, 111, 112, 113, 118, 119, 120, 121, 126, 134, 139, 150, 198, 201, 206, 224, 225, 235, 236, 242, 246, 249, 262]	[3, 6, 8, 9, 10, 36, 37, 38, 39, 46, 48, 49, 52, 56, 57, 58, 59, 61, 62, 63, 71, 71, 74, 82, 83, 84, 85, 95, 96, 97, 98, 99, 100, 103, 104, 106, 107, 114, 120, 124, 128, 130, 132, 133, 134, 135, 139, 140, 141, 142, 144, 145, 147, 151, 152, 153, 158, 161, 169, 176, 178, 179, 180, 181, 182, 184, 185, 186, 189, 193, 194, 197, 198, 200, 205, 208, 209, 210, 212, 216, 217, 226, 231, 232, 233, 234, 235, 236, 237, 238, 243, 247, 248, 251, 252, 253, 254, 255, 255, 259, 261, 264, 266, 267]	[2, 11, 54, 58, 60, 79, 82, 116, 118, 119, 121, 123, 131, 136, 146, 148, 154, 157, 172, 199, 214, 225, 227, 228, 229, 230, 260]

A classification of related works based on the dimension of the control system and control scheme or algorithm is presented in Table 2.3. There are three major observations. Firstly, the single-input-single-output (SISO) control systems are typically implemented with the variations of PID control. This is because PID control is more suitable to achieve SISO control objectives due to the simplicity and robustness of that scheme. Secondly, the complex multi-input-multi-output (MIMO) control systems are implemented using the model predictive controllers (MPC) or linear quadratic regulators (LQR), which have optimization programmers inbuilt in the controller to compute the control decisions under the competing multiple objectives. Thirdly, when there are MIMO control objectives to be achieved, either multiple SISO control systems or a single MIMO controller have been implemented in the existing works.

It is worth noting that none of these control approaches have investigated methodologies to represent the nonlinear dynamics of the software systems explicitly. This research gap is one of the main focuses of this thesis.

Table 2.3: Classification of paper references according to the control system dimension and type of control scheme

	SISO	Multi-SISO	MIMO
Fixed (PID)	[3, 11, 22, 36, 52, 57, 62, 71, 78, 79, 85, 88, 95, 96, 111, 112, 113, 114, 124, 130, 131, 132, 133, 136, 141, 144, 154, 169, 176, 180, 182, 184, 201, 209, 211, 217, 246, 248, 250, 264]	[8, 9, 26, 84, 97, 115, 134, 145, 147, 150, 153, 179, 181, 206, 229, 237, 242, 249, 261, 262]	[58, 229, 231, 232]
Adaptive	[6, 49, 107, 126, 139, 141, 151, 152, 185, 186, 238, 247, 253]	[10, 179, 235, 236, 243]	[63, 71, 74, 103, 104, 106, 108, 128, 140, 142, 161, 178, 254, 255, 255]
MPC	[1, 93, 197]	[23, 224, 229]	[37, 54, 58, 94, 118, 119, 120, 121, 148, 193, 225, 226, 227, 228, 229, 230, 231, 233, 260]
LQR	[234]		[38, 39, 46, 48, 60, 61, 63, 71, 74, 83, 98, 99, 103, 106, 123, 128, 140, 142, 158, 178, 234, 252, 254, 255, 255]
Reconfiguring	[189, 208, 215]	[97, 146, 147, 243, 259]	[212, 232]

2.3.2 Applications of Control Engineering Approaches in Multi-class Shared Resource Environments

In this section, we focus on the related work that proposes dynamic resource management techniques to achieve the absolute and relative performance management objectives in multi-class shared resource environments.

Absolute performance management approaches

Many of the existing approaches address absolute performance management problem. However, most approaches focus on a single control objective (non multi-class systems) [47, 61, 81, 182, 208, 238, 266]. Absolute performance management techniques for the case of multi-class shared resource environments can be found in [41, 145] for connection delay management in web servers and in [180] for database connection pool management.

However, these approaches design multiple independent SISO control loops to manage the performance and resources in a multi-class system. They also ignore the interactions between the inputs and outputs in modelling and control, which could lead to the performance issues [46]. For instance, in systems with a limited amount of resources, increasing the resources for one class implies reduction of resources of another class. These dependencies are not captured in such SISO approaches. Another shortcoming is that they use linear modelling and linear fixed gain control methodologies, disregarding the nonlinear behavior of the system.

The MIMO model based adaptive control is proposed in [105, 108] with the equality constraints on the total resources[3]. However, the equality constraints make it hard to design system identification experiments because of the dependencies between inputs. Furthermore, absolute performance control is also utilized to manage data centers with three or less customer classes in [118, 177]. Work of Kusic et al. in [118] relies on accurate measurements of arrival rates of all classes. Due to the stochastic nature of the workloads obtaining accurate measurements is problematic, which may lead to runtime management issues and temporal instabilities. In addition, the computational and time complexity increases exponentially when the range of input and the prediction horizon increases, because of the large state space exploration problem that has to be solved at each sample instance. Consequently, they require large sampling intervals even to produce sub-optimal solutions. Padala et al. in [177] proposed an adaptive control approach to manage performance properties and hardware level resources of the multi-tire web applications deployed in virtualized data centers. This work suffers due to the limitations of adaptive control, in particular when the workload conditions change rapidly and the requirement of persistence of excitation conditions [13] are violated, the system may encounter large transient responses and instabilities [186, 189].

Relative performance management approaches

The relative performance management scheme is important but a hard control problem, due to the consideration of the ratios of independent system outputs of multiple classes. The relative performance management schemes with feedback control have been utilized to manage web servers [41, 145, 150, 180], storage systems [151] and data centers [177]. The

[3]That is the summation of resource capacities computed for each class has to exactly equal to the total resource amount. More details can be found in Chapter 3.

works in [41, 145, 150, 177, 180], utilize linear model and feedback control, limiting the operating range of the controller to a narrow region that can be linearized, disregarding the severe nonlinearities that exist in the relative management scheme. This approach becomes an issue when the sudden disturbances and conditions move the system to operate away form that the narrow region. Ying et al. in [150] discuss the nonlinearities in the input-output relationship and related issues of relative management scheme. They investigated three different ways to formulate the input and output valuables of the relative management scheme in order to reduce the nonlinearity, while maintaining the scalability of that scheme and the applicability of feedback control. Their final results indicated that taking the ratios of the response time and resource capacities of consecutive client classes based on the priority is the most suitable and effective setting. As a result, that approach was used in their subsequent publications [145]. However, their work was limited to linear fixed gain control. Furthermore, [151, 177] utilize adaptive control in their work to design relative performance management systems.

From the classifications of Section 2.3.1 for general software systems and the analysis in this section with respect to multi-class shared resource environments, typically the control system design approach taken by the existing work is based on a linear model and linear controller. However, to achieve the absolute and relative performance management objectives of multi-class shared resource environments, the control system has to deal with the nonlinear behavior of the system. This is because with respect to the absolute and relative management objectives, the relationship between the performance properties such as response time and throughput (the performance variable controlled) of a single client class and the resource allocation (the manipulated variable) is known to be nonlinear [41, 81, 106, 150]. In addition, the characteristics of the nonlinearity depend on the workload intensity as well as the level of the system stack at which the resources being shared (for instance, hardware and middleware). Consequently, by neglecting nonlinear dynamics, the linear SISO and MIMO control engineering approaches become inadequate when the system operation condition changes.

Adaptive control approaches, both SISO [81, 151, 265] and MIMO [108, 178] can be categorized as nonlinear approaches, since they identify a linear model online capturing the change of operating conditions. However, these approaches also assume persistently exciting conditions and the operating conditions to change slowly. These conditions cannot be assumed in multi-class shared resource environments due to the unpredictable and sudden variations in workloads and operating conditions. As mentioned before, under violations of these assumptions, the management provided by an adaptive control system may show large transient responses and temporal instabilities leading to significant performance and resource management issues [13, 186, 189, 247].

Therefore, in order to design and implement absolute and relative performance management systems for shared resource software systems that share resources at different levels of the system stack and work in a wide spectrum of operating regions with unpredictable workload conditions, nonlinearity in the system dynamics must be sufficiently addressed in the process of modeling and in the process of control system design.

2.4 Off-the-Shelf Design Support for Control Systems

A major impediment for building control systems for software platforms is the lack of implementation frameworks supporting the use of control engineering techniques [203]. These support tools could significantly reduce the knowledge, time and cost required to develop control-based management systems from scratch. Currently, the existing research uses the general purpose software such as Matlab[4], Mathematica[5] and LabVIEW[6] which provide extensive design, analysis and simulation support in the initial stages of the design of a control system. However, it is hard to deploy the executables of the control systems implemented out of these general purpose tools directly into the production software systems. This is because the additional components and runtime environments have to be installed in order to successfully deploy these executables, which is an added runtime performance and management overhead (see work in [149, 177]).

ACME [16] provides a model driven engineering support to generate code for basic controllers with configuration capabilities in different programming languages. However, the extendibility is limited because the code interpreter may require significant modifications to consume new extensions. In [47] an approach is described to automate the design of a control loop by encapsulating the control engineer's expertise into several software agents. Based on the complexity of the requirements and system characteristics such an approach is difficult to apply without a human expert in the design process of even a simple control

[4]http://www.mathworks.com/products/matlab/
[5]http://www.wolfram.com/mathematica/
[6]http://www.ni.com/labview/

system. So that, instead of automating the entire design process, providing supporting tools is more useful to aid the development of control systems.

2.5 Extending State-of-the-Art

This section compares the work proposed in this thesis with the state-of-the-art approaches reviewed in previous sections.

We formulate the multi-objective absolute performance and resource management problem with inequality constraints on the total available resources. Then, well-established MIMO system identification technique is used to model the system, which captures the interactions between the inputs and outputs. This modelling approach therefore does not have assumptions on the arrival rates or probabilistic distribution of the workloads compared to the existing work. However, the main difference of our work is the consideration of the nonlinear behavior of the system explicitly in the design of the MIMO control system. In particular, we investigate two approaches, which include the implementation of nonlinear block-oriented model or multi-model based control systems. Further, model predictive control is used in this work because of its ability to manage MIMO systems with complex constraints.

With respect to relative management objectives in multi-class shared resource environments, this thesis also proposes two nonlinear control system design approaches. Here, the relative performance management problem formulation recommended by Ying et al. in [150] will be used. In addition, avoiding the aforementioned limitations of the linear fixed gain or adaptive control approaches, we propose a nonlinear block-oriented model and a control system with compensators to reduce the impact of severe nonlinearities on the management system. As a consequence, the system can be effectively linearized and a gain-scheduling control mechanism (see, Appendix B) can be implemented avoiding the requirements and assumptions of adaptive control. As the second approach, we investigate a multi-model control system design approach to improve the performance of the control system in multiple distinguishable regions, while maintaining the robustness of the system.

These novel approaches also satisfy all the solution characteristics listed in Section 1.3. Furthermore, most importantly these approaches have shown that they sufficiently address the dominant nonlinear issues exist in the multi-class shared resource environments compared to state-of-the-art approaches.

Apart from the investigation of nonlinear control approaches, one of the outcomes of this thesis is an off-the-shelf class library to support the design, implementation and deployment of control systems proposed in this thesis. However, the development support of the class library is not only limited to multi-class shared resource systems. It can also be used in the development of control system for other types of software systems. We present a general engineering process to build control systems using the class library, and illustrate the extensibility and configurability of the class library. In addition, it also facilitates the engineering process of the control systems by reducing the implementation effort plus the knowledge required of the software engineers. The class library can be directly used for the implementations and deployments in the systems build in Java and .Net, without any additional components or runtime environments.

2.6 Summary

This chapter analyzed the existing work that proposed techniques to design, implement and manage multi-class shared resource software environments. Figure 2.2 illustrates a taxonomy which is the analytical framework we used to classify and review the existing work. Firstly, we looked at the general non-performance management aspects such as maturity levels, customizability, data design and so on. Then, we moved to the performance management approaches, which were divided into open-loop and closed-loop mechanisms. From the closed-loop approaches, dynamic resource management is our focus area. Those approaches were further divided into non-control and control engineering approaches. Control engineering approaches can be classified as linear or nonlinear. Compared to the existing nonlinear approaches, the novelty of the proposed approaches in this thesis is that we focus on block-oriented model based control and multi-model based control systems to manage the performance properties and resources of multi-class shared resource systems. None of the existing work has investigated such techniques. In addition, we presented an overview of the support tools in the existing literature to design, develop and deploy control systems in software environments.

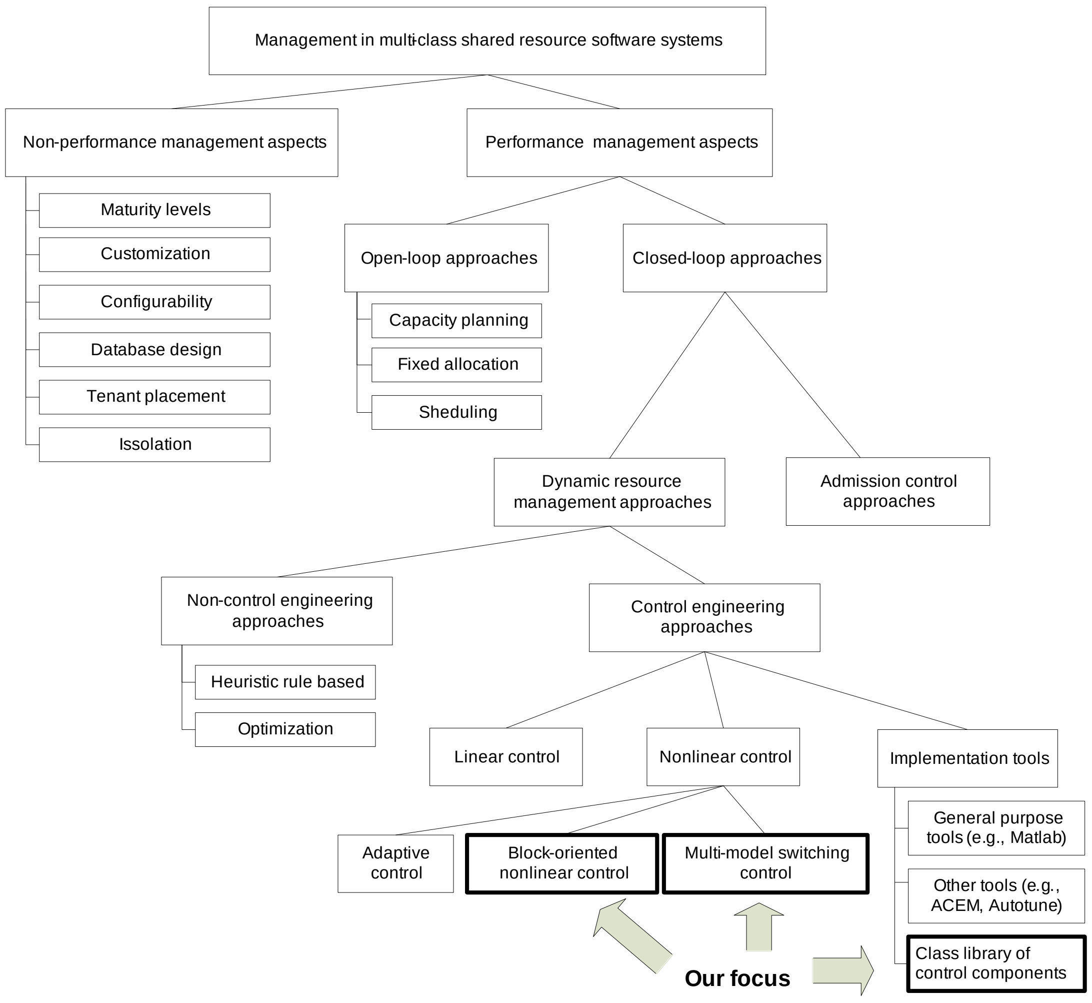

Management in multi-class shared resource software systems
Non-performance management aspects
Maturity levels
Customization
Configurability
Database design
Tenant placement
Issolation
Performance management aspects
Open-loop approaches
Capacity planning
Fixed allocation
Sheduling
Closed-loop approaches
Dynamic resource management approaches
Admission control approaches
Non-control engineering approaches
Heuristic rule based
Optimization
Control engineering approaches
Linear control
Nonlinear control
Implementation tools
Adaptive control
Block-oriented nonlinear control
Multi-model switching control
General purpose tools (e.g., Matlab)
Other tools (e.g., ACEM, Autotune)
Class library of control components
Our focus

Figure 2.2: Taxonomy of existing research and the focus of this thesis

Chapter 3

Research Methodology

3.1 Introduction

This chapter overviews the research methodology followed in this thesis. We begin with the formulation of the runtime performance and resource management problem for multi-class shared resource software systems. The approach and validation mechanisms adopted are then presented briefly. Finally, we cover the details of the simulation environment and experimental case studies.

3.2 Runtime Management Problem Formulation

Section 3.2.1 lists the assumptions and requirements of this work, while Section 3.2.2 defines the management problems that are the focus of this thesis.

3.2.1 Assumptions, Scope and Requirements

We make the following assumptions with respect to multi-class shared resource environments. Based on the assumptions, the requirements and scope of this work will be discussed.

- **Type of the multi-class system.** We focus on the multi-class shared resource environments exist at the application or middleware levels[1]. At the software level, we assume a software application receives workloads from different client classes (classified according to the business objectives), and the available software level resources (e.g., threads and database connection pools) have to be shared between these classes to achieve the performance objectives. Similarly, at the middleware

[1]Although the proposed approaches in this thesis have been already applied for hardware level resource management (see [192]), it is out of the scope of this thesis.

level, we assume a middleware system is providing the ability to deploy software applications and services of multiple customers or organizations.

- **Type of the scheme.** We assume that depending on the performance management objectives, a suitable management scheme is selected and known at the design time. Two dominant types of management schemes are investigated in this work, namely the absolute and relative management schemes. However, the control objectives within these schemes can change at runtime.

- **Performance property.** In this work, we use the response time as the performance property that has to be maintained by the management system. It is the main performance property perceived by the end users of a software system, consequently it is the main performance parameter controlled by most of the existing works (e.g., [32, 40, 81, 108, 118, 145, 178]). According to the survey results in [191], response time is one of the major performance variables used to design control systems in the existing literature.

- **Measurements and decision implementation mechanisms.** We assume that the system has mechanisms to compute the response time of each class in predefined time intervals. This means that the sensors have to be provided either by the target system or the target system itself has the capability to deploy sensors without a significant impact to the rest of the business functionalities. Similarly, we assume that the system has the capability to adjust the partitions of the bottlenecked resources at runtime without restarting the system. For instance, the state-of-the-art vitalization products such as Xen and VMware have already implemented such partitioning schemes in order to enable runtime management by the external entities. In the case of the middleware and software levels, such partitioning and scheduling schemes may not be available. In that case we implement a proportional resource partitioning and scheduling scheme adopted by existing work [145]. See, Section 3.5.2 for more details.

- **Dimension of the system.** The number of classes (say n) served by a single instance of a multi-class system is known at the design time. In this work, we focus on a control system with a static architecture. In the cases where the number of tenants changes over time the control architecture has to be dynamically reconfigured [191]. Such dynamically reconfigurable control systems will not be considered in this thesis.

- **Limits of resource.** We also assume that system profiling has been performed, and the number of total available resource units (say S_{total}) in the system is determined based on the response time requirement of each class. This property is important to maintain the response time within acceptable bounds.

- **Overload management.** We assume that when a class has overloaded a single instance of a multi-class system, a portion of the workload will be rejected to avoid instabilities due to unbounded growth of the workload. If this assumption is not acceptable for the application, an alternative approach would be to migrate the aggressive class to another under-loaded multi-class system instance with sufficient resources or dividing the workload between multiple instances. These techniques are out of the scope of this thesis.

- **Major bottlenecked resource.** There might be multiple resources as the candidates for runtime management. In the ideal scenario, all these resources have to be allocated efficiently between all the classes. However, in such situations the dimensionality of the problem becomes complex. In this work, we only consider the management of a single and the main bottlenecked resource at the level of the system stack where the resource sharing occurs.

3.2.2 Management Problem Definitions

Assume that n customers (here on we refer to them as classes 0, 1 . . ., n-1) are interested in the services provided by a multi-class shared resource system. A performance objective of the service provider is to maintain the specified response time levels or those based on priorities of these classes. These objectives are derived from the business requirements and are to be achieved using the S_{total} number of resources available in the system to service the incoming time varying workloads of these classes. In addition, some amount of resources is reserved during the entire period of operations for each class, in order to avoid starvation of resources and to provide minimum amount of service levels. That is, the management system must reserve a minimum number of resources (say, $S_{i,min}$, where

$i = 0, 1, \ldots n$) that have to be maintained for a specific class i, where $\sum_{i=0}^{n-1} S_{i,min} < S_{total}$. However, under sudden workload changes the discretionary resources $S_{total} - \sum_{i=0}^{n-1} S_{i,min} > 0$ are shared among these classes in order to achieve the required performance objectives. Therefore, this resource management approach is a hybrid approach of resource reservation and dynamic partitioning. Such hybrid techniques are used in [177, 257] and recommended by [75, 145] in the cases of multi-class shared resource systems. An assumption in this hybrid resource management scheme is that there are sufficient discretionary resources (i.e. $S_{total} - \sum_{i=1}^{N} S_{i,min} > 0$) to achieve the required performance objectives when the workloads of these classes exceed the normal workload conditions. Otherwise, the resource management scheme converges to a pure fixed resource allocation scheme.

Control objective: The main objective of this management system is to maintain the average response times $R_0(k)$, $R_1(k)$, …, $R_{n-1}(k)$ under varying/unpredictable workload conditions of n classes based on the absolute and relative performance management objectives (see Sections 3.2.2.1 and 3.2.2.2), while dynamically adjusting the resource partitions or caps ($S_0(k)$, $S_1(k)$, …, $S_{n-1}(k)$), where k is the sample time period. Furthermore, the management system should honour the following constraints related to the total amount of resources and per-class resource reservations at all times.

$$\begin{aligned} &S_0(k) \geq S_{0,min},\ S_1(k) \geq S_{1,min},\ \ldots,\ S_{n-1}(k) \geq S_{n-1,min} \\ &S_0(k) + S_1(k) + \ldots + S_{n-1}(k) \leq S_{total} \end{aligned} \tag{3.1}$$

Using the above general problem definitions, we now specify the absolute and relative performance management objectives in the following subsections.

3.2.2.1 Absolute Performance Management Scheme

In the absolute performance management scheme, we assume that n classes are interested in the services provided by the shared resource environment. The business objective of the system is to maintain the average response time of the workloads R_0, R_1, …, R_{n-1} of these classes at an agreed level $R_{SLA,0}$, $R_{SLA,1}$, …, $R_{SLA,n-1}$ as defined in the service level agreements. This objective will be achieved by calculating the resource caps $S_0(k)$, $S_1(k)$, …, $S_{n-1}(k)$, without violating the constraints defined in equation (3.1).

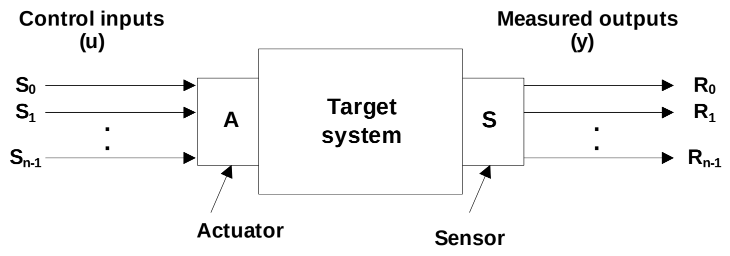


Figure 3.1: Block diagram of a target system controlled by the absolute performance management scheme

In order to achieve these objectives, as illustrated in Figure 3.1 a sensor and actuator have to be deployed in the target system, which provides response time measurements and implements resource allocation decisions in the system respectively.

3.2.2.2 Relative Performance Management Scheme

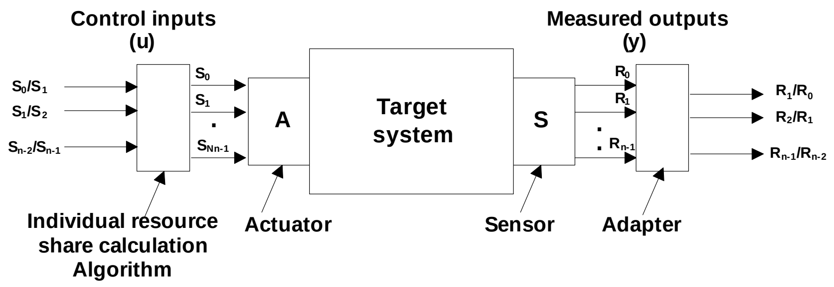


Figure 3.2: Block diagram of a target system controlled by the relative performance management scheme

Let R_i, P_i be the response time and the specified differentiation factor respectively of a class $i = 0, \ldots n-1$. Between the pair of classes i and j, the objective of the relative performance management scheme is to maintain $\frac{R_j}{R_i} = \frac{P_j}{P_i}$ (i $= 0 \ldots n-1, i \neq j)$ at runtime under varying workload conditions. For instance, $\frac{P_1}{P_0} = 2$ means that the response time of class$_1$ is to be maintained twice as of class$_0$ (i.e., class$_0$ has high priority compared to class$_1$). In order to achieve this objective, the resource cap ratio $\frac{S_i}{S_j}$ will be calculated and then converted to individual resource caps $S_0(k)$, $S_1(k)$, …, $S_{n-1}(k)$ adhering to the constraints given by equation (3.1).

Figure 3.2 illustrates the inputs and outputs required to achieve the relative management objectives. Similar to absolute management scheme, a sensor and actuator have to be integrated to the target software system. However, in this scheme, the inputs and outputs have to be transformed to ratios. Firstly, the values of the differentiation factors of these classes are arranged in the ascending order (i.e., descending order of priority). Then, each consecutive class is paired and the ratios are computed between these pairs at the input and the output as shown in Figure 3.2. Two adaptors are connected to the system before the actuator and after the sensor to implement these conversions. The adaptor before the actuator also decodes the ratios back to the individual resource caps using an algorithm[2].

3.3 Approach Overview

To address the management problem set out in Section 3.2.2, a management system needs to be designed and implemented. In this section, the management system architecture and the high-level approach taken by this thesis will be described briefly.

Figure 3.3 illustrates the abstract architecture of a multi-class shared resource software system and its' control loop. This is the same abstract architecture used in the existing works as defined in survey [263]. We start from the top of Figure 3.3 and describe the responsibilities of key components in this system.

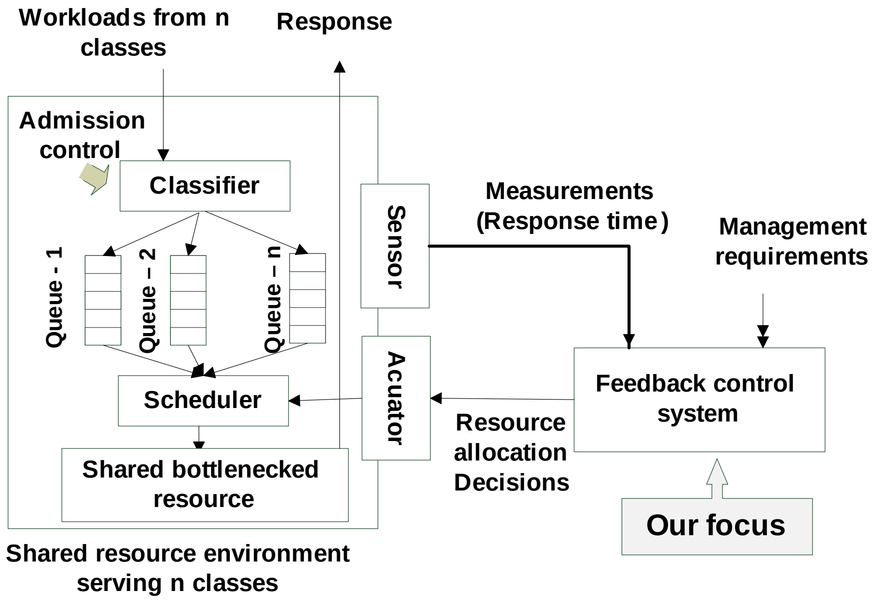


Figure 3.3: Abstract architecture of the multi-class shared resource system and control loop for dynamic performance and resource management

[2]This algorithm is covered in Appendix D in detail.

Workloads. Depending on the behavior of the clients, the workloads of n classes invoke the business operations of the shared resource software system.

Classifier. When a request for service reaches the system, the classifier component classifies the request according to the class and pushes it into the relevant class queue. The classification is performed using the class-specific unique identifier sent with each request of the workload.

Multi-queue system. The multiple queues act as a container of requests of each class waiting to be served by the system. This multi-queue based setting is called *class-aware queuing*, which is important to avoid interference between classes and addresses limitations of the single queue based setting. The moment a request is received, the admission control is also performed looking at the length of each queue to maintain the response time within bounds. The maximum queue length is specified by the designer after looking at the maximum tolerable response time of each class.

Scheduler. The scheduler schedules the requests in a FIFO fashion depending on the resource availability. A responsibility of the scheduler is to schedule the requests within the available resource caps prescribed by the management system accurately in the regular time periods (called the sample time period).

Sensor. A sensor is deployed in the system to measure the average response time of each class. The sensor also sends these measurement data to the management system at predefined time periods.

Actuator. An actuator is implemented in order to inform the scheduler about the resource allocation decisions in each sample time.

Management system. The management system in Figure 3.3 is responsible for making the resource allocation decisions by looking at the measurement data, management objectives and constraints defined in Section 3.2.2.

The main focus of this work is to design and implement such a management system which makes the resource allocation decisions while achieving the absolute and relative performance management objectives at runtime under changing conditions. Our approach is to treat the multi-class shared resource environment as a target system and then compose the management system with a feedback control system. Consequently, the advantages of the well-established control engineering techniques can be used to design the management system in a systematic way. However, in contrast to the existing approaches that are based on linear design methodologies, the main novelty of the proposed feedback control methodology is that it explicitly considers the *nonlinear dynamics* in the management system design, which had not been looked at in the existing literature thus far.

In particular, we present nonlinear control system development techniques based on the variations of Hammerstein and Wiener block oriented model to explicitly capture the nonlinear dynamics of the relative and absolute management systems. Furthermore, instead of representing the system with a single model and controlling the system using a single controller, we present methods to implement the control systems consisting of multiple models and controllers together with the dynamic switching capabilities.

3.4 Validation Mechanism

This section overviews the methodologies used to validate and quantify the effectiveness of the management schemes proposed in this thesis.

Selection of the target system. In order to investigate and characterize the behavior and design a control system, a target software system is essential, i.e. a multi-class shared resource system under study. For this purpose, the existing works have utilized either simulation or experimental case studies of physical software environments (see our survey [191] for quantitative analysis). These two techniques have their own advantages and disadvantages. A simulation environment is useful to evaluate and compare the existing control methodologies with the proposed technique in a controlled environment and conditions. This is because multi-class shared resource environments deployed in physical hardware resources provide variable conditions, even under same settings/inputs due to the noisy and jittery condition of the underlying hardware, operating systems and virtualization platforms. A limitation of simulation environments is that it abstracts away some of the random behavior/noise from the analysis.

As a consequence, this thesis utilizes both simulation studies and experimental case studies to investigate the behavior and then implement and evaluate the proposed management system. For this purpose, a representative simulation system and several multi-class shared resource case study systems sharing resources at different levels of the system stack (e.g., software application, middleware level) have been built in this thesis. The details of these target systems are presented in Sections 3.5 and 3.6. Furthermore, in the evaluations, different control objectives, workload conditions and operating conditions (called

'Cases') are simulated to investigate the behavior of the control system, under versatile settings.

Benchmark to compare. In order to compare the improvements, effectiveness and efficiency of the proposed nonlinear control solutions, we use linear control methodologies, which are the current state-of-the-art. However, different control algorithms can be used to implement a control system (see Appendix B). Although the proposed modelling mechanisms do not depend on the selected control algorithm, different control algorithms might behave slightly differently at runtime under certain operating conditions. Investigation of such effects is out of the scope of this thesis. We used the survey results [191] as a basis to select the appropriate control algorithm based on the problem at hand. With respect to the existing research, in this work, the proportional-integral-derivative (PID) control algorithm will be used to design SISO control solutions, and for MIMO control solutions, model predictive control (MPC) formulations will be used (see Appendix B for more details about these controllers and [191] for statistics).

Comparison metrics. Another important parameter in the validation is the metrics of comparison. We use the *control error* and system *output* signals for this purpose. The control error is the difference between set point signals (desired values) and measured output signals (obtained values). If a control system has achieved the specified management objectives, the control error should be as low as possible. We use a Sum of Square Error (SSE) statistic to evaluate and compare the management provided by the designed control systems. This statistic includes the capabilities of the control system for reacting to the disturbances (overshooting, settling time) and steady sate behavior. Furthermore, we use Minimum (MIN) and Maximum (MAX) statistics of the measured output signals to investigate the variations under sudden disturbances. These two statistics are widely adopted as a measure of overshooting and disturbance rejection capabilities.

Expected outcomes of the validation. The expected outcome for a particular target software system (simulation or case study) under different operation conditions is that the proposed nonlinear control solution should provide better or no-worse statistics for most conditions compared to the linear control counterpart. The better control system for a given experiment shall produce the lowest SSE and MAX, and the highest MIN statistics.

Validation of supporting tools. One of the contributions of this thesis is the off-the-shelf class library of control components to aid the design of control systems for shared resource software environments. In order to validate this claim, we also conduct an empirical study with a group of software engineers to quantify the facilitation provided by this class library to reduce the control system implementation costs and knowledge requirements. The experiment settings are briefly explained below (see Chapter 7 for more details and results).

First, a control system implementation task is documented and given to each participant (software engineer). Each participant has to complete this implementation task in two ways: 1) without using any support tool and 2) with the support of our class library. When the participants complete the task both ways, the source code used in each implementation will be used to compute the lines of code required in each implementation. Furthermore, the time taken for each implementation is also considered. Finally, the average statistics of all the participants are used to make conclusions on the facilitation provided by the class library as opposed to a case where there is no support tool. The expected outcome of this experiment is when our class library is used, the lines of code and the time required for the implementation task is less than when the same implementation is carried out without any support tools.

3.5 Simulation Environment

In this section, we introduce a simulation environment to construct different forms of multi-class shared resource environments, which will be used to investigate the nonlinear characteristics and apply the nonlinear control methodologies proposed in this thesis. Since we consider shared resource environments at multiple levels, in particular at the shared middleware (resources shared are worker threads, cache) and the software application level (resources shared are database connections, domain specific resources), implementing a single case study to cover all these levels is difficult. Consequently, a simulation model is used to represent the general characteristics of these multi-class shared resource environments.

3.5.1 Characteristics and Requirements of a Simulation Model

The main purpose of the simulation system is to represent a model of a multi-class shared resource environment, which can be used to generate measurements and draw conclusions from those measurements.

The requirement for a general simulation model is as follows:

1. same consistent behavior under same input settings,
2. ability to validate the correctness of the simulation model,
3. accurate average statistics of the required system parameters,
4. modifiability, extendibility and scalability,
5. fast and efficient execution.

The requirements of a simulation model, which represents the abstract architecture of a multi-class system (see Figure 3.3) are as follows:

1. simulate multiple (n) classes,
2. accurate measurements of the system outputs (e.g. response times) of each class,
3. valid implementation of the resource allocation decisions,
4. ability to simulate different types of workloads in different intensities over the period of simulation.

3.5.2 Implementation of the Simulation Model

The operations of the multi-class shared resource environments are driven by events occurring in a chronological order. For instance, response to a single request can be described in several events, including request arrival event, request scheduling event and end of resource utilization event. These events in turn update the state of the system as well. One of the tools available to build simulations of such systems driven by events is *discrete event simulation* [18]. Discrete event simulation is widely used to test and analyze new systems and policies before they are implemented as production systems.

A DES model of a system consists of entities (e.g., requests, queues and resources), attributes, events (e.g., request arrival and departure) and activities (operation invocations, statistic collection). For a given time instance, the DES model has a snapshot of the system, which is updated based on the events that is scheduled to happen in that time instance. After the relevant events have taken place, the model is advanced to the next time instance, and the same process is continued till the end of the simulation. During the simulation or at the end of the simulation, the statistics can be gathered to analyze the results of the simulation. Generally, the DES model can be designed in an *event-oriented* and a *process-oriented* point of view. In the event-oriented technique the DES model designer takes the events of the system and how they affect the system state variables as major concerns. On the other hand, the process oriented point of view enables to model the entities, their processes and how the inter-process communication takes place. The event-oriented design produces simulations that can execute faster compared to process-oriented design, however at the expense of modularity, extendibility and the understandability of the model.

3.5.2.1 DES Model of a Multi-class Shared Resource Environment

In this section, we provide implementation details of the simulation environment developed following the guidelines provided in [18]. Here, we have taken the process oriented design technique because it provides modularity, extendibility and convenience to design using the general purpose object-orient programming languages such as Java and C#.Net. We use stochastic inputs and variables in this simulation to represent the behavior of a multi-class shared resource system.

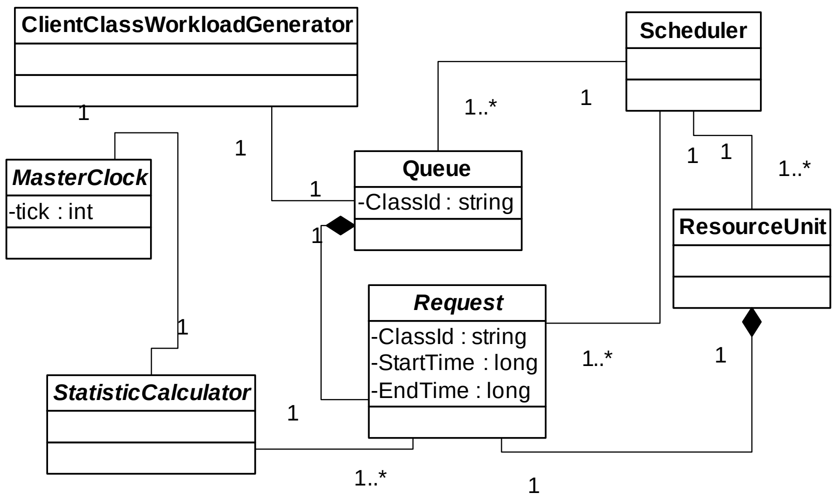


Figure 3.4: High-level class diagram of the simulation model

Figure 3.4 shows the high-level UML (Unified Modeling Language) class diagram of the DES model. The DES model consists of the following entities (components) corresponding to the abstract architecture of Figure 3.3.

MasterClock. This component keeps track of the current time instance of the system and advances the time after all events and activities specific to the current time instance have taken place. It triggers events on the tick (smallest time unit) and major tick (i.e., 1000 ticks), which are similar to a millisecond and a second respectively.

Request. This represents a client request flowing through the simulation model. It has the properties of ClassId, start time, end time and processing time. The processing time is determined by a probabilistic distribution specified by the designer.

ClientClassWorkloadGenerator. This component generates workloads for a specific class. It needs a ClassId, workload script and the corresponding queue instance at the initialization. Then, this component analyses the workload script at each time tick and generates the required number of requests that have to be sent to the system. This component can be configured to simulate deterministic, stochastic and real-world workloads.

Queue. The Queue component is used to represent the client queues. It is a container for the requests generated by the ClientClassWorkloadGenerator and ordered in a first-come-first-out fashion. A simulation model of a multi-class system needs n such Queue instances to represent n classes. Further, the designer can implement the bounds for each queue, which limits the length of the queue. The requests will not be admitted to the queue if this limit has reached.

ResourceUnit. The ResourceUnit entity is an abstraction of a resource unit in a multi-class system. It simulates the time period a resource is reserved, occupied or provisioned to serve a request. For instance, a resource could be worker threads, processing instances, CPU cycles or cache. It has the currently served request, serviced ClassId and status (idle or working) as attributes. At each tick, a ResourceUnit instance simulates the processing time specified on the request it is currently serving. When the request has utilized the resource for the specified period of time, it is assumed to be sent back to the client after stamping the end time.

Scheduler. The scheduler implements the required resource allocation decisions. For instance, if the decision is to maintain 15 and 5 resource units for classes 0 and 1 respectively, this component implements these decisions until the next decision is made. It has the access to the Queue instances of each class, resource units and other state variables. In each tick it executes the following algorithm for each class.

Say S_i and i_{util} are integer variables representing the allocated resources of the i^{th} class and currently utilized resources by the i^{th} class, respectively. The number of resources that can be allocated for i^{th} class at the current time instance is calculated by $Dif_i = S_i - i_{util}$. The Scheduler gets the $Dif_i > 0$ amount of requests from the i^{th} Queue component and then the ResourceUnit instances are initialized with those requests. At the same time, i_{util} variable is updated.

StatisticCalulator. This is the component that computes the measurements required by a management system to observe the status of the system. In particular, it calculates the average response time, throughput and resource utilization for each class on specified time periods. It also has a list of completed requests for each class, which is populated by the ResourceUnit class after servicing the requests. The designer specifies the time interval (so-called sample period) to calculate the statistics. The statistic report generated will be sent to the management system in order to make resource allocation decisions.

The following equations summarize how some of these statistics are calculated for class i.

Given the completed request list for class i, say $List_i$, the throughput of that class is $TP_i = Count(List_i)$, i.e., the number of items in the list. The response time of the j^{th} request in $List_i$ is $r_{i,j} = r_{i,j}.endtime - r_{i,j}.starttime$.

The total response time of all requests in the list is calculated as follows:

$$Tot_i = \sum_{j=0}^{Count(List_i)-1} r_{i,j} \tag{3.2}$$

The average response time is then calculated by $R_i = \frac{Tot_i}{TP_i}$

MainProgram. The designer can use this class to implement the required multi-class system simulation and experiments depending on the requirements. The number of classes and Queue instances has to be created and then the required workload scripts have to be specified in class specific workload generator objects. In addition, the number of resource units that is available in the system has to be specified in the scheduler. Furthermore, the probability distributions to simulate resource reservation time and sample period has to be given depending on the simulation objectives.

3.5.2.2 Assumptions

The assumptions made in the implementation of this DES model are as follows:

1. Typically, the resource allocation decisions for each class have to be implemented at each sample instance. However, some of the resource units may be utilized to serve the requests of a certain other class at that time instance. There are two different ways to implement the resource allocation decisions in such a situation, i.e., *preemptive* and *non- preemptive*. In the preemptive setting, the number of over utilized resources is forcefully taken away in order to allocate them to the specified class. This is a complex policy, which will cause jittery behavior in the system measurements, inconsistent states in the transactions and additional overhead on the shared resource system (e.g., state management) at runtime [143, 145]. In contrast, in the non-preemptive setting, the resource is taken away once the request being processed has been completed. The non-preemptive setting is a desirable configuration for shared resource environments [145]. Thus, we have implemented this non-preemptive setting in the scheduler of this simulation model. However, the inaccuracy in decision implementation can be reduced by selecting the processing (resource reservation) times comparatively smaller than the sample period. For instance, if the processing time varies in ticks range, the sample period can be selected in major ticks. This means the decision made will be implemented during the current sample period before the next decision is made.

2. The end-to-end response time is not considered. The end-to-end response time consists of the communication (or network) delay, connection delay and processing time. The communication delay is hard to control because clients may request services from different locations in the world using different internet connections (e.g., dialup, broadband). Similarly, the connection delay also depends on the connection scheduling techniques implemented by the underlying operating system. Therefore, in this work, we only consider the processing time, which is the controllable component of the end-to-end response time. This is the same approach taken by many existing work (e.g., [145, 150, 226, 229, 231]).

3. Time taken to reschedule the resource is assumed to be negligible. Some types of resources may require time to be rescheduled. For instance, if a virtual machine has to be taken from one class and rescheduled for another class, it may require restarting the virtual machine and installing the client specific application. In this

simulation we assume such delays are negligible.

4. Overhead from the scheduler and the statistic calculator is negligible. The scheduler and the statistic calculators may impose the computational overhead in the real system implementations. However, this simulation model abstracts away such overheads.

3.5.3 Validation of the DES Model

In the general software engineering process, when the implementation phase completes, the next step is to test the implementation. For instance, black-box and white-box testing methods are typically utilized. We also conducted such testing, which showed that this DES simulation model achieves many of the requirements set out in Section 3.5.1. However, in this work, we are interested in the runtime performance characteristics of the system under changing conditions. We have also conducted validations on the runtime behavior of the system theoretically, which is also a major step in the simulation environment development process. For this validation, the queuing theory was used. It is worth noting that to apply the queuing theory, certain assumptions on the system structure, arrival workloads and processing time probability distributions should hold. Although many of these assumptions do not hold in real world systems, we implemented a simulation system adhering to the assumptions of the queuing theory in order to validate the implementation of the simulation model. These validations have shown that the results of the simulation model conform to the theoretical results of queuing theory, which further indicate the validity of this implementation. Details of the validation settings and results are listed in Appendix C.

3.5.4 Simulation Settings

Using the above DES model, we setup a simulation system to apply and validate the proposed nonlinear control theoretic approaches in this thesis. The variables that need to be set and configured are 1) the workload profile of each class, 2) the resource reservation time probability distributions and 3) the total number of available resources (S_{total}).

3.5.4.1 Workload Profiles

The workloads that multi-class shared resource systems may face cannot be generalized. The workload a system can manage depends on the capacity of resources, management requirements and the type of resource being shared (e.g., hardware versus middleware). In addition, the workloads are time-varying, instead of staying constant for the entire period of operations. Such variations in the system inputs are not only limited to software systems, but to other types of physical systems as well. As a consequence, control engineering provides a set of well-established input signals to validate the control systems. They are as follows:

Assume, W_n is the nominal workload that system receives.

Impulse input signal. Formally, $W_{impulse}(k) = 1$ when $k = 0$ and $W_{impulse}(k) = 0$ $k \neq 0$. i.e., the impulse input signal increases the workload to some value greater than W_n for a single sample period. In a real workload this can be considered as a workload spike for a very short time period. However, such spikes for a very short period of time may not affect the performance attributes (e.g., average response time) drastically, consequently the impulse input signal may not be useful for the validations of the control systems for software systems.

Step input signal. Step input signal models a sudden jump in the workload from W_n to some value W_{step} and stays at that value for more than a single sample period. This is one of the widely used input signals to validate the performance of the control systems in control engineering. In addition, most of the applications of feedback control in software systems, including multi-class systems have used step workload changes to validate the performance and resource management capabilities. This is because, such workload changes of even a single class for a long period of time affect the performance attributes (e.g., response time) in a shared resource environment. As a consequence, the control system is forced to redistribute the available resources among classes efficiently, in order to achieve the required performance objectives. The delay in response to such workload variations may cause large transient responses and temporal instabilities in the system.

Ramp input signal. Ramp input linearly increases the workload from W_n to W_{ramp} during a sometime interval. This signal models a gradual increase of workload instead of instantaneous increment of the workload in the case of step input signal.

The main advantage of these input signals is given a linear model of a system, there are well-known design and analysis techniques in control theory to compute performance specifications and behavior of the system. Consequently, after constructing a linear model of a system, we can investigate the load variations that the system can maintain without leading to instabilities. However, a linear model of a system is an estimation of its behavior (not a 100% accurate representation), so that these theoretical evaluations may not be 100% correct. This is also true for systems demonstrating nonlinearities such as the system under investigation in this thesis. Combinations of workload input signals (in particular, step input profiles in time varying fashion) will be used as heuristics to validate and compare the performance of the control systems. In addition, the workloads generated from the real-world workload traces will also be used.

3.5.4.2 Total Resource Amount and Resource Reservation Time Distribution

The following settings have been used in the simulations to represent the behavior of the multi-class shared resource system. The settings will remain the same unless otherwise specified.

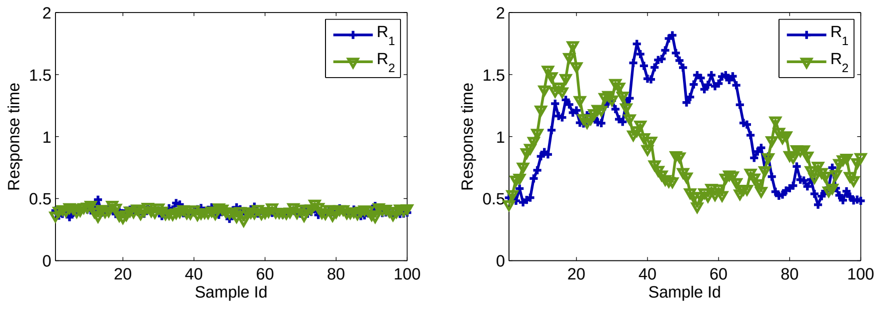
The total amount of resources simulated $S_{total} = 30$. The processing time of each resource unit is selected from a uniform distribution as follows:

$$r(x) = \frac{1}{r_{max} - r_{min}} \text{ for } r_{min} \leq x \leq r_{max} \tag{3.3}$$

$$= 0 \text{ for } x < r_{min} \text{ and } x > r_{max} \tag{3.4}$$

where, $r_{min} = 100$ ticks and $r_{max} = 700$ ticks.

The above settings are selected in order to achieve the tractability of resource allocations between the classes under different experimental conditions. The r_{min} and r_{max} were selected after careful investigation of system outputs under different workload conditions. That is, when the system is running close to the full capacity, the system output should remain within some bounds, according to theoretical and practical system behavior. Figure 3.5 shows a comparison when 30 resource units are allocated to two classes with 30 requests/sec workloads for each class. When the selected bounds are $r_{min} = 100$ and $r_{max} = 700$ ticks, the system is at a steady state. However, under the same settings, when the bounds are $r_{min} = 100$ and $r_{max} = 900$ ticks, the steady state behavior is highly



(a) 100-700 (b) 100-900

Figure 3.5: System behavior under 3.5a) $r_{min} = 100$ and $r_{max} = 700$ ticks 3.5b) $r_{min} = 100$ and $r_{max} = 900$ ticks

variable and unbounded. This is because the variability around the average response time leads to large transient response in the system, even under a constant workload. To avoid such behavior, the resource capacity and the workload intensity have to be selected depending on the bounds. $r_{min} = 100$ ticks and $r_{max} = 700$ are suitable bounds for the selected workload rates and the resource capacity. However, in each chapter we conduct Monte-Carol simulations under different r_{min}, r_{max} and S_{total} values to investigate the impact of these parameters on the presented results.

Furthermore, the uniformly distributed processing time means that any operation invoked in the system is equally likely, which gives a fair weight for each invocation. This is done because there is neither evidence nor a generalization available to represent the invocation patterns of the operations and their system output (e.g., response time) bounds. The uniform distributions have been selected in the existing works [11, 42, 54, 99, 118, 143, 224].

In addition, 2000 ticks (milliseconds) were selected as the sampling time period of the statistic calculation process. The selection of the sample time period has to be carefully done in physical systems. For instance, a small sample time invokes the statistic calculations frequently leading to additional overhead on the system. Furthermore, a short sampling interval affects the variability of the measured average statistics. In contrast, a large sampling time may cause decision delays leading to instabilities under sudden changes of the workloads or other short-term disturbances. These trade-offs have to be considered in the selection of the sample time interval. We selected 2000 ticks after analysis of the workload rates and changes the system may encounter. Further, it reduces the effects of the assumption (1), listed in Section 3.5.2.

3.6 Summary of Experimental Case Studies

As mentioned previously, in addition to the simulation studies, a couple of case studies on real-world physical systems will be also utilized in the validation of this thesis. One of our objectives was to select case studies that share resources at different levels of the system stack. The selected case studies are briefly introduced below. Further details on the architecture and experimental results of these case studies can be found in Chapter 8.

WSO2 Stratos. WSO2 Stratos[3] is a production business process server (BPS) enabling deployments of business processes for its consumers. The latest version of this BPS also supports multi-tenanted deployments, so that, it is a multi-class shared-middleware platform. Although it provides many properties (e.g., security, data isolation) required by a multi-tenanted middleware platform, the performance and resource management capabilities are not provided thus far. Therefore, this middleware platform is a suitable candidate to apply and evaluate the performance management approaches proposed in this thesis.

This case study also enables us to investigate the resource management capabilities of the nonlinear control approaches with respect to the existing linear approaches at the middleware level. The resource that has to be shared between client classes is the business process instances. In addition, being an open-source project helps us to perform the necessary architectural modifications to the BPS easily (e.g., adding per-class queues, sensors and actuators), before integrating the performance and resource management capabilities into the system.

Travel reservation system. The real-world travel reservation system architectures can use the shared application multi-tenanted option to gain many advantages. However, currently, these systems use the traditional method of maintaining separate infrastructure for each client class. In this thesis, we build an experimental case study following the real-world travel reservation architectures in order to investigate the behavior of the proposed nonlinear approaches at the application level. In this system, application domain specific resources have to be shared between client classes to achieve the required performance objectives.

Both these multi-tenanted environments are deployed on hardware (physical or virtual

[3]http://wso2.com/cloud/stratos/

machines) platforms similar to production deployment environments. The distributed components of these environments are connected using isolated networks to avoid unpredictable network interferences. The evaluations conducted on these experimental case studies follow the same validation settings described in Sections 3.4 and 3.5. However, in order to reach conclusive results under noisy and uncontrollable conditions, the same experiment is run multiple (10) times and the average results are compared and analyzed.

Chapter 4

Control-oriented Nonlinear System Identification

4.1 Introduction

One of the major contributions of this thesis is the successful application of novel nonlinear model structures and model identification methodologies to represent the relative and absolute performance management schemes compared to the existing linear identification mechanisms. The focus of this chapter is to present the model identification procedures used, which estimate the existing dominant nonlinearities in the relative and absolute performance management schemes. This enables the feedback control systems developed using these nonlinear models to achieve the management objectives of a multi-class shared resource environment as defined in Chapter 3.

According to the analysis in Chapter 2, the relative performance management scheme has already been adopted by many researchers who work in the field of performance and resource management of multi-class systems. As mentioned, the existing works realize the relative performance management by developing a control system composed of a linear model and controller. However, due to the consideration of the ratios between the system outputs and resource caps (see Section 3.2.2.2), the relative management scheme inherits a highly nonlinear behavior [150]. Furthermore, the target system controlled by the management system also exhibits nonlinear behavior. Due to these reasons, when such linear control systems are deployed in a dynamic environment, it is difficult for them to achieve the business and control objectives under highly varying workloads, resource demands, changing business requirements (priority levels) and disturbances. Consequently, linear modelling techniques are insufficient to capture the nonlinear dynamic behavior, thereby fail to achieve the control objectives of the relative management system. To mitigate the issues of nonlinearities, Section 4.2 of this chapter characterizes the dominant nonlinearities of relative management scheme and presents a SID method to estimate the nonlinearities explicitly.

Section 4.3 of this chapter deals with the absolute performance and resource management scheme as introduced in Section 3.2.2.1. In a nutshell, the absolute performance management scheme controls resources at runtime in order to achieve the specified target values of the system outputs of each class served by the multi-class shared resource system. Therefore, this is a multiple-variable and multiple-objective control problem. In addition, the behavior of system outputs with respect to the variations of the resources (inputs) is nonlinear. Furthermore, there are interactions between workloads of the multiple classes because all classes are served by a limited amount of resources. As a consequence, the above described factors necessitate a runtime performance and resource management methodology that can achieve multiple control objectives under constraints, which also takes into account the nonlinear behavior of the system. In order to achieve these objectives, in Section 4.3 a nonlinear Multi-Input-Multi-Output (MIMO) model structure and model identification technique is presented to estimate the nonlinearities that exist in the absolute management scheme.

4.2 Identification of Relative Performance Management Scheme

This section firstly overviews the formulation of relative management system and analyzes the characteristics of the nonlinear behavior of the relative management scheme and target system, which will affect the closed-loop performance of a linear control system. Secondly, an estimation technique is presented to model the nonlinear dynamic behavior which is subsequently deployed to compensate the identified nonlinearities in order to reduce the impact of the nonlinear behavior on the management system. The results of this new nonlinear SID process are covered at the end of this section.

4.2.1 Formulation of Relative Management System

Section 3.2.2.2 presented the definition of the relative management system for n classes. In order to understand how a relative management system is formulated, we examine the following two cases from the existing literature. The first example presents a case where we have two classes.

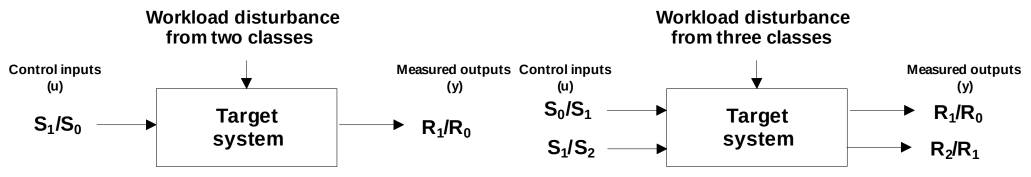
Example 1: Let us consider two classes, $class_0$ and $class_1$. $R_0(k)$ and $R_1(k)$ are the response times of these classes respectively at the k^{th} sample. Further, $S_0(k)$ and $S_1(k)$ are the resource caps, where $S_0(k) + S_1(k) = S_{total} = 30$ and $S_{0,min},\ S_{1,min} = 6$. As defined in the management problem in Section 3.2.2, S_{total} is the number of resources available in the system and $S_{i,min}$, where $i = 1, 2$ are the reserved minimum number of resources for each classes at all times.

In the relative management and proportional resource allocation schemes, the control input is the ratio of resource caps, represented by $\frac{S_0(k)}{S_1(k)}$ and the output variable is the ratio of average response time of the workloads, represented by $\frac{R_1(k)}{R_0(k)}$. For notational simplicity let us denote $\frac{S_0(k)}{S_1(k)}$ and $\frac{R_1(k)}{R_0(k)}$ as $u(k)$ and $y(k)$ respectively. The control objective of this control system is to maintain the response time ratio $(y(k))$ around $\frac{P_1(k)}{P_0(k)}$ (i.e. the reference or set point) depending on the performance differentiation factors $(P_0(k), P_1(k))$ of two classes. Here, it is clearly seen that in order to maintain the response time ratio around the prescribed reference signal, the input signal $u(k) = \frac{S_0(k)}{S_1(k)}$ is to be manipulated. In particular, when the workload conditions from these two classes change, the output variable that is the response time ratio will be affected. Because of the unpredictable nature of the workload conditions, they act like stochastical disturbance to the system. From the systems point of view, this relationship can be represented by the block diagram shown in Figure 4.1a.

The second example examines the case of three classes in a relative resource sharing problem.

Example 2: In this case, we consider three classes, $class_0$, $class_1$ and $class_2$. Similar to example 1, $R_0(k)$, $R_1(k)$ and $R_2(k)$ are the response times at the k^{th} sample and $S_0(k)$, $S_1(k)$ and $S_2(k)$ are the resource caps of these classes, where $S_0(k) + S_1(k) + S_2(k) = S_{total} = 30$ and $S_{0,min}, S_{1,min}, S_{2,min} = 6$. According to the definitions, the input variables of the system are $u_1(k) = \frac{S_0(k)}{S_1(k)}$ and $u_2(k) = \frac{S_1(k)}{S_2(k)}$ and output variables of the system are

$y_1(k) = \frac{R_1(k)}{R_0(k)}$ and $y_2(k) = \frac{R_2(k)}{R_1(k)}$. The main objective of the control system is to maintain the outputs $(y_1(k),\ y_2(k))$ around $\frac{P_1(k)}{P_0(k)}$ and $\frac{P_2(k)}{P_1(k)}$ (i.e. the reference signals) depending on the performance differentiation factors of the classes $(P_0(k), P_1(k), P_2(k))$. Figure 4.1b shows the block diagram of this system.



(a) Two classes (b) Three classes

Figure 4.1: Open-loop relative management system with two and three classes

4.2.2 Classification of Nonlinearities of Relative Management Scheme

In order to analyze the runtime behavior of the relative management scheme, here we continue with the *Example 1* in Section 4.2.1.

When the requirements and policies in *Example 1* are embedded in the design, the control input $u = \frac{S_0(k)}{S_1(k)}$ can only take certain discrete values that can be implemented in the system. These values can be computed as follows:

$$\frac{S_0}{S_1} = \frac{S_0}{S_{total} - S_0} \tag{4.1}$$

Then, by choosing $S_0 = 6, 7, 8, \ldots, 24$, with $S_{total} = 30$, the control variable $u = \frac{S_0}{S_1}$ takes value at $\frac{6}{24}, \frac{7}{23}, \ldots, \frac{23}{7}, \frac{24}{6}$. Figure 4.2 shows the operating points that the controller can choose. These operating points are unequally spaced. The region where $class_0$ workload gets more resources compared to the other is represented by *region 0*. Similarly, the region in which $class_1$ gets more resources is represented by *region 1*. The nominal operating point is when both classes get equal number of resources. In *region 0*, spacing increases towards the right end of the entire operating range whereas in *region 1* spacing decreases towards the left end of the operating range. These nonlinearities caused by the restricted operating points generated due to considering the ratios of the system inputs, exhibit the characteristics of *static input nonlinearities*. This is because the aforementioned nonlinearities do not vary with time. Such static input nonlinearities may affect the performance of a linear controller when it is operating away from the nominal operating region.

Similarly, let us consider the controlled variable $y = \frac{R_1(k)}{R_0(k)}$ that is the ratio of the system outputs. However, unlike the way we computed the range of the control input u

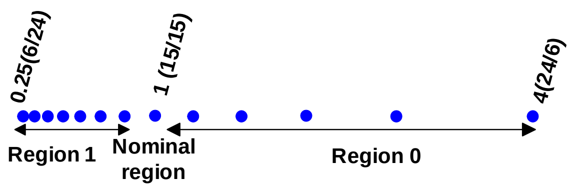
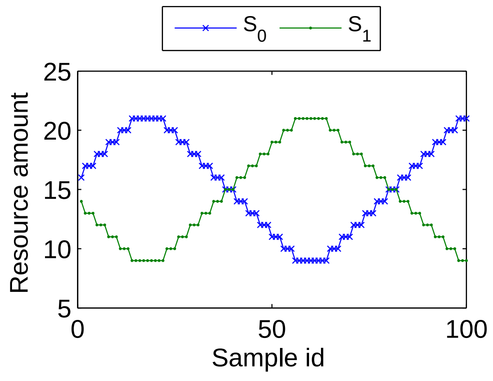


Figure 4.2: Possible discrete operating points (control input)

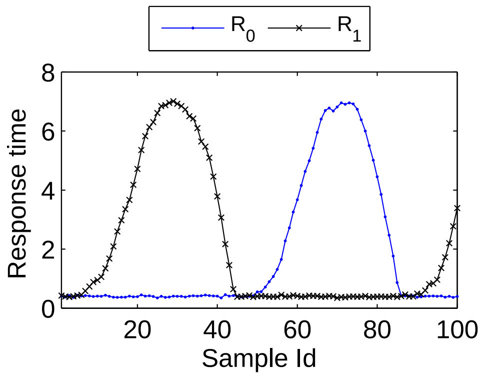
(i.e., by using the linear relationship), the range of output cannot be predetermined. This is because R_0 and R_1 can take large range of values, causing $\frac{R_1}{R_0}$ ratio to have a large set of values depending on the workloads of the corresponding classes. For instance, if $R_0 = 0.4$ (sec) and $R_1 = 1$ (sec) then $\frac{R_1}{R_0} = 2.5$. Similarly, if $R_0 = 1$ (sec) and $R_1 = 0.4$ (sec) then $\frac{R_1}{R_0} = 0.4$. This characteristic could also be shown experimentally by applying a sinusoidal input in the system under constant workloads for both classes 0 and 1. Figure 4.3 shows the input signals, outputs ($R_0(k)$ and $R_1(k)$) of the system and measured output (y). Figure 4.3b indicates that R_0 and R_1 show similar behavior when resources are increased and decreased. When resources are adequate the response time remains at steady state, however when inadequate it increases drastically. Figure 4.3c shows $\frac{R_1}{R_0}$ calculated from the data in Figure 4.3b. It is seen that when R_1 increases, $\frac{R_1}{R_0}$ increases at a high rate. In contrast, when R_0 increases, $\frac{R_1}{R_0}$ decays at a high rate.

This behavior of the system output is clearly nonlinear. In addition, even though R_0 and R_1 values have similar symmetrical behavior with respect to the change of the corresponding resource cap, the divider operator in the controlled output ($y = \frac{R_1}{R_0}$) creates an asymmetric behavior leading to this nonlinearity. Thus, it is evident that there are *output nonlinearities* in the system. Such output nonlinearities may also cause performance degradation in a linear controller. This is because, due to a disturbance of $class_0$ the output behaves in a totally different way compared to a disturbance of $class_1$. Consequently, a single linear controller may behave differently in both regions leading to significant performance and resource management issues. Furthermore, setting the controller parameters (gains) are difficult to provide satisfactory control in both regions.

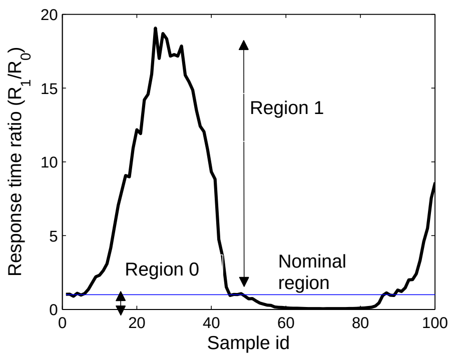
Based on the above characterization of the relative management scheme, the input and output nonlinearities may become a significant issue in designing a management system and providing robust control at runtime. A possible way to reduce the impact of the nonlinear behavior on the management systems is by designing a compensation frame-



(a) Inputs



(b) Outputs



(c) Controlled output (y)

Figure 4.3: The output nonlinearity

work to compensate the existing nonlinearities at runtime. With respect to the relative management scheme, such compensation framework needs to reduce the impact of the aforementioned input and output nonlinearities.

In this work, we utilize a nonlinear modelling approach called Hammerstein-Weiner model for the first time in the literature of performance management of software systems to design such a compensator framework and reduce the impact of static input and output nonlinearities on the management system.

4.2.3 Hammerstein and Wiener Block Structure Model

Hammerstein and Wiener model is a well known nonlinear block-structure model in control literature [15, 69, 70, 173, 195, 221]. As shown in Figure 4.4, the Hammerstein-Wiener block structure model has a linear block surrounded by two nonlinear blocks. The entire model can be divided into two segments called Hammerstein and Wiener.

The Hammerstein model has a nonlinear component preceding a linear component. In contrast, the Wiener model has a linear component followed by a nonlinear component. The combination of these two schemes is referred as the Hammerstein-Wiener model. The input and output nonlinear blocks are assumed to model the static input and output nonlinearities in the system respectively, while the linear block captures the rest of the system dynamics. u and y denote the input and output of the Hammerstein-Wiener block structure respectively. The intermediate variables v and w are not measurable.

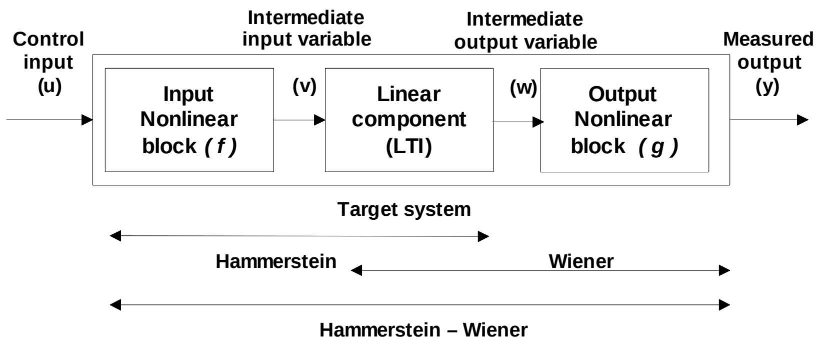


Figure 4.4: Block diagram of Hammerstein and Wiener block structure model

The identification of Hammerstein-Wiener block structure model is complex because of the additional nonlinear blocks compared to the identification of a single linear model. Typically, a linear model of the system is constructed using the input-output $(u - y)$ data gathered form a SID experiment [125]. In contrast, the complexity arises in the estimation of Hammerstein-Wiener model due to the structure selection and parameter derivation of the nonlinear blocks. The nonlinear blocks can be represented by many different nonlinear functions, including piecewise linear, polynomial with degree r, nonlinear-ARMA models and splines [15, 68, 91, 195, 221]. Even though, there are additional parameters to derive, only the input-output $(u - y)$ data can be used in this identification process, because the intermediate (v, w) variables are not measurable. In addition, due to the characteristics of the nonlinearities that exist in the systems, it is hard to provide a identification technique with significant generality or systematic process [87]. Thus, it is important to validate the identified model. However, if some details about these static nonlinearities are known at the design time, from the prior knowledge or an analysis of the system, the modelling process could be simplified.

4.2.4 Model Identification Procedure of Relative Management Scheme

The discussions in Section 4.2.2 indicates that the Hammerstein-Wiener model is a suitable candidate to model the characterized input and output nonlinearities of the relative management scheme. This section presents the model identification methodology based on the Hammerstein-Wiener model for the relative management system.

To estimate the dynamics of the target system as Hammerstein-Wiener model the two nonlinear components have to be represented by the formal relationships with the respective variables at the model estimation stage. After the estimation of the nonlinear components, their inverse functions are estimated to design compensators, which will be used in the implementation of the control architecture at the later chapters to compensate the existing input and output nonlinearities. As mentioned, different types of functions can be used in the design of the compensators. The desirable properties when finalizing the inverse functions are as follows:

Property 1 : the relationship of the dependent and independent variables should be monotonic (i.e., there should not be multiplicities).

Property 2 : low computational overhead[1].

After the integration of the compensators, the open-loop system architecture is shown in Figure 4.5. With the integration of the pre-compensator and the post-compensator, the structure of the target system has transformed significantly. In particular, compared to the original target system variables $u(k)$ and $y(k)$, this new structure transforms the system to operate with intermediate variables $v(k)$ and $w(k)$ as the input and output respectively.

As mentioned before, estimating the Hammerstein-Wiener model for this system is a complex process, due to the additional parameters that have to be approximated for the input and output nonlinear blocks. There are different approaches taken in the literature with various assumptions. There are many approaches focusing on the Hammerstein model [5, 68, 89] or Weiner model [65, 66, 91, 174, 207] individually. However, only few SID approaches estimate the combined Hammerstein-Wiener model.

In this work, we propose a novel approach to reduce the modelling complexity that exists in the relative management system. This estimation process of the Hammerstein-

[1]The computational overhead has to be considered in particular when there are multiple candidates to model the data with similar accuracies.

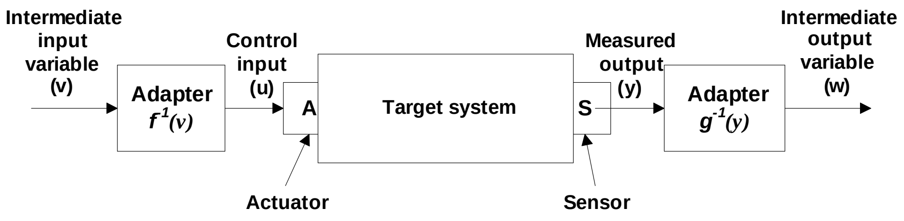


Figure 4.5: Hammerstein-Wiener model based system after the integration of the compensator

Wiener model consists of three steps. In the following sections, we present the estimation process of the two nonlinear components and linear component respectively. Furthermore, the modelling process covered in these sections is for a multi-class shared resource system serving n number of classes, with S_{total} number of resources as defined in Section 4.2.1. It is also noteworthy that the modelling approach treats the software system as a black-box, disregarding the architectural and implementation details of the software system.

4.2.5 Estimating Input Nonlinearity

The input nonlinear component is to capture the relationship between the input (u) and the intermediate linear variable (v). Since the intermediate input variable (v) is not measurable, the estimation of this relationship is not straightforward. Consequently, there are many SID techniques to estimate the Hammerstein model using input-output data $(u - y)$. For instance, [15, 68, 69] propose algorithms to derive parameters for different functions from input and output data. These techniques have used different nonlinear functions including piecewise linear, polynomials and splines [173] to represent the input nonlinear component. However, if the nonlinearity is known at the design time the nonlinear function selection and its parameter estimation process becomes easier. From the analysis in Section 4.2.2, the discontinuous operating points shown in Figure 4.2 may induce significant static input nonlinearity. If we implement equally spaced operating points, a linear controller may provide better performance in the entire operating region. Therefore, assuming that the input nonlinearity is known at the design time, a method is proposed to estimate the nonlinearity avoiding the complexity of estimation based on a nonlinear SID experiment.

Firstly, in order to compute the possible operating points (or the range of u) for the controller in the original system, we can utilize equation (4.1) and calculate points for $S_0 \in \left\{S_{0,min}, S_{total} - \sum_{j=1}^{n-1}(S_{j,min})\right\}$. Here, we have computed the operating points for a controller of first pair of classes, assuming that the rest of the classes are guaranteed the required minimum allocation $S_{i,min}$. Let us represent these $p > 1$ number of operating points as $u = \{u_1, u_2, \ldots u_p\}$. The next step is to replace these unequally spaced operating points u with equally spaced operating points. Since, the range of the intermediate variable is not known, select an arbitrary $v_{min} \leq v \leq v_{max}$ for the intermediate input variable v (the effect of the values selected for v_{min} and v_{max} is investigated in Appendix I). With δv defined as $\delta v = \frac{v_{max} - v_{min}}{p-1}$, the intermediate input variable takes its own operating points as $v_1 = v_{min}, v_2 = v_1 + \delta v, \ldots, v_{l+1} = v_l + \delta v, \ldots, v_p = v_{max}$. Thirdly, map the individual values u_l and v_l in the u and v sets to create data pairs, where $l = 1, 2 \ldots, p$. Those data pairs can be directly used in a look-up table for applications such as compensation. However, for convenience of use, a simple curve fitting may be used to obtain the analytical inverse function ($u = f^{-1}(v)$) to design a compensator. More precisely, it is assumed that the f^{-1} is a polynomial of order m, and is defined for the k^{th} sample as

$$\begin{aligned} u(k) &= f^{-1}(v(k)) \\ &= \alpha_0 + \alpha_1 v(k) + \alpha_2 v(k)^2 + \ldots + \alpha_m v(k)^m \\ &= \phi(k)^T \theta \end{aligned} \tag{4.2}$$

where the coefficient vector $\theta = [\alpha_0 \ \alpha_1 \ \ldots \ \alpha_m]^T$ and the data vector $\phi(i) = [1 \ v(k) \ \ldots \ v(k)^m]^T$. The least squares estimate of the coefficient vector $\hat{\theta}$ is given by

$$\hat{\theta} = (\sum_{i=1}^{p} \phi(k)\phi(k)^T)^{-1} \sum_{k=1}^{p} \phi(i)u(k) \tag{4.3}$$

which derives the coefficients of the polynomial.

After f^{-1} function is estimated, it is used to implement a compensator component and integrated to the system as shown in Figure 4.5 at the input. The same compensation is done for each controller, managing the class pair $i-1$ and i according to the relative management scheme. With integration of this pre-input compensator, the system modelling problem is now reduced to a Wiener structure.

4.2.6 Estimating Output Nonlinearity

The output nonlinearity cannot be estimated using the same approach in Section 4.2.5, because the static output nonlinearity is not known and cannot be decoupled from the other dynamics of the system. In addition, there is no knowledge about the intermediate output variable at the design time. Consequently, the model structure and order of the output nonlinear and linear component have to be estimated as a black-box using SID

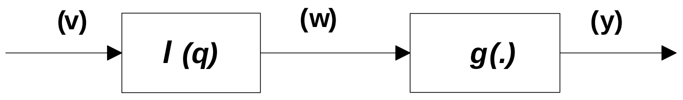


Figure 4.6: Wiener model

experiments.

Now, let us look at the Wiener model structure as shown in Figure 4.6 to understand the parameters that need to be estimated during the model identification process. The output can be represented as a function of $w(t)$ and η as follows

$$y(t) = g(w(t), \eta) \tag{4.4}$$

where, $w(t)$ is the intermediate output variable, η is the parameter vector and t is the time instance. Similarly, the linear function l can be represented as

$$w(t) = l(v(t), \theta) \tag{4.5}$$

where, $v(t)$ is the intermediate input variable and θ is the parameter vector of the linear model.

The main focus of the Wiener model identification is to compute the best η and θ vectors that represent the data from the SID experiment with sufficient accuracy. The *predication error method* [125] is one of the popular methods used to estimate the required parameters in SID. The prediction error method can be formulated as a minimization problem of function V given below.

$$V_N(\eta, \theta) = \frac{1}{N}\sum_{t=1}^{N}(y(t) - \hat{y}(t, \eta, \theta))^2$$

$$(\hat{\eta}, \hat{\theta}) = arg_{(\eta,\theta)} min V_N(\eta, \theta) \tag{4.6}$$

where, $y(t)$ is the measured output data, $\hat{y}(t, \eta, \theta)$ is the predicted output and N is the number of data samples.

This predication error minimization problem can be solved using Gauss-Newton search method [76]. However, the Wiener model is over-parameterized when parameters of both linear and nonlinear components are considered simultaneously. Consequently, numerical problems will be encountered. In order to overcome this issue a simple solution is to fix some of the parameters of one of the components of the model (i.e., in l or g) during the optimization search process [76].

There are different types of identification strategies proposed and adopted in the literature to identify the Wiener model based on the above methodology, including *single step* and *two step* approaches [33, 66, 173]. Basically, in the two step process the nonlinear and linear components are approximated in two stages. In [66, 173, 174], the nonlinear component is identified first followed by the linear component, where as in [33] the linear block is identified first and then the nonlinear block. These two step approaches are known to capture the static nonlinearities much better, however some of the linear or nonlinear dynamics are coupled up to some extent in the component estimated at the first stage. In contrast, single step approaches [87, 90, 91] estimate both components simultaneously. This approach requires less effort in identification, but model accuracy is a trade-off [173].

In this work, a two step identification process is utilized, which consist of two experiments to identify the Wiener system. The first is to identify the static output nonlinearity together with the dynamics. Then, the inverse of the static output nonlinearity is approximated and integrated into the system as shown in Figure 4.5. Afterwards, the second experiment is conducted to identify the linear gains (see Section 4.2.7). This is because the first experiment is not sufficient to identify the gains of the linear system as stated by Kalafatis et al. in [91].

For the first SID experiment a sinusoidal input signal is used, because of its sufficient excitation[2] [13]. The main difference is now the input signal is designed using v instead of u in the original system. The possible values of v are applied as a sinusoidal, with a suitable switching frequency. The system is assumed to be in the idle state before the start of the experiment, with no workloads from both classes. Suitable workloads are then applied for each class and the output is observed for a sufficient amount of time. Afterwards, gathered $(v - y)$ data pairs are divided into two sets called estimation set and the test set for cross validation [125, 155]. The *nlhw* command in the *Matlab: system identification toolbox* [159] implements an iterative Gauss-Newton search method to come up with the model parameters from the input-output data using the prediction error method described above. In order to resolve the issue of over-parameterization, the Matlab implementation fixes some of the coefficients of the linear component to unity. In addition, it is a single step estimation technique, which derives parameters of both

[2]The input signal should consist of frequency components to ensure the excitation of important dynamics of the system.

linear and output nonlinear components simultaneously. The derived linear model is only utilized to approximate the structural details of the linear component and to calculate the intermediate output variable data to estimate the inverse of the output nonlinearity. To represent the linear block of the Wiener structure we use an autoregressive exogenous input (ARX) model. Standard ARX model is as follows:
ARX(n,m,d)

$$y(k) = \sum_{i=0}^{n} a_i y(k-i) + \sum_{j=0}^{m} a_j u(k-d-j) \tag{4.7}$$

where, n, m - the order of the model, (a_i, b_j) - the parameters of the model, d - delay, k - the current sample instance.

A polynomial is a preferred choice to represent the output nonlinearity and widely adopted in the literature (e.g., [70, 90, 173]). Consequently, to represent the output nonlinear block, a polynomial of degree r is used. Then, different model structures (n, m, d, r) are fitted to the input-output $(v - y)$ data of the estimation set using the *nlhw* command. (n, m, d, r) indicate the model structure, where (n, m, d) represents the structure of the linear block and (r) represents the polynomial order of the output nonlinear block. After this model fitting process, the model structure and parameters have to be decided, in order to derive a sufficiently accurate model.

The linear block estimated in this identification process is a function of v and w and the output nonlinear block is a function of w and y, $y = g(w)$. However, to compensate the estimated output nonlinearity, the inverse of the output nonlinearity has to be derived, i.e., $w = g^{-1}(y)$. For this particular purpose, the data of the intermediate variable w has to be calculated. Using the input signal data (v) of the SID experiment, the estimated linear model is simulated to calculate the data points for w. The $(w - y)$ data pairs are then used to estimate an inverse of the output nonlinearity $g^{-1}(y)$ using a suitable type of function. Afterwards, $g^{-1}(y)$ function is implemented and integrated to the software system as a software component at each output (see Figure 4.5).

To summarize, in this identification procedure 1) data from the SID experiment is used to estimate the Wiener model (i.e., both linear and nonlinear components), 2) identified linear model is then simulated to compute intermediate output variable data set $(w(k))$ and 3) $w(k) - y(k)$ data set is then used to estimate the inverse output nonlinear function.

4.2.7 Estimating the Linear Dynamics With Nonlinear Compensation

After integration of both input and output compensators the existing static nonlinearities can be effectively compensated. The gains of the linear component of the Hammerstein-Wiener model is estimated in this step using an experiment conducted under a wide spectrum after applying suitable workloads for each class. The gathered $(v - w)$ data is then used to construct a ARX model (see equation (4.7)). With this step, the Hammerstein-Wiener model estimation procedure of the relative performance and resource management system is complete.

4.2.8 Model Identification Results of Relative Management Scheme

This section covers the model estimation results of the inverse nonlinear functions and the linear component of Hammerstein-Wiener block oriented model. A shared resource system with two classes (namely, $class_0$ and $class_1$) is considered. The resource caps are denoted by $S_0(k)$ and $S_1(k)$ where $S_0(k) + S_1(k) = S_{total} = 30$ and $S_{0,min}, S_{1,min} = 6$. $R_0(k)$ and $R_1(k)$ and $P_0(k)$ and $P_1(k)$ are the response times and differentiation factors of these two classes respectively at the k^{th} sample.

The estimated models are validated with *goodness of fit* (R^2/variance accounted for [81]) index, which is defined as follows:

$$R^2 = \left(1 - \frac{\sqrt{\sum_{i=1}^{N} (z - \hat{z})}}{\sqrt{\sum_{i=1}^{N} (z^2)}} \right) \tag{4.8}$$

where $N, z, \hat{z}$ are the number of samples, measured output and model output respectively.

Estimation of input nonlinearity and its inverse: The first step is to implement the input nonlinear compensator (or pre-input compensator). Using the procedure explained in Section 4.2.5, the set of operating points for the control input u can be computed as $\frac{6}{24}, \frac{7}{23}, \ldots, \frac{23}{7}, \frac{24}{6}$ (see Figure 4.2) for the shared resource environment. Let us select v_{min} and v_{max} as -9 and 9 respectively, deriving $\delta v = 1$. This formulates $v = -9, \ -8, \ldots, 0, \ldots, 8, \ 9$. The mapping of the points of u and v are shown in Figure 4.7. Then, the relationship of u and v are modelled using a suitable function without violating the properties, in particular the *property 1* introduced in Section 4.2.4. From our investigation polynomials and logarithm functions can model the relationship shown in Figure 4.7 with sufficient accuracy. A polynomial is used in this work, which provides

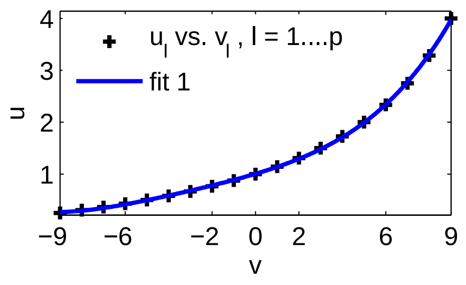


Figure 4.7: Mapping of the u_l and v_l data pairs

much better fitting for larger ranges. As a result, f^{-1} was estimated using a 4th order polynomial with a goodness of (R^2) fit of 0.99. The equation (4.9) shows the model structure and Figure 4.7 shows the model fit.

$$\begin{aligned} u(k) &= f^{-1}(v) \\ &= 7.468 \times 10^{-5} v(k)^4 + 0.001 v(k)^3 + 0.008 v(k)^2 + 0.124 v(k) + 1.005 \end{aligned} \tag{4.9}$$

Estimation of output nonlinearity and its inverse: With the integration of input static nonlinear compensator, the estimation for the system is focused on the Wiener model. The input signal design is an essential part of this stage. We investigated three different input signals, which include the pseudo binary random, sinusoidal and pseudo random signals. Sinusoidal signals showed higher model fits ($R^2 > 0.8$), compared to the other two signals (see, Tables 4.1 and 4.2 for model validation results). This may be because of the sufficient excitation of the sinusoidal signal. As a result, a sinusoidal signal was designed with possible values of v and a switching frequency of 2 samples. 30 requests/sec workloads for each class were applied and the output was observed for 600 sample periods. Data samples between the 1st to 400th samples were included in the estimation set, which was used to construct the model of the system. The rest of the data samples were used as the test set to validate and assess the quality of the model. The model validation results are shown in Table 4.1, where n is the model order for the linear system without time delay, r is the polynomial order for the static output nonlinearity. From Table 4.1, it is evident that the best model structure for the Wiener system is a second order linear model with a fifth order static polynomial model. The lower order models have poor model fits, while higher orders did not improve the fit significantly. Figure 4.8 shows the predicted output from this model in comparison with the test set data.

Table 4.1: Model validation results with a sinusoidal test signal

(n,r)	(1,5)	(2,1)	(2,4)	(2,5)	(2,7)	(3,5)
R^2 Fit	53.4	53.9	80.4	81.7	81.1	81.1

Table 4.2: Model validation results with random input signals

Signal type	(n,r)=(1,5)	(n,r)= (2,5)
Pseudo binary random	66.7	66.8
Pseudo random	70.1	71.5

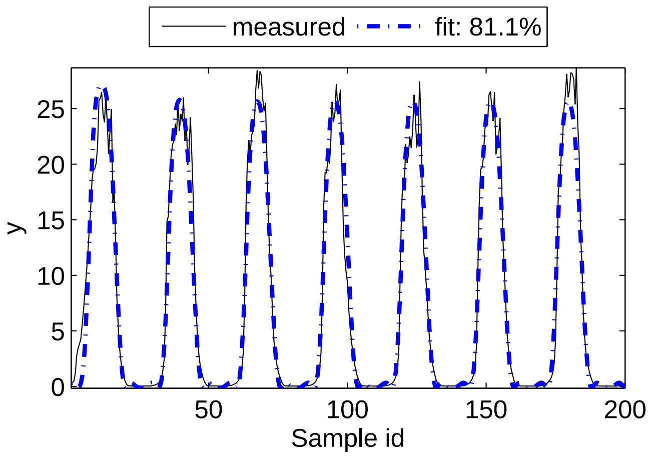
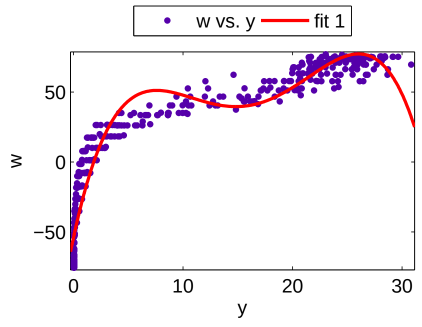


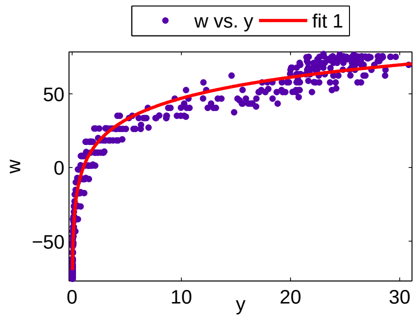
Figure 4.8: Model fit for the case of (n,r) = (2,5)

In order to estimate the inverse static nonlinear model at the output, the estimated linear model was simulated to compute the intermediate output variable data w. The data of set w and the measured output data (y) was then used to model the inverse static nonlinear function. We investigated the suitability of different nonlinear functions in order to implement the inverse static nonlinear component adhering to the properties mentioned in Section 4.2.4. Figure 4.9 illustrates the mode fit for a polynomial, logarithm of the form $w = a \times log(y(k)) + c$ and power function of the form $w = a \times y(k)^b + c$.

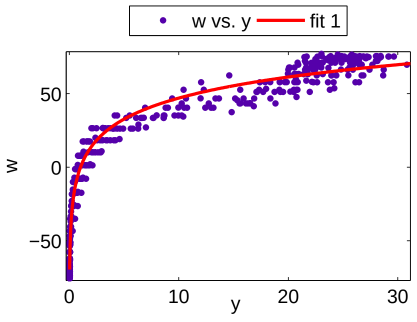
The polynomial fit indicates that there is no monotonic relationship for $w = g^{-1}(y)$, which violates the *property 1*. The logarithm and power functions show monotonic relationships, however the logarithms are much suitable because of the low computational overhead. As a consequence, we modelled the inverse output nonlinear function with a



(a) Polynomial function



(b) Log function



(c) Power function

Figure 4.9: The model fit of the inverse output nonlinearity with different function types

logarithm function. The estimated model with R^2 =0.98 fit is as follows:

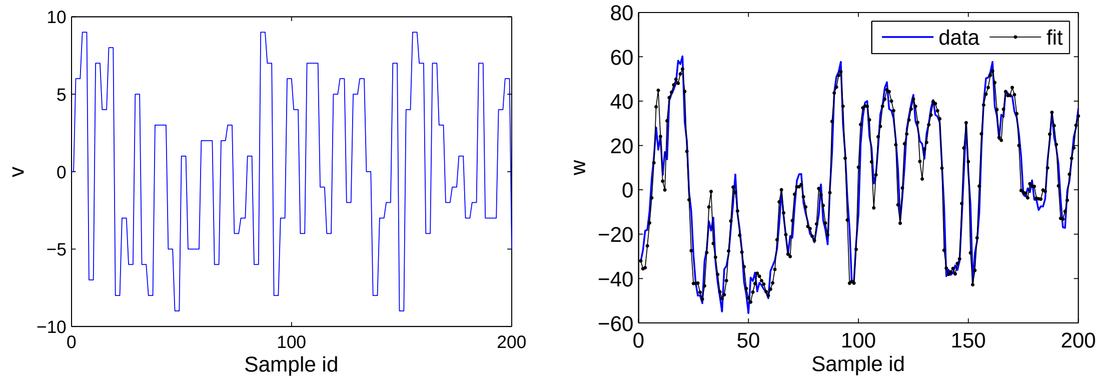
$$\begin{aligned} w &= g^{-1}(y) \\ &= 20.63log(y(k)) - 0.45331 \end{aligned} \tag{4.10}$$

Estimation of the linear model: The linear model estimated in the previous section with the *nlhw* command had a unit coefficient on the input, which means linear system gain has been lumped together with the static nonlinear gains[3]. In order to find the exact gain of the linear system, a second identification experiment is designed after integration of the compensators at the input and output. A pseudo random input signal was used and a 20 requests/sec were used to simulate the workload of each class. A first order model showed high model fit with $R^2 = 0.95$. The estimated linear model is shown in equation (4.11) and Figure 4.10 shows the model fit with the test set data.

$$w(k+1) = 0.85w(k) + 1.97v(k) \tag{4.11}$$

In order to compare the quality of this model, a linear model was also estimated using

[3]This is to avoid the issue of over-parameterization exist in the identification of Wiener model, See Section 4.2.6



(a) Input signal (b) Model fit

Figure 4.10: Input and the model fit of the linear component of Wiener model

the data gathered to construct the linear component of the Hammerstein-Wiener model. However, $u - y$ data was used to estimate the single linear model of the system. The estimated model with $R^2 = 0.88$ fit is shown in equation 4.12. Figure 4.11 shows the model fit. One of the main observations is that due to the output nonlinearity, the model fit is poor in the region where output decays compared to region where output increases at a rapid rate.

$$y(k+1) = 0.81y(k) + 0.72u(k) \tag{4.12}$$

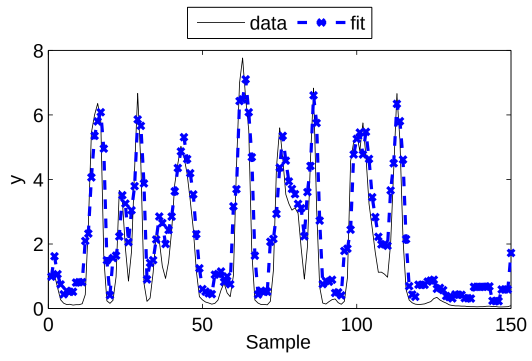


Figure 4.11: The linear model fit

To further evaluate the model estimation process presented in this section, the estimation results for a system with more than two classes are listed in Appendix F, while the experimental results of real-world case studies can be found in Chapter 8. These results also yield the same conclusions presented in this section.

4.3 Identification of Absolute Performance Management Scheme

This section presents the details of the model estimation procedure for absolute management scheme. Subsections 4.3.1, 4.3.2 and 4.3.4 cover the system architecture for absolute performance management scheme, classification of the nonlinearities that exist in the system and new estimation mechanism respectively. Finally, the model validation results will be given in 4.3.7.

4.3.1 Formulation of Absolute Management System

The general settings of the absolute management scheme for a system with n classes can be found in Section 3.2.2.1. In order to explain how the absolute management system is formulated, we take the following example of a system serving two classes.

Example : Let us consider two classes, $class_0$ and $class_1$. $R_0(k)$ and $R_1(k)$ are the response times of these classes respectively at the k^{th} sample. Further, the resource caps are $S_0(k)$ and $S_1(k)$ where $S_0(k) + S_1(k) \leq S_{total} = 30$ and $S_{0,min},\ S_{1,min} = 6$. S_{total} is the total available resources and $S_{0,min},\ S_{1,min}$ are the minimum resource limits of each class. According to the absolute performance and resource management scheme, the objective is to maintain, $R_0(k)$ and $R_1(k)$ around the specified values $R_{SLA,0}(k)$ and $R_{SLA,1}(k)$, while manipulating $S_0(k)$ and $S_1(k)$ in each sample instance. This is because when the workloads of different classes change unpredictably, the output variables will be significantly affected. In addition, when the input $S_0(k)$ changes, the output $R_1(k)$ will be affected. Similarly, $R_0(k)$ will be affected when $S_1(k)$ changes. This indicates that there are interactions between the system variables. In addition, there are tight constraints on the system inputs as well. The open-loop absolute management system is shown in Figure 4.12.

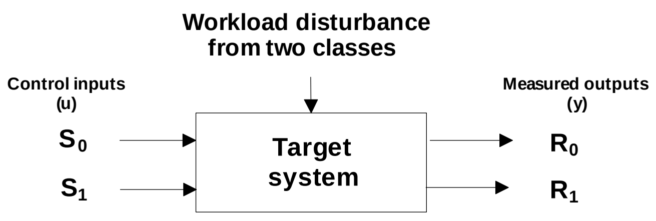


Figure 4.12: Open-loop absolute management system with two classes

4.3.2 Classification of Nonlinearities of Absolute Management Scheme

In this section, we take the example in Section 4.3.1 to identify the nonlinearities that may affect the runtime management system.

In order to investigate the behavior of the inputs $(S_0(k), S_1(k))$ and outputs $(R_0(k), R_1(k))$ we conduct the following experiment. Let us observe the behavior of $R_0(k)$ when $S_0(k)$ is changed from 18 to 6 while maintaining the workload of $class_0$ constant (e.g., 30 requests/sec). Meanwhile, $S_1(k)$ is fixed at 6 under a constant workload maintaining $R_1(k)$ at a steady state. Therefore, the effects of variations in $S_0(k)$ is minimal on $R_1(k)$. Figure 4.13 shows the behavior of the output signal for two such workloads.

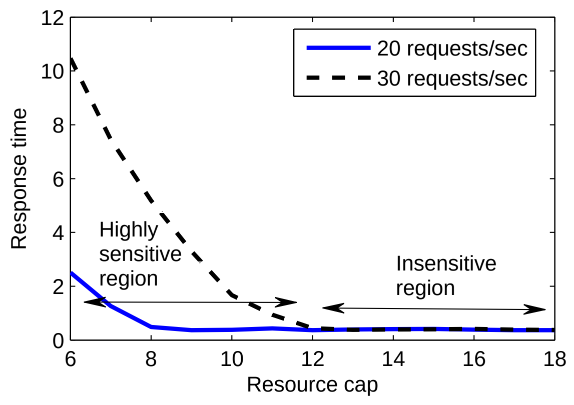


Figure 4.13: Behavior of the output with respect to varying resource cap under different workloads

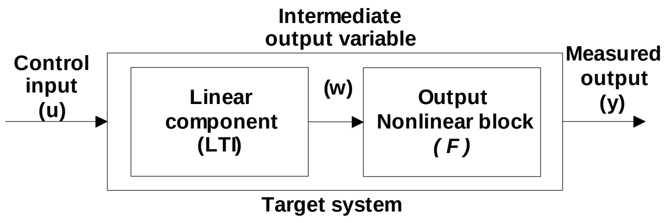
From Figure 4.13 it is evident that the relationship between the input and output is nonlinear. In addition, depending on the workload intensity the nonlinear behavior changes as well. Furthermore, there are two regions in these curves. Firstly, when the resource cap is sufficient to manage the applied workload the response time (or the output) remains at a steady state. This region is called as the *insensitive region* because there are multiple operating points that the management system can select to settle down, without affecting the response time. For instance, for a 30 requests/sec workload a control system can settle to 12 resources or more without affecting the response time. Secondly, in the *sensitive region*, the response time increases at a high rate when the resource cap is insufficient to cater the workload. This means, queues of the system start to fill up. If the resources are not managed efficiently the performance of a particular class may degrade or a large workloads may get rejected while leading to system failures.

Another observation apart from the nonlinear behavior is the resource cap S_i is inversely related to output R_i. i.e., $S_i \propto \frac{1}{R_i}$. Further, the response time curves in Figure 4.13 show that in the sensitive region, the gap between consecutive points increases rapidly. This means, under a sudden workload variation, produced feedback signals miss a lot of information in-between due to the rapid variations of the output signal. The initial investigation with the linear control methodology indicated that such information loss affects the modelling of the system and the runtime control. To reduce the impact of this factor the output is inverted, that is $\frac{1}{R_i}$ is considered as the output. This conversion damps out such high variations. Consequently, inverting the response time reduces the numerical sensitivity leading to better model fits and runtime control. This conversion has been used in many existing works (e.g., [238, 265]) to improve runtime performance management. The same conversion will be used in this work as well. For notational simplicity let us represent $\frac{1}{R_i(k)}$ as $y_i(k)$ and $S_i(k)$ as $u_i(k)$, where $i = 0 \dots n-1$. However, it is worth noting that this simple inversion does not remove the nonlinear behavior exhibited by the absolute management system illustrated in Figure 4.13.

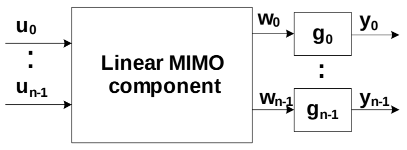
The above case examines when a single class changes its workload or resource cap. However, typically the workloads of different classes vary in unpredictable fashion over time, which demand an efficient management scheme to alter the resource caps at runtime in order to achieve the control objectives. The nonlinear behavior may also impact on the performance management capabilities of a control system. Initial investigations to design a linear MIMO control system to solve the above management problem indicated fair performance and resource management capabilities [193]. However, the idea behind this work is to investigate whether we can further improve the efficiency of this management scheme by considering the nonlinear behavior explicitly at the design time compared to the existing linear control scheme.

Similar to Section 4.2, in this work, the motivation is to estimate the nonlinearities that exist in the target system and then compensate them by integrating a compensator framework. However, the absolute performance management scheme does not incorporate ratios in the inputs and outputs compared to the relative management scheme. As a result, if the inputs ($u_i, i = 0, 1, \dots n-1$) are examined they have equal spacing in-between the consecutive operating points. Therefore, we assume that there are no input nonlinearities, avoiding the need of estimating the Hammerstein component in the modelling process. This means that the model structure considered is a MIMO Wiener block-oriented model.

4.3.3 Wiener Model



(a) SISO Wiener model



(b) MIMO Wiener model

Figure 4.14: Block diagrams of SISO Wiener model and MIMO Wiener model

As covered in Section 4.2, the Wiener model consists of a linear component followed by a nonlinear component (see Figure 4.14a). The nonlinear component captures the static output nonlinearities, while the linear component captures the rest of the dynamics. Section 4.2 provided details about the single and two step identification techniques for SISO Wiener model. Compared to the SISO Wiener model identification, the MIMO Wiener model identification is significantly difficult. This is because if there are interactions between the intermediate variables and the outputs, those have to be estimated in the nonlinear component of the Wiener model [27, 171]. Due to this difficulty, different model structures have been utilized by the existing literature [27, 33, 45, 171, 174, 207, 245]. Some approaches have captured these interactions (e.g., [27, 171]), while some have ignored them (e.g., [45, 174, 207]).

4.3.4 Model Identification Procedure of Absolute Management Scheme

In this work, we use two step design approach similar to Section 4.2. In addition, we make the working assumption that the interactions are negligible between the intermediate variables and the outputs of the system to achieve some convenience in the model identification. Figure 4.14b illustrates the block diagram of MIMO Wiener model considered in this work (similar to work in control engineering literature [45, 174]). The nonlinear component at the each output represents the static nonlinear relationship between the intermediate output variable and the corresponding output variable. In order to compensate the estimated nonlinearities their inverse functions are then estimated. Using the inverse function a set of compensators is developed and integrated into the system before MIMO linear model is estimated to capture the rest of the dynamics using the inputs and intermediate output variables.

In the following sections we introduce the estimation procedures of the nonlinear component at each individual output and MIMO linear model to represent the absolute management system using MIMO Wiener model.

4.3.5 Estimating the Output Nonlinear Components

In order to estimate the output nonlinear block at the each output (see, Figure 4.14b), the MIMO system with n classes can be decomposed into n SISO subsystems. Then, the same estimation methodology presented in Section 4.2.6 can be applied to estimate each SISO subsystem as a SISO Wiener model. This application can be performed in two ways: 1) decompose the system to n subsystems and identify each one individually or 2) use a single subsystem as a representation of all subsystems and identify that specific subsystem. The design complexity of the first approach in a case of a shared resource environment serving multiple classes is high. In contrast, the second approach reduces the design complexity significantly, however makes the assumption that all subsystems illustrate similar nonlinear behavior. Due to low design cost, we adopt the second approach, but in order to validate the assumption of similar nonlinear behavior by all classes, we conduct the following statistical test in a shared resource system serving three classes.

Here we utilize the example in Section 4.3.1 to explain the simulation setup. The resource cap (u_i) of the i^{th} class, where i = 0, 1, 2 is changed from 18 to 6 while applying a constant workload and the respective output (y_i) is observed during that time period. Meanwhile, the resource caps (u_j), $j \neq i$, are fixed at 6 under constant workloads maintaining the respective outputs at the steady states. Thus, changes of the resource allocation settings of the treatment class have low impact on the other classes and vice versa. The same experiment was conducted for all these classes under 5 different workloads. The gathered output data ($y_i(k)$, i = 1, 2, 3) from each of these experiments is used to conduct a *Kruskal-Wallis statistical test* [44], in order to validate the hypothesis that the nonlinear behavior illustrated is similar or significantly different. Here, the null

Table 4.3: Kruskal-Wallis test results comparing data of three classes under different workload conditions

Workload (requests/sec)	p-Value
40	0.9275
30	0.9949
20	0.9101
18	0.8955
12	0.2722

hypothesis (H_0) is that the gathered data from different classes is different, i.e. nonlinear behavior is different from one class to another. The alternative hypothesis (H_1) is that the gathered data from different classes is similar, i.e. the nonlinear behavior is similar between different classes.

The results of the Kruskal-Wallis test shown in Table 4.3 indicate that the p-values are greater than 0.01 for all workload settings. This means that the data sets of all three classes do not show any significant difference. Hence, we can reject (H_0) and accept (H_1) concluding that the outputs of different classes illustrate similar behavior at different workload rates.

The similar nonlinear behavior of each class justifies that design of a single compensator may be enough to compensate the nonlinear behavior of each class. As mentioned before, we use the same design procedure covered in Section 4.2.6 to design this compensator. Select an arbitrary class to conduct a SISO Wiener model based experiment. Without loss of generality let us select $class_0$. Then, design a suitable input signal for $u_0(k)$ to conduct the nonlinear SID experiment. A sinusoidal signal is selected similar to Section 4.2.6. The design of this experiment has to be done without violating any of the hard constraints of the system (see, Section 4.3.1). Then, a suitable workload is applied on the system for the resource caps selected in the signal which would move the system to the sensitive and insensitive regions (see, Figure 4.13). However, maintaining the output in each region for a long time has to be avoided. This is because the estimated data will have multiplicities, which leads to the violation of property 1 listed in Section 4.2.4. Meanwhile other control inputs $u_i(k), i = 1 \ldots n-1$ are fixed at $S_{i,min}, i = 1 \ldots n-1$ under a constant workload maintaining their outputs at a steady state. The SISO Wiener modelling procedure presented in Section 4.2.6 is then applied on the input-output ($u_0(k)-y_0(k)$) data gathered from this experiment to estimate the linear and nonlinear components of the SISO Wiener model. Afterwards, the intermediate variable $w_0(k)$ data is computed followed by the estimation of the inverse nonlinear function $w = g^{-1}(y)$ (lower order polynomial are used in this work).

After the derivation of this inverse output nonlinear function g^{-1}, it is implemented as a software component and integrated just after each output of the system as the nonlinear compensator. The open-loop system with the compensators is shown in Figure 4.15.

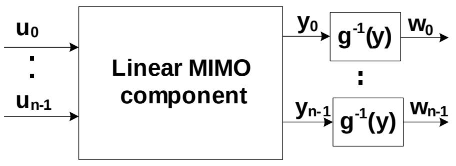


Figure 4.15: MIMO Wiener open-loop system with the compensators

4.3.6 Estimating the MIMO Linear Component

After the integration of the nonlinear compensators at the each output of the system, the system is assumed to be linear, hence the next step is to approximate the rest of the dynamics into linear component of the MIMO Wiener model (see Figure 4.14b).

There are two main techniques. The first technique is to change or excite a single input at a time keeping other inputs at desired (steady state) values and observe the outputs. The same process is then applied for other inputs. Afterwards, the MIMO model is constructed based on the gathered data. The drawbacks of this approach are the time taken to construct the model, high level of manual work, insufficient capture of the input-output interactions in the model and merging of data or models to arrive at the final model [222]. The second technique is to simultaneously excite all the inputs in the system and observe all outputs. The gathered data is then fitted into a model using *multivariable regression*. This approach addresses most of the above limitations, but conducting such an experiment may not be always possible in some systems due to safety reasons. However, if it is possible to conduct such an experiment, this second approach produces a high quality model as recommended in [46, 222] and proved in [67]. The design of the input signal in SID is important to excite the dynamics of the system. The pseudo random binary (PRB), sinusoidal or pseudo random signals are extensively used as input signals in the literature. When designing these input signals, the hard (input) constraints should not be violated.

In this work, using the second approach described above, a MIMO linear SID experiment is designed simulating all the inputs using pseudo random binary (PRB) signals. As a consequence, here we estimate the linear model capturing the interactions between the inputs and the transformed outputs. The levels of the input signal have to be carefully selected without violating any of the hard constraints on the inputs. Suitable workloads have to then be selected in order to maintain the outputs in the required region of operations. This region is selected based on the desired values of the response times ($R_{SLA,i}, i = 0, 1, \ldots n-1$). The experiment is conducted to gather input-output $u(k)$ and $w(k)$, where $u(k) = [u_0(k)\ u_1(k)\ \ldots\ u_{n-1}(k)], w(k) = \left[g^{-1}(y_0(k))\ g^{-1}(y_1(k))\ \ldots\ g^{-1}(y_{n-1}(k))\right]$, which will be used in multi-variable regression to construct MIMO ARX model (see, equation (4.13)).

$$w(k) = \sum_{i=0}^{p} A_i w(k-i) + \sum_{j=0}^{q} B_j u(k-d-j) \tag{4.13}$$

where, p, q - the order of the model, (A_i, B_j) are the parameter matrices with $n \times n$ dimension, d - is the delay and k is the current sample instance.

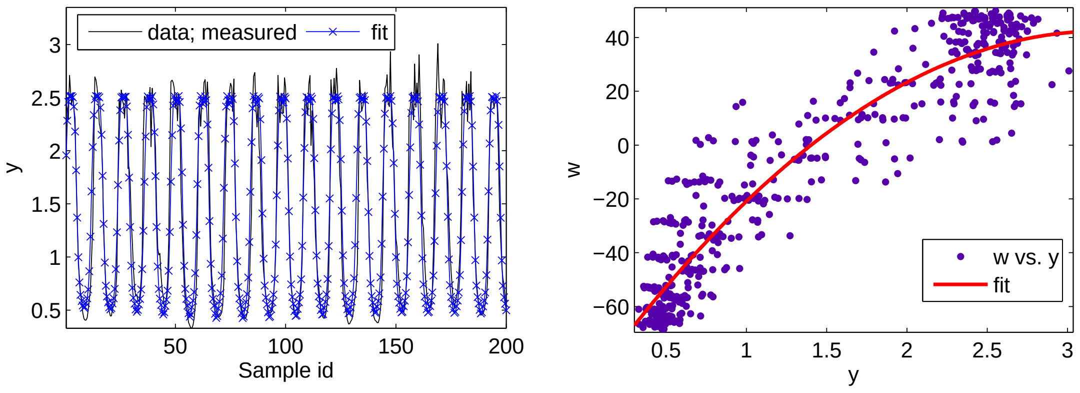
4.3.7 Model Identification Results of Absolute Management Scheme

In this section, we use the settings of the example in Section 4.3.1 and the design procedure covered in Section 4.3.5 to estimate the nonlinear compensator at each output. The MIMO linear model is then identified.

Estimation of output nonlinear component and its inverse: We designed the SISO Wiener SID experiment using a sinusoidal signal in order to simulate the input of $class_0$. The resource caps from 9 to 13 were selected to design the input signal. 25 requests/sec workload was applied by $class_0$ and the output R_0 data was gathered for 600 samples. Meanwhile, the $class_1$ was maintained at the steady state without affecting the behavior of $class_0$. The gathered data pairs were used to estimate a SISO Weiner model and the inverse nonlinear function as explained in Section 4.3.5. Figure 4.16 shows the model fit and equation (4.14) shows the inverse nonlinear function.

$$\begin{aligned} w_0(k) &= f^{-1}(y_0) \\ &= -12.85 y_0(k)^2 + 82.76 y_0(k) - 90.82 \end{aligned} \tag{4.14}$$

The equation (4.14) was then implemented and integrated to the system as a component at each output, transforming the open-loop system as shown in Figure 4.15.



(a) Model fit of the Wiener model estimation (b) Inverse nonlinear function model fit

Figure 4.16: Model fit of the inverse output nonlinear funciton

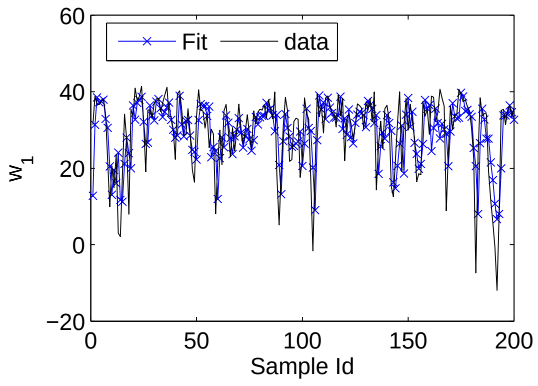
Estimation of MIMO linear model: After the integration of the output compensators, the next step is to estimate the MIMO linear model capturing the relationships between u_0, u_1 and w_0, w_1. Here, we designed a pseudo binary signal selecting 8 and 10 as the resource caps for each of the input, with a switching frequency of 2 samples. 20 requests/sec workloads were applied for both $class_0$ and $class_1$ for 600 samples. The gathered data was then used to estimate a MIMO ARX model. The finalized model is shown in equation (4.15).

$$w(k+1) = \begin{bmatrix} 0.5592 & -0.0316 \\ -0.0101 & 0.6014 \end{bmatrix} w(k) + \begin{bmatrix} 2.9253 & -1.4053 \\ -1.3768 & 2.6580 \end{bmatrix} u(k), \qquad (4.15)$$

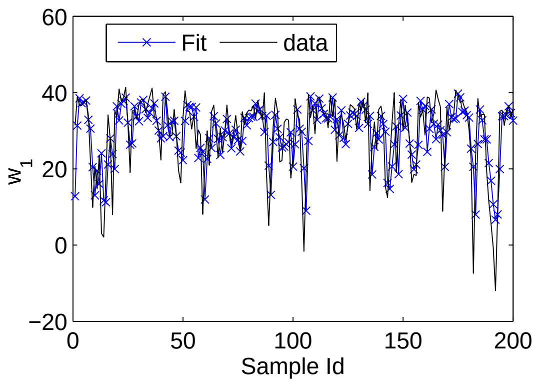
where $w(k) = [w_0(k)w_1(k)]^T$ and $u(k) = [u_0(k)u_1(k)]^T$.

The model fits for each output are shown in Figure 4.17. The model fit is sufficient, however, $R^2 = 0.5$. The second or higher order models did not improve the model fit over R^2=0.5.

In order to compare the nonlinear model, we estimated a MIMO linear model as well. A linear model was estimated under the same settings used to construct the MIMO linear component of the Wiener model (see, Section 4.3). The equation (4.16) shows the model and Figure 4.18 illustrates the model fits. Although the Wiener model has improved the model fit, it is important to note that both models show lower model fits compared to the general model fit threshold qualify as a good model fit (i.e, $R^2 > 0.7$). Chapter 5 further evaluates the performance management capabilities of the control systems implemented



(a) Model fit w_0



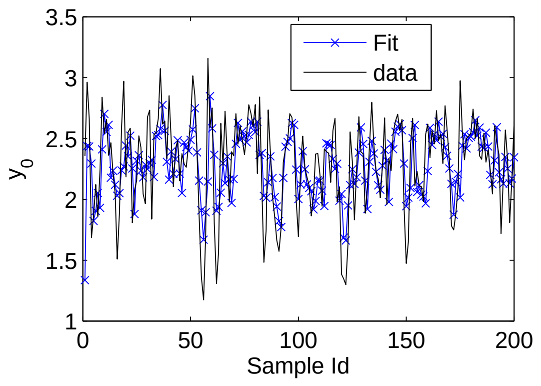
(b) Model fit w_1

Figure 4.17: Model fit of MIMO linear components

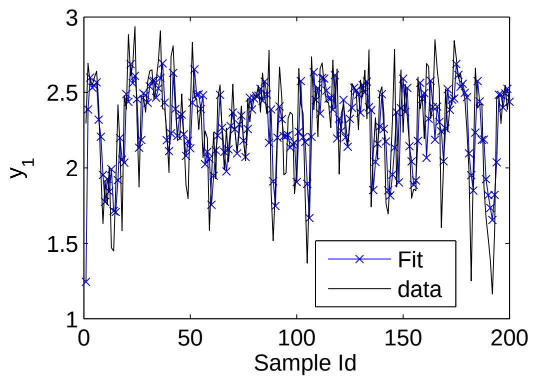
out of these models. In addition, Appendix F covers model validation results of a system serving more than two classes. Chapter 8 presents the model estimation details of absolute management system for a real-world case study.

$$y(k+1) = \begin{bmatrix} 0.4817 & -0.0145 \\ -0.0146 & 0.5131 \end{bmatrix} y(k) + \begin{bmatrix} 0.1446 & -0.0110 \\ -0.0079 & 0.1323 \end{bmatrix} u(k), \tag{4.16}$$

where $y(k) = [y_0(k)y_1(k)]^T$ and $u(k) = [u_0(k)u_1(k)]^T$.



(a) Model fit y_0



(b) Model fit y_1

Figure 4.18: Model fit of MIMO linear model

4.4 Conclusion

This chapter has presented the nonlinear block-oriented model identification approaches to represent the relative and absolute performance management schemes. The model identification procedures have been put forward to capture the characterized nonlinear dynamics of each scheme explicitly, which subsequently enabled the development of nonlinear compensators, reducing the impact of the nonlinear dynamics on the management system. In particular, for case of relative performance management system, the Hammerstein-Wiener model has been used to represent the existing input and output nonlinear dynamics, while a MIMO Wiener model has been used to represent the nonlinear dynamics of absolute management system. This is the first time such models and compensator frameworks have been used to represent multi-class shared resource software system in the literature. In addition, this chapter also provides model validation results of these approaches, which have indicated better model fits compared to the existing linear model estimation methods.

Chapter 5

Performance Management Using Nonlinear Feedback Control

5.1 Introduction

The previous chapter has presented the system identification procedures, to estimate nonlinear block-structure models of the relative and absolute performance management systems. This chapter presents the design and implementation steps of the new control system architectures based on those nonlinear models. The main novelty of the control systems proposed in this thesis is the integration of compensators that will reduce the impact of nonlinear dynamics on the feedback control systems. This is new in the field of performance and resource management studies of software systems. The experimental results will show that the proposed nonlinear control systems provide much better performance and resource management in a multi-class shared resource system in comparison to the traditional linear control methods. Although the nonlinear compensators are added to the control systems, well established control system design methodologies can still be used to provide a systematic and formally grounded processes in the design and implementation of these control systems. This is one of the attractive features of the block-oriented nonlinear models utilized in this work.

Section 5.2 of this chapter covers the Hammerstein-Wiener control system design process, followed by its evaluation in Section 5.3 for the case of relative performance management scheme. Section 5.4 presents the details of MIMO nonlinear control system design method for absolute management scheme while its evaluation is given in Section 5.5.

5.2 Relative Performance Management Using Nonlinear Control

In this section, first, we explain the existing linear control system architecture of the relative management system (see Section 5.2.1). The architecture and design details of the nonlinear control system based on the Hammerstein-Wiener block-oriented model is presented in Section 5.2.2.

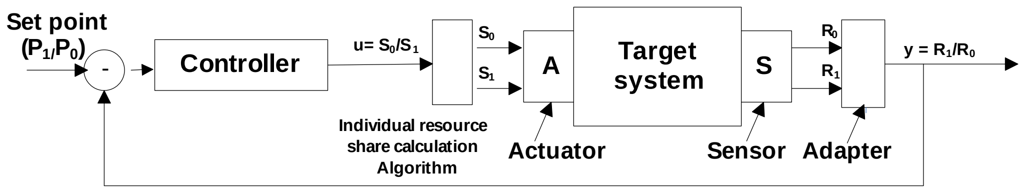
5.2.1 Linear Control System Design for Relative Performance Management

Here, we continue with the two examples covered in Section 4.2. For a system serving two classes (i.e., the *Example-1*), the closed-loop linear control system architecture is illustrated in Figure 5.1a. The control objective according to the relative management scheme is to maintain the response time ratio $y(k) = \frac{R_1(k)}{R_0(k)}$ of (0) and (1) classes around $\frac{P_1(k)}{P_0(k)}$ while computing the resources cap ratio $u(k) = \frac{S_0(k)}{S_1(k)}$. Furthermore, the hard constraints on the total number of resources and reserved per-class resources ($S_{0,min}, i = 0, 1$) have to be maintained by this control system.

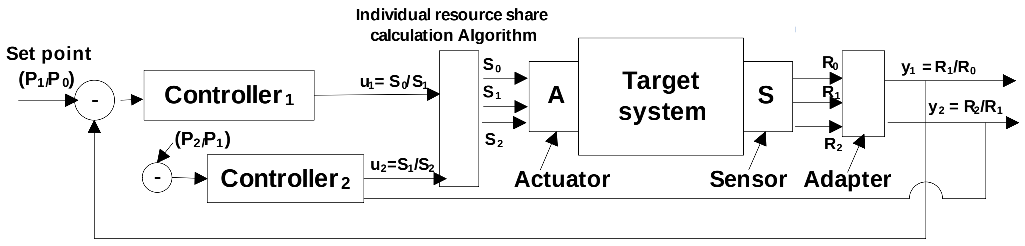
The main challenge under these specifications is maintaining the output around performance differentiation ratio by calculating the resource cap ratio dynamically under unpredictable workload conditions of two classes [145]. To address this challenge, Lu et al. proposed a dynamic propositional resource allocation approach in [145] to achieve the aforementioned relative performance management objectives in a shared resource environment (web server) with multiple classes. Their approach was to develop a linear feedback control system, which automates the computation of resource cap ratio $\frac{S_j}{S_i}$ at runtime.

For the case of a system serving two classes, a single SISO controller is sufficient to achieve the required control objectives. It is also worth noting that, as shown in Figure 5.1a, at each sample instance, computed resource cap ratio $u(k)$ by the controller is sent to the *individual resource cap calculation algorithm* proposed in [145] for the calculation of individual resource caps $S_i(k)$ (i= 0, 1). This algorithm is listed in Appendix D, which also takes into account the constraints imposed on the management system.

In contrast, for the case of three classes as presented in *Example-2* in Section 4.2, the target system becomes a MIMO system (see, Figure 4.1b). As a result, a control system with two loops as illustrated in Figure 5.1b is utilized in the existing literature. Each



(a) Two classes



(b) Three classes

Figure 5.1: Linear control system for relative management with two and three classes

loop $c = 1, 2$ maintains the specified output ratios ($y_c(k) = \frac{R_i(k)}{R_{i-1}(k)}$) around ($\frac{P_i(k)}{P_{i-1}}(k)$) by computing the resource cap ratios $u_c(k) = \frac{S_{i-1}(k)}{S_i(k)}$. Finally, at each sample instance, calculated resource cap ratios $u_c(k)$ by each controller is sent to the individual resource cap calculation algorithm, in order to compute the individual resource caps $S_i(k)$ (i= 0, 1, 2).

A possible question is why multiple SISO controllers have been utilized in the relative management scheme compared to a single MIMO controller. This is because the relative management scheme is complex due to consideration of the ratios, which transforms system inputs and outputs significantly. As a consequence, to calculate the individual resource caps using the algorithm listed in Appendix D, the summation of the individual resource caps should exactly equal to the total number of resources. i.e., $S_0 + S_1 \ldots S_n = S_{total}$. This means, when we calculate the resource cap ratios of multiple classes, they become linearly dependent. As a result, conducting a MIMO system identification experiment using $\frac{S_i(k)}{S_j(k)}$ as inputs becomes mathematically ill-conditioned.

The advantage of this formulation is that it enables maintaining the system outputs according to priority levels under versatile workload conditions and it reduces the dimension of the system to $n - 1$. So that, the assumption of the relative management system design is the interactions between inputs and outputs are insignificant as stated in the existing literature [145, 150, 180]. Therefore, we will treat each relative management system controlling a pair of classes as a SISO system.

5.2.2 Hammerstein-Wiener Control System Design

This section overviews the development of the closed-loop system equipped with the compensators based on the Hammerstein-Wiener model. In Section 4.2, the input and output nonlinear compensators were introduced into the system, subsequently the system can be assumed to be linear. As a result, we do not have to design a complex nonlinear (e.g., fuzzy, neural network based [81]) controller for the system under consideration. With the linear system model identified in Section 4.2.7, we can design a linear controller using the existing design techniques. Any type of linear controller can be used in such a design (see Appendix B), including proportional-integral (PI) and predictive controllers. For this SISO control system design we will use a PI controller, due to its robustness, disturbance rejection capabilities and simplicity [81].

Before explaining the design of the Hammerstein-Wiener nonlinear control system, let us overview the well-established design process of PI controller. Figure 5.2 shows the structure and variables involved in a typical control system. For the simplicity of analysis we have represented the variables in *z-transforms* [81].

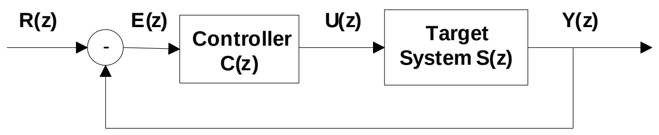


Figure 5.2: Block diagram of a standard control system

The reference (or set point), output and control input variables are represented by R(z), Y(z) and U(z) respectively. E(z) represents the *control error*, computed by $R(z) - Y(z)$. The transfer function of the target system and controller are represented by S(z) and C(z) respectively. The model of the target system estimated by system identification can be used to formulate S(z). Assuming, the model is a first order ARX model, we can represent S(z) as follows:

$$S(z) = \frac{Y(z)}{U(z)} = \frac{b_1}{z + a_1} \tag{5.1}$$

The representation of C(z) depends on the selected controller. For the case of PI controller, the standard equation is as follows:

$$C(z) = \frac{U(z)}{E(z)} = K_p + \frac{K_i}{1 - z^{-1}} = \frac{(K_i + K_p)z - K_p}{z - 1} \tag{5.2}$$

where, K_p (proportional gain) and K_i (integral gain) are called the *gains* of the PI controller. Deriving suitable gains is paramount to maintain the stability and tune the performance of the controller to a satisfactory level. As opposed to the trial and error methods, there are systematic and formal techniques in control engineering to compute the controller gains depending on the performance requirements. The pole-placement design methodology [81] is one such method to compute the gains of the PI controller. In this work, we follow the pole-placement design method as described below.

First, the closed-loop transfer function has to be derived using the equations (5.1) and (5.2) as shown in equation (5.3).

$$\frac{Y(z)}{R(z)} = \frac{C(z)S(z)}{1 + C(z)S(z)} \tag{5.3}$$

The roots of the polynomial in the denominator of equation (5.3) are called the *closed-loop poles*. Theoretically, for closed-loop stability these poles should be within the unit circle or less than 1 in magnitude [81].

Second, according to the above theory, the control system designer can specify the desired locations for the closed-loop poles. This can be done by defining two poles (say, α and β) and deriving the so-called *desired closed-loop polynomial* according to the performance specifications (e.g., settling time and overshooting). The structure of the desired closed-loop polynomial is shown in equation (5.4).

$$Dcl(z) = z^2 - (\alpha + \beta)z + \alpha\beta \tag{5.4}$$

Third, by equating the denominator of equation (5.3) to the desired closed-loop polynomial equation (5.4), we can calculate the gains of the PI controller as follows.

$$z^2 + (b_1(K_p + K_i) - 1 + a_1)z - b_1K_p - a_1 = z^2 - (\alpha + \beta)z + \alpha\beta$$

$$K_p = \frac{\alpha\beta + a_1}{-b_1} \tag{5.5}$$

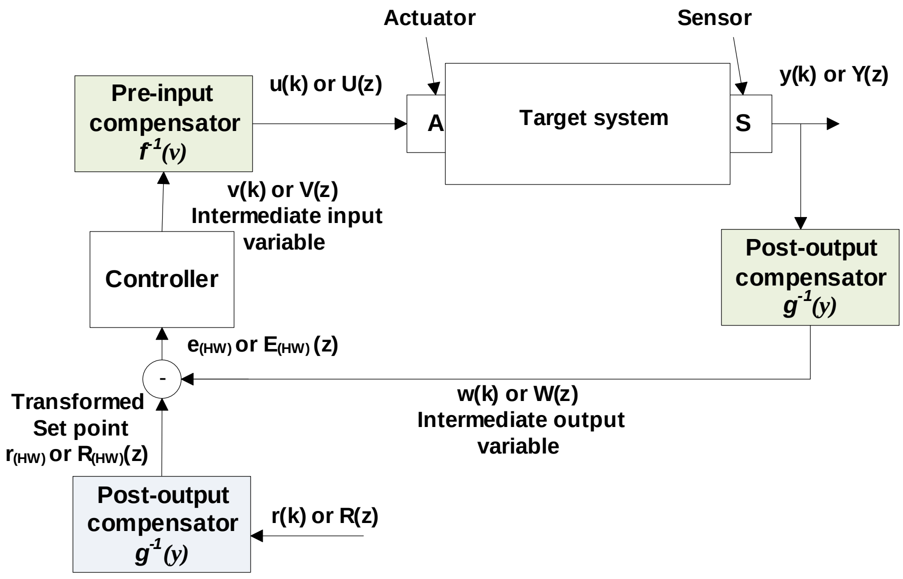
$$K_i = \frac{(\alpha + \beta) + 1 - a_1}{b_1} - K_p \tag{5.6}$$

With this background let us now move on to the implementation of Hammerstein-Wiener control system. As shown in Figure 5.3, the Hammerstein-Wiener control system operates with transformed variables. With the addition of the nonlinear compensators, the target system is represented by the linear component of the Hammerstein-Wiener model (see Section 4.2.7). Hence, the transfer function of the target system $S_{(HW)}(z)$ can be represented as follows:

$$S_{(HW)}(z) = \frac{W(z)}{V(z)} = \frac{b_1}{z + a_1} \tag{5.7}$$

The main difference in equation (5.7) compared to equation (5.1) is the input and output variables. Similarly, the transfer function ($C_{(HW)}(z)$) of the PI controller based on the Hammerstein-Wiener model can be represented by equation (5.8).

$$C_{(HW)}(z) = \frac{V(z)}{E_{(HW)}(z)} = \frac{(K_{i(HW)} + K_{p(HW)})z - K_{p(HW)}}{z-1} \tag{5.8}$$



Actuator
Sensor
Pre-input compensator $f^{-1}(v)$
u(k) or U(z)
A
Target system
S
y(k) or Y(z)
v(k) or V(z) Intermediate input variable
Controller
Post-output compensator $g^{-1}(y)$
e(HW) or E(HW)(z)
-
Transformed Set point r(HW) or R(HW)(z)
w(k) or W(z) Intermediate output variable
Post-output compensator $g^{-1}(y)$
r(k) or R(z)

Figure 5.3: Block diagram of the Hammerstein-Wiener control system

The main changes from the standard form are the transformation of controller error e(k) (or E(z)) and controller input v(k) (or V(z)). $e_{(HW)}(k)$ is computed by $r_{(HW)} - w(k)$, where $r_{(HW)}(k) = g^{-1}(r(k))$ is the transformed reference signal from the original reference signal $r(k)$ and $w(k) = g^{-1}(y(k))$. g^{-1} is the estimated inverse output nonlinear function (see Section 4.2.6). It is also important to note that $v(k)$ computed by the Hammerstein-Wiener controller has to be converted to the original system input $u(k)$ before applying it in the target system. This conversion is performed by the pre-input compensator using $u(k) = f^{-1}(v(k))$, where f^{-1} is the estimated inverse input nonlinear function (see Section 4.2.5). For, notational consistency the gains of Hammerstein-Wiener model based controller are represented by $K_{p(HW)}$ and $K_{i(HW)}$. Although the target system and controller transfer equations are formulated by the transformed variables, we can still use the above described formal control system design methodology (i.e., pole-placement) to compute $K_{p(HW)}$ and $K_{i(HW)}$ using equations (5.5) and (5.6) respectively. α and β have to be decided by the control system designer, while a_1 and b_1 are estimated by the system identification method presented in Chapter 4.

Furthermore, the control system is also constrained by the resource limits of the target system. The input limits of the original system is u_{min} and u_{max}, which are the minimum and maximum values of the input u (see Section 4.2.5 for definition of u). These limits also have to be translated and implemented in the Hammerstein-Wiener control system. In the Hammerstein-Wiener controller, these limits correspond to v_{min} and v_{max}. Consequently, the control inputs $v(k)$ generated by each controller is compared with v_{min} and v_{max} in each sample and rounded-off to the corresponding limit if they are exceeded.

Finally, the resource cap ratios $u_c(k),\ c = 1, \dots n-1$ calculated by the controllers managing class pairs $i-1$ and i have to be decoded back to the individual resource cap for each class $S_i(k)$, $i = 0, 1, \dots n-1$. The algorithm used for this calculation is listed in Table 5.1. This algorithm has extended the algorithm presented in [145] (see Appendix D). The major modification is the conversion of values ($v_c(k)$) computed by the nonlinear control system to the original system inputs $u_c(k)$ using f^{-1} function, before computing the individual resource caps for each class. This algorithm is then implemented in the actuator of the Hammerstein-Wiener control system. After the integration of the PI controller, the final closed-loop control system takes the structure of Figure 5.3.

Table 5.1: Algorithm for resource cap calculation extending [145]

```
$RS_i$: normalized resource allocation of class i relative to
class $n-1$, $i = 0, 1, \dots n-1$.
$RS_i(k) = \frac{S_i(k)}{S_{i-1}(k)}$, where k - sample instance.
$sum$: the sum of the normalized process budgets of all classes.
$M$: total number of resource units.
Begin Algorithm
$RS_{n-1} = 1$
$sum = 1$
$for(int\ j = n-2; j \geq 0; j--)$ {
//Call the Hammerstein controller to get ratio $V_{j+1}(k)$
between QoS class j and j+1.
//Convert $V_{j+1}(k)$ into original share ratio $U_{j+1}(k)$
using the equation estimated in Section 4.2.5)
    $\mathbf{U_{j+1}(k) = f^{-1}(V_{j+1}(k))}$
    $RS_j(k) = RS_{j-1}(k) * U_{j+1}(k)$
    $sum = sum + RS_j(k)$
}
$for(int\ j = n-1; j \geq 0; j--)$ {
    $Sj(k) = M * (RS_j(k)/sum)$
}
End Algorithm
```

5.3 Evaluation of Hammerstein-Wiener Control

In this section, we evaluate the relative performance and resource management capabilities of the nonlinear Hammerstein-Wiener model based control approach. The simulation environment and the settings covered in Chapter 3 is utilized to implement and evaluate the control systems. Furthermore, the criteria set out in Section 3.4 will be used in the comparative analysis.

The evaluation is based on a system with two classes as described in Section 4.2.8 of Chapter 4. In particular, the resource caps are denoted by $S_0(k)$ and $S_1(k)$ where $S_0(k) + S_1(k) = S_{total} = 30$ and $S_{0,min}, S_{1,min} = 6$. $R_0(k)$ and $R_1(k)$ and $P_0(k)$ and $P_1(k)$ are the response times and differentiation factors of these two classes respectively at the k^{th} sample.

In order to implement the Hammerstein-Wiener control system, the pre-input and post-output compensators and the estimated linear model in Section 4.2.8 will be utilized. As mentioned in Section 5.2.2, the PI control algorithm is used to implement the Hammerstein-Wiener controller to provide control at runtime. Using the linear model estimated (see equation (4.11) in Chapter 4) and the pole-placement design methodology, the gains ($K_{i(HW)}$ and $K_{p(HW)}$) are calculated. The closed-loop poles were placed at ($\alpha = 0.5$, $\beta = 0.5$) after analysing the stability, settling time and overshooting specifications[1]. Also, $v_{min} = -9$ and $v_{max} = 9$ are implemented as the saturation limits of the controller. Afterwards, the controller is integrated into the system constructing the final control system architecture shown in Figure 5.3.

To compare and contrast the management capabilities of the Hammerstein-Wiener nonlinear control system (namely, HWCS) under different operating conditions and requirements, we use following two control systems.

1) Hammerstein model based control system - a control system with just the pre-input nonlinear compensator. This control system enables us to understand the problems of the output nonlinearity in isolation.

2) Linear model based control system - a control system without any nonlinear compensators. This control system shows the performance of the existing linear methods and

[1]Depending on the selected values for α and β, the behaviour of the control system varies because of the nonlinear characteristics of the system. Therefore, simulations have to be conducted in order to finalize the values for α and β.

illustrates the issues of the input and output nonlinearities in tandem.

The details of the above two control systems are as follows:

Hammerstein model based control system: As shown in Figure 5.4, the Hammerstein model based control system (namely, HCS) has only the input nonlinear compensator. Thus, it only compensates for the input nonlinearities characterized in the relative management scheme. The same input nonlinear compensator designed in Section 4.2.8 was integrated to the system before the identification of the linear model. The linear model identification setting is same as the linear component identification of the Hammerstein-Wiener model covered earlier (see Section 4.2.7). In fact, the same identification data can be utilized, however, the linear model is constructed using $v - y$ data pairs. The estimated linear model of the Hammerstein model with $\mathrm{R}^2 = 0.87$ is as follows:

$$y(t+1) = 0.90y(t) + 0.24v(t) \tag{5.9}$$

The PI controller of HCS operates with the control equation as follows:

$$C_{(H)}(z) = \frac{V(z)}{E_{(H)}(z)} = \frac{(K_{i(H)} + K_{p(H)})z - K_{p(H)}}{z-1} \tag{5.10}$$

where $E_{(H)}(z) = R(z) - Y(z)$, which is similar to a linear model based control error. However, $v(k)$ is the transformed control signal, which has to be converted to $u(k)$ using f^{-1} function. $K_{p(H)}$ and $K_{i(H)}$ are the gains of the HCS. These gains were also computed by the same settings used in the Hammerstein-Wiener control system design. The final structure of the HCS is shown in Figure 5.4.

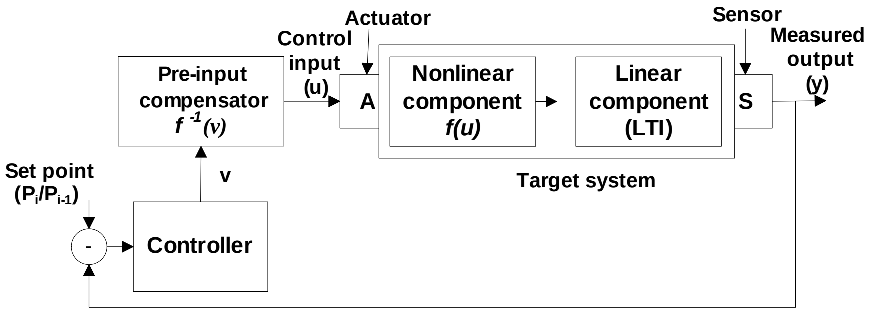


Figure 5.4: Hammerstein model based control system

Linear model based control system: The linear control system (namely, LCS) does not compensate any of the input and output nonlinearities. The linear model estimated in Chapter 4 (see equation (4.12)) is used to derive the gains for the PI controller following the pole-placement procedure covered in Section 5.2.2. A PI controller was then integrated to the control system similar to the structure shown in Figure 5.1a.

Table 5.2: Parameters of the control systems

Parameter	HWCS	HCS	LCS
K_p	0.30	2.06	0.73
K_i	0.12	0.78	0.32
Min saturation limit	-9	-9	0.25
Max saturation limit	9	9	4
Initial input	v(0) = 0	v(0) = 0	u(0) = 1

The summary of the configuration parameters of the above control systems are listed in Table 5.2. The saturation limits and initial control input set in all the control systems correspond to the same values in relation to the original system input. The differences of values are because the controllers in the nonlinear control systems operate with transformed variables.

In following sections we evaluate the performance of HWCS, HCS and LCS in different conditions and requirements. In Section 5.3.1, we compare the performance when the system is running full capacity or lower. By full capacity we mean that all the resource units are occupied during the entire period. In contrast, in Section 5.3.2 the performance is evaluated when the system is running with an extreme overload. That is the system is facing heavy workloads, which cannot be managed even all the resources are utilized 100% of the time. Consequently, admission control has to be implemented to maintain the stability of the system. Also note that, Appendix E covers a number of other conditions. Table 5.3 provides a brief description and the objectives of all the cases investigated in this evaluation.

Tables 5.4 and 5.5[2] quantify the quality of the runtime performance and resource management of all three control systems, which includes the following statistics:

- Sum of squared errors (SSE) = $\sum_{j=1}^{\bar{t}} e(j)^2$,
- Max (maximum output recorded) = max(Y),
- Min (minimum output recorded) = min(Y),

where, $e(k) = r(k) - y(k)$, that is the control error at each sample k, $Y = [y(0), y(1), \ldots, y(\bar{t})]$ is the measured output vector during the experiment conducted for $\bar{t}$ number of samples.

[2]For completeness, they also includes the results of the cases covered in Appendix E.

Table 5.3: The operating conditions and description

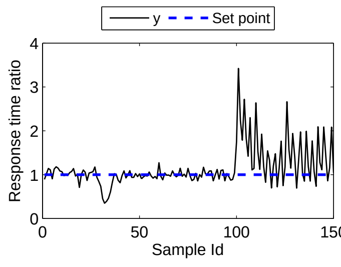
Workload near full capacity or lower		
Case	Set point	Description
A	1	Behavior away from the nominal operating point, where the input nonlinearity is dominant. The disturbance rejection capabilities are also investigated, which show the effects of output nonlinearities.
B	1.5	$class_0$ is more important than $class_1$. The set point is placed in the region where output signal increases at a high rate under sudden disturbance. The disturbance rejection capabilities are investigated under sudden workload disturbances from two classes. This case shows the issues of input nonlinearity.
C	0.6	$class_1$ is more important than $class_0$. The set point is placed in the region where output signal decays at a high rate under sudden disturbance. The disturbance rejection capabilities are investigated. This case shows the issues of output nonlinearity.
Workloads of extreme overloads		
D	1	This is similar to Case A. However, the workloads of classes are increased to extremely high magnitudes to overload the system. It illustrates the issues of the input and output nonlinearities together.
E	2	This is similar to Case B. One class overloads the system while the workload of other classes remains at a nominal rate. This case shows the issues of input nonlinearity.
H	$1.5 \rightarrow 2.25$	Investigate the behavior when the set point signal is changed at runtime abruptly.

5.3.1 Workloads of Full Capacity or Lower

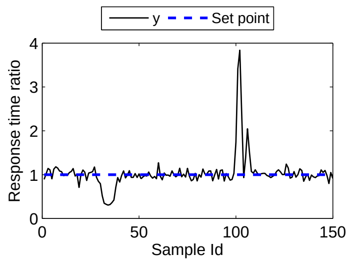
In this section, we maintain the workload conditions at the level that the system can cope with the available resources, meaning that, if the control system reacts accurately and fast the long term performance issues can be avoided under changing workload conditions. As mentioned in Chapter 3, the simulation model with 30 resources can handle 60-65 requests/sec. The cases investigated under these settings are as follows.

Case A*: Performance away from the nominal region*

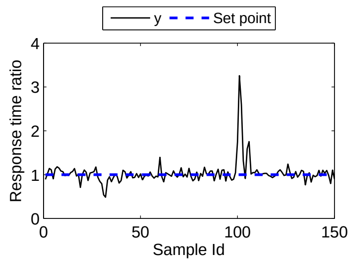
In this case, we test the performance of the control systems when the resource demands are away from the nominal region (see *region 0* and *region 1* in Figure 4.2). That is at the end of *region 0* or *region 1*, where the input nonlinearity is severe. In order to force the system to operate in these regions an experiment is conducted with the workloads of $class_0$ and $class_1$ being increased to the highest capacity separately. Till the 30th sample workload of 20 requests/sec is applied for $class_0$ and $class_1$. Then, at the 30th sample the $class_0$ workload increases to 45 requests/sec. This could be a scenario where a high resource demand for $class_0$, while $class_1$ is at a normal workload rate. Afterwards, at the 80th sample, $class_0$ workload reduces to 20 requests/sec. At the 100th sample, the $class_1$ workload increases to 45 requests/sec from 20 requests/sec. Furthermore, we specify the set point $\frac{P_1}{P_0} = 1$, assuming both classes are equally important. The desired responses of the control systems should be to see overshooting at the output due to the sudden workload disturbances. However, the controllers must reject these disturbances and come up with the suitable resource caps under changing resource demands to maintain the required reference/set point value.



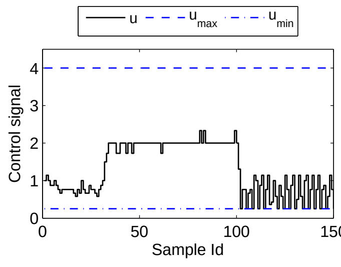
(a) Output of LCS



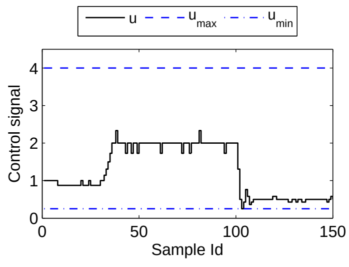
(b) Output of HCS



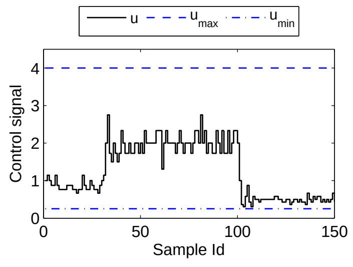
(c) Output of HWCS



(d) Control signal of LCS



(e) Control signal of HCS



(f) Control signal of HWCS

Figure 5.5: Performance of the control systems away from the nominal region (Case A)

The output signals of all the control systems are shown in Figure 5.5. At the start-up workload conditions, the system remains in the nominal region. Therefore, implementing resource caps greater than the minimum resource allocation is sufficient. i.e.,

$S_0(k), S_1(k) > S_{0,min}, S_{1,min}$. Then, let us analyse the performance of the control systems in the region where $class_0$ demands more resources (between 30th and 80th sample). The settling times of LCS, HCS and HWCS are approximately 11, 12 and 8 sample periods respectively. Similarly, the overshooting are 0.7, 0.7, 0.45 respectively. Therefore, the performance of the HWCS is significantly better compared to LCS and HCS. The control signals indicate that the resource caps have been adjusted at the 30th sample by all control systems, due to the disturbance. However, after settling down, all control systems provide similar steady state behavior, achieving the set point with small errors. In contrast, after the 100th sample when $class_1$ demands more resources, the steady state performance of LCS is significantly poor compared to other control systems. It shows highly oscillatory and unstable behavior with a large steady state error after the high workload disturbance of $class_1$. This is an indication that LCS cannot provide effective performance and resource management in *region 1*, when the workload of $class_1$ is high. In addition, performance isolation in that region is significantly poor as well. When the control signal in Figure 5.5d is investigated, there are significant oscillations. The reason for this behavior is the issue of input nonlinearity discussed in Section 4.2.2. The small gaps between the points in *region 1* (see Figure 4.2) affect the control provided by LCS under noisy operating conditions making LCS to jump between several operating points without settling down. However, the performance in this region can be improved by reducing the aggressiveness (gains) of the controller. This adversely affects the performance management and disturbance rejection capabilities when the workload of $class_1$ is high. Consequently, LCS fails to achieve effective performance management in the entire operating region under changing conditions. Furthermore, the discriminative behavior in different regions leads to model uncertainties and loss of flexibility in control system design.

In contrast, HCS and HWCS provide highly satisfactory steady state performance after the disturbance at the 100th sample without affecting the stability. This is because of the integration of the input nonlinear compensator into the control system that reduces the effect of input nonlinearity. However, HWCS settles down 3 samples before with 0.45 less overshooting compared HCS at the 100th sample. Similar improvements have been achieved by the HWCS at the 30th sample as well. As mentioned in Section 4.2.2, how output reacts to $class_0$ disturbance is different from $class_1$ disturbance, which leads to the output nonlinearity. This nonlinearity is compensated by HWCS providing better disturbance rejection in both regions compared HCS.

Therefore, the proposed Hammerstein-Wiener model based nonlinear control methodology provides much better performance and resource management in the entire operating region while providing design flexibility compared to other control systems.

***Case B**: When $class_0$ is more important*

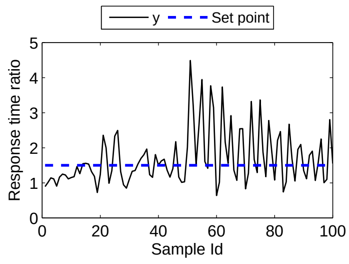
In this case, we maintain different priority levels assuming that $class_0$ is more important. This is translated into the control system by setting reference signal $\frac{P_1}{P_0} > 1$. This means, the control system provides fewer resources to the less important class, reserving more resources to the most important class anticipating a high workload demand. Here, we set $\frac{P_1}{P_0} = 1.5$ and check how the control system maintains the specified differentiation levels. The experiment starts with 20 and 20 requests/sec for $class_0$ and $class_1$ respectively. Then, at the 50th sample $class_1$ increased its workload to 40 requests/sec. Figure 5.6 shows the performance of the control systems, while Table 5.4 summarizes the statistics[3].

The performance of LCS is significantly poor in this case when the high workload disturbance of $class_1$ is applied at the 50th sample. The control signal in Figure 5.6d shows the oscillations similar to what was observed in Case A. This is because when $class_1$ demands more resources, LCS has to operate with the control points which show significant input nonlinearity. Consequently, the performance of the control system degrades significantly compared to the nonlinear control systems. The settling time and overshooting are less in HWCS compared to HCS (see Table 5.4). However, the steady state performance of the HCS is better because the output noise affects the compensator performance of the HWCS.

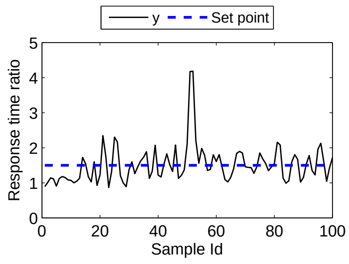
***Case C**: When $class_1$ is more important*

In this case, we maintain the different priority levels, assuming that $class_1$ is more important. This is translated into the control system by setting the reference signal $\frac{P_1}{P_0} < 1$. Here, the reference is set to $\frac{P_1}{P_0} = 0.6$, which is in the region, where the output signal decays when the sudden disturbances are encountered. The experiment starts with 20 and 20 requests/sec for $class_0$ and $class_1$ respectively. Then, at the 50th sample $class_0$ increased its workload to 40 requests/sec. Figure 5.7 shows the outputs of the control

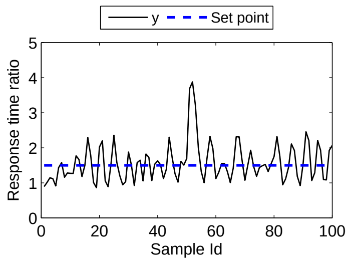
[3]Another case under these settings is covered under Case H in Appendix E.1, where $class_0$ increases workload instead of $class_1$.



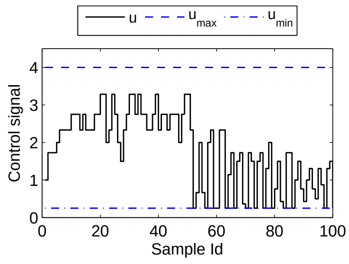
(a) Output of LCS



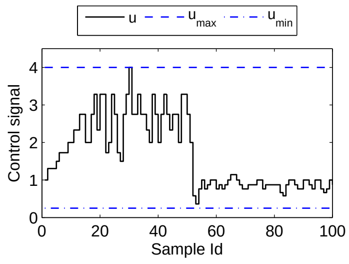
(b) Output of HCS



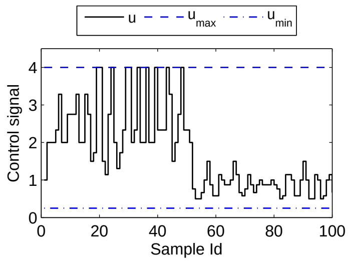
(c) Output of HWCS



(d) Control signal of LCS



(e) Control signal of HCS

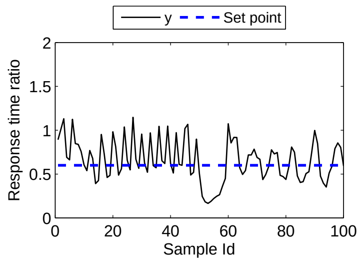


(f) Control signal of HWCS

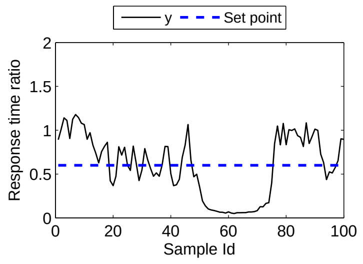
Figure 5.6: Performance of the control systems when $class_0$ is more important (Case B)

systems.

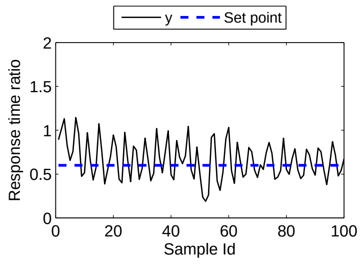
In this case the control systems have to perform in *region 0* (see Figure 4.3) where the output is damped out due to the output nonlinearity (see Figure 4.3). All the control systems show some steady state error due to the effect of queuing delays generated by $class_0$. The main characteristic to look at in this case is the disturbance rejection capabilities at the 50th sample due to the high workload of $class_0$. Under this disturbance, even though R_0 increases at a rapid rate, the output of the control system changes relatively slowly. As a consequence, LCS and HCS, which do not compensate the output nonlinearity, show significantly poor disturbance rejection capabilities, leading to high overshooting and settling times. In addition, the output signal of LCS shows an interesting change of behavior, before and after the disturbance at the 50th sample. This is because of the input nonlinearity. That is before the 50th sample, LCS operates in the region where input nonlinearity is severe, thus the control signal shows oscillatory behavior. However, unlike in Case A and B, it has not affected the performance adversely because the output nonlinearity damps out the variations of the output signal. Then, after the high workload disturbance of $class_0$, LCS has to operate in the region where the gaps between the operating points are larger. Again, the output nonlinearity affects the performance of LCS and the responsiveness of



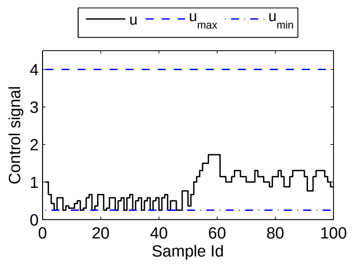
(a) Output of LCS



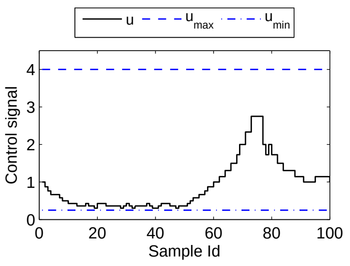
(b) Output of HCS



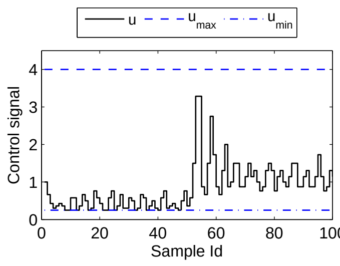
(c) Output of HWCS



(d) Control signal of LCS



(e) Control signal of HCS



(f) Control signal of HWCS

Figure 5.7: Performance of the control systems when $class_1$ is more important (Case C)

LCS decreases. Such unpredictable changes in behavior are also an indication that it is hard to provide consistent performance using LCS under changing workload settings. The input and output nonlinearity compensated HWCS on the other hand, provides better performance management with significantly lower overshooting, settling time and steady state error compared to LCS and HCS[4].

5.3.2 Workloads of Extreme Overload

In this section, we the compare the performance of the control systems under the extreme overloaded cases. In such conditions, the queues of one or more classes may grow rapidly, affecting the performance variables. If this behavior remains for a long time the entire system may become unstable, subsequently leading to system failures. Thus, to avoid such undesirable effects we specify a queue length of 30 for each queue. This is done to avoid the queues growing unboundedly due to the extreme overloads for a long time periods, which will increase the response time unboundedly. The queue limit has to be decided based on the maximum tolerable response times of all classes. When a queue has reached the specified limit, the subsequent incoming requests will be rejected by

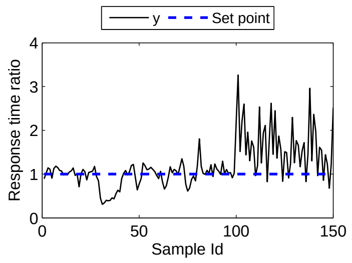
[4]The numerical improvements indicated in Figure 5.7 is small because in this region of operation the output damps out. However, when the response time signal of each class is investigated individually, significant improvements can be seen for the case of HWCS.

the management system. Hence, this is a simple rule based admission control mechanism. However, such simple admission control is sufficient because we adjust the resource caps at runtime, which will effectively achieve the required management objectives under changing workload conditions.

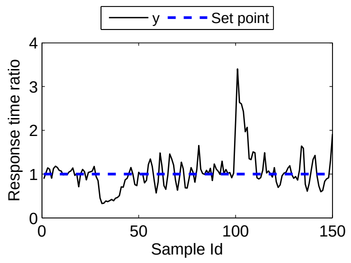
In following cases, we maintain the workload mix of two classes exceeding 200% of the maximum capacity of the system. The average request rejection or loss rates are listed in Table 5.5 for all these cases. The less rejection rates indicate better performance and resource management. A large difference may not be observed because the average loss rate per second is calculated over the total period of overload.

Case D: *Performance in away from the nominal region under overload*

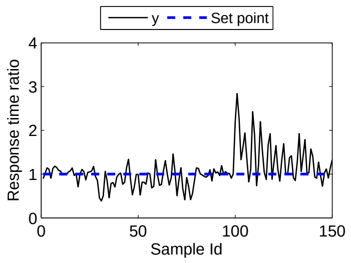
This case is similar to Case A, where control systems are forced to operate in the regions that the nonlinearities are severe. We maintained the same workload settings of Case A, however, in order to overload the system at the 30th sample, the workload of $class_0$ is increased to 100 requests/sec while maintaining the workload of $class_1$ at nominal 20 requests/sec. Similarly, at the 100th sample $class_1$ workload is increased to 100 requests/sec overloading the system. Figure 5.8 shows the outputs of the control systems.



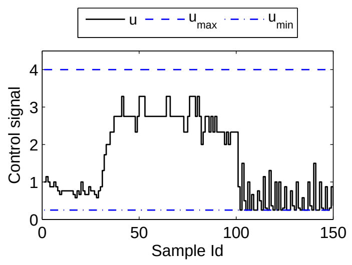
(a) Output of LCS



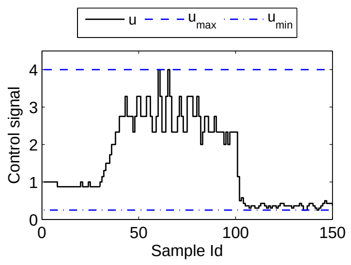
(b) Output of HCS



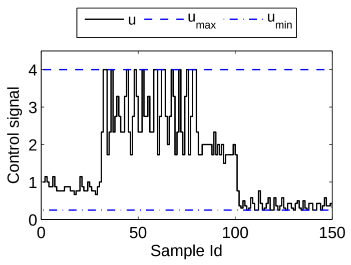
(c) Output of HWCS



(d) Control signal of LCS



(e) Control signal of HCS



(f) Control signal of HWCS

Figure 5.8: Performance in away from the nominal region under overload (Case D)

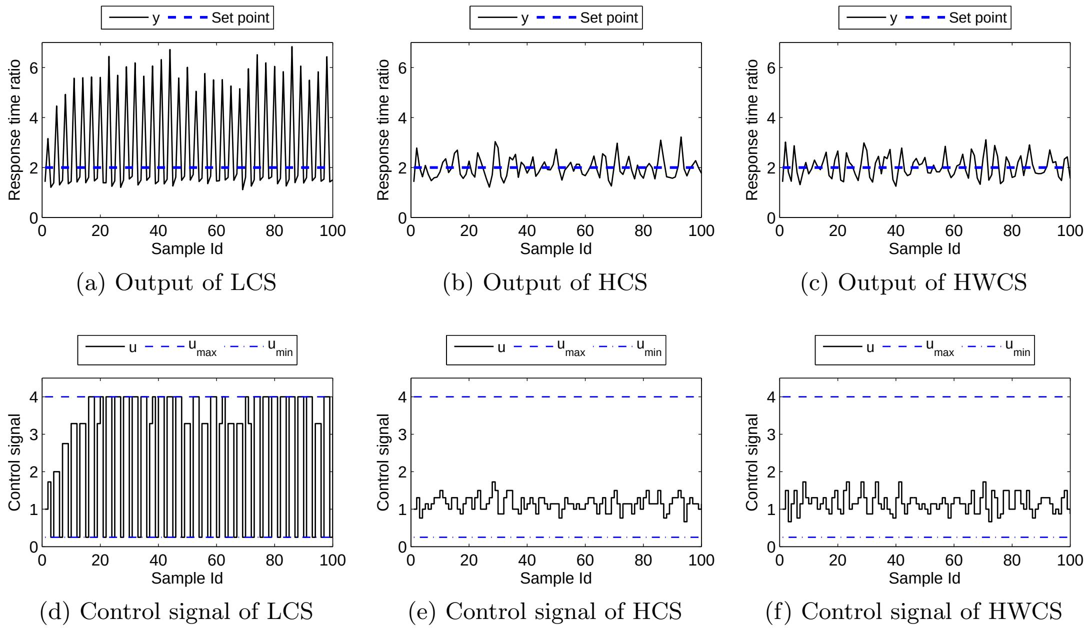
When a class overloads the system, some portion of requests has to be rejected in order maintain the response times of both classes within bounds. Further, under such a condition the response time of the overloaded class increases significantly. However, since the other class is treated equally, even though the workload of that class is less, its response time has to be increased as well by reducing the resource cap. However, the minimum resource reservation is guaranteed for the class running with low workload, in order to progress without total starvation of resources. All the control systems achieve the specified control objective, but as explained in Case A, the input and output nonlinearities of the system significantly affect the performance of LCS and HCS. In particular, the disturbance rejection capabilities are poor in LCS and HCS at the 30th sample due to the output nonlinearity. Again, the input nonlinearity affects the performance of LCS after the 100th sample. In contrast, the input and output nonlinearity compensated HWCS provides significantly better overshooting and settling time without sacrificing the steady state performance compared to HCS and LCS. Table 5.5 compares the average request loss rates of the control systems. LCS shows additional 3 requests/sec workload loss for $class_1$ compared to HWCS and HCS, which means that there are higher request losses during the overloaded period.

Case E*: Performance under overload when* $class_0$ *is more important*

In this case, we set $P_1 : P_0 = 1 : 2$, making $class_0$ as the most important class. Hence, depending on the workload mix $class_0$ should have relatively less workload losses compared to $class_1$. The experiment starts with a 40 and 75 requests/sec for $class_0$ and $class_1$ respectively, which means the less important class has overloaded the system. Figure 5.9 shows the outputs and the control signals of the control systems[5].

The output and control signal of LCS show significant performance issues, including the highly unstable behavior. It is hard to reason this behavior. One possible reason could be due to the initial workload disturbance, LCS hits the u_{min} limit, and the control error generated by that behavior saturates the control input at u_{max}. The same behavior continues in subsequent samples, leading to this unstable performance. When the performance of HCS and HWCS is compared there is no significant difference. Both control systems achieve the required control objectives significantly better than LCS

[5]Another case under these settings is covered under Case I of Appendix E.1. In that case, both classes increase their workloads simultaneously.



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

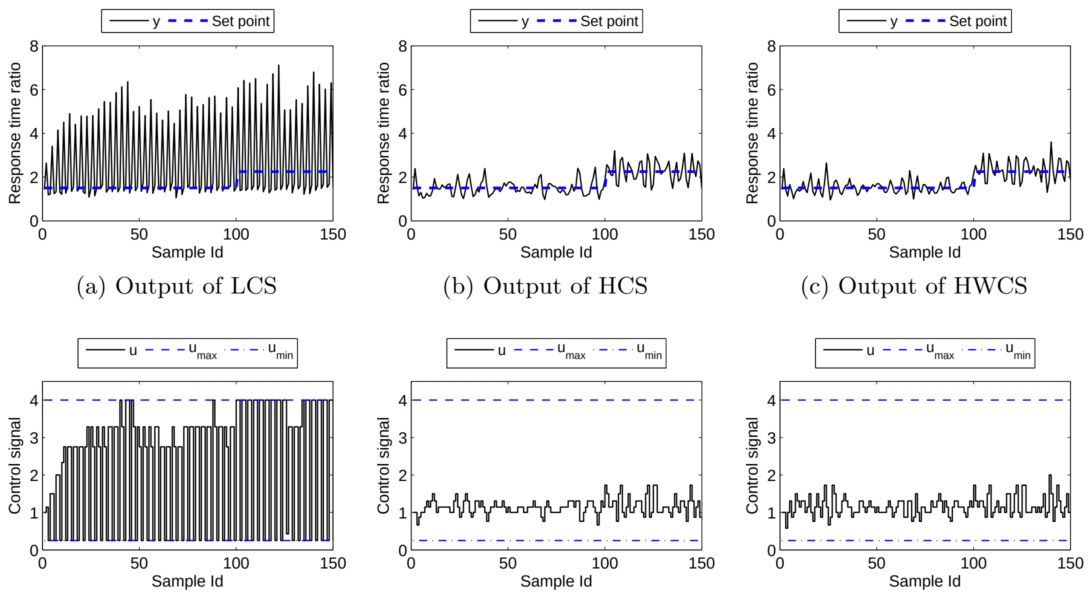
Figure 5.9: Performance under overload when $class_0$ is more important (Case E)

(also compare the SSE statistics of Table 5.4). Further, the loss rates listed in Table 5.5 indicate that LCS rejects additional 8 requests/sec workload of the most important class, where as the nonlinear control systems show no workload losses for that class. Similarly, LCS rejects higher amount of requests of $class_1$ as well, compared to the nonlinear control systems.

***Case F**: Change of differentiation levels (or set point) at runtime time*

This case evaluates the adaptability of the control systems when the business objectives change at runtime. That is when a differentiation level of one class changes, the management objective of the control system has to be changed as well at runtime. This objective can be achieved by changing the reference signal of the control system during the operations. Such changes can be done at runtime by the system administrator without any overhead of restarting the control system, which is an advantage of the relative management scheme. In this case, we maintain the differentiation factors $P_1 : P_0 = 1 : 1.5$ till the 100th sample and change it afterwards to $P_1 : P_0 = 1 : 2.25$. 40 and 75 requests/sec workloads are applied for $class_0$ and $class_1$ respectively, which means that the less important class has overloaded the system. This is similar to Case E conditions. Figure 5.10

shows the outputs and the control signals of the control systems.



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

Figure 5.10: Change of differentiation levels at runtime time (Case F)

The results of this condition are similar to Case E. LCS totally fails to achieve the control objectives. At the same time no adaptability is shown at the 100th sample when the new differentiation level is implemented in the system. Consequently, the performance of LCS in this condition is highly unstable and unsatisfactory. In contrast, HCS and HWCS achieve the required control objectives and adjust rapidly to the change of differentiation levels by tracking the set point signal effectively. Both nonlinear control systems adjust to the change in reference signal in approximately less than 5 samples.

5.3.3 Summary of Results

The performance evaluations conducted in versatile operating conditions and business requirements have indicated that the proposed nonlinear Hammerstein-Wiener control technique provides significantly better runtime performance management compared to the existing linear control approaches under many cases. In particular, when either input or output nonlinearity is severe, the linear control system has shown poor disturbance rejection capabilities and stability issues. As a consequence, designing a single linear controller to operate in the entire operating region is difficult. In contrast, the compensations performed by the Hammerstein-Wiener control system has enabled better disturbance re-

Table 5.4: Statistical summary of control systems managing two classes

Case,Type	SSE				MIN				MAX		
	LCS	HCS	HWCS		LCS	HCS	HWCS		LCS	HCS	HWCS
A	1.001	0.937	0.954		0.687	0.709	0.709		1.243	1.243	1.243
B	64.13	27.896	30.7		0.639	0.869	0.862		4.487	4.184	3.879
C	5.886	12.308	4.884		0.167	0.049	0.190		1.147	1.179	1.144
D	12.574	14.31	9.825		0.308	0.326	0.384		3.263	3.402	2.842
E	507.138	16.978	20.964		1.114	1.217	1.252		6.833	3.219	3.108
F	708.104	22.898	20.822		1.055	0.972	0.954		7.113	3.202	3.599
Cases covered in Appendix E.1											
G	32.893	21.885	11.072		0.349	0.305	0.486		3.422	3.839	3.258
H	15.265	14.372	18.942		0.725	0.869	0.862		2.633	2.424	2.621
I	4.287	5.448	5.539		0.889	0.889	0.889		2.403	2.437	2.429
J	19.812	24.9	26.147		1.314	1.059	1.292		3.183	3.416	3.859
K	6.31	10.947	4.969		1.125	1.062	1.125		2.655	2.352	2.514
L	3.274	4.572	4.385		0.883	0.883	0.883		2.596	2.675	2.898
M	29.44	6.866	5.079		0.611	0.604	0.629		3.326	1.659	1.469
N	195.98	129.731	135.768		0.29	0.255	0.267		3.851	3.664	2.968

Table 5.5: Summary of average loss rates of the control systems managing two classes

Case,Type	LCS			HCS			HWCS	
	$class_0$	$class_1$		$class_0$	$class_1$		$class_0$	$class_1$
D	45	49		46	46		45	46
E	8	85		0	79		0	79
Cases covered in Appendix E.1								
I	41	107		42	107		41	108
J	80	0		80	0		79	0
K	108	41		105	46		107	41

jection without sacrificing the stability of the managed system. This also helps to establish design flexibility compared to the linear control method. When the Hammerstein-Wiener control system is compared with the Hammerstein control system, which compensates only the input nonlinearities indicates that compensation of output nonlinearities is important to achieve better performance in most of the cases. However, the noisy conditions affect the output nonlinear compensation compared to the Hammerstein control system in few cases.

Further evaluations were conducted in Appendix E.1 covering a number other cases. Appendix H investigates the management of a multi-class system having a large amount of resources with significantly more operating points and severe nonlinearities. The evaluation results listed in Appendix H also indicate the superior management provided by this new nonlinear control system compared to the linear counterpart. Moreover, Appendix F covers the model estimation, control system design and evaluation of the Hammerstein-Wiener control approach for a multi-class system with three classes. This evaluation has also shown similar results observed previous section. In the above evaluations, the configuration parameters of the simulation environment were set at the fixed values mentioned in Chapter 3. The effects of the simulation environment variables are also examined in Appendix G using Monte-Carlo simulations in order to investigate the robustness of the proposed nonlinear control system compared to a linear control system. This investigation has demonstrated the robustness of proposed nonlinear control approach and the validity of the results presented in previous sections. Finally, one of the design parameters of the proposed nonlinear estimation and control approach is the range of V (v_{max}, v_{min}). This range was selected arbitrarily in the design of Hammerstein-Wiener control system. The impact of the selected range is investigated and the design guidelines are provided in Appendix I. The applications and evaluations of the Hammerstein-Wiener nonlinear control approach for relative performance management in the real-world shared resource systems are covered in Chapter 8 in more detail.

5.4 Absolute Performance Management Using Nonlinear Control

In Section 4.3 of Chapter 4, the nonlinear dynamics of absolute management scheme were identified using MIMO Wiener model structure. A compensator was designed and connected at each output to compensate the nonlinear dynamics. Afterwards, a MIMO linear model was estimated. This section presents the MIMO Wiener control system design procedure for absolute performance management.

5.4.1 Linear Control System Design for Absolute Performance Management

In this section, we continue the example system with two classes covered in Section 4.3 to illustrate the linear control system architecture for absolute performance management. The main objective of the absolute performance management system is to maintain the system outputs $R_0(k)$ and $R_1(k)$ of two classes around $R_{SLA,0}(k)$ and $R_{SLA,1}(k)$ respectively under unpredictable workload conditions, while adjusting the resource caps $S_1(k)$ and $S_1(k)$. In addition, the scheduler should work within the following constraints[6].

$$\begin{aligned} &S_0 \geq S_{0,min}, S_1 \geq S_{1,min} \\ &S_0 + S_1 \leq S_{total} \end{aligned} \tag{5.11}$$

Figure 5.11 shows the control system architecture for absolute performance management. It is clear that the controller should have the ability to deal with multiple control objectives and constraints at runtime. The inputs to the controller are the set point signals ($R_{SLA,0}(k)$, $R_{SLA,1}(k)$) and feedback signals from the system ($R_0(k)$, $R_1(k)$). The outputs of the controller are the resource caps ($S_0(k)$, $S_1(k)$), adhering to the constraints specified above.

Unlike the case of relative scheme, the absolute scheme incorporates the system inputs and outputs directly without any transformations. Therefore, a general MIMO controller can be utilized. In addition, especially, when the system is encountering overloaded work-

[6]The equality condition is relaxed with inequality condition in order to avoid the mathematical ill-conditioning. See [193] for experimental results.

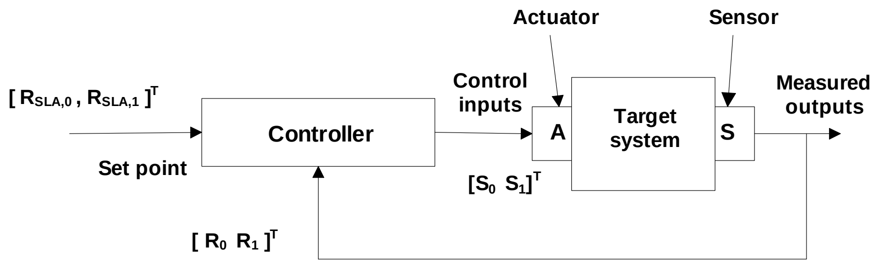


Figure 5.11: Block diagram of absolute performance and resource management system

load conditions, the absolute performance management scheme may not always be able to maintain the outputs at the prescribed fixed values ($R_{SLA,0}(k)$, $R_{SLA,1}(k)$). Consequently, the absolute performance management can be only performed under a limited range of workload conditions compared to the relative performance management [108, 145].

5.4.2 MIMO Wiener Model Based Control System Design

The control problem is to maintain the agreed levels of response time $R_{SLA,i}, i \in \{0, \dots, n-1\}$ of each class under varying workloads and resource demands while allocating the limited amount of resources among them without violating the hard constraints on the system. To deal with such multi-objective constrained control problems, model predictive control (MPC) is widely adopted [21, 223] (see Appendix B). Therefore, MPC is well suited for absolute performance and resource management in multi-class systems, which has multi-input and multi-output configuration and operational constraints. However, as opposed to the linear MPC, the MIMO Wiener MPC operates with transformed feedback signals due to the integration of compensators (see, Section 4.3). An attractive feature of the Wiener model is that it integrates static nonlinearity into the control system design by preserving some of the numerical properties of the original linear MPC design and the constraint problem [173]. This means that we can use a linear MPC formulation and standard quadratic programming solver.

As mentioned, the Wiener MPC operates with transformed variables, in particular, the standard output variable y(k) has to be replaced by the intermediate variable w(k). As a consequence, the cost function of the standard MPC has to be transformed as follows.

$$J(k)_{(W)} = (R_{s(w)} - W)Q_{(w)}(R_{s(w)} - W)^T + \Delta U R_{(w)} \Delta U^T \qquad (5.12)$$

where, $R_{s(W)} = g^{-1}(R_s)$ is the transformed reference signal from $R_s = [R_{SLA,0} \dots R_{SLA,n-1}]$ that is the data vector for the future reference signal, $W = [w(k+1|k)^T \; w(k+$

$2|k)^T \ldots w(k+N_p|k)^T]^T$ is the transformed output vector, that is the data vector for the predicted output, and $\Delta U = [\Delta u(k)^T \; \Delta u(k+1)^T \; \ldots \; \Delta u(k+N_c-1)^T]^T$ is the control input. The weight matrices $Q_{(W)}$ and $R_{(W)}$ are assumed to be symmetric, non-negative and positive definite. N_p is the prediction horizon and N_c is the control horizon.

The first-term in equation (5.12) incorporates the deviation or error of the system output compared to the desired values. The second-term incorporates the controller effort. The objective is to minimize the error with minimal control effort. This objective has to be achieved by minimizing the cost function J subject to the hard constraints on the resources as follows:

Minimize J

Subject to:

$$\underbrace{\begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix}_{n\times n}}_{\alpha} u(k) \geq \underbrace{\begin{bmatrix} S_{0,min} \\ S_{1,min} \\ \vdots \\ S_{n-1,min} \end{bmatrix}}_{\beta} \tag{5.13}$$

$$\underbrace{\begin{bmatrix} 1 & 1 & \ldots & 1 \end{bmatrix}_{1\times n}}_{\omega} u(k) \leq S_{total} \tag{5.14}$$

However, since the optimization is performed based on Δu, these constraints will be converted into functions in terms of the parameters $\Delta u(k)^T, \Delta u(k+1)^T, \ldots, u(k+N_c-1)^T$. In addition, to solve this constraint problem using a standard quadratic programming solver, the constraints have to be represented in the form of $M\Delta U \leq \gamma$. These considerations are incorporated as follows:

$$u(k) = u(k-1) + \Delta u(k) \tag{5.15}$$

Then, using equation (5.15), equation (5.13) can be converted to

$$\alpha \Delta u(k) \geq \beta - \alpha u(k-1),$$

$$-\alpha \Delta u(k) \leq -\beta + \alpha u(k-1), \tag{5.16}$$

and similarly, equation (5.14) can be converted to

$$\omega \Delta u(k) \leq S_{total} - \omega u(k-1) \tag{5.17}$$

Finally, the above formulated constraints in equations (5.16) and (5.17) are represented in compact $M\Delta U \leq \gamma$ form as follows:

$$\underbrace{\begin{bmatrix} -\alpha \\ \omega \end{bmatrix}}_{M} \Delta u(k) \leq \underbrace{\begin{bmatrix} -\beta \\ S_{total} \end{bmatrix} + \begin{bmatrix} \alpha \\ -\omega \end{bmatrix} u(k-1)}_{\gamma} \tag{5.18}$$

The problem of minimizing equation (5.12) subject to equation (5.18) imposes constraints on the current input $u(k)$ only (i.e. assuming $N_c = 1$), but the constraints can be applied on the inputs of a even larger control horizon N_c. Such constraints can be also implemented by extending equation (5.18).

It is also useful to note that MPC can handle soft constraints on the outputs and state variables. This is out of the scope of this work. However, if the output constraints are considered they have to be converted to the intermediate output variables w_i as follows:

$$g^{-1}(Y_{min}) \leq W(k) \leq g^{-1}(Y_{max}),$$

where, $Y_{min} = [y_{min,0}(k)\ y_{min,1}(k)\ \ldots\ y_{min,n-1}(k)]^T$ and $Y_{max} = [y_{max,0}(k)\ y_{max,1}(k)\ \ldots\ y_{max,n-1}(k)]^T$ are the vectors containing bounds of the output for each class.

5.4.3 Implementation and Tool Support

In this section, we present the details of MPC design, which includes the state-space model transformation, controller development and quadratic programming solver implementation.

Formulation of the state-space model

In Chapter 4, we estimated a MIMO transfer function model for the case of absolute performance management using system identification. However, the standard state-space model is a popular method used to design MPC in the existing literature because of the simplicity to handle multi-variable systems [223]. We therefore convert the MIMO transfer function model to the state-space realization as follows. The standard form of the state-space model is shown in equation (5.19).

$$\begin{aligned} x_m(k+1) &= A_m x_m(k) + B_m u(k), \\ w(k) &= C_m x_m(k) \end{aligned} \tag{5.19}$$

where, w is the intermediate output variable of the Wiener model, u is the manipulated variable (control input) and x is the state variable vector. In this work, a technique called *non-minimal state space realization* presented in [223] is used to convert MIMO ARX model into the above state space model. The advantage is that the state space vector can be formulated by measurable inputs and outputs, so that we can avoid the requirement of having an observer [223].

Let us represent the MIMO ARX model of order n as

$$w(k+1) + F_1 w(k) + F_2 w(k-1) + \ldots + F_n w(k-n+1) = \tag{5.20}$$

$$H_1 u(k) + H_2 u(k-1) + \ldots H_n u(k-n+1) \tag{5.21}$$

Then, by selecting the state space vector
$x(k) = \left[w(k)^T \ldots w(k-n+1)^T \;\; u(k-1)^T \ldots u(k-n+1)^T\right]$, the non-minimum state space realization can be represented as follows:

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k), \\ w(k) &= Cx(k) \end{aligned} \tag{5.22}$$

where[7],

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix}, \; B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \; C = \begin{bmatrix} C_1 & C_2 \end{bmatrix},$$

$$A_1 = \begin{bmatrix} -F_1 & -F_2 & \ldots & -F_n \\ I & o & \ldots & o \\ \vdots & \vdots & \ddots & \vdots \\ o & o & \ldots & o \end{bmatrix}, A_2 = \begin{bmatrix} H_2 & H_3 & \ldots & H_n \\ I & o & \ldots & o \\ \ldots & \ldots & \ldots & \ldots \\ o & o & \ldots & o \end{bmatrix}$$

$$A_4 = \begin{bmatrix} o & o & \ldots & o \\ I & o & \ldots & o \\ o & I & \ldots & o \\ \ldots & \ldots & \ldots & \ldots \\ o & o & \ldots & o \end{bmatrix} B_1 = [H_1 \;\; o \ldots \;\; o]^T, B_2 = [I \;\; o \ldots \;\; o]^T, C_1 = [I \;\; o \ldots \;\; o]^T.$$

Example: For the first order MIMO ARX model shown in equation (4.16), the non-minimum state-space realization is given below.

$$A = \begin{bmatrix} 0.4817 & -0.0145 \\ -0.0146 & 0.5131 \end{bmatrix}, \quad B = \begin{bmatrix} 0.1446 & -0.0110 \\ -0.0079 & 0.1323 \end{bmatrix}, \quad C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where, $x(k) = [w_0(k) \;\; w_1(k)]^T$.

Implementation of the MPC using Laguerre functions

As mentioned at the start of this section, the basic idea behind MPC is to compute the

[7]o is a matrix of zeros in corresponding dimensions, A_3 is a zero matrix.

future control trajectory. This control trajectory can be regarded as an impulse response of a stable dynamic system. A *Laguerre model* based on *Laguerre functions* can be also used to describe the impulse response of a stable system. Based on this concept, (Wang, 2009) proposed a MPC implementation framework based on Laguerre functions [223]. The advantage of this approach is that when there is rapid sampling and complicated dynamics in the system, the computation of control inputs does not require a large number of parameters. Furthermore, the computational overhead can be reduced for MIMO systems. In this work, we use the MPC formulation in [223] to implement MIMO MPC. Here, we briefly describe the design of Laguerre functions based MPC, however, for more details refer [223].

The incremental input, for a future sample instance can be described using Laguerre functions as follows:

$$\Delta u(k + k_i) = L(k_i)^T \eta \tag{5.23}$$

where, k_i is a future sample, $L(k) = A_l L(k-1)$ and $\eta = [c_1 \;\; c_2 \ldots c_N]^T$. $L(0)^T$ is computed by $\sqrt{\beta}\left[1 \;\; -a \;\; a^2 \;\; -A^3 \ldots (-1)^{N-1} a^{N-1}\right]$, where $0 < a < 1$ is the pole of the discrete time Lageurre network, N is the number of terms in the network and $\beta = (1 - a^2)$. A_l is a $N \times N$ matrix and a function of a and β. $c_1, c_2 \ldots c_N$ are Laguerre coefficients.

Then, according to the formulation in [223], the B matrix of equation (5.22) is partitioned into n columns (B_1, $B_2 \ldots B_n$). The incremental vector for n inputs is represented by $\Delta u(k) = [\Delta u_1(k) \;\; \Delta u_2(k) \;\; \ldots \Delta u_n(k)]^T$, where ith incremental input $\Delta u_i(k) = L_i(k)^T \eta_i$. This means each control signal is expressed by a Lageurre function with a pole a_i and number of terms N_i. Using these variables, the properties of Lageurre functions and state variable data at the kth sample, the future state information can be predicted for a future time instance p using equation (5.24).

$$x(k + m|k) = A^p x(k) + \sum_{j=0}^{p-1} A^{p-j-1} \left[B_1 L_1(j)^T \;\; B_2 L_1(j)^T \ldots B_n L_n(j)^T\right] \eta \tag{5.24}$$

where, $\eta^T = \left[\eta_1^T \;\; \eta_2^T \;\; \ldots \eta_n^T\right]$.

Afterwards, the minimum value of the cost function in equation (5.12) is computed and translated into the Laguerre function terms with the idea of computing the optimal solutions for the cost function. The optimal solution without considering any constraints is given by

$$\eta = -\Omega^{-1}\psi x(k) \tag{5.25}$$

where,

$\Omega = \sum_{m=2}^{N_p} \phi(m)Q\phi(m)^T + R_w$,

$\psi = \sum_{m=1}^{N_P} \phi Q A^m$,

$\phi(m)^T = \sum_{j=0}^{p-1} A^{p-j-1} \left[B_1 L_1(j)^T \;\; B_2 L_1(j)^T \ldots B_m L_m(j)^T\right]$, $Q = C^T C$ and N_p is the prediction horizon. Finally, the computed η is used to calculate the input vector $\Delta u(k)$ as follows.

$$\Delta u(k) = \begin{bmatrix} L_1(0)^T & o_2^T & \ldots & o_n^T \\ o_1^T & L_2(0)^T & \ldots & o_n^T \\ \vdots & \vdots & \ddots & \vdots \\ o_1^T & o_2(0)^T & \ldots & L_n(0)^T \end{bmatrix} \eta \tag{5.26}$$

where, $o_j^T, j = 1, 2 \ldots n$ is a row vector of zeros with the same dimension to $L_k(0)^T$. Using Δu and equation (5.15), $u(k)$ can be computed which is applied in the system as the resource allocation.

Implementation of the quadratic programming solver

The definition of the quadratic programming problem with respect to Laguerre functions formulation is as follows:

$$J = \eta^T \Omega \eta + 2\eta^T \psi x(k) \tag{5.27}$$

$$M\eta \leq \gamma \tag{5.28}$$

where, J is the cost function, and equation (5.28) represents the constraint set.

The *active set* method is one of the widely adopted methods based on Lagrange multipliers to solve such standard quadratic optimization problems. In the active set method, the constraints are categorized into active and inactive constraints at each stage. The active constraints are the constraints where equality (=) conditions are satisfied at the current stage of optimization. The inactive constraints are the constraints where inequality (<) conditions are satisfied. The Lagrange multipliers enable us to perform this categorization. The formula to calculate Lagrange multipliers is given below.

$$\lambda = -(M\Omega^{-1}M^T)^{-1}(\gamma + M\Omega^{-1}\psi x(k)) \tag{5.29}$$

If elements of λ are positive, then the corresponding constraints are categorized as active, else the constraint is inactive. Once, this categorization is performed the inactive constraints are removed from the problem, while active constraints are used as the working constraint set of the optimization problem. Iteratively, the constraint problem is reformulated and solved till all constraints are satisfied. The solution for η is then calculated by equation (5.30).

$$\eta = -\Omega^{-1}(M^T\lambda + \psi x(k)) \tag{5.30}$$

The active set method is hard to program and the computational overhead is large when there are many constraints [223]. In addition, the active set method can lead to termination of the runtime optimization process, because it requires matrix inversions, which may not always be invertible.

In this work, we implement a quadratic programming solver called *Hildreth's quadratic program*, which can be classified under *primal-dual method* [223]. This algorithm is relatively easy to program compared the active set method. Furthermore, Hildreth's quadratic programming solver does not require matrix inversions, and consequently the optimization process can be carried out without interruptions, which is important in real-time optimizations to avoid critical failures in the control system. However, the drawback is that under conflicting constraints/situations, suboptimal solutions can be generated.

The details of Hildreth's quadratic programming procedure are as follows. After obtaining the Lagrange multipliers (λ) vector, we focus on just one component (say λ_i)≥ 0 at each step. This means at each stage cost function is optimized using a single component. To minimize the objective function, λ_i will be adjusted, however, in order to reach the minimum value if λ_i is required to be < 0, λ_i is set 0. Then, the algorithm proceeds to $i + 1$ element and continues the same process. Once the final λ is computed, equation (5.30) can be used to compute the optimal solution. The algorithm in detail can be found in [223].

The above discussed MPC formulation and quadratic program solution implementations are available in our configurable C#.NET and Java class library, which will be introduced in Chapter 7. In addition, a MATLAB implementations can be found in [223].

A limitation of the proposed controller design technique is that the control decisions generated by the control system are continuous values. As a consequence, *rounding-off policies* have to be implemented to convert the continuous values to the discrete values (resource caps) that can be implemented in the software system.

5.5 Evaluation of MIMO Wiener Control System

In this section, we evaluate the performance of the MIMO Wiener nonlinear control system (namely, WMPC) under different operating conditions and business requirements. The same system configurations used in Section 5.3 are used in this evaluation as well.

Here, we continue with the compensator and models estimated in Section 4.3.7. In order to implement the MIMO Wiener controller, the transfer function model in equation (4.15) was converted into the non-minimum state space model as described in Section 5.4.3. The control and the optimization program covered in Section 5.4.3 were then configured with the tuning parameters after careful investigation of the performance. The parameters of the Laguerre functions based MPC for a system with two inputs were set at ($a_1 = 0.45$, a_2 =0.45 and $N_1, N_2 = 1$). In addition, $N_p = 15$ and $u(0) = [15\ 15]^T$. It is worth noting that the controller showed poor performance at low gains (R_w) because of the low model fits (see Section 5.3). As a result, R_w was set to $500 \times I_{(2\times2)}$.

In order to compare the performance, we designed a MIMO linear model based control system (namely, LMPC) as well. The MIMO linear model estimated in Chapter 4 is utilized in this implementation. All the parameters of LMPC were set to the same values as WMPC apart from R_w, which was set to $0.2 \times I_{(2\times2)}$.

In following cases we evaluate the performance of WMPC and LMPC in different conditions and requirements. Table 5.6 lists the details of cases investigated and their objectives. Note that, the workload conditions utilized in the following cases are the same workload conditions used in Section 5.3 for the evaluation of the relative performance management scheme. As mentioned, the LMPC shows fair performance without instabilities in the case of absolute performance management scheme because of less intensive nonlinearities that exists in the system compared to the relative performance management scheme. In the following cases we compare the performance of the proposed nonlinear control technique to investigate whether further improvements can be made compared to the linear counterpart. Our main conjecture is that when the system has to operate in the sensitive region due to the sudden workload variations, the performance management can be improved due to the compensation of the nonlinearities at each output. In particular, we claim that WMPC provides better disturbance rejection capabilities, without sacrificing the steady state behavior. Table 5.7 and 5.8 summarize the results.

Table 5.6: Operating conditions and objectives of absolute performance management

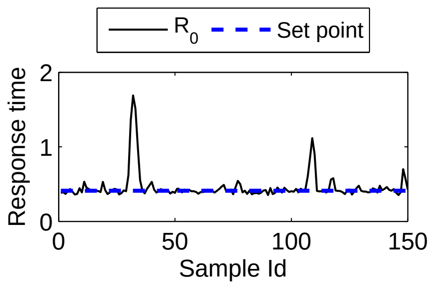
Case	Description
A	Workloads of both classes are increased in separate time periods. Investigating disturbance rejection capabilities under sudden workload variations. Both set points are placed in the 'insensitive' region.
B	Workloads of both classes are increased simultaneously. Here both classes are competing for equal amount of resources. This is a condition with high interactions between the inputs and outputs. Both set points are placed in the 'insensitive' region.
C	Set points are placed in the 'sensitive' and 'insensitive' regions, investigating the performance differentiation and disturbance rejection capabilities together.
D	Behavior of the control systems under extreme overloads.

5.5.1 Performance Management Under Different Cases

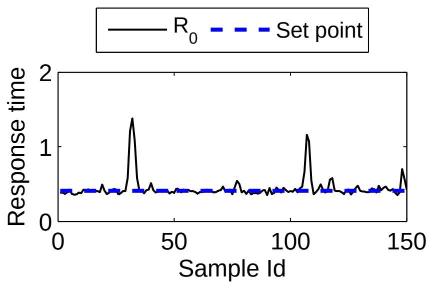
Case A : High workloads separately

In this case, we place the reference signals of both classes in the insensitive region, making $R_{SLA,0}, R_{SLA,1} = 0.41$ seconds. In order to evaluate the efficiency of resource cap adjustments at runtime under varying workloads we change the workload of $class_0$ and $class_1$ to the highest capacity separately. Till the 30th sample a workload of 20 requests/sec is applied for $class_0$ and $class_1$. Then, at the 30th sample, $class_0$ workload increases to 45 requests/sec. Afterwards, at the 80th sample the $class_0$ workload reduces to 20 requests/sec. At the 100th sample, $class_1$ workload increases to 45 requests/sec from 20 requests/sec. Such, workload changes necessitate the control systems to efficiently adjust the resource caps, otherwise the delays may lead to the degradation of performance of both classes.

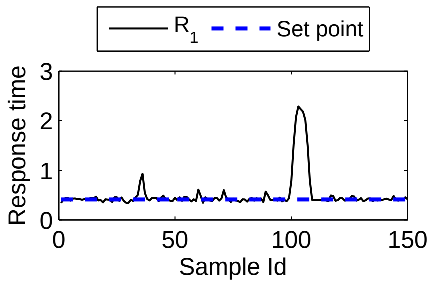
Figure 5.12 shows the performance of the MIMO linear and nonlinear control systems. The general observation of both LMPC and WMPC is that when the high workloads are applied at the 30th and 100th sample there is overshooting at the corresponding outputs. This is the expected behavior because the controllers take time to adjust the resource caps according to the new resource demands. However, when the two control systems are compared, the overshooting and the settling time of LMPC at the disturbances are higher than WMPC. The Max statistics of R_0 in LMPC is around 1.7 seconds compared to 1.4 seconds of WMPC (see Table 5.7 and 5.8). This is a reduction of 300 (18%) milliseconds



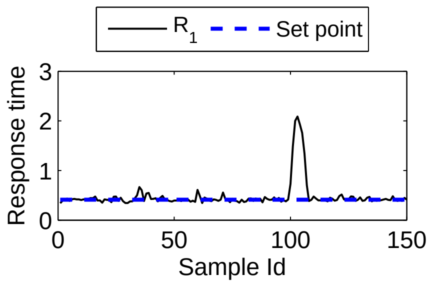
(a) Output R_0 of MIMO LMPC



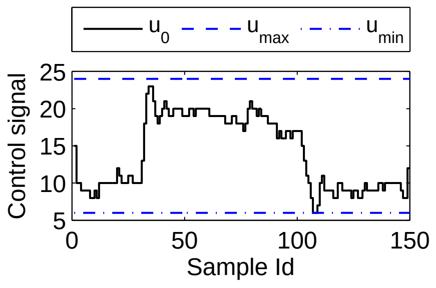
(b) Output R_0 of MIMO WMPC



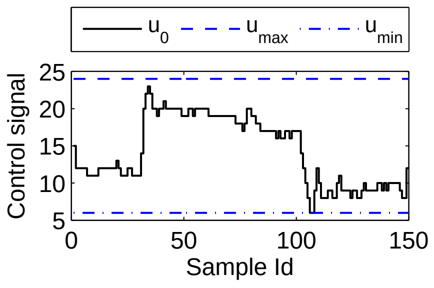
(c) Output R_1 of MIMO LMPC



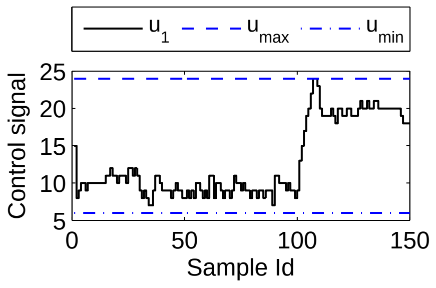
(d) Output R_1 of MIMO WMPC



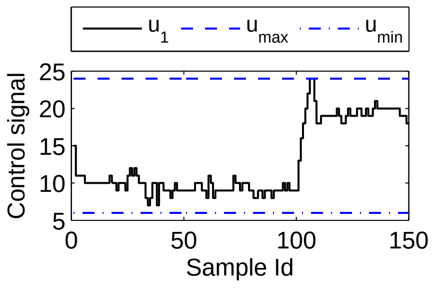
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC



(g) Control signal u_1 of MIMO LMPC



(h) Control signal u_1 of MIMO WMPC

Figure 5.12: Performance management under high separate workloads (Case A)

of overshooting. Similarly, close to 200 milliseconds of overshooting has been reduced by WMPC compared to LMPC at the 100th sample. Now, a possible question is did WMPC sacrifice the steady state behavior (or error) while reducing the overshooting. This is because, the overshooting and steady state behavior are typically, competing performance attributes of a control system. Table 5.9 and 5.10 illustrate the steady state error as SSE for 1000 samples. That is we maintained the same high workload after the disturbance for 1000 samples to examine the steady state error. It indicates that better steady state performance of WMPC. As a consequence, our claim of better steady state behavior and less overshooting is justified by this experiment. When the control signals are compared it is evident that LMPC shows vulnerability to noisy conditions, leading to oscillations in both control signals at the steady state compared to WMPC. Further, it is worth noting that the implemented MPC and optimization solutions have not violated any of the hard constraints imposed on the system during this experiment (see Figures 5.12e, 5.12f, 5.12g, 5.12h).

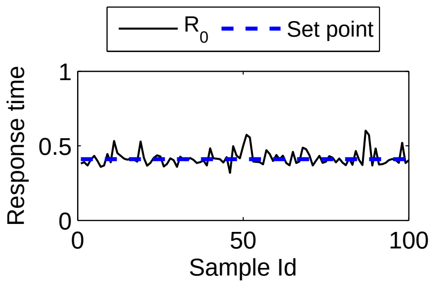
Case B: High workloads simultaneously

In this case, we increase the workload of both classes simultaneously to the highest capacity. Consequently, in this condition there are significant interactions between the inputs and outputs, because both classes are competing for an equal amount of resources. Here, $class_0$ and $class_1$ start-off by sending 20 requests/sec each till the 50th sample and afterwards both classes increase their workloads to 30 requests/sec simultaneously. $R_{SLA,0}$ and $R_{SLA,1} = 0.41$ seconds.

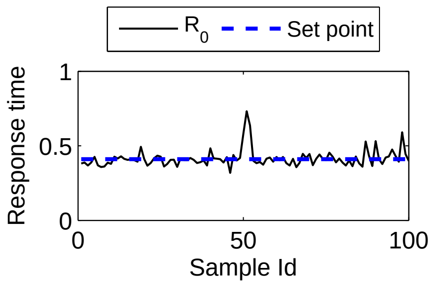
Figure 5.13 indicates that the both outputs show overshooting due to the disturbance at the 50th sample. For instance, LMPC shows a large overshooting in R_1, while WMPC shows a overshooting in R_0. As a result, the statistics in Table 5.7 and 5.10 indicate slightly better performance of LMPC for the case of R_0 but much poor performance for R_1 compared to WMPC. However, both controllers settle to 15:15 resource caps after the 50th sample because both classes are applying the same workload. The control signals of LMPC show oscillations indicating vulnerability to noisy conditions at the steady state compared to WMPC. This is further justified by the long-term steady state error statistics in Table 5.9 and 5.10. Although this is a condition where high interactions exist between the inputs and output, WMPC has not shown any performance issues.

Case C: Different response time requirements

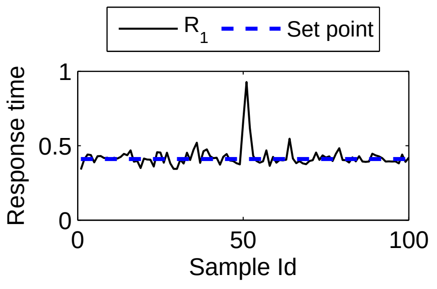
In this case, we set the response time requirements of different classes in different regions. That is in the sensitive and insensitive regions by setting $R_{SLA,0} = 0.41$ (sec) and $R_{SLA,1} = 0.6$ (sec) respectively. Therefore, $class_0$ gets much better response time, while $class_1$ gets relatively higher response time. The experiment starts with 20 and 20



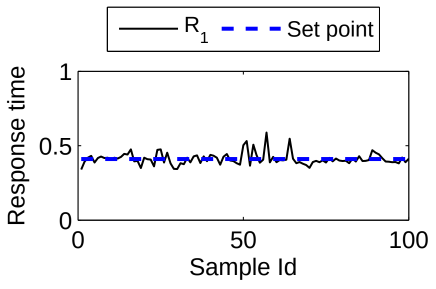
(a) Output R_0 of MIMO LMPC



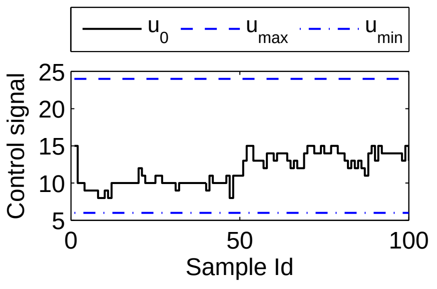
(b) Output R_0 of MIMO WMPC



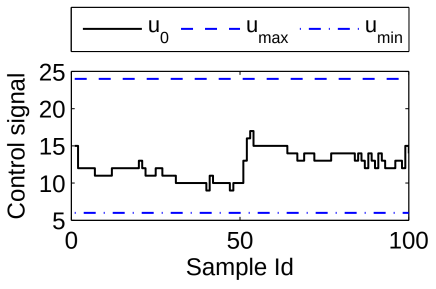
(c) Output R_1 of MIMO LMPC



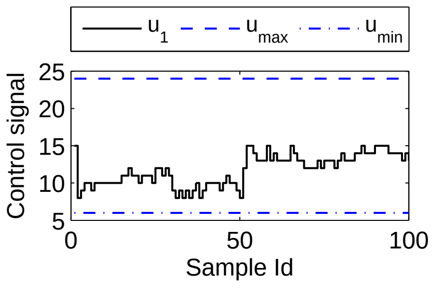
(d) Output R_1 of MIMO WMPC



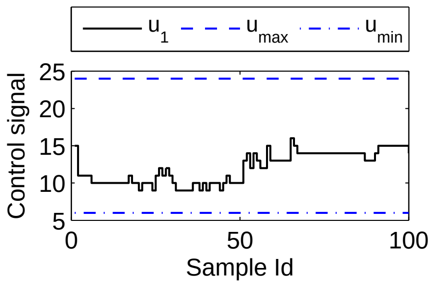
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC



(g) Control signal u_1 of MIMO LMPC

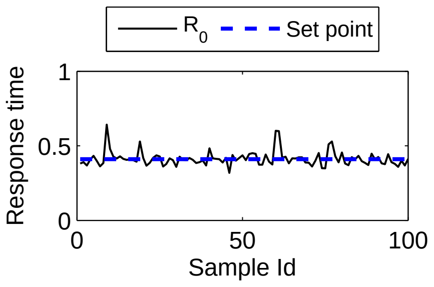


(h) Control signal u_1 of MIMO WMPC

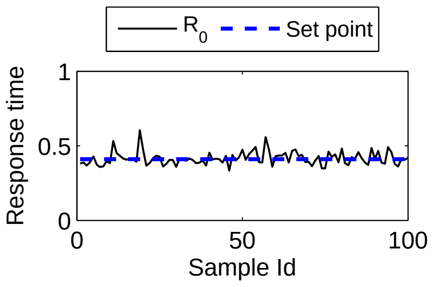
Figure 5.13: Performance management under simultaneously high workloads (Case B)

requests/sec for $class_0$ and $class_1$ respectively. Then, at the 50th sample $class_1$ increases its workload to 40 requests/sec.

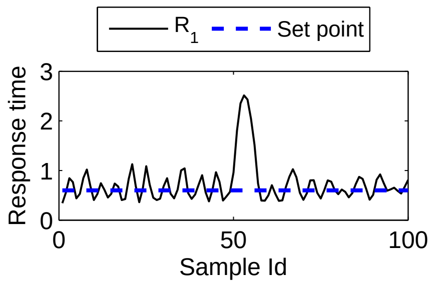
Figure 5.14 shows the responses of the control systems. In this condition, we assumed that $class_0$ is more important. Furthermore, reference signal (set point) of $class_1$ is placed



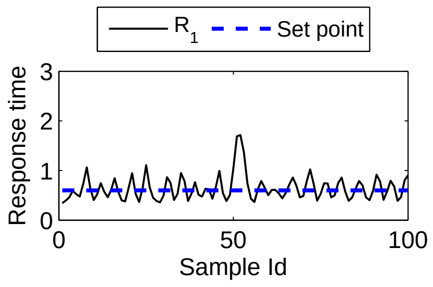
(a) Output R_0 of MIMO LMPC



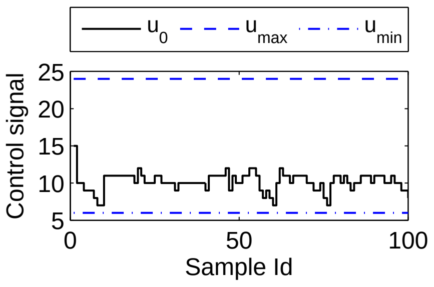
(b) Output R_0 of MIMO WMPC



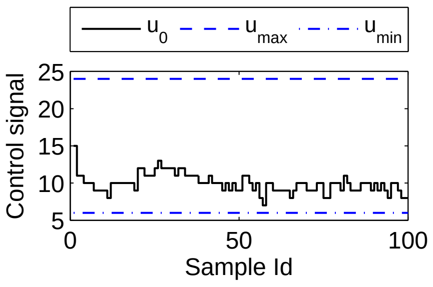
(c) Output R_1 of MIMO LMPC



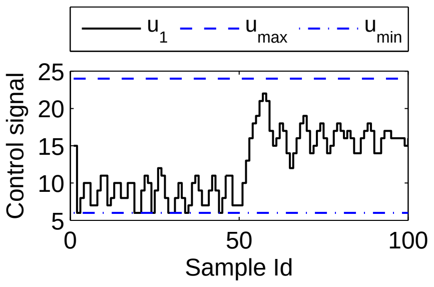
(d) Output R_1 of MIMO WMPC



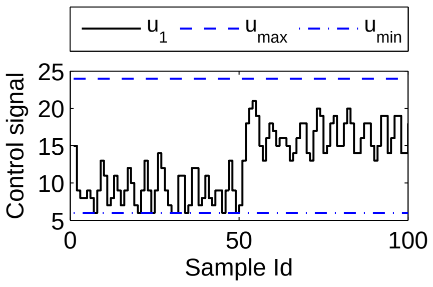
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC



(g) Control signal u_1 of MIMO LMPC



(h) Control signal u_1 of MIMO WMPC

Figure 5.14: Performance management under different reference values (Case C)

in the sensitive region of the response time curve (see Figure 4.13). Consequently, the resource allocation is delayed for $class_1$ in order to maintain the response time in the sensitive region. There is not much difference in the case of R_0 in both control systems (WMPC shows a slight improvement, see Table 5.7). In contrast, the steady state performance of LMPC is much poor compared to WMPC for the case of R_1. This is because R_1 is operating in the sensitive region where queuing delays and nonlinearity affect the performance of the control system. The compensation of the nonlinearity has reduced these effects producing much better performance in the sensitive region. In particular, the overshooting and the settling time have been improved with significantly better steady state behavior. See Tables 5.7, 5.8, 5.9 and 5.10.

Case D: Extreme overloaded condition

In this work, the absolute performance and resource management problem was formulated as a set point tracking problem using feedback control. A main known issue of the absolute management scheme is that the steady state value of the response time signal depends on the workload disturbance of each class under overloaded conditions [108, 145]. As a result, it is hard to specify a set point, because the controllability is lost for some workloads. In this case, we examine the performance management capabilities of the above control systems with an overloaded workload setting. Although the set point specified is not achieved, the optimization capabilities of the model predictive control system implemented in this work provides performance differentiation under changing workload conditions. We set $R_{SLA,0} = 0.41$ and $R_{SLA,1} = 1$ seconds and then overload the system by applying 75 requests/sec for each class.

The responses of both control systems shown in Figure 5.15 indicate similar performance and resource management capabilities. It is evident that the control objectives or set points of both classes are not achieved. However, the performance differentiation characteristics have been illustrated by the proposed control solution, because they have maintained R_1 much higher than R_0. An important observation is that the control signals have saturated (i.e., hit the maximum or minimum levels) indicating that this is the best possible resource allocation setting to achieve the prescribed optimization and control objectives. We will therefore not compare the management capabilities of WMPC and LMPC, under overloaded cases for absolute performance management scheme.

5.5.2 Summary of Results

In the above evaluations, the proposed nonlinear control system has shown significant performance improvements compared to the existing linear control system in all cases. In particular, we observed efficient disturbance rejection capabilities without sacrificing the

Table 5.7: Summary of statistics of $class_0$ output (R_0) under different cases

Case	LMPC			WMPC			Dif(WMPC -LMPC)		
	SSE	MIN	MAX	SSE	MIN	MAX	SSE	MIN	MAX
A	5.47	0.353	1.69	3.505	0.353	1.381	-1.965	0	-0.309
B	0.258	0.319	0.602	0.331	0.319	0.732	0.073	0	0.130
C	0.251	0.319	0.642	0.202	0.334	0.604	-0.049	0.015	-0.038
Cases covered in Appendix E.2									
D	0.993	0.317	0.732	0.524	0.317	0.59	-0.469	0	-0.142
E	26.977	0.155	1.287	23.217	0.155	1.518	-3.760	0	0.231

Table 5.8: Summary of statistics of $class_1$ output (R_1) under different cases

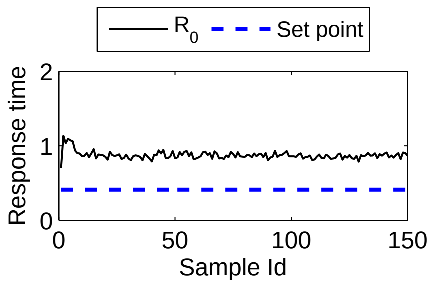
Case	LMPC			WMPC			Dif(WMPC -LMPC)		
	SSE	MIN	MAX	SSE	MIN	MAX	SSE	MIN	MAX
A	18.707	0.340	2.288	12.041	0.340	2.087	-6.666	0	-0.201
B	0.492	0.340	0.928	0.161	0.340	0.589	-0.331	0	-0.339
C	18.066	0.340	2.515	6.395	0.340	1.715	-11.671	0	-0.800
Cases covered in Appendix E.2									
D	0.928	0.335	0.839	0.603	0.335	0.668	-0.325	0	-0.171
E	24.527	0.216	1.526	22.787	0.216	1.526	-1.740	0	0

Table 5.9: Steady state statistics of $class_0$ output (R_0) under different cases

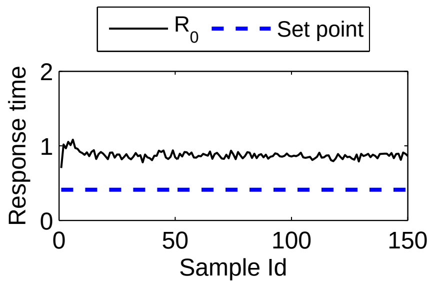
Case	LMPC		WMPC		Dif	
	SSE	Max	SSE	Max	SSE	Max
Case A after $class_0$ disturbance	1.507	0.659	1.153	0.679	-0.969	-0.260
Case A after $class_1$ disturbance	3.264	0.819	1.962	0.798	-1.188	-0.145
Case B	1.795	0.683	1.148	0.683	-0.643	-0.315
Case C	2.552	0.798	1.679	0.695	-19.171	-0.800

Table 5.10: Steady state statistics of $class_1$ output (R_1) under different cases

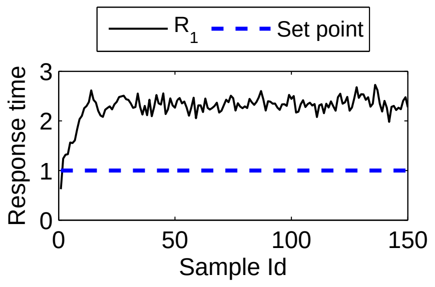
Case	LMPC		WMPC		Dif	
	SSE	Max	SSE	Max	SSE	Max
Case A after $class_0$ disturbance	2.477	0.876	1.658	0.656	-0.819	-0.220
Case A after $class_1$ disturbance	1.032	0.561	0.998	0.625	-0.034	0.064
Case B	1.371	0.635	1.075	0.613	-0.296	-0.022
Case C	36.312	1.290	27.053	1.139	-9.259	-0.151



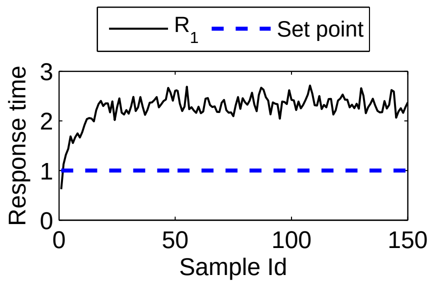
(a) Output R_0 of MIMO LMPC



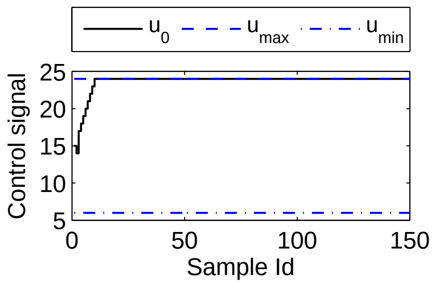
(b) Output R_0 of MIMO WMPC



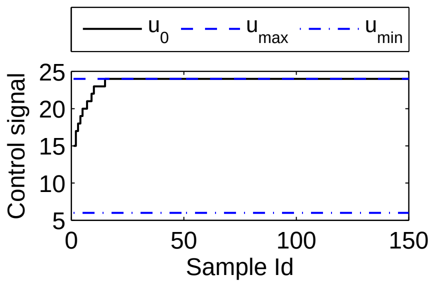
(c) Output R_1 of MIMO LMPC



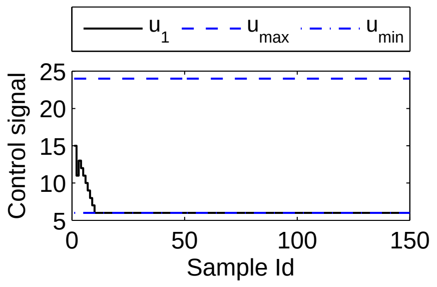
(d) Output R_1 of MIMO WMPC



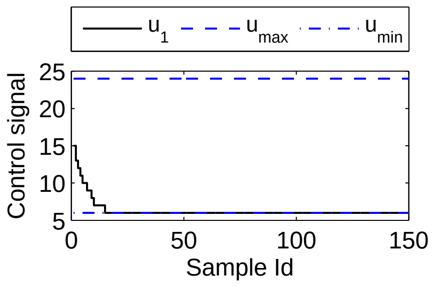
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC



(g) Control signal u_1 of MIMO LMPC



(h) Control signal u_1 of MIMO WMPC

Figure 5.15: Performance management under overload (Case D)

steady state behaviour of the system. When the reference signals (SLAs) have been placed in the sensitive region, the disturbances have been rejected efficiently under large workloads variations by the new MIMO Wiener MPC, showing low overshooting and settling time compared to the linear MPC.

Further evaluations of absolute performance management under different workload patterns (e.g., ramp and real-world workloads) can be found in Appendix E.2. In addition, a MIMO Wiener controller is designed, developed and evaluated in a multi-class system with three classes as well. The results are listed in Appendix F. Moreover, in order to investigate the effects of the configuration parameters of the simulation environment, further simulations are conducted in Appendix G using Monte-Carlo simulations. The results under these conditions did not invalidate the results presented in the above section. Chapter 8 further applies and validates the MIMO Wiener control approach in a real-world software environment.

5.6 Conclusion

This chapter have presented the design and development details of the new nonlinear control systems equipped with nonlinear compensators to achieve the relative and absolute performance management objectives of multi-class shared resource systems.

For the relative performance management, a new management system architecture based on the Hammerstein-Wiener model and equipped with pre-input and post-output compensators was used, in order to reduce the impact of the input and output nonlinear dynamics respectively. We also formulated the MIMO absolute performance and resource management problem of a multi-class system using a MIMO Wiener model predictive control. The nonlinearities at each output was represented using a MIMO Wiener model, and then compensators were designed and integrated at each output to reduce the impact of the nonlinearities on the management system. The runtime control and optimization problem of the transformed system was solved by MPC formulated with the transformed variables.

In summary, compared to the existing linear control techniques, the proposed nonlinear control techniques have successfully achieved the desired attributes required by the management system of a multi-class shared resource software environment, listed in Section 1.3. In addition, although the nonlinearities were explicitly considered in the design, we were still able to use well-established control engineering techniques to design the proposed nonlinear control systems providing systematic and formal design processes.

However, the nonlinear block-oriented models are most useful in the cases where nonlinearities are static. If this is not the case, integrated nonlinear blocks may amplify the non-static nonlinearities degrading the performance of the system [91]. Such behavior was observed in some of the cases where the output noise impacted the operations of the compensator, subsequently affecting the performance of the nonlinear control system. In such situations, the on-line approximation techniques (e.g., [65, 91]) that estimate the output nonlinearity or its inverse may provide improvements. Furthermore, one of the concerns in the self-managed software systems is the computational demand of the management system (for example, the Hammerstein-Wiener model control system with respect to relative management) [203]. The nonlinear control systems impose an additional computational overhead compared to a linear control system because of the integration of the nonlinear compensators. The computational efficiency also depends on the selected nonlinear function type (e.g., polynomial and log). We have quantified the computation overhead in [194].

Chapter 6

Performance Management Using Multi-Model Self-Managing Control Schemes

6.1 Introduction

In Chapter 4 and 5, we illustrated the nonlinear characteristics of relative and absolute management schemes. A single linear model typically captures the system behavior in a particular operating region under certain operating conditions. The experimental results in Chapter 4 and 5 indicated such linearization is inherently problematic because a software system has to work in a spectrum of operating regions with changing conditions and un-modelled system dynamics. However, in different operating regions, different models may estimate the system dynamics much better, while different controllers may achieve the required performance objectives effectively. We refer to the nature of a software system needing to operate across multiple regions and therefore needing multiple models to characterize the system behavior as the *multi-model* characteristic of the target system.

In order to improve the efficiency of a single linear model based control system, the earlier chapters proposed nonlinear block-oriented modelling techniques and control systems equipped with nonlinear compensators, which showed significant improvements compared to a linear control system. However, compensation of nonlinearity is not the only way to capture the multi-model (or nonlinear) characteristics. Other types of nonlinear modelling approaches could be: (i) design multiple static models that can capture and cope with the system behavior in different operating regions; (ii) develop models and algorithms that can adapt and learn at runtime; (iii) combination of (i) and (ii). After capturing the behavior, multiple controllers have to be designed and integrated to the control system, including the ability to detect the change of the operating regions and to switch between the appropriate controllers at runtime. These solutions demand control systems that can dynamically reconfigure themselves and select the suitable model and controller with little or no human intervention. We call such control systems *self-managing control systems*. A self-managing control system provides a high-level of adaptive capability, because it reconfigures the structure of the control system at runtime, according to the changes in the operating regions. However, deciding on the stable switching algorithms and its variables is a significant challenge under disturbances and changing dynamics. The focus of this chapter is to investigate suitable control engineering approaches to design new self-managing control system architectures to manage performance and resources of multi-class shared resource environments.

Designing a multi-model self-managing control system for relative performance management is a challenging task. As characterized in Chapter 4, there are input and output nonlinearities which affect the control system at runtime. From the cases studied in Chapter 5, we observed that for some cases, a single or both nonlinearities could affect the performance of the target system depending on the region of the input and output the control system is operating in. As a consequence, selecting just a single state variable of the system as the switching variable is problematic. Therefore, the relationships between input and output have to be considered together to represent the dynamics of the target system in multiple regions.

An approach called Multi-Model Switching and Tuning (MMST) adaptive control [117, 164, 166] has been proposed by (Narendra and Balakrishnan, 1993) to overcome the limitations of linear control and to address problems of adaptive control when the conditions change fast and frequently. MMST adaptive control enables integration of multiple models and controllers, and automates the selection of the appropriate model or controller to achieve the desired control objectives in a way that takes account of different operating regions of a target system. As such, MMST adaptive control can be categorized as a self-managing control technique. The existing performance management studies of software systems have used either linear or adaptive control techniques. However, due to the multi-model characteristics and fast changing conditions, MMST adaptive control is a suitable candidate to implement management systems for software systems.

In the first part of this chapter, we investigate and evaluate a relative management system design methodology based on the MMST adaptive control. The objective of this work is to investigate whether MMST adaptive control provides any performance improvements over the existing linear or adaptive control schemes. In order to achieve this objective, based on the input and output regions of relative management system, two models are designed and corresponding controllers are implemented, followed by integrating them into a MMST switching scheme. This new self-managing control system has shown the capability to autonomously detect the change of operating regions and then select the most suitable controller to provide control decisions in that particular operating region at runtime for most experimental conditions.

The second part of this chapter examines a switching scheme to achieve absolute performance management objectives. For the case of absolute performance management, the steady state error is higher when an output is operating in the sensitive region compared to when it is operating in the insensitive region. In order to provide satisfactory performance in both regions under disturbances, a single linear model based controller has to sacrifice performance in both regions to some extent. In this chapter, we propose a new rule-based multi-model self-managing control system design method to implement the absolute management system. The switching is performed by observing the reference signals of each class (i.e., $R_{SLA,i}, i = 0, 1, \ldots, n$). The management capabilities of this scheme are also evaluated later on in this chapter and have shown significant design flexibility and performance improvements. Furthermore, in contrast to MMST adaptive control, this rule-based approach makes the switching decisions based on the reference signals, which do not depend on the environmental conditions.

6.2 Relative Performance Management Using Multi-Model Control

This section overviews a multi-model control system design mechanism for relative performance and resource management. The proposed design methodology is based on the MMST adaptive control [117, 164, 166]. Firstly, we introduce the basic concepts of MMST adaptive control. Secondly, we formulate the problem and then the design methodology of the relative management system according to the MMST adaptive control will be covered.

6.2.1 A Brief Overview of MMST Adaptive Control

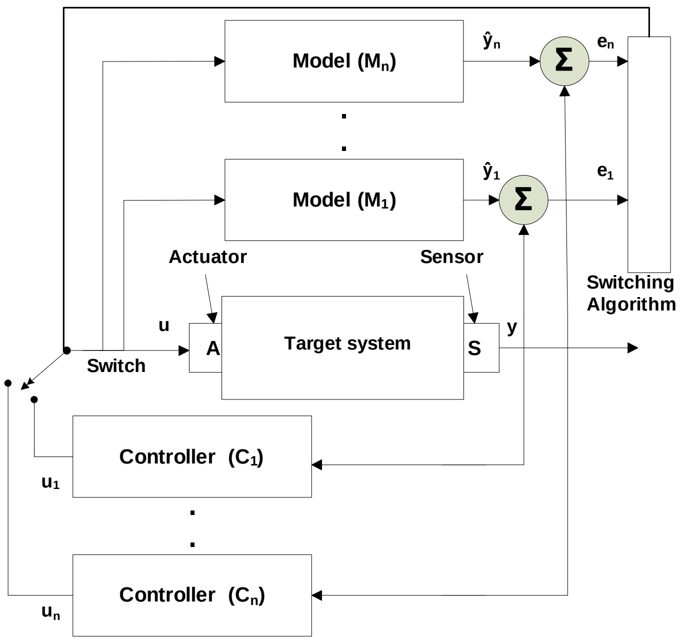


Figure 6.1: Block diagram of MMST adaptive control system

The MMST adaptive control was proposed by (Narendra and Balakrishnan, 1993) to improve the transient response of adaptive control systems in the presence of model uncertainties [166]. It is a concept inspired by biological systems [117]. Biological systems have the ability to select an appropriate action for a specific situation from a collection of behaviors. MMST uses the same concept by selecting the most suitable controller for the current environment that the system is in. Figure 6.1 shows the main components of MMST adaptive control.

The input and output of the target system are represented by u and y respectively. There are n number of models $(M_1, M_2, \dots M_n)$ describing the relationship between u and y for different operating conditions, which provide estimations for the system output, simultaneously. The estimates from these n models are denoted by $\hat{y}_1, \hat{y}_2, \dots \hat{y}_n$. Similarly, there may be maximum of n controllers, with each corresponding to a model. Although there are multiple controllers, only a single controller can be connected in the control loop to make the control decisions at a given time instance. Thus, the most appropriate model and controller for the system and environment condition have to be selected to make the control decisions at runtime. The responsibility of the switching algorithm is to select the appropriate model and corresponding controller based on some criteria that will improve the performance of the target system. There are multiple switching algorithms discussed in [168]. All of these algorithms are based on the prediction errors of the models ($e = y - \hat{y}$). The prediction error provides the indication that at the current instance, which model fits the current operating conditions of the system. Hence, the integration of this model and the corresponding controller into the control loop should improve the performance of the control system [168]. The model evaluation and selection steps of the switching algorithm are summarized as follows:

Model evaluation:

$$J_i(k) = \alpha e^2(k) + \beta \sum_{r=0}^{k} e^2(r), \forall i = 1, 2, \ldots n \tag{6.1}$$

Model selection:

$$J_{min}(k) = min\{j_i(k)\}, i = 1, 2, \ldots n \tag{6.2}$$

k is the time instance. α, $\beta \geq 0$ are parameters that should be carefully decided by the designer. The first and second term of equation (6.1) are called the instantaneous-term and long-term components respectively. If $\alpha > 0$, $\beta = 0$, only the instantaneous part is utilized. In this case switching may be frequent, leading to performance degradation [166]. If $\alpha = 0$, $\beta > 0$ only the long-term component is active, hence the switching may be infrequent, again possibly lead to performance degradation [166]. In the model evolution step, the J index of each model is calculated based on equation (6.1) using the prediction error data of each model. In the model selection step, the model that produces minimum J index (J_{min}) is selected and the corresponding controller is integrated into the control loop. In addition, T_{min} is another parameter called waiting time period which specifies the time that the control system has to wait before selecting the next controller to control the system [164, 166]. The index in equation (6.1) is most suitable in time-invariant environments. For time-varying environments long term error accumulation will affect the J indexes. In such conditions the performance of the MMST adaptive control systems can be improved by calculating the performance index in a finite window ($T \geq 0$) as illustrated in equation (6.3).

$$j_i(k) = \alpha e^2(k) + \beta \sum_{r=k-T+1}^{k} e^2(r), \forall i = 1, 2, \dots n \tag{6.3}$$

The above discussion presents the general concepts of the MMST adaptive control. Going further, different types of multi-model schemes have been evaluated and formal stability proofs are provided in [164, 165]. These multi model schemes are as follows.

Type 1: *All adaptive models*- in this scheme all the system models ($M_i, i = 1, 2, \dots n$) are estimated by the on-line identification (estimation) algorithms [13, 125]. The corresponding controllers use the parameter estimations to come up with the control input u. This scheme is computationally inefficient. In addition, if the environment remains unchanged for a long time, all the adaptive models will converge to the same parameter neighbourhood which reduces the advantage of having multiple models. In addition, when a sudden disturbance occurs, models may not react to it rapidly, without re-initialization due to the inherent characteristics of adaptive control [117].

Type 2: *All Fixed models*- This scheme addresses some of the limitations in the type 1 scheme by integrating fixed models and fixed gain controllers. Fixed gain control is generally not regarded as an adaptive technique, but because of the switching capabilities this scheme can be considered as an adaptive reconfiguring control technique. However, fixed models can only represent a finite number of operating regions or conditions. As such, this scheme assumes that there is always one of the models that closely approximate the system behavior. Therefore, to satisfy this assumption and the stability requirements we may have to build a large number of fixed models.

Type 3: *One Adaptive model and one Fixed model*- In this scheme, initially the fixed model may be selected since the adaptive model takes time to converge at the startup. However, when the adaptive model converges, it will often outperform the fixed model. This scheme is simple and addresses some of the limitations in the above two schemes, however the same limitations of adaptive control exist under fast changing conditions. Consequently, much improvement in the performance may not be achieved compared to a single adaptive model based control.

Type 4: *Adaptive models and Fixed models*- Different types of schemes can be formulated combining adaptive and fixed models. Two main configurations are discussed in [117, 164]. The first configuration includes *n-1 fixed models and one adaptive model.*

From prior knowledge of the system's operating and environment conditions, $n-1$ number of fixed models can be designed. Then, the adaptive model is run free of interference to capture the system dynamics that is not captured by the fixed models. On the other hand, the second configuration includes another adaptive model, i.e., involving *n-2 fixed models and 2 adaptive models.* The second adaptive model is re-initialized with the parameters of the best model in the current time instance. The main purpose of this adaptive model is that after re-initialization it may converge faster to the new model parameters so that the transient responses may be improved under sudden disturbances. To achieve effective performance under these two configurations, the design of the fixed models has to be done after carefully analyzing the available prior knowledge on the system and its environment.

The above discussion provides the objectives, features and some limitations of different MMST adaptive control schemes. MMST adaptive control is an *indirect adaptive control* [13] scheme since it depends on the model selection or estimation before providing the control decisions. Furthermore, it is a form of *reconfiguring control* (see Appendix B) because MMST adaptive control changes the models, controllers, components and the architecture of the control system at runtime, depending on the changing conditions. Interested readers are referred to [164, 168] for the details of simulation studies and the stability proofs of these MMST schemes in guaranteeing that the system will not be unstable due to the switching and tuning behavior.

6.2.2 Analysis of MMST Adaptive Control for Management of Software System

Several papers have illustrated the problems of fixed [107, 151] and adaptive [92, 189] control to manage software systems. MMST adaptive control combines the qualities of both of these regimes. In general, MMST adaptive control can be used to represent a software system with multiple models. Furthermore, it can self-manage the control system without any human intervention, reducing the system administration effort. However, there is a set of questions in the design of such a scheme for a software system. The main questions are: (i) which schemes to use, (ii) how many fixed or adaptive models and how to identify them and (iii) how to configure the switching algorithm. Answers to these questions are application or requirements dependent. At the design time, suitable models and controllers have to be formulated, depending on the available knowledge about the operating conditions, system behavior and physical analysis [117]. However, precise knowledge is not required, because the models are estimations of system dynamics, which are not always 100% accurate. Considering the characteristics of the software systems, type-2 and type-4 schemes may be most suitable. This is because, type-1 is computational inefficient and type-3 may not provide improvements compared to the basic adaptive control. For systems where some prior knowledge is available about the operating conditions, different fixed models can be approximated and integrated utilizing MMST type-2 scheme. If there is little or no prior knowledge about the system or the systems that change often, there is the need to integrate adaptive models together with the fixed models using the MMST type-4 scheme. However, if there is little prior knowledge, fixed models can be uniformly placed in the model parameter space as proposed in [167]. It is also recommended to start with a small number of models and include more models depending on the performance observed. Since software systems investigated in this thesis are typically nonlinear and time-varying [81], the switching scheme in equation (6.3) may be most suitable by setting $\alpha = 0$, $\beta = 1$ to achieve predictable and consistent switching. T is recommended to be set to a small value to avoid large transient responses if the conditions are fast varying.

6.2.3 Relative Performance Management Using MMST Adaptive Control

In Chapter 4, the nonlinear characteristics of the relative management system were conceptualized as the input and output nonlinearities. Instead, in this section, after conceptually fragmenting the dynamics of the system into multiple regions, a linear model is used to represent the dynamics of each region. Then, using the identified models a MMST adaptive control system is implemented.

6.2.3.1 Model Identification

Figure 4.2 and 4.3 in Chapter 4 illustrate two regions of the input and output of the relative management system, namely region (0) and (1). In these two regions the dynamics of the system are significantly different. As a result, representing the system with a single linear model and then designing a single controller indicated significant performance issues. In this section, the system dynamics in these two regions will be estimated with two linear models. For this purpose, two system identification experiments have to be conducted.

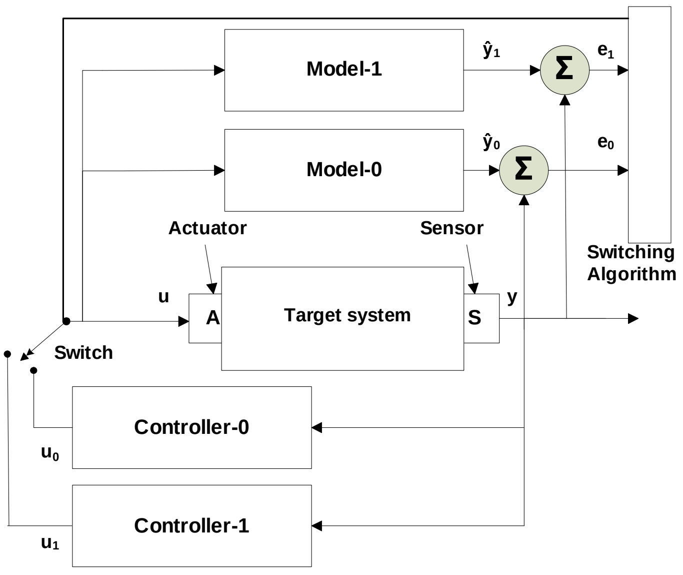
Firstly, to compute the possible operating points (or the range of u) of the system, we can utilize equation (4.1) and calculate the points for $S_0 \in \left\{S_{0,min}, S_{total} - \sum_{j=1}^{n-1}(S_{j,min})\right\}$. These computed operating points are for a controller managing the first pair of classes, assuming that the rest of the classes are guaranteed the required minimum allocation $S_{i,min}$. The point that both $class_0$ and $class_1$ get equal amount of resources is called as nominal operating point. The rest of the operating points are grouped into *region 0* and *region 1* where $class_0$ or $class_1$ workloads get more resources respectively. The first system identification experiment is conducted using the operating points in *region 0*, by applying high workload disturbance for $class_0$ and a nominal workload for $class_1$. As a consequence, the system output will remain in *region 0* (see, Figure 4.3). The gathered $u - y$ data is then used to estimate the dynamics of the system. Similarly, the dynamics in the *region 1* is estimated using a second system identification experiment. Let us denote the models estimated from the first and second experiment as *model-0* and *model-1* respectively.

6.2.3.2 MMST Adaptive Control System Design

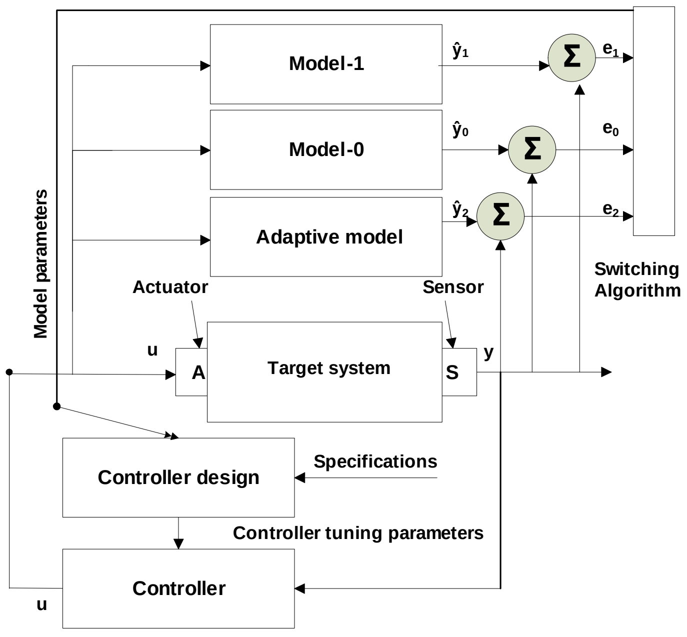
As discussed in Section 6.2.2, MMST-Type 2 and 4 schemes are more suitable to implement control systems for software systems. In this section, using the two models identified in Section 6.2.3.1, implementation steps of MMST-Type 2 and 4 schemes are covered. The design procedure is similar to Chapter 5, where a control system is designed for the first pair of classes in the system. The same MMST adaptive control system is then integrated to manage the performance and resources of each consecutive pair of classes.

Implementation of MMST-Type 2 scheme

As shown in Figure 6.2a, MMST-T2 scheme requires integration of the fixed models representing different operating regions and the respective controllers. After identification of the models (see Section 6.2.3.1), the next step is to construct the suitable controllers and configuring the switching parameters. Two types of discrete PI control laws are listed in Appendix B. Equation (B.1) illustrates the *position/full-value form* of the PI control law, which is widely used to manage software systems (e.g., see [61, 81, 183]). However, in the switching control systems this control law may cause performance issues. For instance, when the control is switched from one controller to another due to a large disturbance, the



(a) MMST-T2



(b) MMST-T4

Figure 6.2: The block diagrams of implemented MMST control system architectures

integral term (the second term) in equation (B.1) may have different values, which may lead to implementation of different or unsuitable control inputs by the new controller. As a consequence, there may be large transient output responses, failing to achieve *bump-less* transfers [258]. In contrast, the *velocity/incremental form* of the PI law illustrated in equation (B.2) is well known in implementing bump-less transfers in the case of mode and controller switching systems [258]. This is because the control input of the previous sample ($u(k-1)$) is used as a relative point and the incremental part (computed by summation of the second and third terms) is added to it. Consequently, $u(k)$ may not have large deviations from $u(k-1)$ even after the controllers are switched. As a consequence, in this work, the velocity form of PI control is implemented to achieve smooth or bump-less transfers, when the controllers are switched abruptly in and out of the control system. This is the same PI control algorithm presented in Chapter 5.

Next, the gains (K_p, K_i) for the two controllers have to be decided, which provide the control in *region 0* and *region 1.* As seen from the experimental results and observations in Chapter 4 and 5, it is hard for a single PI controller to satisfy the performance objectives and stability of the system under many different conditions. An aggressive controller (with large gains) is needed to achieve the performance objectives in *region 0* (called *controller-0*), whereas a comparatively less aggressive controller (with smaller gains) is needed to provide control in *region 1* (called *controller-1*). The aggressive controller will take larger steps in *region 0* and reach the desired control inputs in *region 0*, which has larger gaps between the operating points. However, in *region 1* because of the spacing between the control inputs is small (see Figure 4.2), the large steps taken by the aggressive controller will create instabilities. In contrast, the less aggressive controller taking smaller steps settles to appropriate operating points in *region 1*, avoiding such instabilities. But the less aggressive controller will show large settling times and performance degradation when it is operating in *region 0.* The gains for these regions can be finalized using the simulation studies by integrating each controller separately to the control system. The pole-placement design methodology introduced in Chapter 5 can be used here as well[1].

After finalizing the models and controllers, {controller-0, model-0} and {controller-1, model-1} are grouped together. The final step is to configuration of the switching al-

[1]In order to adjust the gains for each controller, the α and β values of the desired closed-loop equation have to be changed.

gorithm. This step includes, selection of the parameters of the switching algorithm, α, β, (T) and the start-up controller[2]. The model evaluation algorithm with a finite window illustrated in equation (6.3) is used, because of the time-varying operating conditions. Extensive simulations and testing have to be conducted in order to fine tune these configuration parameters. Consequently, design heuristics have to be used in this implementation to reduce the development costs (see, Section 6.3.3).

Implementation of MMST-Type 4 scheme

MMST-Type 4 (from here on, MMST-T4) scheme is a complex scheme compared to MMST-T2 because of the addition of the adaptive models. Figure 6.2b shows MMST-T4 scheme with a free running adaptive model integrated together with the estimated two fixed models. Hence, this is a MMST-T4 configuration with $n-1$ fixed models and an adaptive model.

The controller is a single self-tuning PI controller (with a similar structure to Figure B.1b), which computes the controller gains given the parameters of the model and the design specifications. The two fixed models (model (0) and (1)) and a first order ARX adaptive model (implementing the recursive least squares algorithm [13, 125]) are evaluated by the switching algorithm using equation (6.3) and subsequently the best model is selected at each switching period. The parameters of the selected model are then given to the controller design component to compute the gains of the self-tuning PI controller. Furthermore, the design specification includes the desired closed-loop equation with α and β. The velocity form is used in the self-tuning PI controller as well (see [81] for more details). Afterwards, *model-0*, *model-1* and the adaptive model are integrated to the control system, with the self-tuning PI controller as the controller. In addition to the design parameters of MMST-T2, here the parameters of the adaptive model (e.g., forgetting factor [13]) have to be set as well.

Finally, the selection of the suitable switching algorithm parameters is paramount after conducting the simulations to avoid chattering and instabilities.

[2]*controller-0* or *controller-1* can be selected as the start-up controller to provide control till the first switching decision is made by the MMST control system. The start-up controller operates in the first T samples.

6.3 Evaluation of MMST Adaptive Control Systems

In this section, we provide the implementation details and evaluation results of MMST-T2 and T4 control systems designed for relative performance management. The simulation settings used in Chapter 5 will be used here as well.

Model identification. The first step is to compute the input range and fragment it into *region 0* and *region 1*. Using the procedure explained in Section 6.2.3.2, the set of operating points for the control input u is $\frac{6}{24}, \frac{7}{23}, \ldots, \frac{23}{7}, \frac{24}{6}$ (see Figure 4.2). *region 0* includes operating points $\frac{15}{15}, \frac{16}{14}, \ldots, \frac{23}{7}, \frac{24}{6}$ and *region 1* includes operating points $\frac{6}{24}, \frac{7}{23}, \ldots, \frac{14}{16}, \frac{15}{15}$. A system identification experiment is then conducted to capture the dynamics in each of these regions, simulating the suitable workloads. The points in *region 0* were used to design a pseudo random signal for the system identification experiment. 45 requests/sec and 15 requests/sec were selected to simulate $class_0$ and $class_1$ workloads respectively, indicating high workload and resource demands for $class_0$. This experiment was carried out for 600 sample periods and the gathered input-output $(u - y)$ data was used to estimate the model for *region 0*. The data samples till the 400th period were included in the estimation set and the rest of the data samples formed the test set. A first order ARX model was used to fit the data with sufficient accuracy (i.e., *model-0*). A similar experiment was carried out in the *region 1* as well. Again, a first order ARX model was used to fit the data with sufficient accuracy (which we call *model-1*). The model parameters and structure of *model-0* and *model-1* are shown in equations (6.4) and (6.5) respectively, with R^2 fit over 75%. When these two models are compared, although the output coefficients are similar, the input coefficients are significantly different. As a result, the step responses of these models are significantly different from each other. This means that these two models represent different dynamics, therefore they are suitable to implement multi-model control systems.

$$y(k+1) = 0.63y(k) + 0.14u(k) \tag{6.4}$$

$$y(k+1) = 0.64y(k) + 0.96u(k) \tag{6.5}$$

MMST-T2 implementation. As mentioned in Section 6.2.3.2, aggressive

Table 6.1: Parameters of the control systems

Parameter	controller -0	controller-1
K_p	0.84	0.24
K_i	0.42	0.06
Min situation limit	0.25	0.25
Max situation limit	4	4
Initial input	u(0) = 1	u(0) = 1

(controller-0)[3] and relatively less aggressive (controller-1)[4] controllers were designed to provide control in *region 0* and *region 1* respectively. The controller parameters are shown in Table 6.1. It is worth noting that *controller-0* is more aggressive than the linear controller used in the simulation studies of Chapter 5. On the other hand, *controller-1* is significantly less aggressive than the linear controller used in Chapter 5.

We set α and β to 0 and 1 respectively to achieve consistent switching under noisy conditions. (T) was set to 3, to trade-off between chattering and reaction time to changing conditions/regions. Furthermore, without loss of generality*controller-0* was set as the start-up controller.

MMST-T4 implementation. The same parameters specified in Section MMST-T2 were used in this implementation as well. The forgetting factor of the adaptive model was set to 0.94 (around the standard value recommended by [13]). The specifications to the control design component is the desired pole locations of the self-tuning PI controller, which were placed at (0.7 and 0.7). It is important to emphasize that due to the highly dynamic operation conditions, tuning of the adaptive controller was significantly difficult and the performance was poor in most cases.

In the following sections we validate the performance of MMST-T2 and T4, under the operating conditions and business requirements used in Chapter 5. See Section 5.3 for the details of the settings used in the following cases. In order to compare the performance of a single linear model based control system, the results produced by LCS in Chapter 5 are presented here as well. We also present the results of a standard self-tuning PI adaptive controller (namely, ACS) implemented with the same settings used in the MMST-T4 control system. It is also worth noting that the results for only some cases will be presented, however, a summary of the statistics for all cases is listed in Tables 6.2 and 6.3. The rest

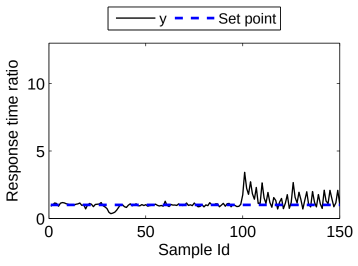
[3]by placing poles of the desired closed-loop equation at $\alpha = 0.45$ and $\beta = 0.45$

[4]by placing poles of the desired closed-loop equation at $\alpha = 0.8$ and $\beta = 0.8$

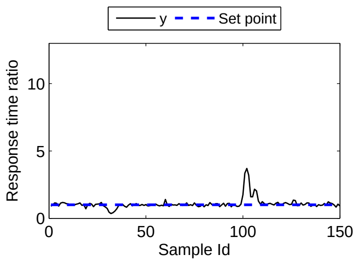
of the evaluations can be found in Appendix E.3.

6.3.1 Workloads of Full Capacity or Lower

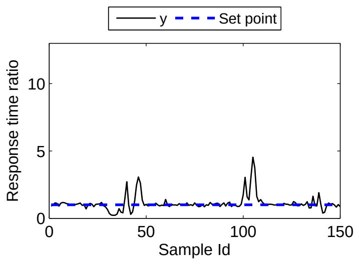
Case A*: Performance away from the nominal region*



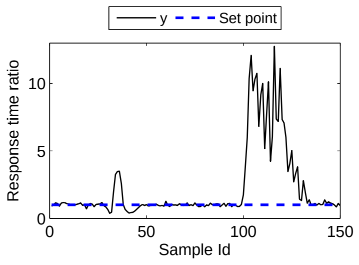
(a) Output of LCS



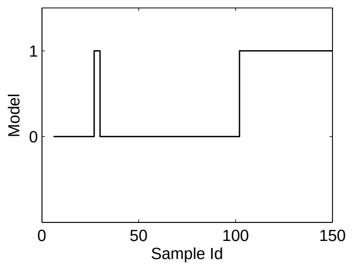
(b) Output of MMST-T2



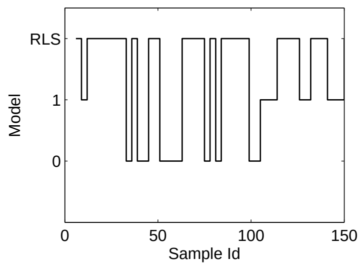
(c) Output of MMST-T4



(d) Adaptive



(e) Model switching signal of MMST-T2



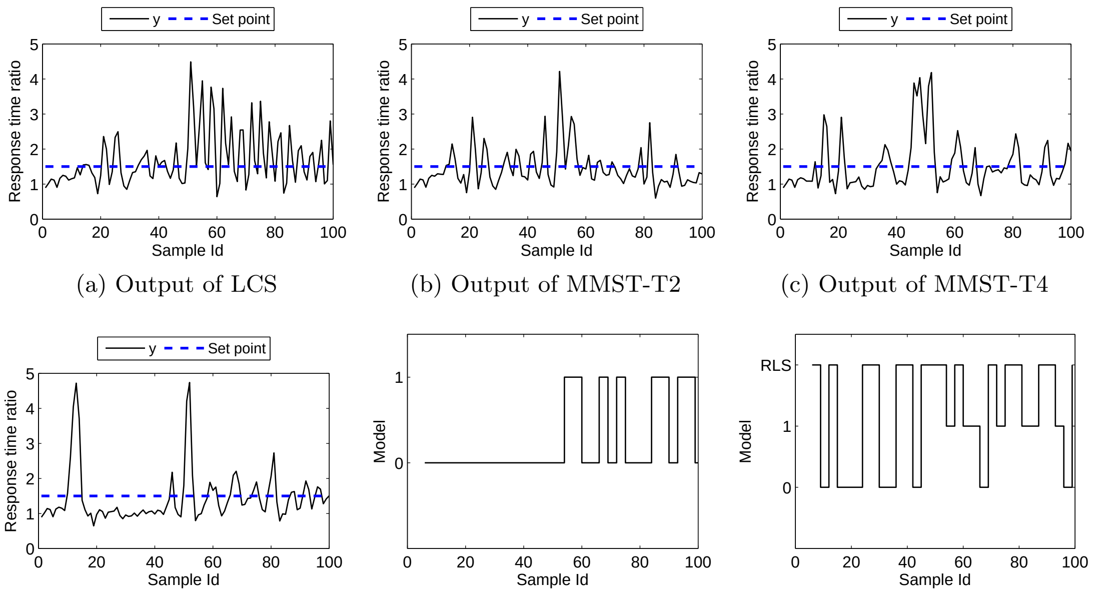
(f) Model switching signal of MMST-T4

Figure 6.3: Performance of the control systems away from the nominal region (Case A)

In this case, LCS showed significant performance issues and instabilities after the disturbance at the 100th sample. This issue has been resolved by MMST-T2 control system, which combines the performance of two controllers particularly designed to operate in each region. The model switching signal in Figure 6.3e illustrates that *controller-0* designed to operate in *region 0* was selected by the switching scheme autonomously, by detecting the change of workload conditions at the 30th sample. Similarly, *model-1* and *controller-1* were selected to operate after the 100th sample, reducing the effect of poor performance illustrated by LCS.

Although MMST-T4 shows poorer performance compared to MMST-T2, it shows much better control compared to LCS and adaptive control in this case. The adaptive controller shows large transient error after the disturbance at the 100th sample. This is because of the inherent limitation of adaptive control in reacting to sudden changes in conditions. However, *model-1* was selected by the MMST-T4 switching algorithm most of the time after the 100th sample which captures the dynamics sufficiently in *region-1*, improving the performance compared to the standard adaptive controller. It is interesting to note that the adaptive model has been selected by MMST-T4, when the adaptive model has settled to the new parameter estimates after the disturbances.

***Case B**: When $class_0$ is more important*



(a) Output of LCS (b) Output of MMST-T2 (c) Output of MMST-T4

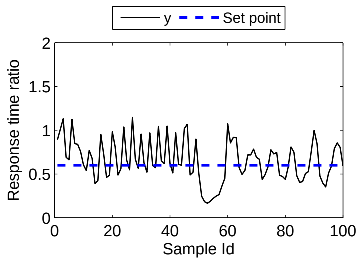
(d) Output of ACS (e) Model switching signal of MMST-T2 (f) Model switching signal of MMST-T4

Figure 6.4: Performance of the control systems when $class_0$ is more important (Case B)

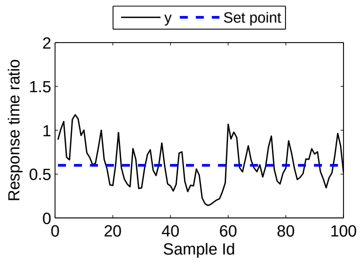
In this case, all control systems show deviations from the set point, in particular at the 50^{th} sample due to the workload disturbance (see Figure 6.4). The statistics in Table 6.2 indicates that MMST-T2 scheme provides better performance compared to LCS, adaptive and MMST-T4 schemes. Although chattering was observed in the case of MMST-T2 scheme (see Figure 6.4e), *controller-1* was selected most of the time after the 50^{th} sample indicating *model-1* is closer to the conditions in this case. However, the noise of the output signal leads the control system to falsely detect the change of dynamics, subsequently to select *controller-0* abruptly. The bump-less transfers implemented by the PI controller mitigate the issues of chattering to some extent, indicating to better performance compared to LCS. Both control systems that included an adaptive model showed significant issues under these conditions.

***Case C**: When $class_1$ is more important*

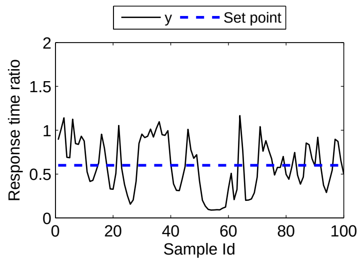
Figure 6.5 shows the performance of the control systems under Case C settings.



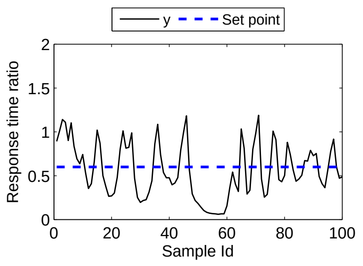
(a) Output of LCS



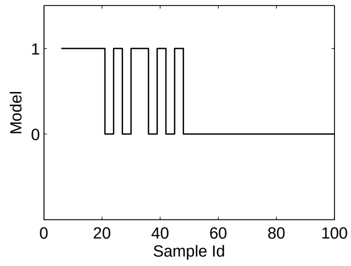
(b) Output of MMST-T2



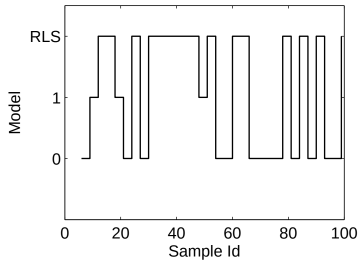
(c) Output of MMST-T4



(d) Output of ACS



(e) Model switching signal of MMST-T2



(f) Model switching signal of MMST-T4

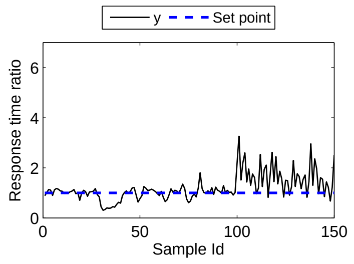
Figure 6.5: Performance of the control systems when class$_1$ is more important (Case C)

MMST-T2 outperforms MMST-T4 and ACS, while providing similar performance to LCS. Again chattering was observed during the first 50 samples till the disturbance, however *model-0* and *controller-0* were selected afterwards. Even though combination of the two controllers led to chattering, the performance has not significantly affected compared to LCS, because of the bump-less transfers. The poor performance of the MMST-T4 control system is caused by the poor performance of the adaptive model and chattering under the changing conditions.

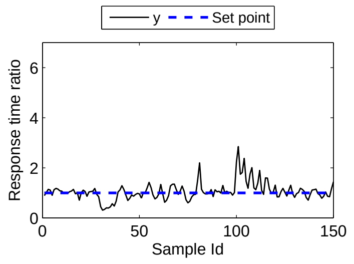
6.3.2 Workloads of Extreme Overload

***Case D**: Performance in away from the nominal region under overload*

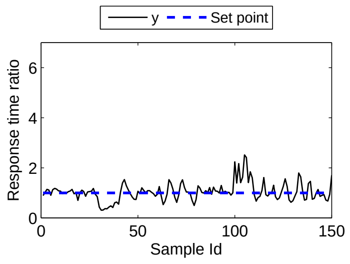
The results of the control systems shown in Figure 6.6 and Table 6.2 indicate that MMST-T2 has outperformed LCS, ACS and MMST-T4 control systems. The better performance is achieved by the combined performance of the *controller-0* and *controller-1*, which are autonomously selected depending on the operating conditions (See Figure 6.6e). Further, no chattering was observed in this case. MMST-T4 also provides better performance compared to LCS and ACS. The model switching shown in Figure 6.6f indicates that the adaptive model was selected infrequently compared to other two models. As a result, MMST-T4 shows similar behavior to MMST-T2 scheme. Further, less workload



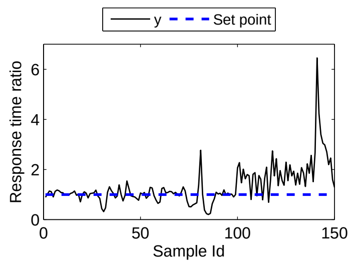
(a) Output of LCS



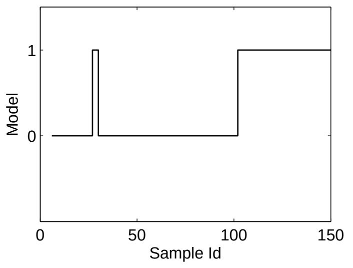
(b) Output of MMST-T2



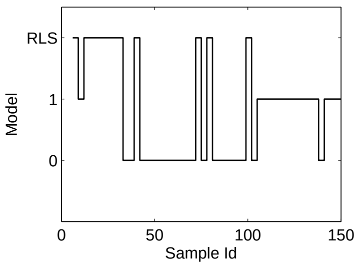
(c) Output of MMST-T4



(d) Output of ACS



(e) Model switching signal of MMST-T2



(f) Model switching signal of MMST-T4

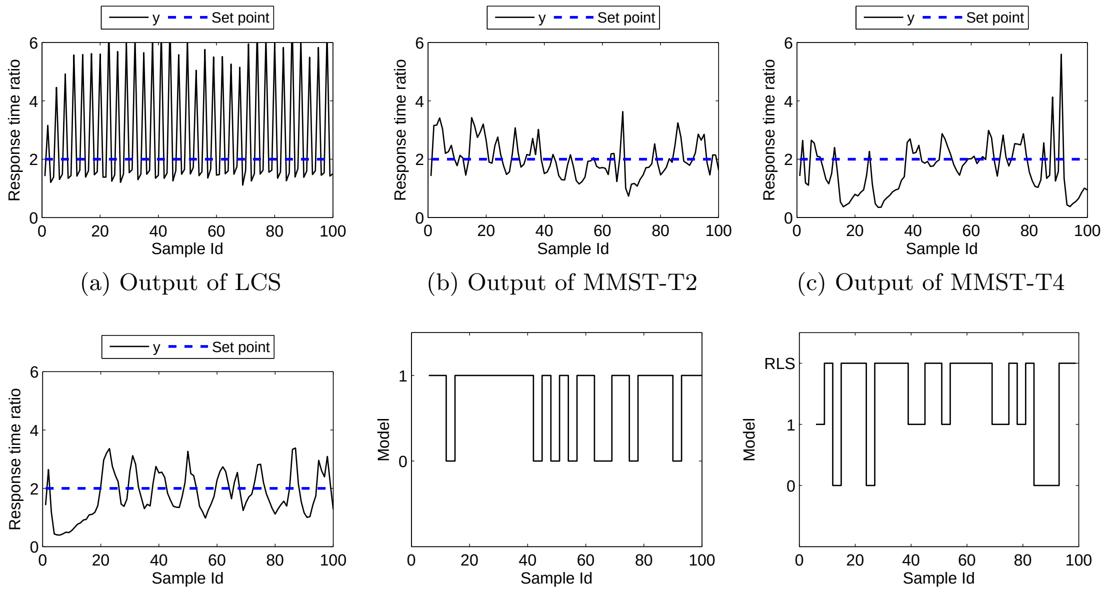
Figure 6.6: Performance of the control systems away from the nominal region under overload (Case D)

was rejected by the two MMST control systems compared to two other control systems (see Table 6.3).

***Case E**: Performance under overload when class$_0$ is more important*

Figure 6.7 shows the performance of the control systems under Case E settings. MMST-T2 control system has selected the *model-1* most of the time providing better performance compared to LCS, MMST-T4 and ACS. Although the results summarized in Table 6.2 indicate improvements for this case, the long-term execution of the MMST-T2 control system showed instabilities similar to LCS. This is because, selection of the *controller-0* which is even more aggressive than the LCS, led to instability which the MMST-T2 control system was unable to recover from. A possible solution to improve the performance in this case was to change the T parameter of the switching algorithm to as short as 1 or 2, instead of 3 used in this case. However, such small T values affects the results of the other cases examined earlier.

In summary, when MMST-T2 and LCS is compared, the general observations are MMST provides better performance in cases where output noise is lower (e.g., Case A, D), avoiding the instabilities and performance degradation of a single linear controller.



(a) Output of LCS (b) Output of MMST-T2 (c) Output of MMST-T4

(d) Output of ACS (e) Model switching signal of MMST-T2 (f) Model switching signal of MMST-T4

Figure 6.7: Performance under overload when $class_0$ is more important (Case E)

However, the issues inherit to switching control systems such as chattering were observed in the cases that show noisy output signals (e.g., Case B, C, E and F), subsequently leading to transient responses. Although the velocity form of the PI controller substantially mitigated the repercussions of chattering, the other parameters of the switching algorithm have to be still selected properly depending on the case. Therefore, MMST-T2 scheme has to be tuned and used only after careful analysis based on the simulations. It is also worth noting that, MMST-T2 control approach was less effective in tackling the nonlinearities that exist in the relative performance management system compared to the Hammerstein-Wiener approach presented in Chapter 5.

When MMST-T2, MMST-T4 and ACS are compared, the general conclusion from the results in Table 6.2 is that MMST-T2 provides better performance. The problems of the other control systems are because of the fast changing dynamics, noise and nonlinearities that exist in the relative management system affect the closed-loop model estimation at runtime when there are sudden variations in operating conditions. Consequently, the gains computed at runtime by MMST-T4 and ACS are either too high or too low causing instabilities or larger transient responses compared to LCS and MMST-T2. As a result, incorporating an adaptive model in these cases leads to many performance management problems.

6.3.3 Impact of the Tuning Parameters of MMST Adaptive Control

In this section, we illustrate the importance of selecting suitable tuning parameters of the MMST adaptive control in order to achieve the required performance objectives avoiding the limitations inherent to switching control systems. Furthermore, a set of guidelines is specified for deciding the tuning parameters in such implementations. We use 'Case A' settings, which provided consistent performance and model switching to investigate the effects of these parameters.

6.3.3.1 Impact of Number of Models

Theoretically, for the case of MMST-T2 scheme, when the number of fixed models increases the performance may become much better [117]. However, the downside is the computational overhead and chattering under fast changing conditions. Consequently, the suitable number of models has to be decided carefully, after investigating the prior knowledge available on the system. We investigated the behavior of MMST-T2, when another model is added to explicitly represent the nominal input and output region (model-2) of relative management system. In addition, another controller was designed to operate in that region as well (controller-2).

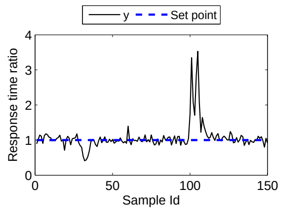
Figure 6.8a shows the output and model switching signals of the control system. The interesting observation is that *controller-2* was selected till the 30th sample because it was explicitly designed to represent the nominal region. The overall performance was satisfactory, however abrupt model switching was observed compared to the performance of Case A in Section 6.3. In particular, when the control system operates in *region 1*, chattering occurs for short time period affecting the steady state behavior slightly. Consequently, compared to the performance with two models, addition of another model led to chattering, thereby indicating slight performance degradation for Case A and other cases as well.

Table 6.2: Statistical summary of control systems managing two classes

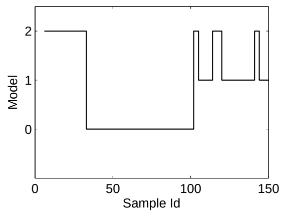
	SSE					MIN					MAX			
Case	MMST2	LCS	MMST4	ACS		MMST2	LCS	MMST4	ACS		MMST2	LCS	MMST4	ACS
A	24.369	32.893	50.108	1357.45		0.349	0.349	0.22	0.377		3.702	3.422	4.539	12.744
B	16.971	15.265	49.199	44.225		0.755	0.724	0.485	0.645		2.935	2.633	4.037	4.719
C	5.973	5.866	8.757	9.359		0.141	0.166	0.089	0.06		1.177	1.148	1.166	1.189
D	19.484	44.811	20.24	103.186		0.308	0.308	0.309	0.202		2.846	3.263	2.512	6.45
E	37.675	507.139	86.099	63.244		0.735	1.114	0.349	0.4		3.628	6.833	5.593	3.38
F	147.107	708.103	7.743	75.787		0.697	1.0553	0.883	0.356		6.638	7.112	2.499	3.777
For cases covered in Appendix E.3														
G	1.009	1.001	1.737	1.109		0.687	0.687	0.472	0.589		1.243	1.243	1.243	1.243
H	32.458	64.130	54.617	59.049		0.598	0.639	0.668	0.645		4.215	4.486	4.182	4.737
I	4.076	4.287	17.936	11.867		0.889	0.889	0.328	0.889		2.436	2.403	2.429	2.883
J	1.079	1.012	2.761	3.091		0.291	0.314	0.282	0.174		0.769	0.761	1.116	0.941
K	0.49	0.629	0.685	0.971		0.381	0.377	0.416	0.422		0.889	0.889	0.95	1.028
L	3.125	3.274	7.743	8.674		0.883	0.883	0.883	0.883		2.585	2.596	2.499	2.834
M	6.471	29.440	13.415	127.708		0.528	0.611	0.349	0.14		1.469	3.326	3.079	9.119
N	124.23	195.980	170.153	216.039		0.242	0.290	0.169	0.141		3.549	3.851	3.885	5.747

Table 6.3: Summary of average loss rates of the control systems managing two classes

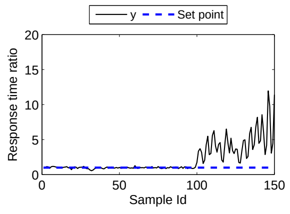
	MMST2			LCS			MMST4			ACS	
Case,Type	$class_0$	$class_1$		$class_0$	$class_1$		$class_0$	$class_1$		$class_0$	$class_1$
D	45	47		45	49		46	47		45	54
E	0	79		8	85		9	71		4	75
For cases covered in Appendix E.3											
I	41	108		41	107		45	103		42	107
J	80	0		80	0		80	0		83	0
K	108	41		108	41		107	42		107	41



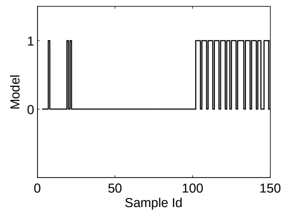
(a) Output with three models



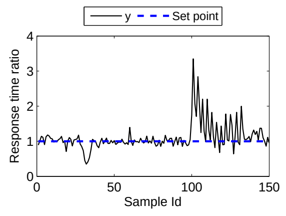
(b) Model switching signal with three models



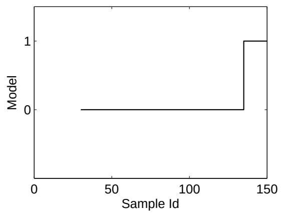
(c) Output with $T = 1$



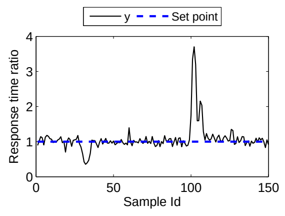
(d) Model switching signal with $T = 1$



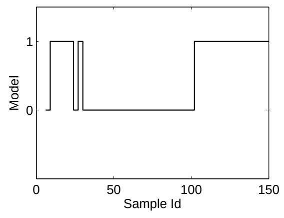
(e) Output with $T = 15$



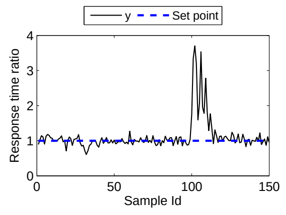
(f) Model switching signal with $T = 15$



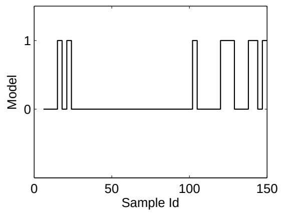
(g) Output with different start-up controller



(h) Model switching signal with different start-up controller



(i) Output with $\alpha = 1$



(j) Model switching signal with $\alpha = 1$

Figure 6.8: Performance of MMST2 with the different parameter settings

6.3.3.2 Impact of Short Finite Time Window

For this experiment, the time window T was set at 1 and 15. Figure 6.8c shows the performance and switching behavior for the case of $T = 1$. Compared to the results in Section 6.3, which used $T = 3$, the performance under low T causes frequent chattering. The performance in *region-0* (after 30th sample) is much better compared to *region 1* (i.e., after the 100th sample). This is mainly because less chattering occurred during the time period from the 30th sample to the 100th sample. However, due to highly discontinuous operating points in *region 1*, controller switching occurs frequently, leading to high steady state error and oscillatory behavior. The high switching period of 15 on the other had delays the selection of the suitable controller, leading to the performance issues (see, Figure 6.8e). However, issues related chattering are significantly less.

The general heuristic is that if the system has fast varying dynamics it is better to have low T, but if it is not that fast varying, relatively high T can be used. The simulation studies may provide a means to evaluate such performance characteristics before setting this parameter in a real system.

6.3.3.3 Impact of the start-up controller

For the experiments in Section 6.3, *controller-0* was used as the start-up controller to provide control till the first switching decision is made by the control system. In this section, we set *controller-1* as the start-up controller to check if it has an effect on the switching behavior initially or afterwards. The performance and switching behavior shown in Figure 6.8g is similar to the results of Section 6.3. Although after brief selection of *controller-0* at the 3rd sample, the controller-1 was used till the 30th sample. However, the overall performance under this setting is identical to the output of Case A of Section 6.3. Therefore, the selected start-up controller has not affected the performance of the control system apart from the initial switching behavior. Based on this result, we conclude that the selection of the start-up controller can be done arbitrarily.

6.3.3.4 Impact of Using Only the Instantaneous Component of the Switching Algorithm (When α= 1 and β = 0)

In this experiment, we only use the instantaneous component of the switching algorithm compared to those in Section 6.3, which used only the long-term component. The output of the system (see Figure 6.8i) illustrates high settling time compared to the performance in Section 6.3. The switching behavior in Figure 6.8j is also different from that of Section 6.3. Especially, at the 100th sample, chattering can be observed causing slight deviations in the output. In addition, there is an inconsistent model switch before the 30th sample. Such inconsistent and unpredictable switching behavior is caused because of the utilization of only the instantaneous component. However, when the long-term component is added to the switching logic, this effect can be removed. The simulations that had $\beta = 1$ and small values of α (=0.1 to 0.3) showed model switching and performance similar to Case A in Section 6.3.

Further evaluations of MMST adaptive control can be found in Appendix F.3 for the case of a system serving three classes. Monte-Carlo simulations are also conducted to check the impact of the simulation settings, on the result presented in this section (see Appendix G). Furthermore, experiment results of MMST adaptive control in a real-world case study can be found in Chapter 8.

6.4 Absolute Performance Management Using Multi-Model Control

In this section, we present a switching control system design technique based on multiple models to improve the performance of the absolute performance management scheme.

From the results of linear MIMO control system in Chapter 5, the controller showed high steady state error when the set points are placed in the sensitive region (see Figure 4.13). In order to capture the behavior in the both insensitive and sensitive regions the linear model was designed limiting the range of the system outputs in a narrow region spanning both insensitive and sensitive regions. As a consequence, in the model estimation and controller tuning processes, trade-offs had to be made to provide satisfactory performance in both regions. For instance, if only one class operates in the sensitive region, a model could be estimated and a controller could be implemented to provide performance for that specific requirement, without estimating the behavior in both regions. Afterwards, these models and the linear MIMO controllers can be combined with a simple rule-based switching algorithm, instead of using the complex switching rules such as in MMST adaptive control. The following subsections provide details of the model identification and switching algorithm design steps.

6.4.1 Model Identification

Depending on the possible requirements, models can be estimated based on the system dynamics in different regions. For instance, Figure 6.9 shows where R_{SLA}, i.e. the reference could be placed in the output regions. The possible cases include, both classes having set points in just insensitive or sensitive regions and one class specifying the set point in the sensitive region, while the other in the insensitive region. Once the required R_{SLA} is decided, the operating output regions can be finalized. Subsequently, models have to be estimated to capture the dynamics in those output regions using the MIMO system identification procedure presented in Section 4.3.6. According to Figure 6.9, four models have to be estimated to cover all possible combinations. However, two models are enough, after $R_{SLA,1}$ and $R_{SLA,2}$ have been defined. Finally, the step responses of these models have to be analysed to check whether these models show similar characteristics. If the models are similar, the number of models can be reduced by removing the similar models.

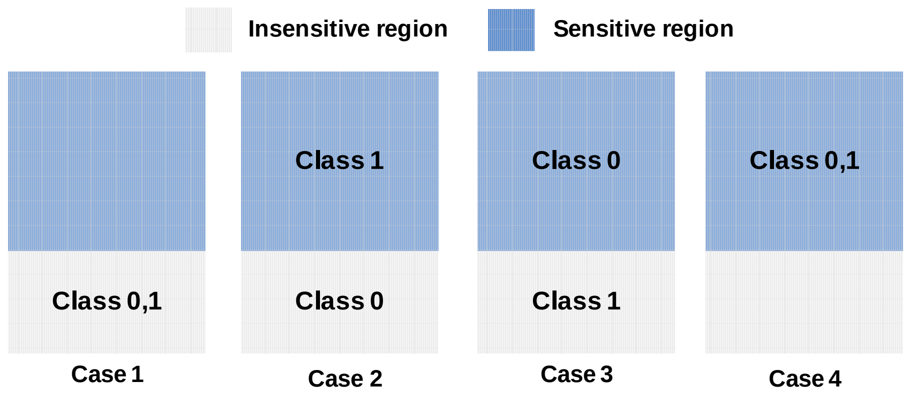


Figure 6.9: Output regions where the set points (R_{SLA}) could be placed for two classes

6.4.2 Switching Control System Design

In order to design the switching control system, the individual MIMO linear MPC controllers have to be developed from the finalized number of models and then tuned after the simulation studies depending on the operating conditions and requirements. It is worth noting that the MPC control and constraint problem formulations are the same as in Chapter 5 for all these controllers. After the design of controllers, the switching rules have to be implemented to switch between the controllers. As mentioned in the previous section, the set point signals are the suitable scheduling variables to make the switching decisions at runtime.

Example: For a system with two classes, let us say two models are estimated for case 1 and case 2 shown in Figure 6.9. Then, two controllers are designed based on those models, namely LMPC$_1$ and LMPC$_2$ respectively. Subsequently, the switching rules are implemented as follows.

If ($R_{SLA,0}(k) < level_1$ and $R_{SLA,1}(k) < level_1$) then execute LMPC$_1$ to compute the input vector of that controller, i.e., $U_1(k)$

Else If ($R_{SLA,0}(k) < level_1$ and $R_{SLA,1}(k) \geq level_1$) then execute LMPC$_2$ to compute the input vector of that controller, i.e., $U_2(k)$.

where, $level_1$ is the variable which divides the output region into insensitive and sensitive regions, $U_1(k) = [u_0(k) \;\; u_1(k)]$ and $U_2(k) = [u_0(k) \;\; u_1(k)]$. $u_0(k)$, $u_1(k)$ are the resource caps of $class_0$ and $class_1$ respectively. Although there are abrupt switching between the controllers when the set point signals change, bump-less transfers can be achieved because the MPC operates with incremental input variable $\Delta u(k)$, similar to the velocity form of PI controller. See, Chapter 5 for the formulation and implementation of MPC.

The above example covers only a limited number of rules, however, depending on the requirements more models and controllers can be integrated to the control system. Even though the design complexity is high at design time, the on-line computational overhead is quite low, because only one controller solves the control and constraint problem at a given time.

6.5 Evaluation of Multi-model Control System for Absolute Management

In this section, we design a multi-model self-managing control system for absolute performance management. We use a system with two classes in this study with the same settings of Chapter 5. Here, we continue with the example covered in the previous section.

Model identification and control system design

The output region of each class is divided into two regions by selecting $level_1$ variable

Table 6.4: Controller parameters of two controllers

Parameter	LMPC_1	LMPC_2
a_1, a_2	0.45, 0.45	0.45,0.55
N_1,N_2	1,1	1,1
R_w	0	1.5
u(0)	$[15\ 15]^T$	$[15\ 15]^T$

as 0.5 (sec). The linear model and linear control system used in Chapter 5 (see equation (4.16)) were used to represent the dynamics in the region below $level_1$. To represent the dynamics in the region above $level_1$, another model was identified (see equation (6.6)). When the models in equation (4.16) and equation (6.6) are compared there are differences in the pole locations and step responses.

$$y(k+1) = \begin{bmatrix} 0.4730 & -0.0130 \\ -0.0614 & 0.6650 \end{bmatrix} y(k) + \begin{bmatrix} 0.1400 & -0.0053 \\ -0.0534 & 0.1188 \end{bmatrix} u(k), \tag{6.6}$$

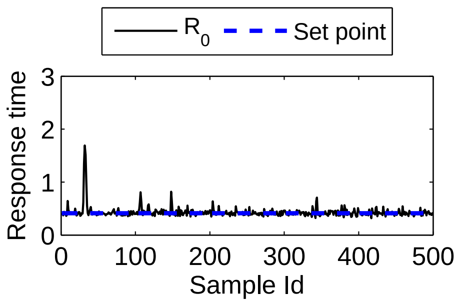
Then, two MIMO predicative controllers were designed based on the methodology presented in Section 5.4. Table 6.4 lists the tuning parameters of the two controllers called as LMPC_1 and LMPC_2. a_1 and a_2 are the poles of Laguerre network, while N_1 and N_2 are the number of terms in the network. The main difference to the controller designed in Section 5.4 is that LMPC_1 is operating with much higher gain (R_w). This would reduce the overshooting when the set points are placed in the insensitive region. In contrast, LMPC_2 operates with a lower gain (R_w), in order to improve the steady state behavior in the sensitive region.

Finally, the following switching logic was implemented in the control system.

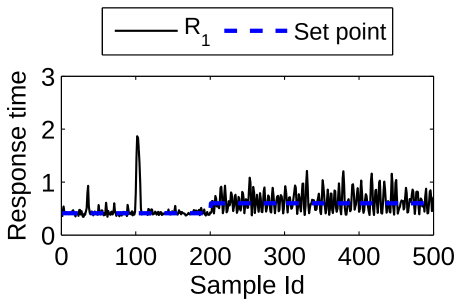
If ($R_{SLA,0}(k) < 0.5$ and $R_{SLA,1}(k) < 0.5$) then execute LMPC$_1$ and implement $U_1(k)$
Else If ($R_{SLA,0}(k) < 0.5$ and $R_{SLA,1}(k) \geq 0.5$) then execute LMPC$_2$ and implement $U_2(k)$.

Now, we evaluate the absolute performance and resource management capabilities of this new multi-model self-managing control system (namely, MMMPC). Here, we use combined operating conditions of Case A and C described in Chapter 5 to evaluate absolute performance management. We start-off with Case A, where references of two classes are set to 0.41 seconds (i.e., the references are placed according to case 1 in Figure 6.9). Then, at the 200th sample reference of $class_1$ is increased to 0.6 seconds, while workload settings of Case C is applied on the system (i.e., references are placed according to case 2 in Figure

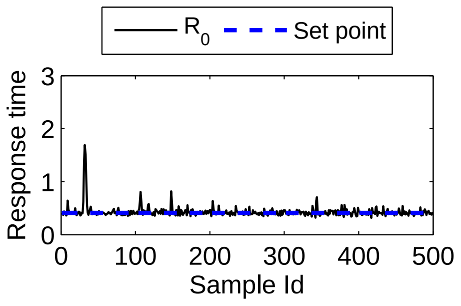
6.9). These operating conditions provide an effective way to validate the capability of the multi-model self-managing control system in different regions (see Table 6.5 for statistics). Figure 6.10 shows the performance of the control systems. This figure also includes the outputs of individual LMPC$_1$ and LMPC$_2$ for comparison purposes.



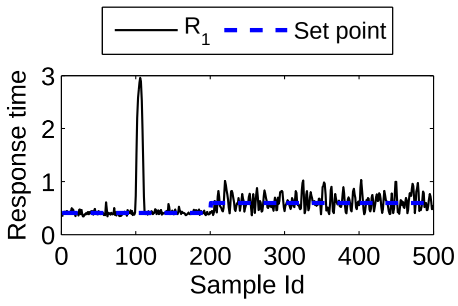
(a) Output R_0 of LMPC$_1$



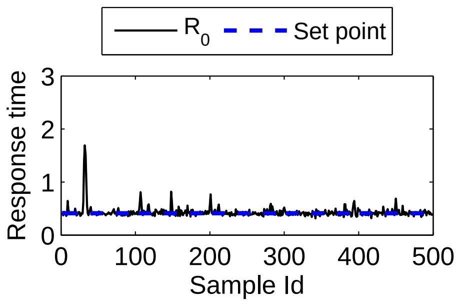
(b) Output R_1 of MIMO LMPC$_1$



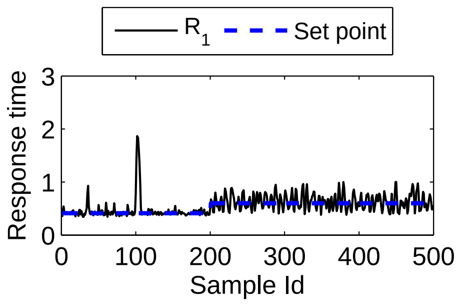
(c) Output R_0 of MIMO LMPC$_2$



(d) Output R_1 of MIMO LMPC$_2$



(e) Output R_0 of MMMPC



(f) Output R_1 of MMMPC

Figure 6.10: Performance management of multi-model self-managing control system for absolute management

When LMPC$_1$ performance is compared with the other control systems, it is evident that output of $class_1$ has higher steady state error after the 200th sample. In contrast, performance of LMPC$_2$ shows larger overshooting due to the disturbances of Case A before the 200th sample. The MMMPC on the other hand combines the performance of these

Table 6.5: SSE statistics of the control system at different set points

	R_0(SSE)	R_1(SSE)
$R_{SLA,0} = 0.41$, $R_{SLA,1} = 0.5$		
LMPC$_1$	5.718	92.937
LMPC$_2$	19.464	122.723
MMMPC	5.884	91.668
$R_{SLA,0} = 0.41$, $R_{SLA,1} = 0.6$		
LMPC$_1$	5.71	143.222
LMPC$_2$	19.395	163.544
MMMPC	5.921	134.247
$R_{SLA,0} = 0.41$, $R_{SLA,1} = 0.7$		
LMPC$_1$	5.76	207.291
LMPC$_2$	19.288	220.986
MMMPC	5.859	192.302

control systems and provides better performance under both cases. In addition, Table 6.5 summarizes the SSE statistics of both output signals, when the reference signal of $class_1$ is set at different levels. It is evident that when the set point signal of $class_1$ is close to the insensitive region the improvements obtained by MMMPC is small (e.g., 0.5), compared to the cases where the set point is placed further away from the insensitive region (e.g., 0.7). Another observation is that SSE for R$_0$ signal is slightly high in MMMPC compared to $LMPC_1$. This is because, at the 200th sample LMPC$_2$ is integrated to the control system abruptly by the MMMPC, which has the start-up settings. As a consequence, there is a deviation for the case of R$_0$ from the reference value compared to the stand-alone $LMPC_1$. However, even with such abrupt switching, the steady state behavior has not affected significantly and MMMPC settles down fast without showing any instabilities. Furthermore, because of the simple switching mechanism, chattering will not occur based on the changes in operating conditions. This is because switching will only happen if the references of the classes are changed by the system operator.

In addition, it is worth noting that this multi-model approach shows relatively poor performance compared to the nonlinear MIMO Wiener model based control. However, this is simple and intuitive approach compared to the MIMO Wiener control.

6.6 Conclusion

This chapter has presented new approaches based on the multiple models, controllers combined with switching capabilities to manage performance and resources of multi-class shared resource systems.

Firstly, the relative management system was formulated with two different regions and then two models were estimated, subsequently two controllers were designed to provide control in each region. A switching scheme was then implemented based on MMST adaptive control to select the most suitable controller at runtime. This is the first time such an approach has been used in the software performance management literature. The evaluations comparing the MMST-T2 and T4 schemes with traditional linear and adaptive control have indicated successful performance of MMST-T2 control system for a limited set of cases. However, in some cases chattering was observed, which is an inherent issue that exist in the switching control systems. Chattering is caused by the noise in the output signals, which leads the control system to falsely detect the change of operating conditions and subsequently switch to an inappropriate controller at runtime. If we compare the performance of MMST-T2 control system with the Hammerstein-Wiener control system, former provides better performance in Cases H, I and M, while the later provides better performance for the other cases. From the above evaluations, the overall performance of the Hammerstein-Wiener control system is much better and stable compared to MMST-T2 control system. However, MMST-T2 provides satisfactory performance when noise does not affect the output signals, and it is easy to design and develop compared to the Hammerstein-Wiener control system, which requires nonlinear system identification methods.

Secondly, the absolute performance and resource management problem was also formulated by a multi-model and multi-controller based control system with simple switching logic. The reference signal (R_{SLA} specification) was used as the switching decision variable, which provided effective way to select the appropriate controller based on the operating region. The evaluation results also indicated improvements in performance and resource management compared to an individual linear control system.

The main limitation of the proposed multi-model switching control approaches is the design complexity compared to a single linear control system. In both proposed approaches, multiple models and controllers have to be designed which is a manual design task. In addition, there is a slight computational overhead compared to a linear control system due to the requirement of evaluating the switching logic at each sample interval. Furthermore, as observed, chattering is a vital problem at runtime, which may lead to temporal instabilities or large transient responses. In order to mitigate this issue, the selection of the parameters of the switching algorithm has to be performed carefully by using simulation studies. A set of heuristics are also provided in this chapter, which would be useful in tuning of MMST adaptive control schemes.

Chapter 7

Support Tools to Build Control Systems for Software Environments

7.1 Introduction

In the previous chapters, we presented a set of nonlinear control engineering techniques to implement performance management systems for multi-class shared resource environments. In order to build adaptive software systems based on such control engineering methodologies, a designer needs to construct and test a range of models and associated controllers and then integrate these components into the software system. This is not a straight-forward task because the background knowledge required of the software engineer is substantial due to the need for rigorous mathematical foundations of control theory [265]. This skills barrier combined with the complexity of the task means that the engineering cost of control systems is high. These barriers to building self-adaptive software systems could be lowered if the software engineers had well tested and extendable implementation tools and frameworks [203]. In this chapter, we describe a process for developing basic or self-managing control systems with the support of an extendable and configurable off-the-shelf class library implemented based on a reference model.

The class library basically provides control components to implement a range of different control regimes including PID, predicative and self-tuning controllers. These control components can be also used to implement the nonlinear Hammerstein-Wiener and Multi-Model Switching control systems with little implementation overhead. Furthermore, tailor made control systems for a specific software system can be implemented by using a combination of control components included in this class library. Consequently, the proposed step by step engineering process and the class library reduce the engineering costs and abstract away some of the knowledge required in building control systems for software systems.

We start-off by introducing an engineering process for control system development in Section 7.2. Section 7.3 presents the reference model and design of the class library. Section 7.4 covers example use cases to illustrate how the engineering process and class library can be used to design self-managing control systems. Section 7.5 evaluates the extensibility, configurability and the facilitation provided by our class library.

7.2 An Engineering Process for Developing Self-managing Control Systems

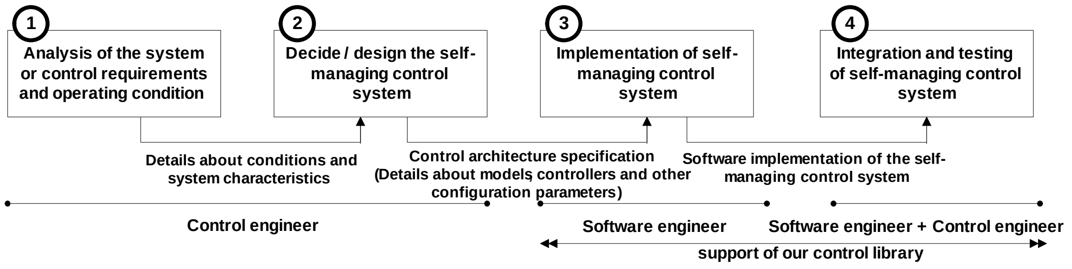

1
Analysis of the system or control requirements and operating condition
2
Decide / design the self-managing control system
3
Implementation of self-managing control system
4
Integration and testing of self-managing control system
Details about conditions and system characteristics
Control architecture specification (Details about models, controllers and other configuration parameters)
Software implementation of the self-managing control system
Control engineer
Software engineer
Software engineer + Control engineer
support of our control library

Figure 7.1: The design process

Figure 7.1 illustrates the steps involved in the design process of the self-managing control system. Similar engineering processes are followed in other industries such as chemical and manufacturing [73]. It is also worth noting that, in this section we mainly focus on the implementations of self-managing control systems introduced in Chapter 6, because such control schemes require different complex control components (e.g., adaptive models, controllers and switching schemes). Although the emphasis is given to complex self-managing control systems, the same steps can be followed to implement the basic control systems as well. The implementation of a control system based on the formal control engineering techniques requires a wide range of knowledge in that discipline, which the software engineers are not generally familiar with. As such, the first two phases of the design process require the skills of a control engineer.

The first phase is to establish the control requirements of the system. Here, the details of control objectives, system requirements, and the environmental conditions need to be investigated in order to identify the distinct operating regions of the target system. In addition, data traces from the production systems or special offline experiments need to be conducted to collect data. The objective of phase 2 is to determine the suitable control architecture. This involves the analysis and investigation of the control objectives together with the collected data to determine the type of control system required for the particular software system. Subsequently, the components required to build a control system have to be identified. For instance, if it is a MMST scheme, the mathematical models to capture the behavior in different operating conditions have to be estimated followed by the design of the controllers.

The third phase is to implement the control system as software components, following the software engineering principles. However, a control engineer typically does not have enough expertise to do this implementation. Similarly, a software engineer does not have enough expertise to engage in the previous two steps. Using our class library, this knowledge gap can be reduced. The design specification document (namely, *control architecture specification*) with a particular self-managing control scheme, parameters of the models and controllers will be given to the software engineer by the controller engineer after the second stage. Given the control architecture specification, the software engineer can use our class library and create and configure the required models and controllers with the specified parameters. In addition, if the required control scheme is supported off-the-shelf by the class library, that scheme can be used without any new implementation effort. Otherwise, an existing scheme can be extended or a new scheme can be implemented using the basic control components provided by our class library.

In the fourth and final phase, the implemented control system has to be connected to the sensors and actuators of the software system, followed by rigorous testing with simulations of desired operating conditions to validate the suitability of the implementation. This step needs to be conducted by the control engineer in conjunction with the software engineer. After the test and validation of the control system, the final fine tuning of the configuration parameters of the self-managing control scheme has to be done. If the designed system fails to achieve the control objectives, the control engineer should go to the first stage again and redesign an appropriate control scheme.

7.3 Reference Model and Class Library

This section presents the reference model used as the basis to build our class library and its implementation details.

7.3.1 Reference Model of Class Library

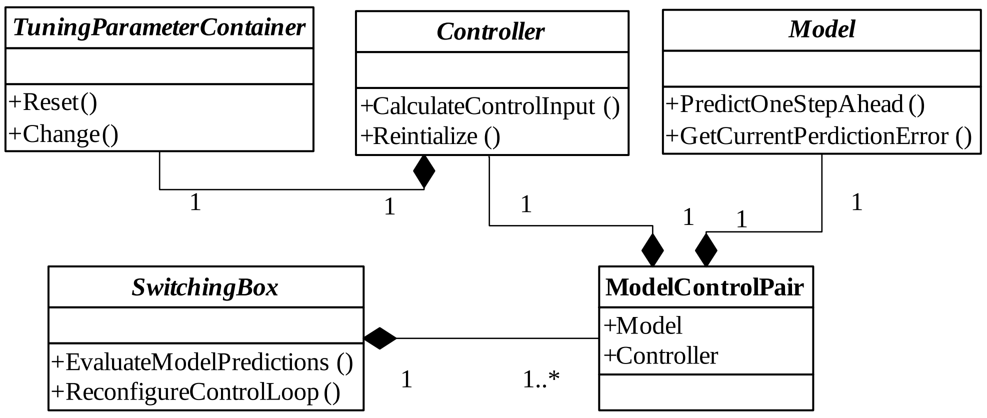


Figure 7.2: The reference model

Figure 7.2 shows the class diagram of the proposed reference model for the class library in Unified Modelling Language (UML). The *Model* is an abstract class which represents the behavior of the target system. Typically, it is represented using an ARX model. The abstract methods such as *PredictOneStepAhead* and *GetCurrentPredictionError* are implemented according to the model structure and parameters. This component will be especially useful to implement multi-model self-managing schemes presented in Chapter 6. The *Controller* is also an abstract class representing the control algorithms that need to be implemented. The *CalculateControlInput* method has to be extended to implement the specific control algorithm, which will return the control input for the current sample instance. Furthermore, the sensor variables and actuator variables have to be connected to this component. The *TuningParameterContainer* is another abstract class which is used by the *Controller* class to represent the tuning parameters (e.g., gains) of the controller. It helps to abstract away the different configurable parameters of different controllers from the main implementation of the control algorithm. It also provides additional convenience to make the reconfigurations of the tuning parameters of the controller at runtime without affecting the state of the controller. The *ModelControlPair* class associates the models representing different conditions and the respective controllers designed to provide control under those conditions. It is particularly useful to implement multi-model self-managing control systems.

The *SwitchingBox* class is the main component which implements the reconfiguration decisions of the control loop. For instance, in order to implement the MMST adaptive control schemes, *EvaluateModelPredictions* method in this class can be extended to evaluate the model predications according to the model evaluation algorithm and select the best model based on the model selection criteria. The *ReconfigureControlLoop* method on the other hand can be used to implement the control loop reconfiguration logic required by a control system at runtime.

7.3.2 Implementation of the Class Library

A major contribution of this chapter is the implementation of the above reference model with a rich set of standard control components and algorithms that can be effectively used in the implementations of different types of control systems for software systems. Figure 7.3 shows the UML class diagram of the library in detail.

The current implementation provides the following off-the-shelf standard control components:

Control algorithms and schemes

1. PID controller (algorithm [81])
2. Model predictive controller (algorithm [223])
3. Indirect self-tuning regulator (algorithm [13])
4. Self-tuning PID controller (algorithm [81])
5. All four MMST schemes (algorithm [168])

Models

1. First order and Higher order models
2. Adaptive model (recursive least square algorithm [13])

This class library is currently implemented using *Java* and *C#.Net*[1]. It can readily

[1]available from `http://www.ict.swin.edu.au/personal/tpatikirikorala/dowloads/SwinMMSTFramework_dll.zip`

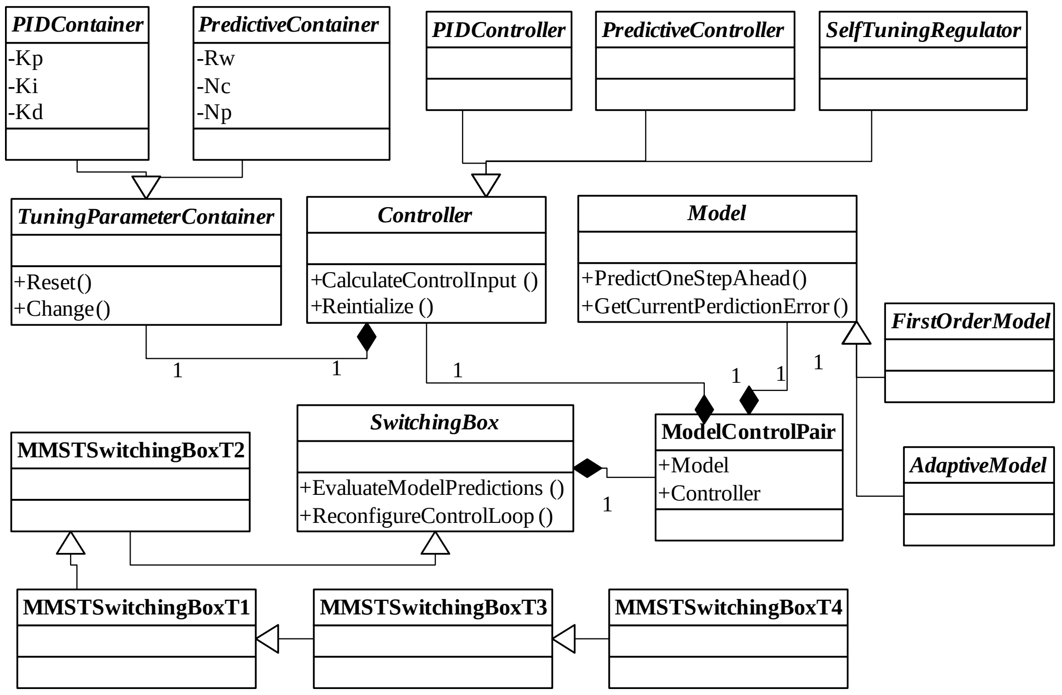


Figure 7.3: UML class diagram of the library

be integrated into a software system implemented in Java or any of the programming languages supported by the *.Net* framework.

7.4 Example Use Cases

In this section, we illustrate how the above class library and the engineering process introduced in Section 7.2 can be used to implement two different self-managed control systems for relative performance management (similar to Chapter 6). Here, we assume that the first two steps of the design process have been already carried out by the control engineer. As shown in Chapter 6, these two initial steps include identifying the models (*model-0* and *model-1*), deciding the tuning parameters of the controllers (*controller-0* and *controller-1*) and configuring the switching algorithm. The next step involves implementation of the control system using software components. For this purpose, our class library can be used. However, in order to show different use cases of the library, let us assume that the control engineer decided to implement and test two types of self-managing schemes, one based on MMST-type 2 scheme (same as in Chapter 6) and the other based on a gain-scheduling scheme equipped with a model predictive controller. Let us further assume that the details of the components and parameters have been documented as a control architecture specification by the control engineer at the end of the second stage and then given to the software engineer in order to carry out the third stage of the design process. The main contents of the control architecture specification are summarized in Tables 7.1 and 7.2.

For the first use case, the software engineer can use our control library directly to implement the required MMST-Type 2 scheme. This is because, as shown in Figure 7.3, the class library provides off-the-shelf implementations of all MMST schemes. The steps involved in this implementation are 1) initiating two first order ARX model instances with the model parameters as specified in Table 7.1, 2) initiating two instances of PI controllers conforming to the specified tuning parameters, 3) pairing the relevant models and controllers in a collection of *ModelControlPair* instances, 4) initiating *MMSTSwitchingBoxT2* instance with the parameters of the MMST-Type 2 scheme and finally 5) connecting the sensors and actuators to the instance of *MMSTSwitchingBoxT2* class. The next step is to test the implementation under different operating conditions.

In order to complete the second use case of implementing a gain-scheduling scheme with a MPC, the software engineer has to extend our class library. Such schemes are not offered off-the-shelf by the class library, because it is hard to generalize the logic and the variables involved in the gain-scheduling schemes. There are two ways to implement this scheme using the class library. The first way is to extend the *SwitchingBox* class and implement the model prediction evaluation and selection algorithms and the reconfiguration logic of MPC from scratch. However, this is unnecessary because *MMSTSwitchingBoxT2* class already provides some of the required functionalities. Alternatively, the software engineer can override the *ReconfigureControlLoop* method in the *MMSTSwitchingBoxT2* class and implement the new functionalities of the gain-scheduling logic shown in Table 7.2, while reusing the rest of the functionalities provided by that class as is. Therefore, with a little extension, the new control architecture can be implemented without a major engineering overhead.

7.5 Evaluation

In this section, we discuss the configurability and extensibility of our class library and evaluate the facilitation provided by it in the development of a complex control system.

Table 7.1: Control architecture specification for MMST-T2 scheme

Switching algorithm
$T = 3$, α=0, $\beta = 1$ (see equation 6.3,6.2) ,
Startup controller - Controler-0
Models
Two first order ARX models for region 0 and 1
Model-0 : y(k+1)= 0.63 y(k)+ 0.14u(k)
Model-1 : y(k+1)= 0.64 y(k)+ 0.96 u(k)
Controllers
Two PI controllers in velocity form, for region A and B
Controler-0 :$K_p = 0.84$, $K_i = 0.42$
Controler-1 : $K_p = 0.24$, $K_i = 0.06$

Table 7.2: Control architecture specification for gain scheduling scheme

Switching algorithm
$T = 3$, α=0, $\beta = 1$ (see equation 6.3,6.2),
Startup controller settings- Setting 0 (see below)
Models
ARX models for region 0 and 1
Model-0 : y(k+1)= 0.63 y(k)+ 0.14u(k)
Model-1 : y(k+1)= 0.64 y(k)+ 0.96 u(k)
Controller parameters
Single MPC with tuning parameters
Setting 0: R_w =0.5, N_p =10, N_c=5
Setting 1: R_w =8.0, N_p =10, N_c=5
Gain scheduling rule/logic
At each T^{th} time instance
If (model 0 is selected) change MPC parameters to Setting 0
Else change MPC parameters to Setting 1

7.5.1 Configurability

The off-the-shelf implementations of the controllers, models and the self-managing control schemes offered by the class library can be configured with the required design parameters by populating the exposed properties of each control component. As shown in the first use case of Section 7.4, when the control engineer comes up with the tuning parameters of the controllers, models and self-managing control schemes, the software engineer has to select the suitable classes and instantiate them by populating the properties according to the specifications of the control engineer. Thus, the engineering and testing effort and the mathematical knowledge required of the software engineer can be minimized to a large degree, compared to the implementation of these components from scratch.

Although this thesis focuses on more complex self-managing control schemes, the basic control components available from the library (see Figure 7.3) could also be readily used to create simpler control schemes if deemed appropriate by the control engineer.

7.5.2 Extensibility

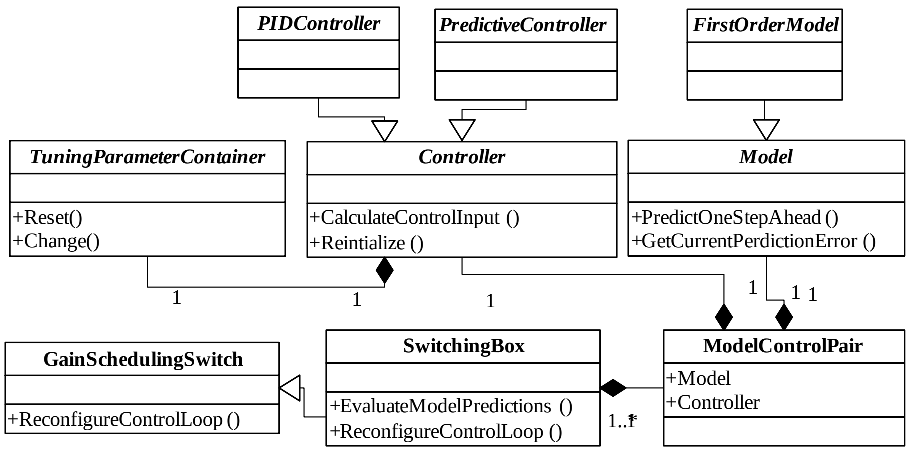


Figure 7.4: Gain-scheduling scheme with a predictive controller extending the reference model

The reference model and the associated class library can also be extended to implement different types of self-managing control schemes. Figure 7.3 shows many different extensions performed to the reference model, in order to implement the class library. In addition, Figure 7.4 shows how this reference model and class library can be further extended to implement the multi-model gain-scheduling control system described in Section 7.4. The *GainSchedulingSwitch* class either can extend the SwitchingBox class as shown in Figure 7.4 or extend *MMSTSwitchingBoxT2* class to implement the required gain-scheduling scheme. Alternatively, to implement other gain-scheduling schemes with different scheduling variables, the required scheduling logic can be implemented by extending the *SwitchingBox* class and adding reference to the runtime data of the scheduling variables.

This class library provides many degrees of freedom for extensions. The controllers, models and self-managing control schemes that are not provided by our class library can be plugged-in and used without affecting the rest of the implementation. For example, if a linear quadratic controller [81] is needed, the implementation of that controller can be done, extending the *Controller* class. Similarly, if a nonlinear neural network model is needed, the same approach can be taken to enrich the class library by extending the *Model* class. Even with these extensions, the existing self-managing schemes can be configured to use the new components without affecting their implementations. Similarly, if new switching schemes have to be implemented the *SwitchingBox* class can be extended. For instance, the gain-scheduling configuration, which is not offered by the class library off-the-shelf, can be implemented with little engineering effort by extending the available components. In addition, any of the controllers available from the library can be used to design self-managing control schemes by mixing the capabilities of different control algorithms. For example, PID controllers and model predictive controllers can be combined in a scheme depending on the requirements, operating conditions and constraints.

7.5.3 Facility

Quantifying the extent to which a software framework, such as the class library presented in this chapter facilitates software engineering practice is a challenging task. The source lines of code (SLoC) is one of the basic proxy measures of the engineering effort [40]. The .Net version of class library consists of 16,000 SLoCs, which do not have to be implemented again in order to design a self-managing control system for a software system. However, SLoCs do not measure either the knowledge required of the engineer, the complexity of the task or the subsequent time taken to come up with a design and implementation.

To further evaluate the extent of facilitation, we conducted an experiment with a group of software engineers, in order to capture the knowledge requirements, time and effort needed to implement a self-managing control scheme. The task was to implement the self-managing control architecture shown in Figure 6.2a, given the parameters in Table 7.1. All participants were software engineers with more than 2 years of experience of developing production software systems.

This experiment was conducted in three stages. In the first stage, each participant received a control architecture specification describing the control system to be implemented. They then participated in an oral presentation delivered by a control engineer and the business analyst, which described the details of the control architecture specification followed by a question and answer session. In the second stage, the participants were familiarized with the experimental setup by preparing the workstation of the participant with the experiment instrumentation. The instrumentation included a simulation of a target software system with a sensor and actuator, graphing tool (to check the output of the target system), a test case, the class library with the documentation and a time sheet to enter the start time and end time of tasks.

After setting up the environment the third and main stage of the experiment was performed. Each participant had to implement the specified control system in two ways: 1) to implement it from scratch (called T1) and 2) to implement it using our class library (called T2). The order in which these tasks were undertaken was altered between participants. This is done in an attempt to account for any learning effect that would allow a participant to complete the second task in a quicker time than they would have done without prior experience of the problem and a solution. An implementation was regarded as complete when the test case ran successfully. During the third stage a control engineer was available to ask any question related to control theory or control architecture specification. These questions were collected during the experiment for a qualitative analysis. Apart from the standard documentation, no additional information related to our class library was given to the participants. After the completion of the experiment, the time sheets and the source code implemented by the participants were collected for further analysis.

Table 7.3: Results of the experiment

ID	Experience (years)	order	Time (Mins)		SLoCs	
			T1	T2	T1	T2
1	4	T1-T2	78	47	163	109
2	2	T2-T1	107	55	397	95
3	3	T1-T2	67	23	169	93
4	3	T2-T1	91	48	326	125
5	2	T1-T2	130	43	189	92
6	5	T1-T2	180	35	233	87
7	2	T2-T1	115	45	283	108
8	3	T2-T1	199	63	314	121
9	3	T2-T1	142	48	219	108
10	3	T1-T2	184	65	152	98
11	9	T1-T2	119	34	179	93
12	4	T2-T1	142	56	311	137
13	5	T1-T2	167	39	222	92
14	4	T1-T2	158	40	217	97
Avg.			134	46	241	104
Diff.			**88**		**137**	

Table 7.3 summarizes the findings of the experiment. The prominent observation is that the time taken to complete T2 (using the library) is low compared to T1, irrespective of the order in which the tasks were done. In addition, in all cases the SLoCs written were lower for T2 than T1. However, while SLoCs required for T1 varied drastically, the SLoCs for T2 across participants was much more consistent. The analysis of the code indicated that SLoCs were high for some participants (e.g., P2) because the implementation was extendable and configurable compared to other implementation which lack those characteristics (e.g., P3). It is also interesting to note that, the design for T1 produced by the participants who did the implementation with the class library (T2) first, showed the high-level design characteristics of our class library. This is an indication that there is an effect on the design depending on the order of the tasks. The time consumed for both T1 and T2 on the other hand varied a lot from the participant to participant. This is because of the speed of coding, debugging and testing varied between the participants. In summary, on average 88 minutes in time and 137 SLoCs in implementation effort have been reduced when the class library is used compared to implementing the entire task from scratch.

Apart from the above time and source code related comparisons, we also analysed the questions asked of the control engineer during the implementations, to get an idea of the knowledge requirements for control system implementations. From the collected questions in the case of no library support (T1), the control engineer had to engage in the testing of the implementation by answering many questions till the test case was successfully run. Thus, the test and validation cost was significantly high in T1. In addition, the lack of understanding of the sampling time caused selection and retention of inappropriate data in the models and the switching algorithms leading to erroneous outcomes. Furthermore, differentiating the system output, input, and controller input, output and the model prediction and the original system output was an issue for some participants.

In contrast, in the case where the library was used (T2), none of the above problems were observed in relation to the mathematical background of the implementations. This is because such details are encapsulated inside the control components. The documentation provided the required information on these components and how to use them in an implementation. Some questions were asked about the technical terms used in the documentation with respect to the terms in the control architecture specification.

Although the scheme that the participants had to implement in this experiment was one of the simplest self-managing control systems, the findings of this experiment indicate the use of the library facilitated the assigned task by lessening the time, effort and knowledge requirements. For schemes with higher complexities and mathematical foundations (e.g. adaptive, MPC control) we can speculate that even greater benefits would accrue from using this class library.

7.6 Conclusion

To build self-adaptive software systems based on the control engineering methods proposed in this thesis, the software engineers have to design, implement and test complex control components from scratch. This is a challenging task because the background knowledge required of the software engineer is substantial due to the need for rigorous mathematical foundation behind the control components. In order to reduce the engineering costs of control systems for software systems, in this chapter we have put forward an engineering process and a class-library. This class library provides many standard control components useful to implement deferent control regimes and it can be easily configured and extended to accommodate new requirements. An empirical study conducted with a group of software engineers has shown that this library facilitates the implementations of control systems significantly compared to implementing such systems from scratch.

Chapter 8

Experimental Case Studies

8.1 Introduction

This chapter shows the applicability of the nonlinear control approaches presented in this thesis to the real-world multi-class shared resource systems. In particular, we study two different multi-class shared resource software environments sharing resources at the middleware and application levels of the system stack[1] (see Chapter 1). These studies therefore help us to demonstrate the management capabilities of the proposed approaches across different levels of the system stack. These systems are built using well-established programming languages and software components. We introduce the business scenarios and objectives briefly below, before going into the details of the system architectures and experimental results. Firstly, we cover WSO2 BPS which is a shared middleware environment. Secondly, the details of a travel reservation system are presented which shares resources at the application level.

WSO2 BPS. WSO2[2] is one of the leading open-source enterprise software platform providers, offering services to companies such as ebay[3]. From their product suite, WSO2 Stratos business process server (BPS)[4] is a multi-tenanted middleware environment which provides out of the box support to deploy software workflows for multiple clients (or tenants) using a single instance. It is a shared-middleware system, in terms of the definitions

[1]Another case study, investigating the hardware level resource management capabilities of proposed approaches can be found in [192]. A VM environment built based on the RUBiS benchmark application is used in that case study, with the objective of managing response times of two RUBiS applications by controlling CPU capacities of two VMs at the hypervisor level.

[2]http://wso2.com/

[3]http://www.ebay.com/

[4]http://wso2.com/cloud/stratos/

in Chapter 1. WSO2 BPS already provides data, security and execution isolation, however, performance isolation is not so far provided [187]. In this work we extend the WSO2 BPS to manage performance of multiple customers while sharing the resources effectively at runtime. More details are provided in Section 8.2[5].

Travel reservation system. The travel reservation systems depend on Global Distribution Systems (GDS) which have information about the flight availabilities and their tax and fare rules. There are different GDS providers dominating in different parts of the world. Amadeus, Sabre and Galileo are some of these leading large scale GDS providers in the business today. The information services of a particular GDS can be used by the travel solution development organizations to build travel solutions for travel agencies. The communication between the GDS and travel solution is enabled by communication sessions. Only a limited number of sessions is available for each travel solution due to the costs involved in purchasing them. The travel solution provider exposes a standard web service interface which can be used to design the travel reservation websites for travel agencies. This web service platform, maintained by the travel solution provider is a multi-class shared resource system. As a consequence, the resources available at the level of web service platform have to be shared by the travel solution provider to maintain the performance properties of the travel agent websites. The main bottlenecked resource in this system is the sessions provided by the GDS[6].

In the following subsection, we use WSO2 BPS case study to present the applicability and experimental results of the approaches proposed for relative performance management. Similarly, the travel reservation case study is used to show the applicability and experimental results of the approaches proposed for absolute performance management.

8.2 Relative Performance Management at the Middleware Level

In this study, we use the WSO2 Stratos Business Process Server to share and control resources at the middleware level to achieve relative management objectives. WSO2 Stratos Business Process Server[7] is a multi-tenanted workflow engine which executes busi-

[5]We would like to thank Mr. Waruna Ranasinghe and Mr. Denis Weerasiri from WSo2 for collaborating in this work.

[6]We would like to thank Mr. Dharshana Batagoda (senior architect at GoQuo) and Mr. Kushan Chathuranga (senior software engineer at CodeGen) for assisting us in the development of this scenario.

[7]`http://wso2.com/products/business-process-server/`

ness processes compliant with WS-BPEL standard, and is built on top of WSO2 Carbon platform[8]. WSO2 Stratos BPS also supports data and execution isolation for multiple tenants [187]. Figure 8.1 shows its high level architecture. Tenant administrators and authorized users can manage and monitor the business process deployments and business process instances via the graphical administrative console. A user of a tenant can consume a business process via a business process endpoint, which is a standard web service endpoint. *WSO2 Stratos Identity Server (IS)* provides security services such as authentication and authorization of tenants and users. *WSO2 Manager* is used to provision and manage tenants, including their subscriptions and billing. Business process artefacts for each tenant are kept in *WSO2 Stratos Governance Registry* which is a multi-tenanted governance tool that follows the shared database and shared schema multi-tenanted data architecture pattern defined in [43]. WSO2 Stratos BPS uses Apache ODE[9] as its BPEL execution run-time. ODE-Axis2 Integration Layer provides three main services: i) BPEL process and process instance management, ii) tenant-aware request dispatching, and iii) communication with partner services defined in a BPEL process. Integration Layer is also responsible to expose a BPEL process as an Axis2[10] Web Service. In the current multi- tenanted BPS instance, a single ODE process engine is shared by multiple tenants. Therefore, a workload of a single tenant may adversely affect the performance of other tenants. Consequently, a mechanism is required to manage the performance, resources and workloads of different tenants in a single BPS instance, which is the focus of this study.

8.2.1 Implementation

In order to enable the performance and resource management in the BPS many modifications had to be done. In particular, tenant-aware queues, response time monitors/sensors for each tenant and resource partition scheduler were integrated. Figure 8.1 shows the architecture of the BPS after these modifications, which corresponds to the standard multi-class shared resource system architecture presented in Chapter 3. The *Tomcat transport layer* receives the requests from the users of tenants, and forwards the requests to *Axis2 message handler chain*. Upon processing the request in the handler chain, an Axis2 message context is created, and the information about the tenant (so-called tenant

[8]`http://wso2.com/products/carbon/`
[9]`http://ode.apache.org/`
[10]http://axis.apache.org/axis2/java/core/

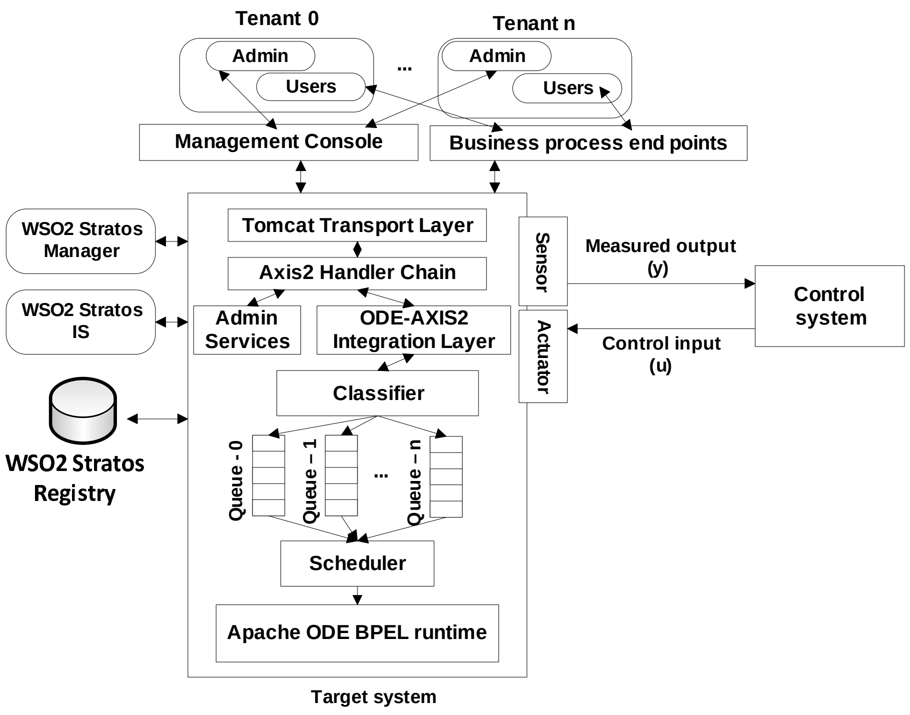


Figure 8.1: Block diagram of WSO2 BPS

domain) from the request is used to identify the corresponding BPEL process. When *ODE-Axis2 Integration Layer* receives an Axis2 message context, the message context is classified based on the available tenant information and puts it into the message queue corresponding to the tenant. The thread that processed the request waits until a notification of the result is available, in order to send back the response to the client. The management system informs the *Scheduler* via the actuator about the process instance caps for each tenant. Here, the scheduler takes in to account the process instance caps (S_i, $i = 0, 1, \ldots n-1$) and current usage to schedule the requests from each tenant's queue to be sent to ODE runtime. In addition, the average response time of requests in a 2 seconds sample window is calculated by the sensor for each tenant (R_i, $i = 0, 1, \ldots n-1$) and sent to the management system. The response time of a request includes the waiting time in the tenant's queue and execution time in the ODE runtime.

8.2.2 Experiment Setup

Here, we consider a BPS with two tenants ($n = 2$). The BPS and database was deployed in a virtual machine (VM) with two 2.67 GHz CPUs and 3 GB memory. We used the LoanProcess[11] as the deployed business process for each tenant, which invokes three partner services sequentially. The workload generators and partner web services were deployed in two VMs each with a 2.67 GHz CPU and 2 GB memory. After initial profiling the maximum concurrent process instances S_{total} was set to 20. Although higher S_{total} increases the throughput, the response time was significantly affected as well (e.g., $S_{total} = 30$ increased response time around 100 ms). In addition, $S_{1,min}, S_{2,min} = 4$.

8.2.3 Hammerstein-Wiener Control System Design

This section gives the design details of the Hammerstein-Wiener control system, including the two compensators and controller to achieve the relative performance management objectives. We follow the design process presented in Chapter 4 and 5.

Firstly, to design the pre-input compensator the possible operating points for u were calculated as $\frac{4}{16}, \ldots, 1, \ldots, \frac{16}{4}$. Then, the points of the intermediate variable v were selected as values $-6, -5, \ldots - 1, 0, 1 \ldots 5, 6$ by setting $\delta v = 1$,$v_{min} = -6$ and $v_{max} = 6$. Following the design process in Chapter 4, a fourth order polynomial was used in the estimation of the inverse input nonlinear function (see equation (8.1)) with a goodness of fit of 0.99. Figure 8.2a shows the model fit. This function was then implemented as a software component/compensator and integrated into the BPS.

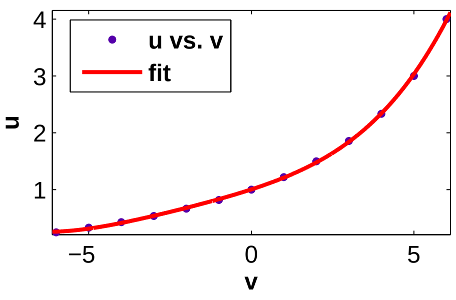
$$u = f^{-1}(v) = 0.0003828 * v^4 + 0.003445 * v^3 + 0.01722 * v^2 + 0.1857 * v + 1.006 \quad (8.1)$$

With the integration of the input static nonlinear compensator, the next step is to design the output nonlinear compensator. Following the design process in Chapter 4, a sinusoidal signal was designed with possible values of v and 40 requests/sec workloads were applied for each tenant to gather output data for 500 sample periods. Data samples between the 1st to 350th samples were included in the estimation set and the rest were used as the test set. 1st order ARX model and 4th order polynomial was sufficient to represent the system as a Wiener model with a R^2 fit of 0.86. After computing the w data, the output inverse nonlinear function was then represented by the equation (8.2), with close to 0.97 fit (see Figure 8.2c for the model fit).

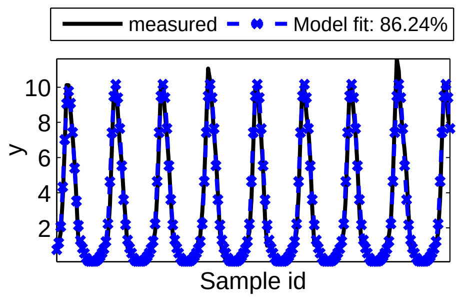
$$w = g^{-1}(y) = 7.48log(y) - 0.08 \quad (8.2)$$

For the second SID experiment, a pseudo random input signal and 35 requests/sec workload for each tenant were used. The estimated linear model is given in equation (8.3)

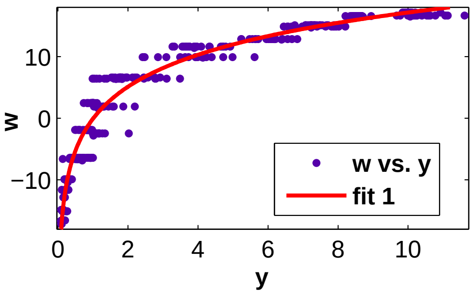
[11]It is a sample BPEL process available at `https://svn.wso2.org/repos/wso2/branches/carbon/3.2.0/products/bps/2.1.2/modules/samples/product/src/main/resources/bpel/2.0/LoanProcess/`



(a) Inverse input nonlinearity



(b) Model fit for Wiener model



(c) Inverse Output nonlinearity

Figure 8.2: The model fit of the inverse input and output nonlinearity

with $R^2 = 0.81$ model fit.

$$w(t+1) = 0.79w(t) + 0.58v(t). \tag{8.3}$$

The final step is to implement the *Hammerstein-Wiener control system* (namely, HWCS) using the linear model and pole-placement design method (see Chapter 5). The finalized parameters after placing poles at (α, β= 0.7) are $K_p = 0.47$, $K_i = 0.16$ and v(0) = 0.

8.2.4 MMST-T2 Control System Design

In this section, we present the details of MMST-T2 control system design based on the approach presented in Chapter 6. The idea is to design two models and controllers to represent the *region 0* and *region 1* and then implement the switching scheme of MMST-T2 control system. Firstly, to capture the behaviour of the system when tenant$_0$ gets more resources the operating points of $\frac{10}{10}, \frac{11}{9}, \ldots, \frac{16}{4}$ were selected to design a pseudo random signal. A high workload for tenant$_0$ was applied keeping the workload of *tenant*$_1$ at a low rate. Gathered data samples from this experiment was used to estimate the model for *region 0* with R^2 fit of 93% (see equation (8.4)). A similar experiment was conducted for *region 1* with the input set $\frac{9}{11}, \frac{8}{12}, \ldots, \frac{4}{16}$ to estimate the model shown in equation (8.5). Two controllers were then designed to provide control in each region. An aggressive controller with $K_p = 1$, $K_i = 0.44$ and u(0) = 1 and less aggressive controller with $K_p = 0.22$, $K_i = 0.11$ and u(0) = 1 were used for *region 0* and *region 1* respectively. In addition, the configuration parameters of MMST-T2 were set as $\alpha = 0$, $\beta = 1$, and $T_{min} = 3$.

$$y(t+1) = 0.84y(t) + 0.05u(t) \tag{8.4}$$

$$y(t+1) = 0.71y(t) + 0.77u(t) \tag{8.5}$$

8.2.5 Experiment Results

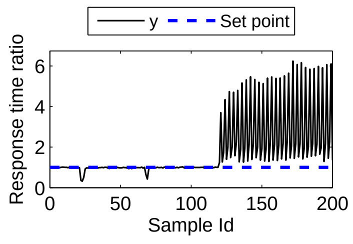
This section compares the performance and resource management capabilities of the control systems designed in Sections 8.2.3 and 8.2.4, under different settings.

In order to compare the management provided by the HWCS and MMST-T2 we also implemented a *linear control system* (namely, LCS), using the linear model in equation (8.6). The same data used in the second SID experiment was used to construct this model with a fit of 0.67. Furthermore, similar to HWCS implementation, the poles were placed at 0.7 and a controller was designed with K_p and K_i 0.64 and 0.25 respectively and u(0) = 1.

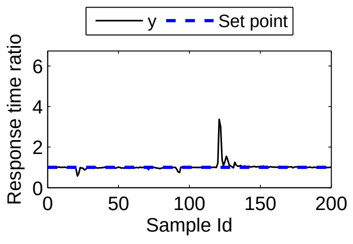
$$y(t+1) = 0.72y(t) + 0.36u(t). \tag{8.6}$$

8.2.5.1 High Workload Separately

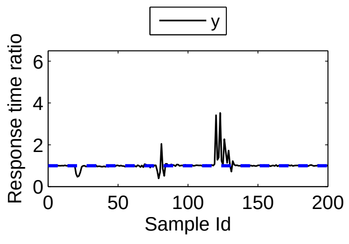
This experiment compares the performance of the control systems when the total workload from two tenants is within the system capacity. Here, each tenant increases its workload to a high level requiring more resources than the other at separate time periods. Till the 20th sample, a workload of 25 requests/sec is applied for $tenant_0$ and tenant$_1$. Then, at the 20th sample $tenant_0$ workload increases to 60 requests/sec. This could be a scenario where a high resource demand for tenant$_0$, while $tenant_1$ is at a lower workload rate. Afterwards, at the 90th sample $tenant_0$ workload reduces to 25 requests/sec. At the 120th sample, $tenant_1$ workload increases to 60 requests/sec from 25 requests/sec. The set point $(\frac{P_1}{P_0})$ is fixed at 1, where both classes are treated equally. Further, the workload settings are such that both tenants require more than $S_{min,i}$ process instances to cater the workload demand. Otherwise the performance isolation is automatically implemented due to the hybrid resource management. Under these conditions, the expected behavior is to adjust the process instance caps efficiently so that, the tenant with high workload rate gets more resources. The outputs and control signals are shown in Figure 8.3.



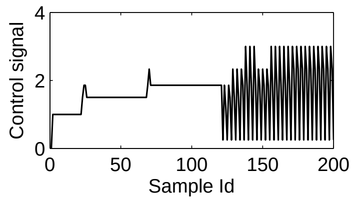
(a) LCS output



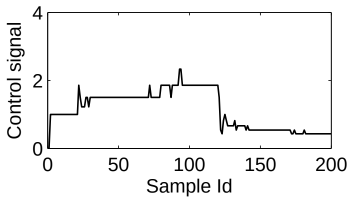
(b) HWCS output



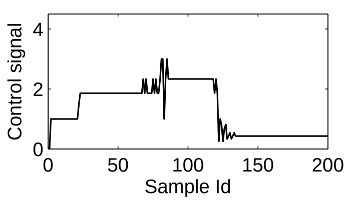
(c) MMST-T2 output



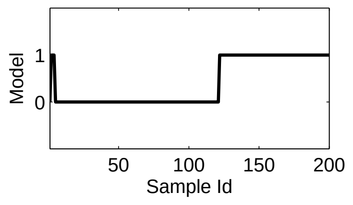
(d) LCS control signal



(e) HWCS control signal



(f) MMST-T2 control signal



(g) MMST-T2 model switch

Figure 8.3: Performance of the control systems under high workloads of each tenant separately

In this case study also we see the performance issues observed in the previous chapters for the case of LCS. The settling time and overshooting due to the disturbance at the 20th sample is significantly high compared to HWCS and MMST control systems. This is because when $tenant_0$ workload increases, the output signal decays due to the output nonlinearity in that region. Then, when LCS operates in the region where the input nonlinearity is severe, an oscillatory behavior is observed at the output after the disturbance of $tenant_1$ at the 120th sample.

In contrast, HWCS and MMST provide much better control in this experimental setting with no significant instabilities. In the case of HWCS the compensators have reduced the impact of the nonlinearities providing better performance than LCS. In addition, MMST has effectively selected the appropriate controller for the particular region avoiding the instabilities observed in LCS. The switching behaviour is shown in Figure 8.3g.

Table 8.1 summarizes the statistics of all the control systems. It is evident that HWCS has provided better performance compared to the other two control systems. MMST on the other hand has outperformed LCS significantly. Although chattering is not observed in this case, the combined performance of two linear controllers of MMST have shown transient responses, when the operating regions are changed abruptly.

8.2.5.2 Different Priority Levels Between Tenants

In this case, we assess the performance of the above designed controllers in the case of different differentiation factors, which needs effective performance differentiation when the system is running under the full capacity. The performance of the control systems is compared in this section when the set point is $\frac{P_1}{P_0}= 1.5$. For this case 25 and 55 requests/sec are applied for $tenant_0$ and $tenant_1$ respectively. Table 8.1 shows the results of the control systems.

The performance of LCS is similar to what was observed in Section 8.2.5.1. In particular, due to high workload of tenant$_1$, LCS has to operate in the region where input nonlinearity is severe. Consequently, LCS produces highly oscillatory outputs and unstable behavior in the system showing larger SSE among the control systems. In contrast, the nonlinearity compensated HWCS provides significantly better steady state behavior compared to LCS with the lowest SSE. In the case of MMST, the SSE statistics are higher than HWCS, because of the chattering. The $model_1$ and corresponding controller was selected most of the time, however switching to the other model led to the temporal instabilities. The performance of MMST is therefore, significantly poor for this case compared to HWCS. This behavior was also observed in the simulation studies presented in Chapter 6.

8.2.5.3 Overloaded Condition

In this case, we compare the performance of the control systems in the case of persistent overload of a single tenant. $tenant_0$ sends 25 requests/sec workload, while $tenant_1$ sends 150 requests/sec workloads. This means that $tenant_1$ has overloaded the system. We fixed the set point at $\frac{P_1}{P_0}= 3$, providing better performance to tenant$_0$.

The results under this condition is also similar to Section 8.2.5.1 (see Table 8.1).

HWCS outperforms both other control systems providing much stable and satisfactory performance. MMST also provide better performance, however chattering was observed for short periods of time leading to temporal instabilities.

Table 8.1: The statistics of LCS, HWCS and MMST

Section	LCS			HWCS			MMST		
	SSE	MIN	MAX	SSE	MIN	MAX	SSE	MIN	MAX
8.2.5.1	601.8	0.45	6.48	12.41	0.52	3.42	18.19	0.46	3.57
8.2.5.2	997.74	1.2	7.8	16.81	0.98	2.88	85.57	0.66	4.31
8.2.5.3	561.52	1.43	8.88	17.13	1.71	4	45.22	1.41	4.62

Further experimental results of the above approaches can be found in our previous publications [189, 190, 194].

8.3 Absolute Performance Management at the Application level

In this study our focus is on the absolute performance management of aforementioned travel reservation system. In this system, the resources that have to be shared and controlled are at the application level. In particular, the limited set of sessions provided by the third party GDS has to be controlled. The details of the implementation of the travel reservation system and experiment results are covered in the following sections.

8.3.1 Implementation

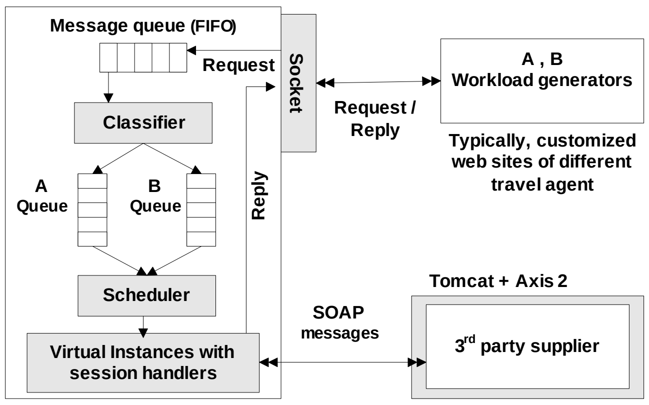


Figure 8.4: Architecture of the travel reservation system

Flight reservation system was developed implementing the architecture shown in Figure 8.4. The main difference of this implementation compared to the system in Section 8.2 is that the source code of the reservation system should also include the tenant or class specific implementations. This is because the classes exist at the application level. As a consequence, the tenant-specific queues, schedulers, sensors and actuators are implemented in the same source code.

A client of a travel reservation system can access the services of the reservation system by connecting to the server socket. After connection is made, the clients can send different messages invoking different service methods. When a message is received at the message queue, a time stamp (t_1) is applied and then the request is classified according to the client class and put to the relevant client queue. The scheduler accesses these queues in a first-come first-serve (FIFO) fashion, and assigns these messages to a virtual application instance with client specific method pointers and virtually partitioned session handlers to be sent to the 3^{rd} party supplier. When the server receives the response from the 3^{rd} party supplier, it is sent back to the client through the socket. Another time stamp (t_2) is applied before the response is written to the client socket.

8.3.2 Experiment Setup

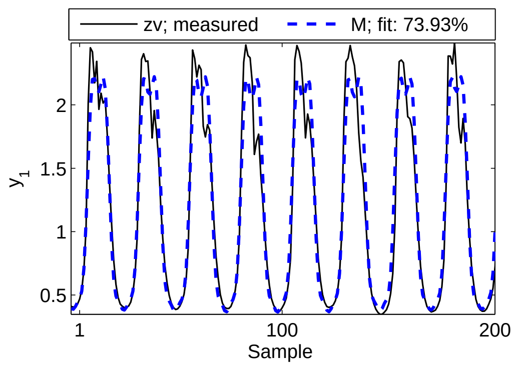
The system was deployed on a machine with a Intel Core(TM)2duo E8400 CPU@3.00 GHz 2.99 GHz processor and 2 GB memory. To simulate workloads of two agents (namely, A and B), we used tailor made workload generators which was deployed on a separate machine with Core(TM)2duo E6550 CPU@2.33 GHz 2.33 GHz processor and 3 GB memory. The target software system and the controllers were built in C#.Net and the 3^{rd} party supplier component was designed using Java as a web service deployed in a Tomcat 5.5 with Axis 2 web service engine. The two machines were connected via 1 Gps Ethernet. The total amount of sessions S_{total}[12] was set to 30 and the minimum resource allocation settings are $S_{1,min}, S_{2,min} = 6$. With these setting the prototype system can handle a 65 requests/sec workload without overloading the system. For notational simplicity $y_1 = \frac{1}{R_1}$ and $y_2 = \frac{1}{R_2}$ denote outputs, where R_1 and R_2 are the response time of agent A and B, while u_1 and u_2 denote their session caps respectively.

8.3.3 MIMO Wiener control system design

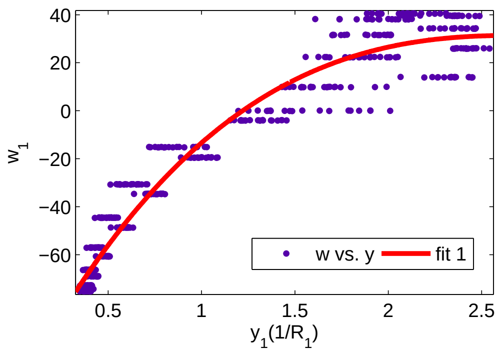
Estimation of output nonlinear compensator: Here, following the modelling approach proposed in Chapter 4, the nonlinear compensator is designed. To gather input-output data, we designed a SID experiment changing u_1 from 9 to 13 in a sinusoidal

[12]Typically, for small scale travel agents subscribe from 5 to 15 sessions, while large scale travel agents subscribe with larger (e.g., 30) sessions.

fashion and workload of 28 requests/sec for agent A. u_2 was kept fixed at 6 applying 10 requests/sec workload for agent B. Then output y_1 was observed for 600 samples. The gathered data set was used to estimate the SISO Wiener model. A second order ARX model and 4^{th} order polynomial was used to fit the data with goodness of fit (R^2) 0.74. Figure 8.5a, shows the model fit. The next step is to estimate the inverse of the estimated nonlinear component. For that, estimated ARX model was simulated with the input signal to calculate the data of intermediate variable w_1. Then, (w_1- y_1) data pairs were used to estimate an inverse of the output nonlinearity using curve fitting. Figure 8.5b shows the fit for a 3^{rd} order polynomial with R^2 over 0.95. The equations (8.7) shows the inverse nonlinear polynomial. Afterwards, this equation was implemented as a software component and integrated at the each output of the system to compensate the nonlinearities.



(a) Model Fit of $(u_1 - y_1)$



(b) Model Fit of $(w_1 - y_1)$ Inverse nonlinear function

Figure 8.5: Model fit of SISO Wiener model

$$\begin{aligned} w(k) &= f^{-1}(y_1) \\ &= 4.954y_1(k)^3 - 47.64y_1(k)^2 + 148.1y_1(k) - 118.8 \end{aligned} \tag{8.7}$$

Estimation of MIMO linear component: The next step is to approximate the rest of the dynamics into the linear component of the Wiener model. For this purpose, a MIMO SID experiment was designed simulating both inputs using a pseudo random binary signals. We selected 10 and 14 sessions as the levels for both u_1 and u_2 signals with switching frequency of 5 samples. Then, the workloads of 25 requests/sec were applied to simulate agent A and B workloads. With these settings, an experiment was conducted and the outputs w_1 and w_2 were observed for 600 samples. The gathered data was used to construct the MIMO model using multivariable regression. Equation (8.8) shows the transfer function of the model with close to $R^2 = 0.7$.

$$w(k+1) = \begin{bmatrix} 0.3744 & 0.0795 \\ 0.0764 & 0.3492 \end{bmatrix} w(k) + \begin{bmatrix} 1.2823 & -0.0439 \\ -0.0005 & 1.3017 \end{bmatrix} u(k), \tag{8.8}$$

where, $w(k) = [w_1(k)\ w_2(k)]^T$ and $u(k) = [u_1(k)\ u_2(k)]^T$.

Controller implementation: Using the above MIMO model and the MPC desgin procedure proposed in Chapter 5, a MIMO MPC controller was designed, which also takes into account the constraints imposed on the system. The N_p and N_c are set to 15 and 1 respectively in the implemented controller. The parameters of the Laguerre network for the two input case were set at (a_1 = 0.3, a_2 =0.3 and $N_1, N_2 = 1$). The $R_{(w)}$ was set to $225 \times I_{(2\times2)}$. $u(0) = [15\ 15]^T$.

8.3.4 Experiment Results

Here, to provide a comparative performance analysis a linear model predictive control (LMPC) system is introduced. Equation (8.9) shows the linear MIMO model. All parameters of LMPC were same as WMPC, but (a_1 = 0.35 , a_2 =0.3) and R was set to $0.5 \times I_{(2\times2)}$.

$$y(k+1) = \begin{bmatrix} 0.4244 & 0.1504 \\ 0.1314 & 0.3726 \end{bmatrix} y(k) + \begin{bmatrix} 0.0881 & -0.0115 \\ -0.0081 & 0.0971 \end{bmatrix} u(k) \tag{8.9}$$

The following subsections show the performance of the control system under different conditions.

8.3.4.1 High Workloads Separately

In this section, the performance of the control systems is evaluated by setting $R_{SLA,A}$ and $R_{SLA,B}$ to 0.41 seconds, i.e. both set points are placed in the insensitive region. This setting shows how the available resources are allocated to achieve the desired performance when workloads of agent A and B are increased to the maximum capacity of the system in separate time periods. The experiment begins with A and B sending 15 requests/sec. These workload settings require the controller to allocate resources more than the minimum allocation allowed (i.e. $u_1, u_2 \geq 6$) to achieve the set points. Then, at the 30th sample, agent A workload increases from 15 to 47 requests/sec, maintaining the maximum workload capacity. At the 80th sample workload of agent A reduces to the nominal request rate of 15 requests/sec. Finally, at the 100th sample, B workload increases to 47 requests/sec from 15 requests/sec. The outputs and control signals of the control systems are shown in Figure 8.6.

The results observed in this experiment are similar to the simulation results of Chapter 5. The overshooting at the disturbances is significantly reduced by WMPC without affecting the steady state behavior compared to LMPC, in particular at the disturbance of agent B. Although the disturbance rejection can be improved by increasing the aggressiveness of LMPC, the steady state behaviour is adversely affected. Table 8.2 shows the summary of the results, indicating performance improvements of WMPC for R_1 and R_2.

8.3.4.2 High Workloads Simultaneously

In this experiment, agents A and B start off by sending 15 requests/sec each till the 50th sample and afterwards both agents increase their workloads to 30 requests/sec simultaneously. Here, both workloads are demanding and competing for an equal amount of resources at the same time. The outputs and control signals are shown in Figure 8.7.

The common behavior of both control systems is that at the 50th sample there is a small overshooting in the output signals due to the disturbance applied. Till the 50th sample, WMPC and LMPC resource allocation setting varies and settles at multiple levels. However, the system output is not affected in both control systems. The reason for this is in the under-loaded workload conditions (here, the system is running half the capacity) the system can settle to many resource allocation settings without causing any control error. However, for the simultaneous increment of workloads at the 50th sample, the desired allocation is only around $S_1 : S_2 = 15 : 15$. Both control systems achieve this desired resource allocation setting and maintain it consistently after the 50th sample. It is important to note that, under these settings there are significant interactions between inputs and outputs of the control system. Although we ignored these interactions in the nonlinear block of WMPC, it has not affected the performance of WPMC. In fact, when statistical results are compared (see Table 8.2), there are improvements in the case of WMPC compared to LMPC.

8.3.4.3 With Different SLAs (set points)

In this section, set points are fixed at $R_{SLA,A} = 0.41$ seconds and $R_{SLA,B} = 0.6$ seconds, to evaluate the performance differentiation capabilities of the control systems when there are differences in SLAs. i.e. a one set point is placed in the sensitive region while the other in the insensitive region. For this case, 20 requests/sec workload was applied for both agents till the 50^{th} sample. At the 50^{th} sample, agent B workload was increased to 40 requests/sec. Figure 8.8 shows the system response and the control signals.

The response of both control systems illustrate the common characteristic of high overshooting at the 50^{th} sample due to high workload of agent B. However, the steady state performance of R_2 is oscillatory in both control systems. This is because $R_{SLA,B} = 0.6$ is located in the sensitive region of the response time curve. As a consequence, there is no single resource allocation setting that will maintain the R_2 at the required value. WMPC performance illustrates significant reduction in overshooting and settling time at the 50^{th} sample. The statistical comparison in Table 8.2 shows that performance of R_2 is significantly better in the case of WMPC, indicating much better steady state and disturbance rejection characteristics compared to LMPC. However, there is a slight impact on R_1 in the case of WMPC compared to LMPC. This is because WMPC reacts aggressively to the high disturbance of agent B, subsequently affecting the resource cap of agent A for a short period of time.

Table 8.2: The statistics of LMPC and WMPC

Case	Output	WMPC		LMPC		Diff (LMPC-WMPC)	
		SSE	MAX	SSE	MAX	SSE	MAX
8.3.4.1	R_1	1.874	1.296	5.170	1.700	3.30	0.40
	R_2	13.489	2.445	30.807	2.981	17.32	0.54
8.3.4.2	R_1	0.067	0.580	0.252	0.705	0.19	0.13
	R_2	0.047	0.522	3.777	0.687	3.73	0.16
8.3.4.3	R_1	0.197	0.699	0.038	0.539	-0.16	-0.16
	R_2	8.227	1.656	11.542	2.216	3.32	0.56

8.3.5 Multi-Model Switching Control

Together with the general linear model shown in equation (8.9) and the controller (namely, $LMPC_1$), we combined another model and a controller (namely, $LMPC_2$) to operate in the region when the agent B's set point is placed in the sensitive region (0.5

to 0.8 seconds) (see equation (8.10)). $LMPC_2$ was designed with the same settings of N_p and N_c set to 15 and 1 respectively. The parameters of the Laguerre network were set at ($a_1 = 0.4$, a_2 =0.7 and $N_1, N_2 = 1$). The $R_{(w)}$ was set to $0.001 \times I_{(2\times 2)}$. In addition, the $R_{(w)}$ of the $LMPC_1$ was set to 0.1, to provide better disturbance rejections in the insensitive region. The switching logic is similar to the rules in Chapter 6. This multi-model switching control system is called MMPC.

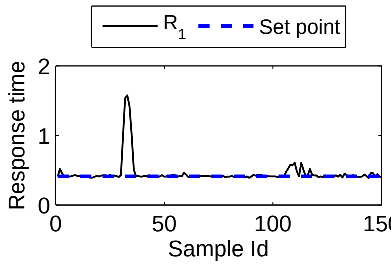
$$y(k+1) = \begin{bmatrix} 0.4478 & 0.0218 \\ -0.1915 & 0.7755 \end{bmatrix} y(k) + \begin{bmatrix} 0.0631 & 0.0386 \\ -0.0201 & 0.0999 \end{bmatrix} u(k) \tag{8.10}$$

The same workload conditions and operating conditions of Section 8.3.4.1 were used in the experiment till the 200th sample. Thereafter, the workload of agent A and B was set to 20 requests/sec and 40 requests/sec respectively, while altering the set point of agent B to 0.6 seconds. The results are shown in Figure 8.9.

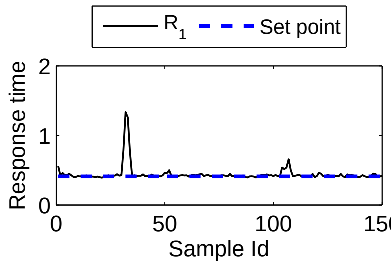
The LMPC$_1$ provides better performance till the 200th sample for both R$_1$ and R$_2$, but the steady state error of R_2 after the 200th sample is significantly high compared to both other control systems. Similarly, although LMPC$_2$ provides better performance after the 200th sample, the performance before the 200th sample is poor for the case of R_1 when the sudden workloads disturbances are encountered for both agents separately. MMPC which combines the performance of both control systems provides better results compared to both individual linear control systems. Furthermore, the simple switching rules implemented in this control system have provided stable switching behavior and have not created any issues inherent to switching control systems.

8.4 Conclusion

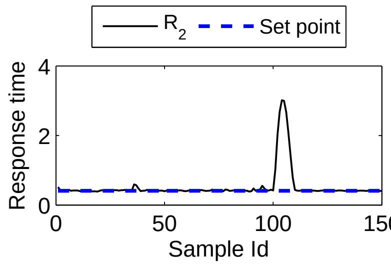
This chapter has applied the proposed nonlinear control approaches in two real-world experimental case studies: the first shares the resources at the middleware level, while the second shares the resources at the application level. The relative performance management capabilities of Hammerstein-Wiener and MMST adaptive nonlinear control methods have been investigated using the first experimental case study. In addition, the absolute performance management capabilities of MIMO Wiener and multi-model nonlinear control schemes have been compared with the exiting linear MIMO control approaches using the second case study. The results of these studies have shown that these new nonlinear control approaches outperform the existing linear approaches in many cases. These results have also illustrated the applicability of the proposed approaches in the real-world shared resource environments that exist at different levels of the system stack.



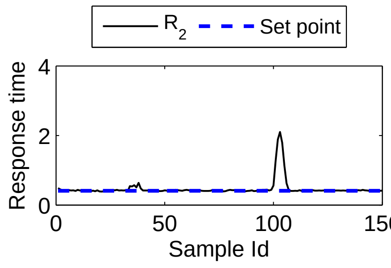
(a) Output R_1 of MIMO LMPC



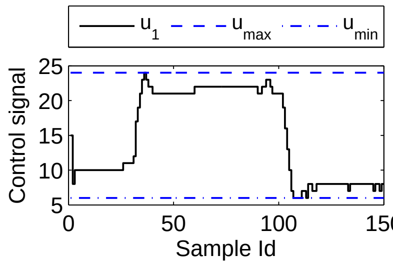
(b) Output R_1 of MIMO WMPC



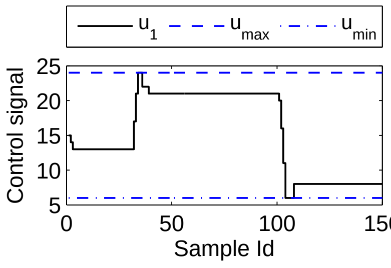
(c) Output R_2 of MIMO LMPC



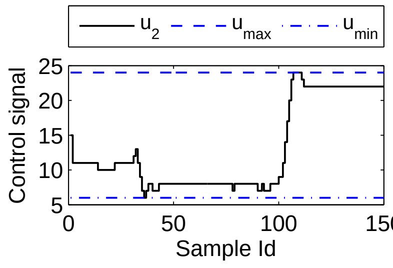
(d) Output R_2 of MIMO WMPC



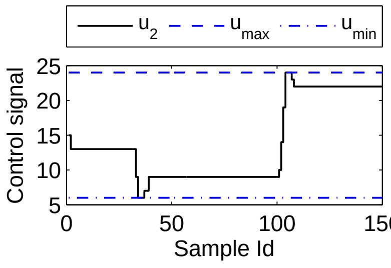
(e) Control signal u_1 of MIMO LMPC



(f) Control signal u_1 of MIMO LMPC

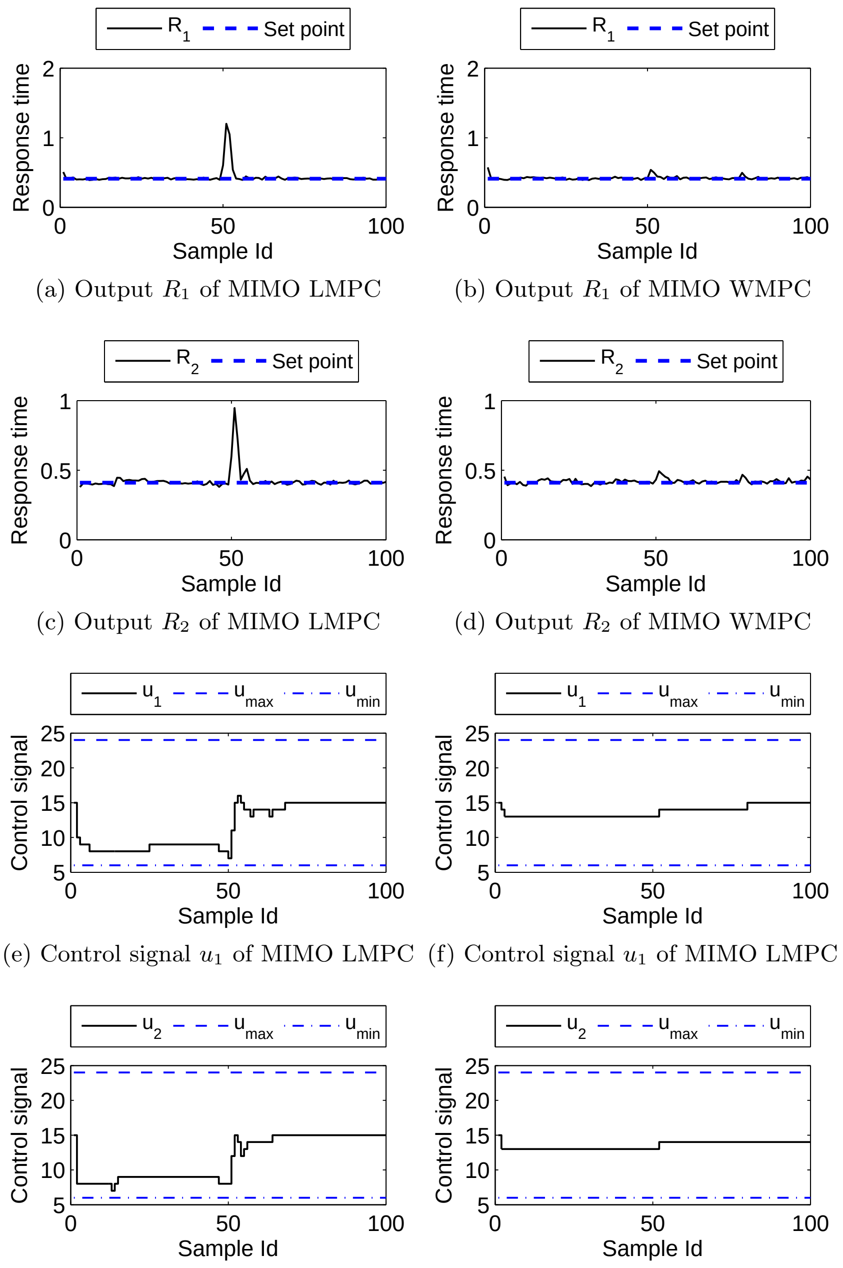


(g) Control signal u_2 of MIMO LMPC



(h) Control signal u_2 of MIMO WMPC

Figure 8.6: Performance management under high separate workloads



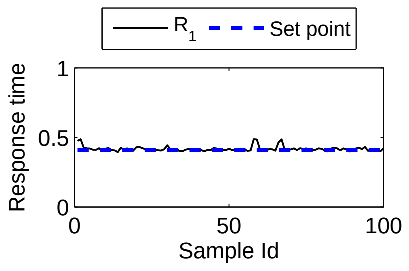
(a) Output R_1 of MIMO LMPC (b) Output R_1 of MIMO WMPC

(c) Output R_2 of MIMO LMPC (d) Output R_2 of MIMO WMPC

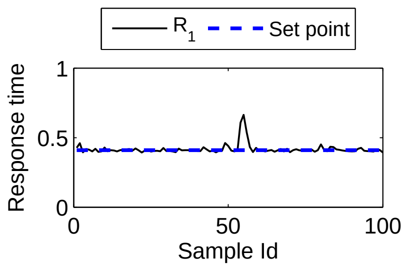
(e) Control signal u_1 of MIMO LMPC (f) Control signal u_1 of MIMO LMPC

(g) Control signal u_2 of MIMO LMPC (h) Control signal u_2 of MIMO WMPC

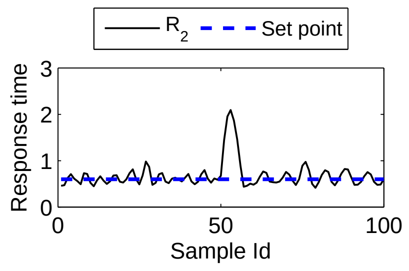
Figure 8.7: Performance management under simultaneously high workloads



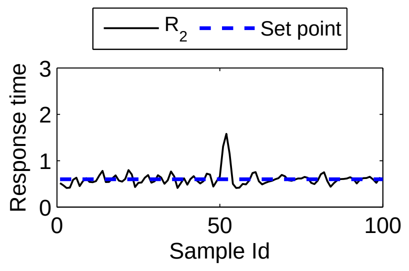
(a) Output R_1 of MIMO LMPC



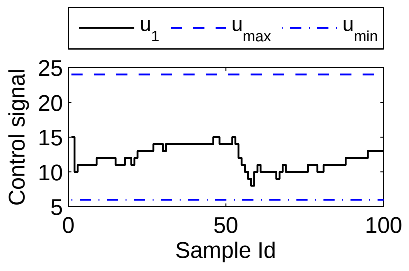
(b) Output R_1 of MIMO WMPC



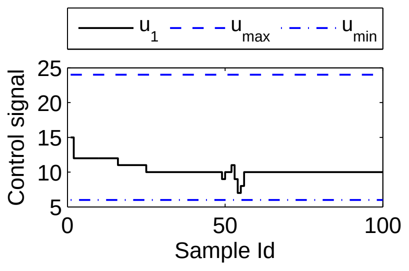
(c) Output R_2 of MIMO LMPC



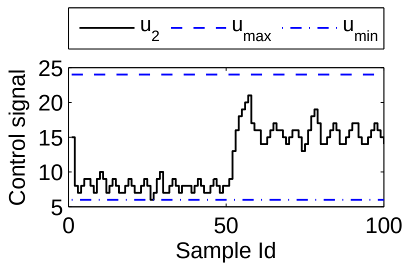
(d) Output R_2 of MIMO WMPC



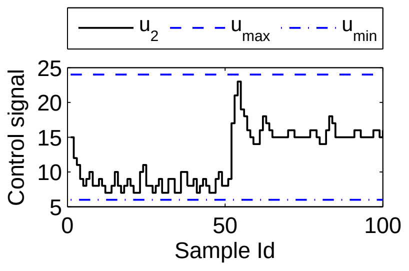
(e) Control signal u_1 of MIMO LMPC



(f) Control signal u_1 of MIMO LMPC

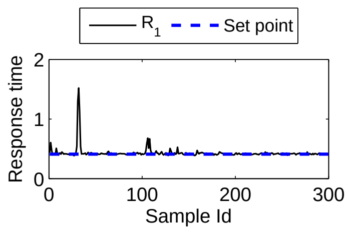


(g) Control signal u_2 of MIMO LMPC

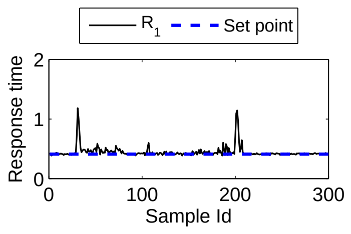


(h) Control signal u_2 of MIMO WMPC

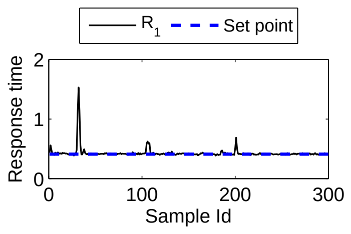
Figure 8.8: Performance management under different reference values



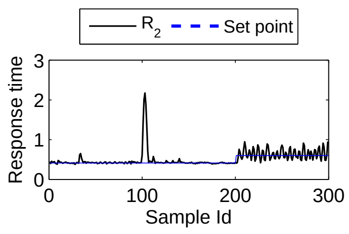
(a) Output R_1 of $LMPC_1$



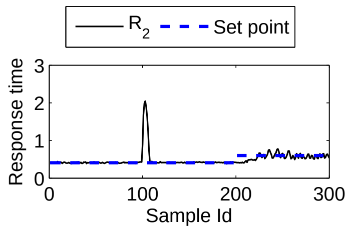
(b) Output R_1 of $LMPC_2$



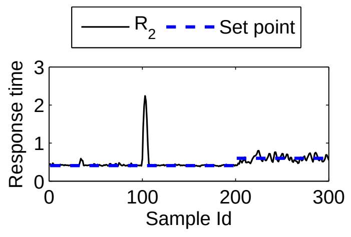
(c) Output R_1 of MMPC



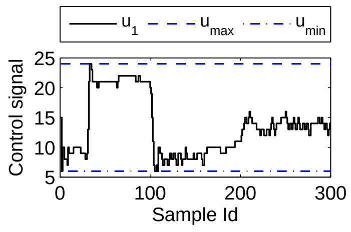
(d) Output R_2 of $LMPC_1$



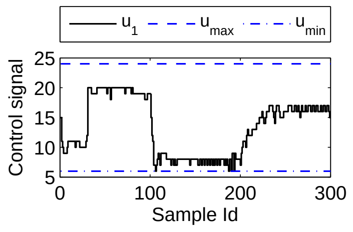
(e) Output R_2 of $LMPC_2$



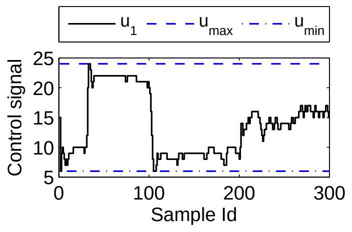
(f) Output R_2 of $MMPC$



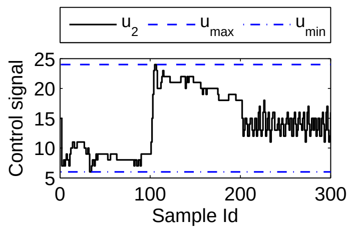
(g) Control signal u_1 of $LMPC_1$



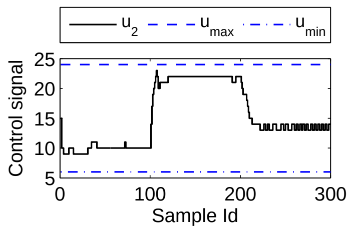
(h) Control signal u_1 of $LMPC_2$



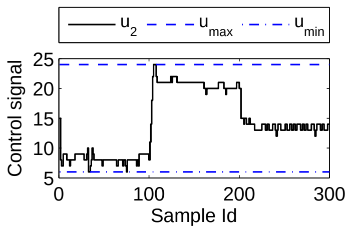
(i) Control signal u_1 of $MMPC$



(j) Control signal u_2 of $LMPC_1$



(k) Control signal u_2 of $LMPC_2$



(l) Control signal u_2 $MMPC$

Figure 8.9: Performance management of LMPC$_1$, LMPC$_2$ and MMPC

Chapter 9

Conclusions

Due to the popularity of shared resource software environments, dynamically managing a limited amount of resources between multiple customer classes, to deliver the required performance properties such as response time has become a challenging research problem. This is because, a management system of a multi-class shared resource software environment has to be designed 1) to achieve diverse performance objectives of the multiple customers under highly dynamic and unpredictable workload conditions, 2) to honour the limits and constraints on the resources, 3) to operate at the different levels of the system stack (e.g., middleware and application levels) and 4) to handle the nonlinear dynamics of the system and control schemes. Addressing this research problem has been the main focus of this thesis.

The primary research focus has been to investigate nonlinear feedback control methodologies to automate the control of resources at runtime to achieve relative and absolute performance management objectives of multi-class shared resource software environments. A literature review on the existing performance and resource management techniques has been performed to identify the research gaps. The research goal and method were then formulated to characterize the nonlinear dynamics of the relative and absolute performance management systems. As major research contributions of this thesis, novel nonlinear model estimation techniques and control architectures have been proposed to implement the relative and absolute performance management systems. The proposed approaches have been evaluated using simulation studies and real-world experimental case studies. From these evaluations, the proposed nonlinear approaches have shown significant improvements compared to the existing control methods. Furthermore, we have also presented a configurable and extensible supporting tool to implement complex control systems for software environments in a cost-effective manner.

The following sections present a summary of work and summary of findings of this thesis, followed by the limitations and directions for future research.

9.1 Summary of Work

In this thesis, we have presented novel nonlinear control approaches, which have made several contributions to the field of performance management of multi-class software systems. In particular, these new approaches have reduced the impact of nonlinearities on the management system by explicitly considering the nonlinear dynamics in the system modelling and control system designing stages. In a nutshell, the research work presented in this thesis is as follows.

Control-oriented nonlinear system identification

- ▶ For relative performance management, due to consideration of ratios of the performance properties (response time) and the resource caps between classes, severe nonlinear dynamics exist at the system inputs and outputs. In this work, we have characterized the existing nonlinear behavior as input and output nonlinearities for the first time using a block-oriented model called the Hammerstein-Wiener model. A system identification method is then presented to explicitly model the input and output nonlinearities together with the rest of the dynamics.
- ▶ For absolute performance management, the nonlinear dynamics exist because of the nonlinear behavior of the performance property with respect to the resource cap of each class. In this case, since the management system is MIMO, the system identification of these nonlinear dynamics was a challenging task. A new modelling approach based on a simplified MIMO Wiener model has been proposed in this thesis. This approach decomposes the MIMO system into SISO subsystems and subsequently identifies the nonlinear dynamics at each output individually. Afterwards, a MIMO linear model is used to represent the rest of the dynamics.
- ▶ In addition to the above modelling approaches, another simple and intuitive approach to capture the nonlinear dynamics of a system is to estimate multiple linear models to represent the dynamics of multiple regions. In this thesis, we have also investigated this

approach to represent the nonlinear dynamics of the relative and absolute management systems and proposed methods to divide a wide operating region into multiple regions, which are represented by multiple linear models.

Nonlinear control system design and management

- ▶ Based on the estimated Hammerstein-Wiener model, a control system architecture equipped with nonlinear compensators has been put forward for the first time, to reduce the impact of input and output nonlinearities of the relative scheme, on the management system. These compensators were integrated externally to the target system and consequently no modifications were required to the target system. A PI control system based on the transformed variables has been implemented to automate the management of resources of multiple classes at runtime.
- ▶ A MIMO MPC control technique based on the estimated MIMO Wiener model structure has been presented for absolute performance management in this thesis. This control architecture also consists of nonlinear compensators at each output of the target system. In addition, after integration of the compensators, the conventional MIMO MPC and constraint problems have been transformed to operate with the intermediate output variables. Furthermore, a MPC system based on the Laguerre functions and a quadratic programming solver based on the primal-dual method have been presented to implement the MIMO Wiener control system, in order to automate the absolute performance management of multiple classes at runtime.
- ▶ Using the estimated multiple linear models to capture the dynamics of the system in multiple regions, a control system consisting of multiple controllers and switching schemes has been proposed for relative management. The switching scheme was designed based on an adaptive control scheme called multi-model-switching and tuning adaptive control. In contrast, a different rule-based switching scheme equipped with multiple controllers has indicated better performance for the case of absolute management. The simple rule-based scheme has been designed using the reference signal of each class as the scheduling variable.

Implementation support tools

- ▶ In order to support the engineering process of control systems, a class library with a set of standard control components has been developed as one of the outcomes of this thesis. This class library can be readily integrated to the software systems developed using Java and .Net platforms. This class library consists of models, control algorithms and optimization programs which can be configured or extended in a cost effective manner.

Evaluations using simulation, experimental and empirical studies

- ▶ The above mentioned modelling and control solutions have been also evaluated comparing them to existing linear and adaptive control methods using simulation studies, as well as two real-world experimental case studies sharing resources at different levels of the system stack. In these evaluations, the benchmark multi-class systems serving two and three classes have been utilized. In addition, to show that these results do not depend on the simulation parameters, we have conducted and presented the results of Monte-Carlo simulations.
- ▶ In order to evaluate the facilitation provided by the aforementioned class library, we have also conducted an empirical study involving a group of software engineers.

9.2 Summary of Findings

This section lists the findings of this thesis.

- ▶ The nonlinear Hammerstein-Wiener control technique presented to achieve the relative performance objectives has shown that the input and output compensators have effectively reduced the impact of nonlinearities that exist in the relative performance management system, compared to the linear control counterpart. In particular, for majority of cases, much better steady state behavior has been shown when the system is operating in the region where input nonlinearity becomes severe while the disturbances have been rejected efficiently in the region where the output nonlinearity damps out the output signal. However, the noisy conditions have affected the performance of the output compensator indicating slight performance degradations for some cases.
- ▶ The nonlinear MIMO Wiener control system has achieved the absolute performance management objectives significantly better compared to a MIMO linear controller. The integrated compensators at the each output have reduced the impact of the nonlinearity on the absolute management system providing significantly better disturbance rejection capabilities, without sacrificing the steady state performance for all cases.
- ▶ The proposed multi-model scheme based on the MMST adaptive control has shown better performance in the cases where noise does not affect the output signal. In

such cases, MMST control scheme has selected the suitable model and controller autonomously under fast changing unpredictable conditions delivering better performance compared to the linear and adaptive control approaches. However, a major observation was that the noisy output signals led the control systems to falsely detect the change of conditions, thereby leading to chattering and large transient responses. It is also worth noting that the velocity form of the PI controller, which operates with the incremental control input has implemented bump-less transfers, contributing to better performance even under frequent switching of controllers. Although the multi-model approach is a simple and intuitive approach, the overall performance comparison has indicated that the Hammerstein-Wiener control approach is much more stable and efficient approach for relative performance management.

- ► The multi-model approach presented for absolute performance management has shown that the simple rule-based switching algorithm provides effective control in the insensitive and sensitive regions of the system output compared to an individual controller tuned to operate in a single region. In addition, this approach has provided design flexibility and stable switching under changing conditions.
- ► Referring to Chapter 1 where we have listed the quality attributes, a management system of multi-class shared resource systems need to possess, it is evident that proposed nonlinear control approaches (i) achieve the absolute and relative performance objectives under unpredictable disturbances and events significantly better compared to the existing linear approaches in most cases, (ii) adjust the resource caps efficiently at runtime under sudden workload changes and resource demands, (iii) provide performance isolation between classes, (v) enable dynamic resource management at different levels of the system stack and (v) provide systematic and formally grounded design processes[1].
- ► The empirical study conducted with a group of software engineers has indicated that the engineering process and the class library presented in this thesis have facilitated software engineers in the implementations of complex control systems compared to implementing them from scratch. The results have shown that the class library has significantly reduced the implementation time and costs together with the knowledge required of the software engineer.

[1]However, in the cases of MMST adaptive control the parameters of the switching algorithm have to be selected using a set of heuristics.

9.3 Limitations

Although proposed nonlinear control approaches have shown significant improvements, there are some limitations encountered during the design stages of the control systems and during the runtime operations of the control systems.

- *Model estimation effort.* The identification procedures of block-oriented models are relatively complex compared to the linear SID. In particular, to identify the output nonlinear and linear components of the block-oriented models, two SID experiments have to be conducted. Similarly, several SID experiments have to be carried out, in order to identify the required models for the multi-model schemes as well. Therefore, the model estimation effort is high for the approaches proposed in this thesis compared to the linear SID approaches. However, the existing design support tools (e.g., Matlab SID toolbox[2]) can still be used for all the proposed nonlinear model estimations, avoiding the need of additional implementation efforts.
- *Computational overhead at runtime.* The control systems equipped with compensators induce additional computational overhead due to the compensator framework compared to a linear control system. Similarly, the switching control systems require execution of the switching logic/algorithms, which impose computational overhead in comparison to a linear control system. However, all these additional components require computations of simple mathematical operations, which introduce only an insignificant overhead in the current state-of-the-art software environments.
- *Tuning of MMST adaptive control schemes.* There is no formal process to derive the switching algorithm parameters of MMST adaptive control. To mitigate this limitation, a set of design heuristics has been presented in Chapter 6.
- *Chattering.* Chattering is a well-known issue that exists in switching control systems such as MMST adaptive control. To overcome this issue, the tunning parameters of the switching algorithm have to be selected carefully after simulation studies.

9.4 Future work

In this thesis, we have presented new nonlinear control engineering approaches to implement performance management systems for shared resource environments. This is still a new area of study in the software performance management literature. There is

[2]`http://www.mathworks.com.au/products/sysid/`

more work to be done in the future to address the existing limitations and further improve these nonlinear techniques. This section overviews the directions for future research.

The proposed model estimation approach for absolute performance management neglected the interactions between the intermediate output variables and system outputs. However, it is important to further investigate the impact of this assumption. New nonlinear identification methods have to be devised to capture these interactions in a systematic way, which is a major future work.

One of the assumptions of Hammerstein-Wiener model estimation method was that the output nonlinear dynamics do not change over time. However, in some cases, we observed that noisy signals affect the compensation, subsequently the controller reacts to those conditions aggressively. This is because output nonlinearity is not always static. In order to mitigate this issue, on-line identification of the output nonlinearity may be useful. Such a technique has been proposed in [91].

The investigated case studies shared resources only at the application and middleware levels. In the future we also intend to apply the proposed nonlinear approaches at the hardware level using the virtualization technology.

In order to mitigate the design overhead of MMST adaptive control, the online model learning and model retention techniques have been proposed in [167, 196]. The investigation of such techniques is another future research direction.

Appendices

Appendix A

Control System Design

Two main design steps have to be carried out in order to design a control system in a systematic way. First, sufficiently accurate model of the system has to be estimated. Second, a suitable controller has to be selected, tuned and tested. Following two subsections of this appendix provide the general background on these two steps of designing a control system.

A.1 System Modeling

In order to design a control system, a formal relationship between the control inputs and the outputs has to be constructed. In control theory this relationship is referred to as the *model* of the system. Many physical systems are governed by first principle models (such as mass, Newtonian, electricity laws) which can be used to model the behavior of the system. However, when the software systems are considered, such first principle models do not exist or highly complex limiting the use of well-established control engineering methodologies to design a controller, unless linearization or other approximation techniques are used to simplify the model. Some work, use queuing models as such analytical model to describe the behavior of software systems. The limitations of queuing models are they are not fine grained enough to capture the behavior of the system [81, 177]. In addition, the performance of the queuing models is significantly poor, when they are used in runtime control with small time intervals [81, 177]. Moreover, the complexity and the performance of such models drastically degrade in multi-class systems under changing workload conditions [30]. Furthermore, assumptions have to be made on arrival, service rates and their distributions which may not hold in certain dynamic software environments [81, 177].

The other possible solution is to treat the entire system as a black-box and conduct a system identification experiment. System identification (SID) provides a rich body of literature and methodologies to represent the behavior/dynamics of the system with mathematical models using experimental input-output data [125]. In a SID experiment, to gather input-output data, a specially designed (persistently exciting [125]) input signal is applied on the system and output is observed for a sufficient period of time. The pseudo random binary (PRB), sinusoidal or pseudo random signals are extensively used as input signals in the literature. Then, the input-output data is fitted into a suitable model with sufficient accuracy using linear regression. For this purpose, typically Linear Time Invariant (LTI) models are used [81]. Autoregressive Exogenous input (ARX) models have been used to represent the behavior of software systems widely [81]. The standard equation of ARX model is shown in equation (A.1).

$$y(k) = \sum_{i=0}^{n} a_i y(k-i) + \sum_{j=0}^{m} b_j u(k-d-j) \tag{A.1}$$

where, n and m are the order of the model, a_i and b_j are the parameters of the model, d is the delay (time intervals taken to observe a change of input in the output) and k stands for the current sample instance. The order and other parameters of this model are derived using linear regression techniques [81, 125]. These are the basic concepts behind a SISO SID experiment.

In a MIMO system, however, the above system identification process becomes complex due to the involvement of multiple variables. There are two main techniques. The first technique is to change/excite a one single input at a time keeping other inputs at desired (steady state) values and observe the outputs. The same process is then applied to other inputs. Afterwards, the MIMO model is constructed based on the gathered data. The obvious drawbacks of this approach are the time taken to construct the model, high level of manual work, insufficient capture of the input-output interactions in the model and merging of data/models to arrive at the final model [222]. The second technique is to simultaneously excite all the inputs in the system and observe all outputs. The gathered data is then fitted into a model using *multivariable regression*. This approach addresses most of the above limitations of the first approach. But, conducting such an experiment may not be always possible in some systems due to safety reasons. However, if it is possible to conduct such an experiment, this second approach produces a high quality model as recommended in [46, 222] and proved in [67]. The MIMO ARX model is generally utilized to represent the behavior of software systems. The standard equation of MIMO ARX model is as follows:

$$y(k) = \sum_{i=0}^{n} A_i y(k-i) + \sum_{j=0}^{m} B_j u(k-d-j) \tag{A.2}$$

where, n, m - the order of the model, $(A_i,\ B_j)$ are the parameter matrices with $N \times N$ dimension (assuming, there are N inputs and outputs), d - is the delay and k is the current sample instance.

A.2 Controller Design and Specifications

After modeling the behavior of the system, the second step includes controller selection, design, simulation and testing. There are many types of controllers available in control literature including proportionalintegralderivative (PID) and predictive controllers. These controllers are most suitable to control, linear systems. However, with a good feedback mechanism and tuning these controllers may provide satisfactory performance in the systems with nonlinearities [81]. More details of these controllers will be provided in Appendix B.

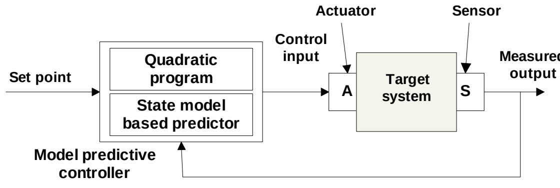
After the selection of a particular type of controller, the controller has to be configured according to the system characteristics and target system model. All of these controllers have the tuning parameters (also called *controller gains*), which have to be set to suitable values to achieve the required runtime performance specifications and maintain the stability of the system. There are several well established formal techniques to aid the designer to select the gains of the controller while analyzing the closed-loop performance of the system. The pole-placement design and root locus design are such techniques (for more details refer [81, 175]). Both these techniques have been used to derive the gains of the controllers in the literature. The theoretical range for each controller gain can be decided given the model of the system, however the final controller gains are decided depending on the ability to achieve the performance specifications after running the simulations and test cases. There are four major performance specifications widely considered in the control system design. They are stability, settling time, overshooting and steady state error (for more details refer [81, 143]).

Another important design concern is so-called saturation limits of the control input (signal). For instance, consider a volume controller of a television. A volume controller has the minimum and maximum limit of volume. The volume can only be controlled between these limits. Similarly, in a shared resource system, there are minimum and maximum resource limits that have to be honoured when the resources are allocated between multiple classes. These limits are often called *saturation limits* in the control literature. These saturation limits have to be imposed as hard constraints in the controller after $u(k)$ is computed by the controller in each sample time instance. i.e., $u_{min} < u(k) < u_{max}$, where u_{min} and u_{max} are the lower and upper bounds of the control input $u(k)$ respectively. These limits are typically implemented as rules or by running an optimization algorithm.

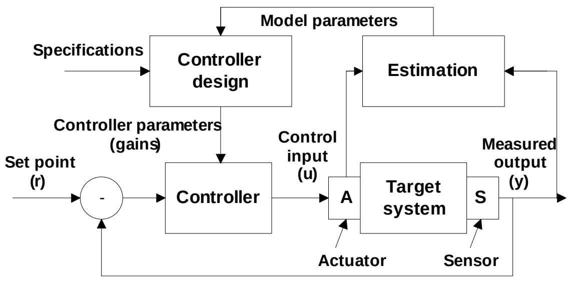
Appendix B

Different Control Schemes

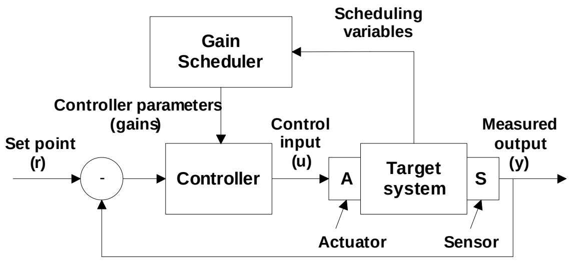
In this appendix, we provide a brief introduction to some of the widely used control schemes.



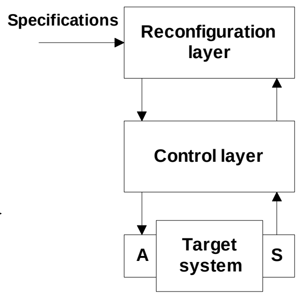
(a) Model predictive control system



(b) Self-tuning adaptive control system



(c) Gain scheduling control system



(d) Reconfiguring control system

Figure B.1: Block diagrams of different feedback control schemes

Fixed-gain control

The structure of a fixed-gain control scheme is the same to that of Figure 2.1. For instance, different variations of the Proportional Integral Derivative (PID) controller is one such fixed-gain controller widely used in existing work due to their robustness against modeling errors, disturbance rejection capabilities and simplicity [81]. Two different formations of the PI control algorithm are shown in equations (B.1) and (B.2), called position/full-value form and velocity/incremental form respectively. K_p (proportional gain) and K_i

(integral gain) are the tuning parameters of the PI controller, which have to be selected to achieve the desired system performance specifications.

$$u(k) = K_p e(k) + K_i \sum_{j=0}^{k} e(j) \tag{B.1}$$

$$u(k) = u(k-1) + (K_p + K_i)e(k) - K_p e(k-1) \tag{B.2}$$

Fixed gain control design has a number of advantages. The fixed gain controllers are useful to automate parameter tuning tasks of the software system and to achieve performance goals with less human interventions. In addition, such controllers are relatively easy to design and formal techniques exist for stability analysis and tuning of the controller. They can deliver the desired performance to a limited extent under varying workloads and changing operating conditions. However, fixed-gain controllers have some limitations. The model of the system often captures the dynamics in a particular operating region under certain operating conditions that can be characterized using a single linear model [81]. The target system usually exhibit complex behavior, including different behaviors under different operating conditions. If there are a number of dimensions across which operating conditions can vary, constructing a signal model and selecting gains to satisfy all operating conditions can be difficult [107]. Consequently, under dynamic and unpredictable variations, the performance of a fixed-gain controller can degrade because the control algorithm and the gains remain unchanged at run time. Thus, the single fixed gain controller alone cannot provide an effective solution to provide control under multiple operating regions of a software system [151, 177, 189].

Model predictive control

The general idea behind Model predictive control (MPC) is to optimize the future behavior of the system output by computing the trajectory of the control inputs. Firstly, using the model of the system and the feedback signals, the behavior of the system output is predicted over $k+N_p$, where k is the current time sample and N_p is called the *prediction horizon.* Then, the objective of the predictive control is to maintain the predicted future output sufficiently close to the desired set point value, subject to various constraints on input, output or combination of them that have to be optimized within the prediction horizon. MPC produces a sequence of control inputs $(\Delta u(k), \Delta u(k+1|k), \Delta u(k+2|k) \ldots \Delta u(k+N_c|k))$ that would achieve these objectives. N_c is called the *control horizon.* Note that the MPC operates with change/increment of input $\Delta u(k)$. Given u(0) the control input, $(u(k))$ for each sample k can be calculated by $u(k) = u(k-1) + \Delta u(k)$. However, only $\Delta u(k)$ is used to calculate the control input $u(k)(= u(k-1) + \Delta u(k))$, which will be implemented on the system in the current time sample, while discarding the rest of the sequence according to the *receding horizon control principle* [21, 223]. The same process continues in the next sample intervals by sliding the prediction horizon one time step ahead while incorporating the feedback signal.

The algorithm of MPC involves a cost function and constraint problem as follows:

$$J = (R_s - Y)^T(R_s - Y) + \Delta U^T R^w \Delta U \tag{B.3}$$

where

$$Y = \begin{bmatrix} y(k+1|k)^T & y(k+2|k)^T \ldots & y(k+N_P|k)^T \end{bmatrix}^T,$$

$$U = \begin{bmatrix} \Delta u(k)^T & \Delta u(k+1)^T \ldots & \Delta u(k+N_c-1)^T \end{bmatrix}^T,$$

$R_s = \begin{bmatrix} 1 & 1 \ldots 1 \end{bmatrix} r(k)^T$, is the reference/set point signal, and $R^w = r_{w(m\times m)} I_{(m\times m)}$, where r_w is the control penalty vector to alter the aggressiveness of the controller.

The first term in equation (B.3) incorporates the deviation of the system output compared to the desired value. The second term incorporates the controller effort. The objective is to minimize the deviation with minimal control effort. This objective can be achieved by minimizing the cost function J subject to constraints as follows:

$$\begin{aligned} & \textit{Minimize } \boldsymbol{J}\textit{:} \\ \textit{Subject to :} & \qquad\qquad (B.4) \\ & u_{min} \le u \le u_{max} \\ & \Delta u_{min} \le \Delta u \le \Delta u_{max} \\ & y_{min} \le y \le y_{max} \qquad (B.5) \end{aligned}$$

The imposed constraints depend on the system requirements. Typically, hard constraints on control input are implemented as shown in equations (B.5), to impose saturation limits as discussed in Appendix A. Figure B.1a shows that MPC requires a model predictor to predict the future behavior and a standard quadratic programming solver to solve the constrain problem online.

Model predictive control is useful because of its ability to optimize future behavior under complex constraints. In addition, it is highly suitable and efficient for controlling multi-variable (MIMO) systems with large constraint sets. However, it requires sufficiently accurate model because it depends on the model predictions to predict the future behavior of the system. In addition, under conflicting constraints the quadratic optimization may fail to achieve the optimal solutions leading to sub-optimal decisions.

Self-tuning adaptive control

Self-tuning (adaptive) control addresses some of the limitations of fixed gain controllers by dynamically estimating the model parameters and gains of the controller to achieve the high-level design objectives. As shown in Figure B.1b, adaptive controllers have a parameter adjustment loop, which derives these required parameters at runtime [13]. The parameters of the target system's model are estimated by the *Estimation* component, while the *Controller design* component uses these estimated model parameters and high-level control objectives provided by the user to compute the gains of the controller. The Self-tuning Regulators (STR) [13, 81] have been often applied as an adaptive control scheme in software systems. There are two types of STR designs. The indirect-STR uses the estimations of the system model to subsequently derive the controller parameters. In contrast, the direct-STR reformulates the model estimation algorithm to compute controller parameters directly [13, 107].

In this sense adaptive control captures the behavior in multiple operating regions of the target system. However, a basic assumption of adaptive control is that the model parameters remain constant or vary slowly over time [13, 117]. This means adaptive control does not cope well with rapid or large changes in operating conditions [166, 265]. Fast changing conditions can be seen in software systems, such as sudden workload spikes, 'slashdot'-effects, component failures and the garbage collection process. Adaptive control also has other limitations, such as computational cost due to online estimation and design. The start-up performance may not be satisfactory since it takes time to come up with the estimations. Furthermore, adaptive control methods require the input signal to contain a sufficient range of frequencies to excite the system (so-called persistently exciting condition) for fast and accurate model estimation [13, 107].

Gain scheduling

Figure B.1c shows the block diagram of a gain scheduling control system. The operating regions or states are decided based on the scheduling variables exposed by the target system. For instance, the request arrival rate can be used as the scheduling variable and rules can be formed to describe the operating regions and the controller gains for a particular operating region [81]. Then, these rules are implemented in the gain scheduling component. At runtime when the rules are satisfied the relevant controller gains are updated in the controller by the gain scheduling component. In contrast to adaptive control, gain scheduling does not have a model estimation component. Instead, it uses a predefined logic/rule based evaluation to change the controller online. Thus,the computational load may be less. Furthermore, some design flexibility can be achieved by changing the gains of the controller depending on the operating regions of the target system compared to fixed gain control. However, an issue with this technique is that the target system has to provide the required useful scheduling variables. The performance of a software system is influenced by complex interactions between different factors (e.g., workload arrival rates and the CPU usage), and their relationship to different operating regions makes it difficult to establish reliable heuristics, rules and thresholds that can determine which gains are appropriate at any given time. In addition, there are no systematic or well defined ways to implement the scheduling logic or rules [81].

Reconfiguring control:

The adaptive control schemes provide more flexibility compared to the non-adaptive scheme by adjusting the controller parameters online. However, the controller algorithm and the organization of the components in the control loop stays fixed overtime [156]. For different operating conditions and disturbances different control algorithms may provide better control [208]. Reconfiguring control scheme is a conceptual approach with the main idea to change the control algorithms, models and architecture of the control system to deal with the changing operating regions of the target system. Figure B.1d illustrates the conceptual layered architecture of reconfiguration control. The control layer consists of the control system (including the controller) providing the control in the current time instance. The responsibility of the reconfiguration layer is to reconfigure the architecture of the control layer so that the control objectives of the target system can be achieved under requirement or environmental changes. This approach is useful to provide control under multiple operating regions of the software system. For instance, multiple fixed gain controllers can be integrated into this scheme with a mechanism of selecting a suitable one at runtime. However, there are tradeoffs between the design complexity and the runtime overhead on the system due to the additional reconfiguration layer [81]. In addition, to come up with different control schemes prior information about the system and environmental conditions may be required. Chattering is another issue that can occur in reconfiguring control, i.e., the control system frequently changes between controllers or different loop configurations without providing desired control. This could lead to drastic performance degradations [156].

Appendix C

Validation of the DES model

This appendix presents the validation details and results of the DES model used to represent the multi-class shared resource environment. Well-established queuing theory will be used in this validation.

C.1 Conformant to Little's Law

One of the fundamental results of queuing theory was developed by John Little in 1960's, which is used as a basic building box in the development of theories for large scale queuing systems. Little's law is defined as follows:

For a queuing system in steady state, if the mean time waiting in the system is $W = E(T)$, and the mean number of customers entering the system is λ, then the mean number of customers in the system is given by $E(L) = W \times \lambda$.

This result applies to any queuing system and even to systems within a system. However, system has to be at steady state, meaning that the arrival rate should be less than the service rate of the system. The architecture of a multi-class shared resource system is composed of queues (see Figure 3.3), therefore the simulation model presented in Chapter 3 can be validated using Little's law. In addition, this validation also confirms whether there are any message losses in the simulation.

In order to conduct the validation, we constructed a queuing simulation using the constructs introduced in Section 3.5.2. We used two class workload generators and queues for each class with 5 resource units in this validation. We used 18 combinations of stochastic arrival and service rates for the exponential distribution to simulate the workloads and processing times of both classes, which provide sufficient data to conduct a statistical test to compare the results. All these combinations were selected while maintaining the system at the steady state. Each experiment was conducted for 50,000 ticks. The *StatisticCalulator* instance was used to compute the final statistics of the experiment including, the average response time, average arrival rates and average number of customers in the system. In these calculations the system was considered composed of two sub systems, each providing services to a single class. The *total number in these two sub systems* computed using the Little's law was compared with the *measured total number in the system* when both sub systems considered together.

A summary of results are listed in Table C.1. The results indicate that the measured number in the system is equal to the theoretical calculations of the Little's law. Hence, the multi-class system simulations implemented from the DES model described in Section 3.5.2, conform to the Little's law. This result also indicates that all the requests input to the system have left the system. Furthermore, the implementation of the DES model including the statistical calculations is also correct.

Table C.1: A comparison based on Little's law

Average waiting time Class 1(W_1)	Average waiting time Class 2(W_2)	Total number of customs class 1(N_1)	Total number of customs class 2(N_2)	Measured average number of customers in the system	Calculation of little's law $W_1 \times N_1$ + $W_2 \times N_1$
54.537	51.490	93	104	0.208	0.208
46.253	49.351	999	993	1.904	1.904
28.235	29.221	1238	1243	1.425	1.425
14.127	14.392	1676	1658	0.950	0.950
17.091	17.265	2030	1944	1.365	1.365
14.42547	14.40802	2536	2468	1.442	1.442

C.2 Conformant to Other Queuing Principles

In addition to the above validations, we also conducted other validations using general queuing system principles, comparing the simulation behavior to a single-server queuing system (M/M/1) and multi-server queuing system (M/M/c). The results indicated there is no statistically significant difference between the simulation results and theoretical result. Interested readers can refer [188] for more details.

Appendix D

Algorithm for Resource Cap Calculation in Relative Management System

This appendix presents the algorithm proposed in [145] to compute individual resource caps for each class in a multi-class shared resource system.

Table D.1: Algorithm for resource cap calculation proposed in [145]

```
$RS_i$: normalized resource allocation of class i relative to
class $n-1$, $i = 0, 1, \ldots n-1$.
$RS_i(k) = \frac{S_i(k)}{S_{i-1}(k)}$, where k - sample instance.
$sum$: the sum of the normalized process budgets of all classes.
$M$: total number of resource units.
Begin Algorithm
$RS_{n-1} = 1$
$sum = 1$
$for(int\ j = n-2; j \geq 0; j--)$ {
//Call the controller to get ratio $U_{j+1}(k)$
between QoS class j and j+1.
      $RS_j(k) = RS_{j-1}(k) * U_{j+1}(k)$
      $sum = sum + RS_j(k)$
}
$for(int\ j = n-1; j \geq 0; j--)$ {
      $Sj(k) = M * (RS_j(k)/sum)$
}
End Algorithm
```

Appendix E

Additional Cases For Multi-class Shared System With Two Classes

In this appendix, we present further evaluation results of relative and absolute performance management systems designed in previous chapters to control shared resource system with two classes. Section E.1, Section E.2 and Section E.3 cover results related to Hammerstein-Wiener, MIMO Wiener and MMST control systems respectively.

E.1 Relative Performance Management of Two Classes Using Hammerstein-Wiener Nonlinear Control

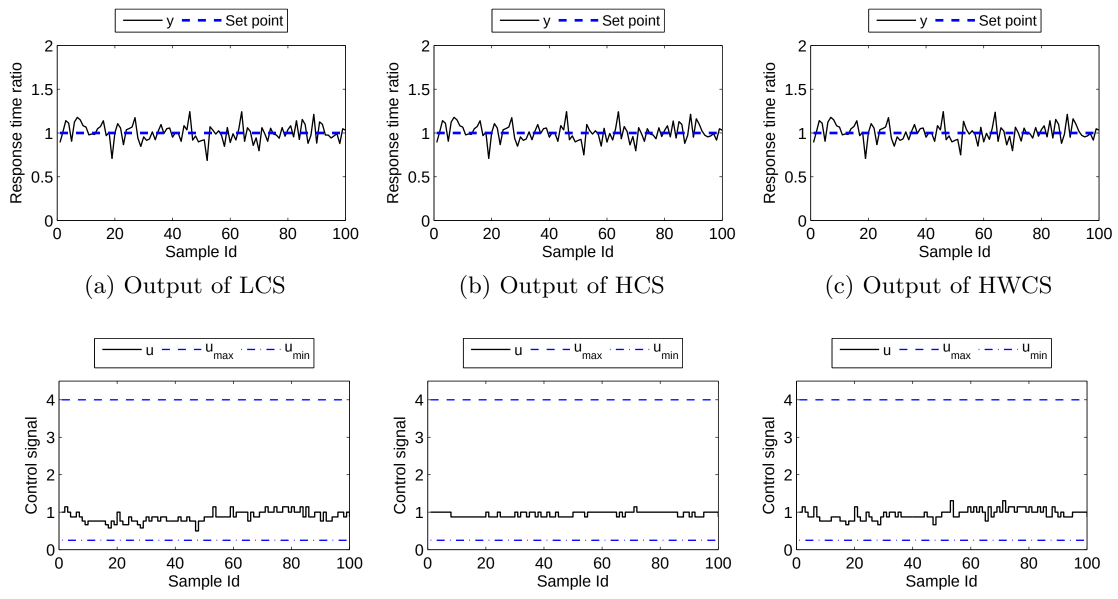
This section follows a similar structure to Section 5.3, but provides additional cases not covered in Section 5.3 under workloads of, full capacity or lower, extreme overloads and different patterns.

E.1.1 Workloads of Full Capacity or Lower

***Case G**: Performance in the nominal region*

In this case the performance of the controllers are compared in the nominal operating region (see Figure 4.2) where resource demands for both classes are similar. In the under-loaded conditions, one of the possible requirement is to treat both classes as equally important which can be translated in the relative control scheme using performance differentiation factors $P_1,\ P_0 = 1$. This is because, since the system has enough resources to cater the both workloads, it is fair to provide similar performance for both classes. Consequently, in order to maintain the control systems in the nominal regions we apply same workloads for both classes and fix the set point at $\frac{P_1}{P_0} = 1$. The class$_0$ and class$_1$ start off by sending 20 requests/sec each till the 50th sample and afterwards both classes increase their workloads to 30 requests/sec simultaneously. The performance and resource management of the HWCS, HCS and LCS are shown in Figure E.1.

Although the statistics listed in Table 5.4 indicate that there are differences in the control system outputs, these differences are insignificant. The control signals of the control systems show the variations around the operating point $\frac{S_0}{S_1} = 1$ or $S_0 : S_1 = 15 : 15$,



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

Figure E.1: Performance of the control systems in the nominal region (Case G)

which is the nominal operating point. The system output is not affected even due to the high workload disturbance at the 50th sample because all controllers settle down to resource caps $S_0 : S_1 = 15 : 15$ at the startup, which is sufficient to cater the workload disturbance at the 50th sample. As expected the LCS provides satisfactory performance in this region, however HCS and HWCS does no worse. Therefore, all control systems achieve the performance objectives under this condition.

***Case H**: When Class$_0$ is more important*

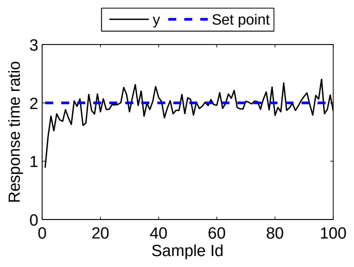
All the settings are same as Case B in Section 5.3. The experiment starts with 20 and 20 requests/sec for class$_0$ and class$_1$ respectively. Then at the 50th sample class$_0$ increases its workload to 40 requests/sec. The set point is at 1.5.

All control systems show some oscillatory behavior because of the noise generated in the output due to queuing delays of less important class. The SSE statistics shown in Table 5.4 indicate slightly better performance of LCS compared to nonlinear control systems. This is because the control system operates in region 0 where the spacing in-between operating points is large. Hence, the LCS shows less effect to the output noise compared to HWCS. The output noise further affects the performance of the post-output compensator, leading to slight performance degradations. This is because the output nonlinearity is not always static compared to the input nonlinearity. However, the MIN and MAX statistics are still better for the case of HWCS compared to LCS. The HCS on the other hand provides best performance under this condition because the noise at the output does not affect the performance of the controller.

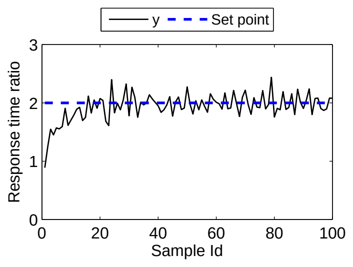
E.1.2 Workloads of Extreme Overload

Case I*: Performance under overload when Class*$_0$ *is more important*

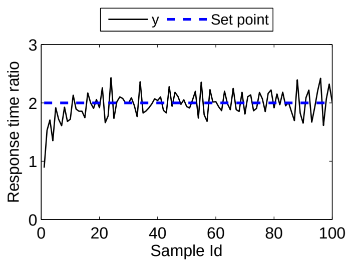
All the settings are same as Case E in Section 5.3. In this case, both classes overload the system by applying 75 requests/sec each.



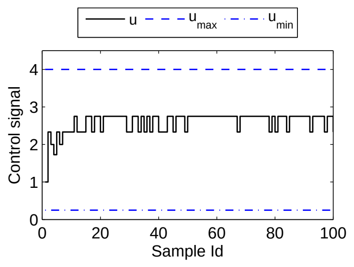
(a) Output of LCS



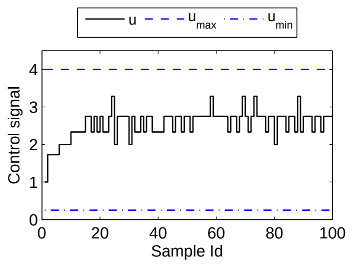
(b) Output of HCS



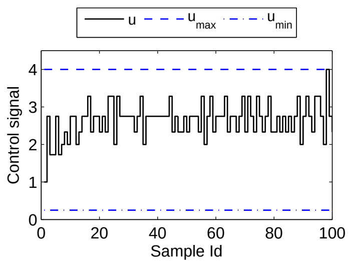
(c) Output of HWCS



(d) Control signal of LCS



(e) Control signal of HCS



(f) Control signal of HWCS

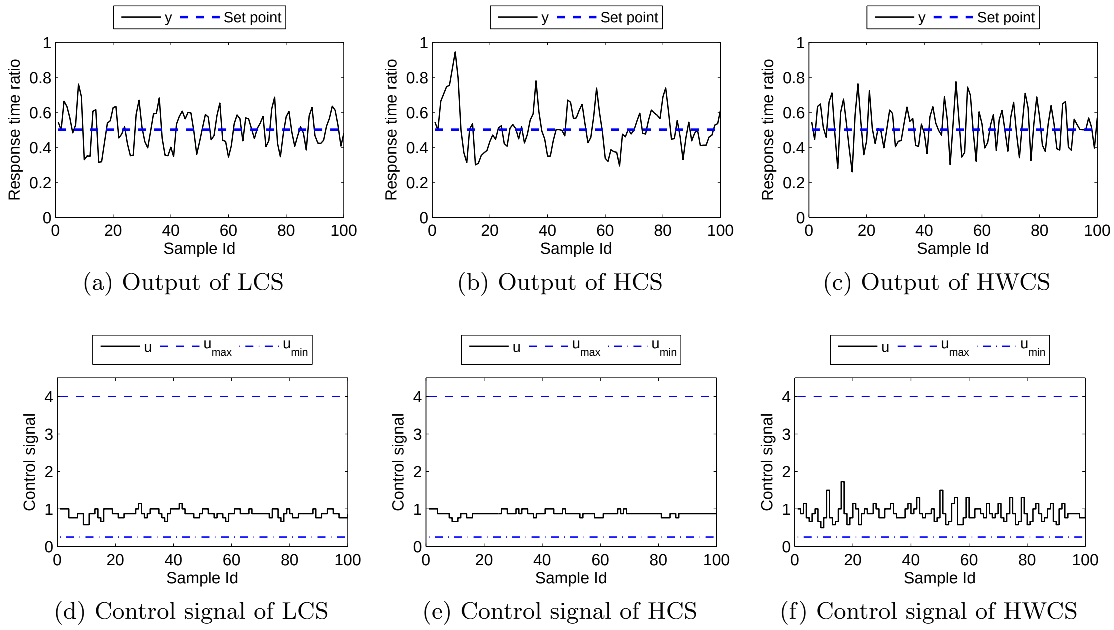
Figure E.2: Performance under overload when Class$_0$ is more important (Case I)

In this case, there is no much difference in the performance management (see Figure E.2). All control systems achieve the required control objectives. The slightly better performance of LCS is because, since class$_0$ overloads the system, the operating points lies in region 0, where the input nonlinearity does not affect the LCS under noisy conditions.

Case J*: Performance under overload when Class*$_1$ *is more important*

In this case, we set the reference signal to be $\frac{P_1}{P_0} = 0.5$, making class$_1$ as the most important class. In this experiment, 75 and 40 requests/sec for class$_0$ and class$_1$ are applied respectively, which means that the less important class has overloaded the system. Figure E.3 shows the outputs and the control signals of the control systems.

Here, the control systems are operating in the region where output nonlinearity damps out the variations of the output signal. Further, since class$_0$ has overloaded the system the resource allocation points are in the nominal region and region 1. These factors contributes to the better performance of LCS, however it is apparent that it reacts slowly to the changes in the output due to the effects of the output nonlinearity. Similar, behavior is observed in HCS as well, because it does not compensate for any output nonlinearities. In contrast, HWCS efficiently detects the variations in the output due to the compensation of



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

Figure E.3: Performance under overload when Class$_1$ is more important (Case J)

the output nonlinearity and efficiently achieves the control objectives compared to other control systems. However, due to the noisy conditions the inaccuracies in the output compensation make HWCS to over react, creating somewhat oscillatory behavior.

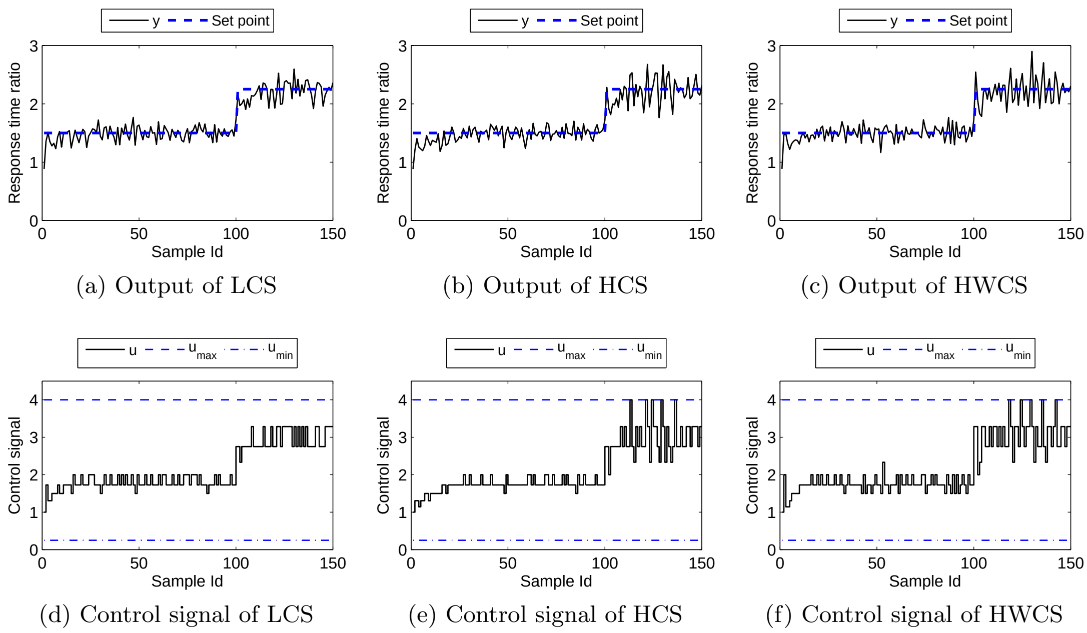
Case K*: Change of differentiation levels (or set point) at runtime time*

All the settings are same as Case F in Section 5.3. However, both classes overload the system by applying 75 requests/sec each.

All control systems achieve the required control objectives and adapt at the 100th sample to the change of reference signal. There is not much difference in the performance management. However, HWCS adapts to the change of reference signal faster compared to LCS and HCS. It settles to the new differentiation level in 6 samples compared to 12 and 10 samples taken by LCS and HCS respectively. The longer settling times of LCS and HCS are due to input and output nonlinearities respectively. However, the better steady state performance of LCS is because the control systems are operating with control points in region 0. However, the same reason affects the reaction time to the set point signal, because it settles down slower than HWCS and HCS.

E.1.3 Workloads Following Different Patterns

In Section 5.3 and the above cases, we utilized time varying workload conditions to force the operating system to operate in different regions. In particular, we used step like workload disturbances to compare the performance management under sudden changes in workload mixes of two classes. In this section, we apply different workload patterns to further investigate the performance and resource management capabilities of the control



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

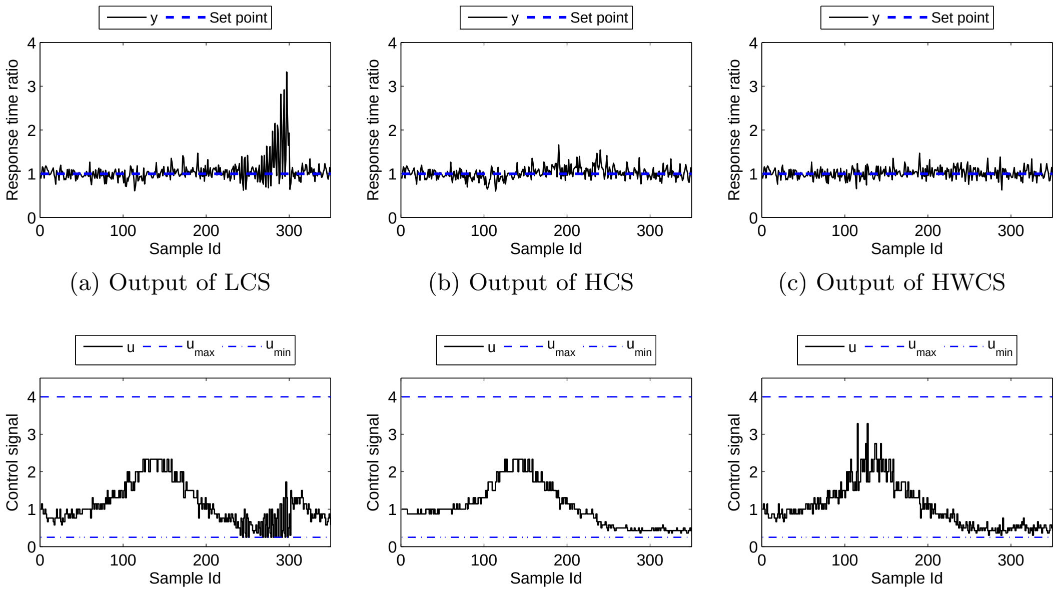
Figure E.4: Change of differentiation levels at runtime time (Case K)

systems. We use the control objectives of Case A in Section 5.3.1. This setting gives us an effective way to investigate the performance management of the control systems and how they deal with the input and output nonlinearities together, when the system is running in different operating regions.

***Case L**: Performance and resource management under ramp workloads*

The definition of the ramp workload is presented in Section 3.5.4.1. In this case, we increase the workload gradually in a ramp like fashion. The experiment starts with both class$_0$ and class$_1$ applying 20 requests/sec workloads till the 10th sample. From the 10th to 100th sample workload of class$_0$ is increased from 20 to 45 requests/sec, at a rate of 0.25 requests/sec^2. The 45 requests/sec workload is maintained till the 150th sample and then it drops to 20 requests/sec soon after. At the same point workload of class$_1$ is increased from 20 to 45 requests/sec till the 250th, at a rate of 0.25 requests/sec^2. The workload remains at that level till 300th sample and drops back to 20 reuqests/sec. Figure E.5 shows the outputs and control signals of the control systems.

Performance management under gradual workload increase of class$_0$ is satisfactory in all the control systems. Under gradual workload changes the resource caps are also adjusted gradually in order to achieve the control objectives. However, LCS shows small overshooting just after the 100th sample. This is because, the gap between consecutive operating points becomes larger due to input nonlinearity in region 0, so that the time taken to move to the consecutive points becomes larger as well. Similar overshooting is observed in HCS at the same stage, which is caused by the output nonlinearity. Then,



(a) Output of LCS (b) Output of HCS (c) Output of HWCS

(d) Control signal of LCS (e) Control signal of HCS (f) Control signal of HWCS

Figure E.5: Performance and resource management under ramp workloads (Case L)

when class$_1$ workload increases gradually, the performance of the LCS starts to degrade around 230th sample. This is again due to the severe input nonlinearity in the region 1 of the operating points. Thus, the control signal shown in Figure E.5d becomes oscillatory around that time period. In contrast, HCS and HWCS provides much better performance management under ramp workloads of class$_1$. However, HWCS provides much better performance due to the compensation of input and output nonlinearity under gradual workload changes of both classes.

***Case M**: Performance and resource management under real-world workload*

In this case, we apply a workload composed from a real-world workload trace of a software system. Many workload traces from real world applications can be accessed from `http://ita.ee.lbl.gov/html/traces.html`. These workload traces are also used in the existing literature to evaluate the performance management capabilities of the control systems designed to manage software systems. For this experiment we used the workload traces of *EPA web server* `http://ita.ee.lbl.gov/html/contrib/EPA-HTTP.html`, which has workload rates that can be managed by the shared resource environment used in the previous sections, without any modifications or scaling to the workloads. After decoding the workload rates of the files, the workloads were composed for 2250 samples for class$_0$ and class$_1$ as illustrated in Figure E.6. Figure E.7 shows the outputs and control signals of the control systems.

The statistics of the control systems are summarized in Table 5.4. It is evident that both nonlinear control systems provide better performance (see SSE statistics) compared

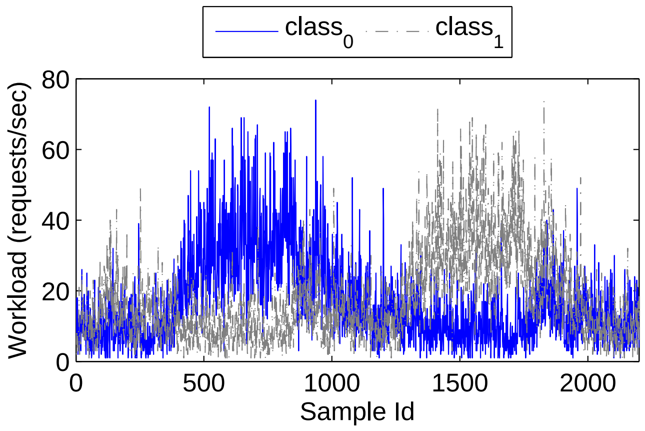
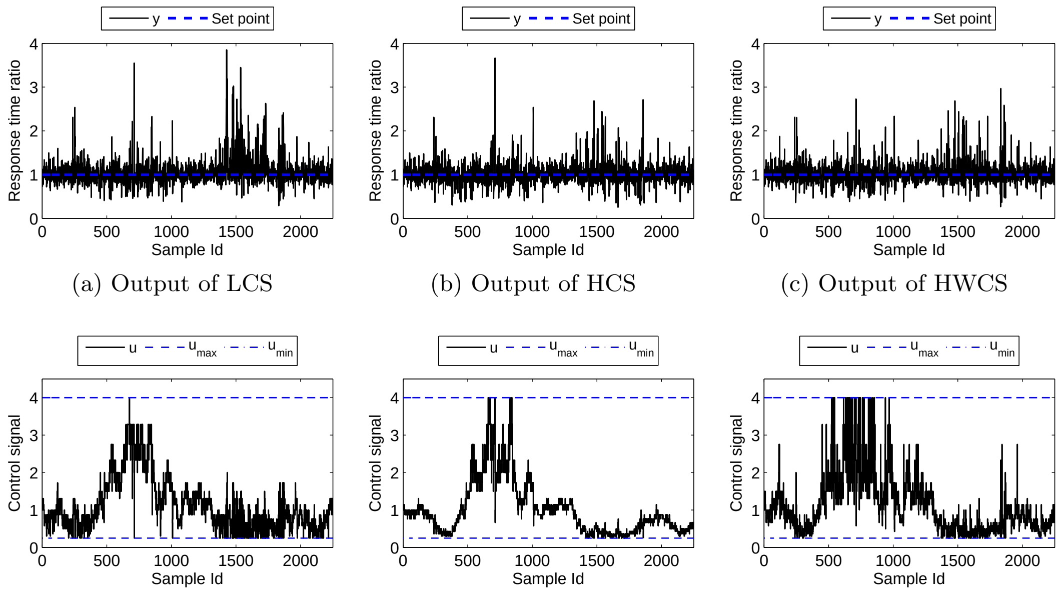


Figure E.6: Real world workloads for class_0 and class_1



(a) Output of LCS

(b) Output of HCS

(c) Output of HWCS

(d) Control signal of LCS

(e) Control signal of HCS

(f) Control signal of HWCS

Figure E.7: Performance management under real world workload (Case M)

to LCS. Further, overshooting or the output bounds are much larger in the case of LCS as well. A performance degradation is caused around the 1500th sample when class_1 workload demands more resources, where LCS shows vulnerability. The SSE statistics of HCS is slightly smaller compared to HWCS however, min and max statistics are much better in the case of HWCS.

E.2 Absolute Performance Management of Two Classes Using MIMO Wiener Nonlinear Control

In this section, additional cases are investigated for absolute management using MIMO Wiener control system designed in Chapter 5. The same settings used in Section 5.5 will be used here as well.

Case E*: Performance and resource management under ramp workloads*

In this case, we increase the workload gradually in a ramp like fashion. Workload conditions of *Case L* in Section E.1.3 will be used. Figure E.8 shows the output and control signals of the control systems.

Under the ramp workload the resources caps are gradually increased in order to achieve the control objectives by both control systems. The statistics in Tables 5.7, 5.8 indicate better performance and resource management provided by WMPC. Further, the control signals generated by the WMPC is less oscillatory compared LMPC, consequently much better steady state behavior and disturbance rejection capabilities can be observed. The compensation of nonlinear behavior has improved the performance management under the ramp workload conditions as well.

Case F*: Performance and resource management under real-world workload*

In this case, we apply a workload trace from a real-world web server (same as in *Case M* of Section E.1.3). Figure E.9 shows the outputs and control signals of the control systems.

The noisy workloads of the real-world workload trace demand efficient disturbance rejection capabilities from the management systems. The control signals of LMPC shows significant oscillatory behavior compared to WMPC under these workload conditions as well. The steady state performance of WMPC is better compared to LMPC from the statistics of the Tables 5.7, 5.8. However, there are only slight improvements in statistics because the workload did not have sudden fluctuations making the control systems to operate in the sensitive region of the response time curve, where the compensation of the WMPC becomes more useful.

E.3 Relative Performance Management of Two Classes Using MMST Adaptive Control

In this section, further cases are investigated for relative management based on MMST adaptive control. The same settings of Chapter 6 will be used. These cases also corresponds to the cases in Section E.1.

Case G*: Performance in the nominal region*

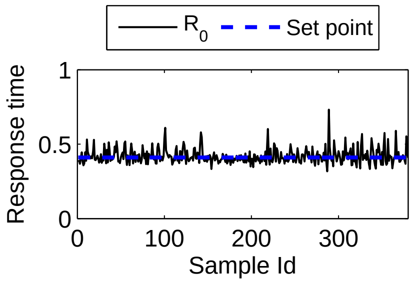
Figure E.10 shows the performance of the control systems. It is evident that the MMST-T2 and ACS provide better performance in this case even with the sudden workload increase at the 50th sample. There are deviations from the set point in the case of MMST-T4 around the 50th sample. The model switching signal shown in Figure E.10d indicates that the MMST-T2 has used model-0 and controller-0. In contrast, MMST-T4 has used the adaptive model most of the time (see Figure E.10e). This is an indication that the adaptive model has captured the dynamics not covered by either model 0 and 1. Although MMST-T2 has used model-0 which estimates the dynamics of region-0, the performance of that control system is much better under changing workload conditions. In the case of MMST-T2, a model switching was observed around the 24th sample. However, the performance was not affected because of the selection of velocity form which implements bump-less transfers.

Case L*: Performance and resource management under ramp workloads*

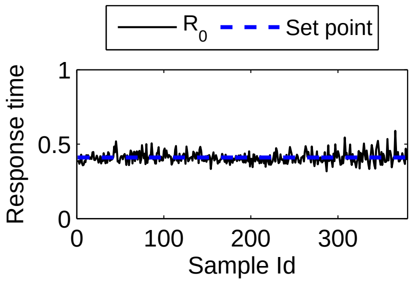
The results under this case are shown in Figure E.11 and Table 6.2. In contrast to Case A, where workload conditions change instantaneously, here the workload conditions change gradually. Such conditions could lead to significant performance issues in the switching algorithm because the overlapping of the regions, which the models were estimated. However, none of those issues were observed in the case of MMST-T2. The frequent chattering occurred during the short time periods of this experiment did not cause any performance degradation, indicating much better performance compared to LCS. In addition, the gradual changes in the workload improved the performance of MMST-T4 and ACS. However, when the system gradually moves to region-1, where the input nonlinearity is severe both control systems showed large transient responses compared to MMST-T2. The transient responses of ACS are significantly larger than MMST-T4, because the switching algorithm selected model-1 which is suitable to operate in that region, improving the performance of MMST-T4.

Case M*: Performance and resource management under real-world workload*

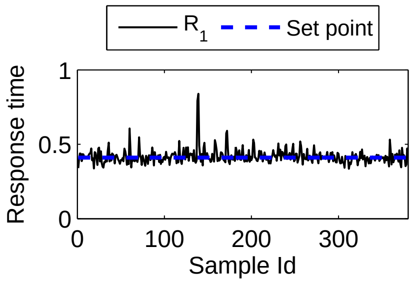
In this case, MMST-T2 control system which combines two models and controllers shows better performance than LCS, MMST-T4 and ACS. Due to the noisy workload the switching decisions made by MMST control systems show chattering (see Figures E.12d and E.12e). The bump-less transfers implemented by the controllers improved the performance of MMST-T2 compared to the other control systems. The MMST-T4 and ACS also provides better performance in this case. Although MMST-T4 showed significant chattering it shows better performance compared to both LCS and ACS.



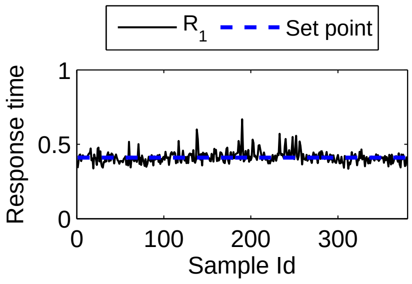
(a) Output R_0 of MIMO LMPC



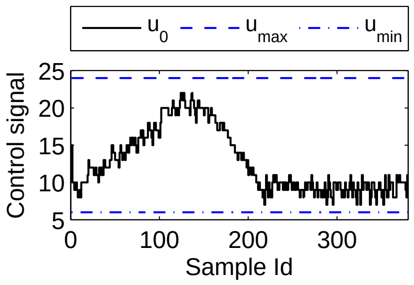
(b) Output R_0 of MIMO WMPC



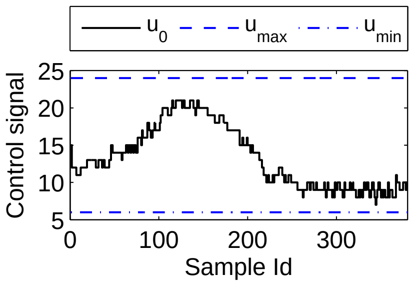
(c) Output R_1 of MIMO LMPC



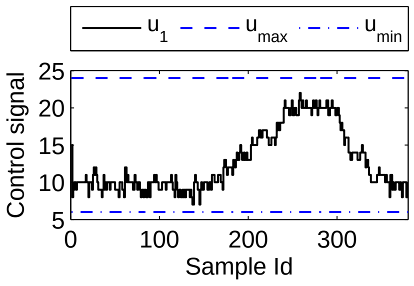
(d) Output R_1 of MIMO WMPC



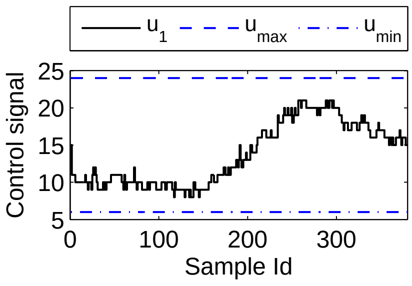
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC

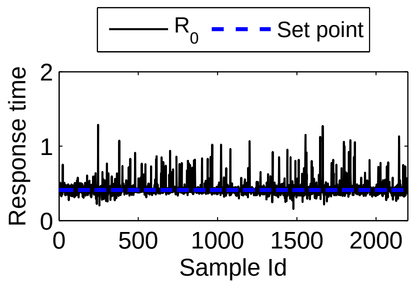


(g) Control signal u_1 of MIMO LMPC

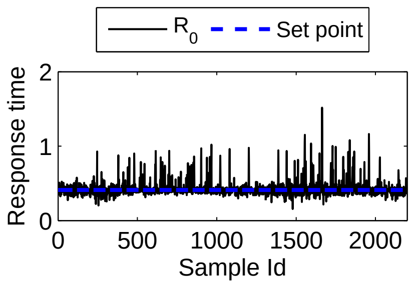


(h) Control signal u_1 of MIMO WMPC

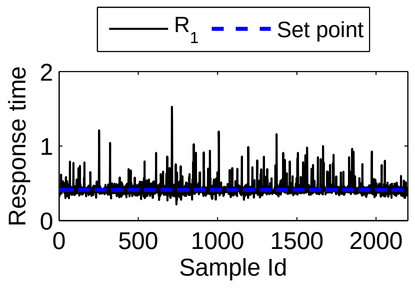
Figure E.8: Performance management under Ramp workload (Case E)



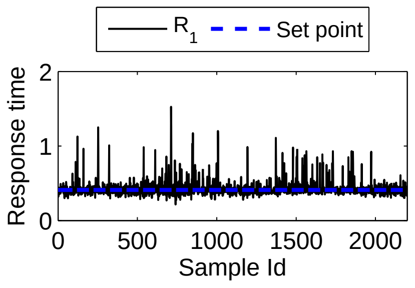
(a) Output R_0 of MIMO LMPC



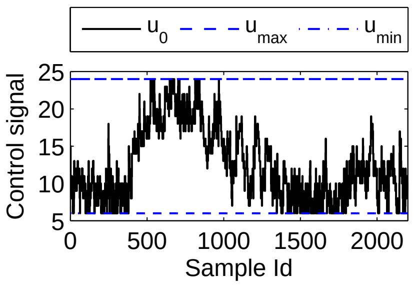
(b) Output R_0 of MIMO WMPC



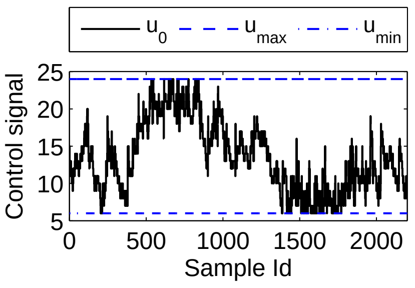
(c) Output R_1 of MIMO LMPC



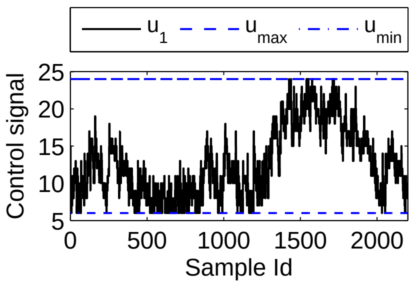
(d) Output R_1 of MIMO WMPC



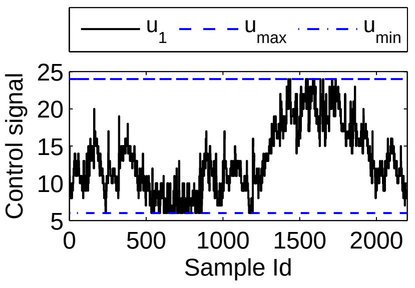
(e) Control signal u_0 of MIMO LMPC



(f) Control signal u_0 of MIMO LMPC

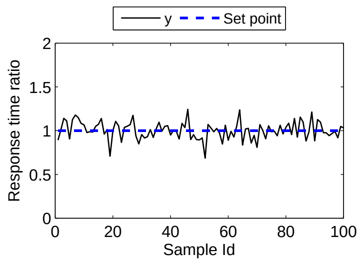


(g) Control signal u_1 of MIMO LMPC

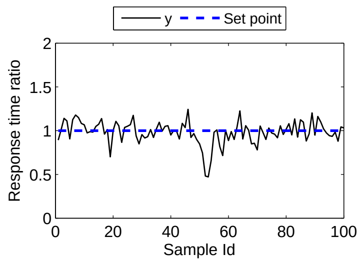


(h) Control signal u_1 of MIMO WMPC

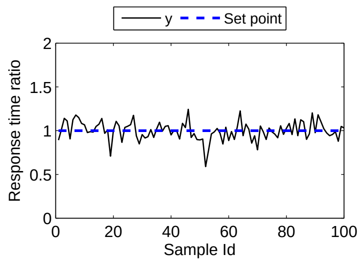
Figure E.9: Performance management under real-world workload trace (Case F)



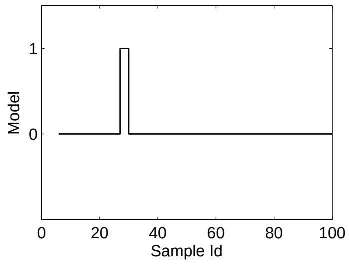
(a) Output of MMST-T2



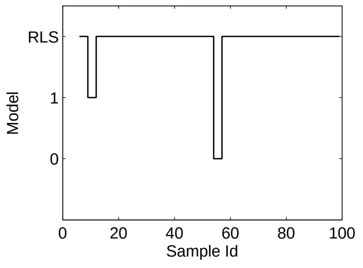
(b) Output of MMST-T4



(c) Adaptive

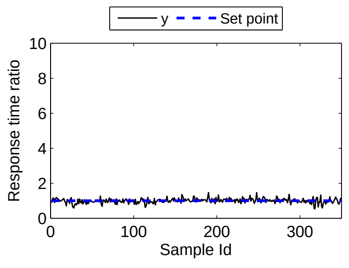


(d) Model switching signal of MMST-T2

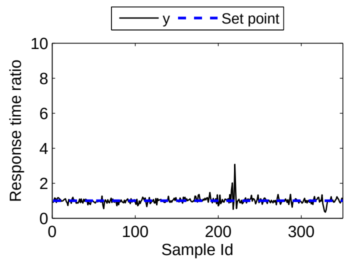


(e) Model switching signal of MMST-T4

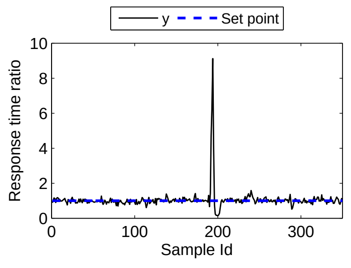
Figure E.10: Performance of the control systems in the nominal region (Case G)



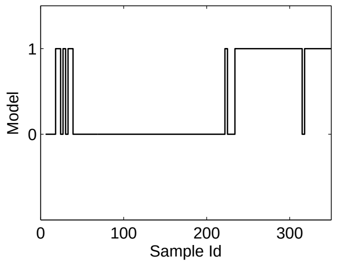
(a) Output of MMST-T2



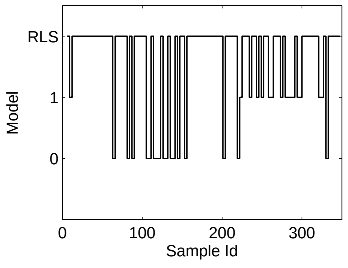
(b) Output of MMST-T4



(c) Output of ACS

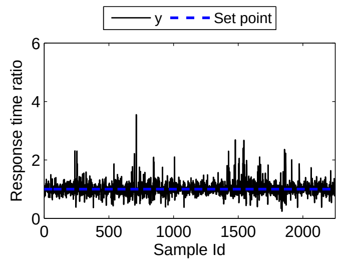


(d) Model switching signal of MMST-T2

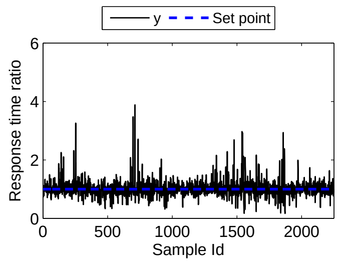


(e) Model switching signal of MMST-T4

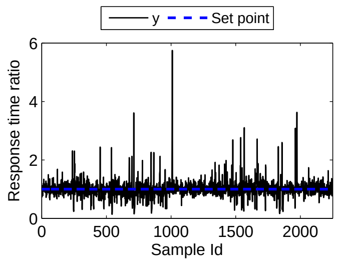
Figure E.11: Performance and resource management under ramp workloads (Case L)



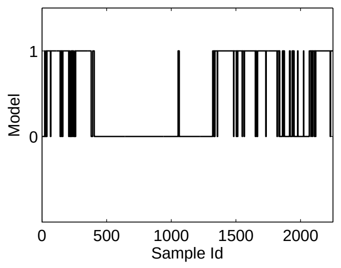
(a) Output of MMST-T2



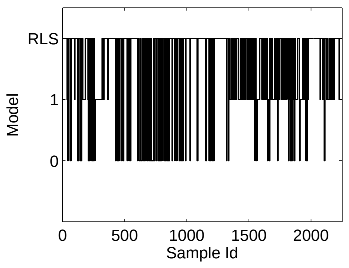
(b) Output of MMST-T4



(c) Output of ACS



(d) Model switching signal of MMST-T2



(e) Model switching signal of MMST-T4

Figure E.12: Performance management under real-world workloads (Case M)

Appendix F

Nonlinear Control of Multi-Class System With Three Classes

In Chapters 4, 5 and 6, the proposed nonlinear model estimation and control system design approaches were applied in a multi-class system with two classes. In order to show how these approaches scale for a system with more than two classes, in this appendix we utilize a multi-class shared resource system with three classes to apply the nonlinear model estimation and control system design techniques and then evaluate the management capabilities of the control systems under versatile conditions. The configuration details of the shared resource system with three classes are as follows.

Assume a shared resource environment with three classes, namely $class_0$, $class_1$ and $class_2$. Let us denote the resource caps as $S_0(k)$, $S_1(k)$ and $S_2(k)$, where $S_0(k) + S_1(k) + S_2(k) = S_{total} = 30$ and $S_{0,min}, S_{1,min}, S_{2,min} = 6$. $R_0(k)$, $R_1(k)$ and $R_2(k)$ are the response times of these classes respectively at the k^{th} sample.

F.1 Relative Performance Management of Three Classes Using Hammerstein-Weiner Nonlinear Control

This section validates the scalability of the proposed Hammerstein-Wiener model based control approach, by applying it on a system serving three classes defined above. The input variables of the system are defined as $u_1(k) = \frac{S_0(k)}{S_1(k)}$ and $u_2(k) = \frac{S_1(k)}{S_2(k)}$ and output variables of the system are defined as $y_1(k) = \frac{R_1(k)}{R_0(k)}$ and $y_2(k) = \frac{R_2(k)}{R_1(k)}$ with respect to relative management scheme. The main control objective of the control system is to maintain the outputs ($y_1(k)$, $y_2(k)$) around $\frac{P_1(k)}{P_0(k)}$ and $\frac{P_2(k)}{P_1(k)}$ (i.e. the reference signals) depending on the performance differentiation factors of the classes $(P_0(k), P_1(k), P_2(k))$.

Section F.1.1 covers the model estimation and control system design process. Section F.1.2 presents the comparative evaluation of the designed nonlinear control system with the existing linear control system.

F.1.1 Model Identification and Control System Design

With the settings of three classes the system becomes MIMO, compared to a system serving two classes. However, as described in Chapter 4, the control system design is performed considering only two classes. Then, the same control system is used to control consecutive pairs of classes in the system. The model estimation process, explained in Section 4.2.4, can be used for this design as well. Here, we design a control system for $class_0$ and $class_1$ pair, assuming that the $S_{2,min} = 6$ resource cap is guaranteed for $class_2$. This is the only alteration to the design process, rest of the design procedure remains the same as for the case of a system with only two classes.

Input nonlinear compensator design: When $S_{2,min} = 6$ is guaranteed, $class_0$ and $class_1$ pair gets total of 24 resources to share. Then, the same procedure explained in Section 4.2.4 is followed, where the set of operating points for the control input u can be computed as $\frac{6}{18}, \frac{7}{17}, \ldots, \frac{17}{7}, \frac{18}{6}$. Lets select v_{min} and v_{max} as -6 and 6 respectively, deriving $\delta v = 1$. This formulates $v = -6, \ -5, \ldots, 0, \ldots, 5, \ 6$. The mapping of the points of u and v is shown in Figure F.1. The relationship of u and v is then modeled using a 4th order polynomial (f^{-1}) with a goodness of (R^2) fit of 1. Equation (F.1) shows the structure of the model and Figure F.1 shows the model fit.

$$\begin{aligned} u(k) &= f^{-1}(v) \\ &= 0.0001v(k)^4 + 0.0016v(k)^3 + 0.0130v(k)^2 + 0.1620v(k) + 1.0010 \end{aligned} \tag{F.1}$$

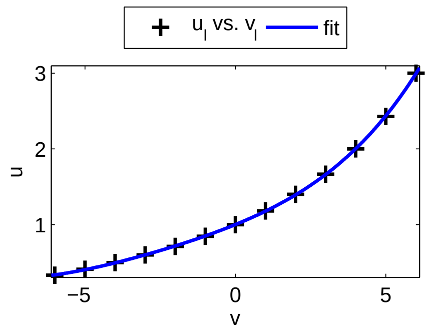


Figure F.1: Mapping of the u_l and v_l data pairs for system with three classes

Output nonlinear compensator design: After the integration of input static nonlinear compensator, a sinusoidal signal was designed with the possible values of v and a switching frequency of 2 samples. A 25 requests/sec workload for each class was applied and the output was observed for 600 sample periods. First order ARX model with a fifth order polynomial was sufficient to represent the Wiener model with $R^2 = 0.84$. Figure F.2 shows the predicted output in comparison with the test set data.

The inverse static nonlinear model at the output ($w = g^{-1}(y)$) was then estimated

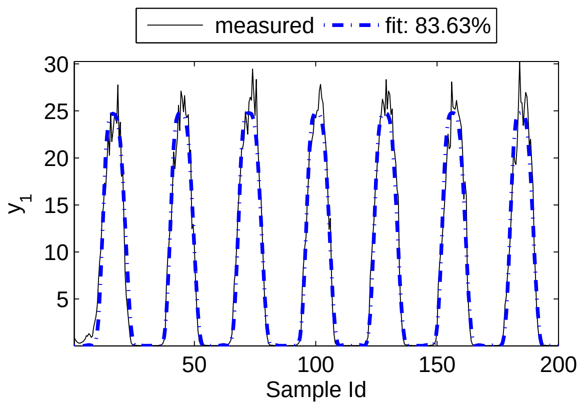


Figure F.2: Model fit for the case of (n,m,d,r) = (1,1,0,5)

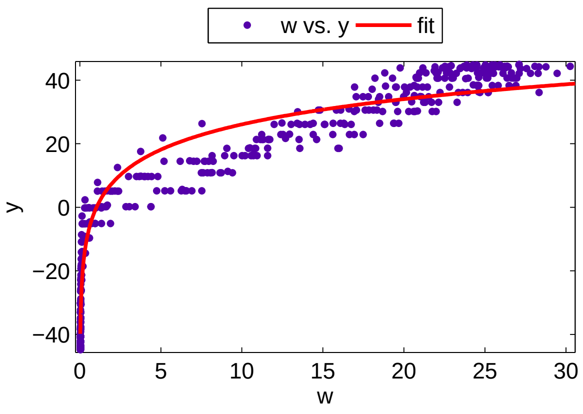


Figure F.3: Model fit of inverse nonlinear compensator function

using the $w - y$ data with R^2 =0.95 fit (see Figure F.3) as follows:

$$\begin{aligned} w &= g^{-1}(y) \\ &= 11.48log(y) - 0.36 \end{aligned} \tag{F.2}$$

Linear model estimation: The second identification experiment was designed after the integration of the compensators at the input and output to estimate the gains of the linear component. A pseudo random input signal and 25 requests/sec was used to simulate each class workload in this experiment. A first order ARX model was fitted to the data with $R^2 = 0.90$. The estimated linear model is shown in equation (F.3).

$$w(t+1) = 0.81w(t) + 1.65v(t) \tag{F.3}$$

Controller design: Using the linear model estimated (see equation (F.3)) and the pole-placement design methodology, we calculated the gains (K_i and K_p) for the controller. The closed-loop poles were placed at ($\alpha = 0.5$, $\beta = 0.5$). Also, $v_{min} = -6$ and $v_{max} = 6$ were set as the saturation limits in the controller.

According to relative management scheme as defined in Chapter 5, subsequent to the implementation of two nonlinear compensators and controller for class$_0$ and class$_1$ pair, a similar closed-loop system is integrated to manage class$_1$ and class$_2$ pairs as well. The final control system is shown in Figure F.4.

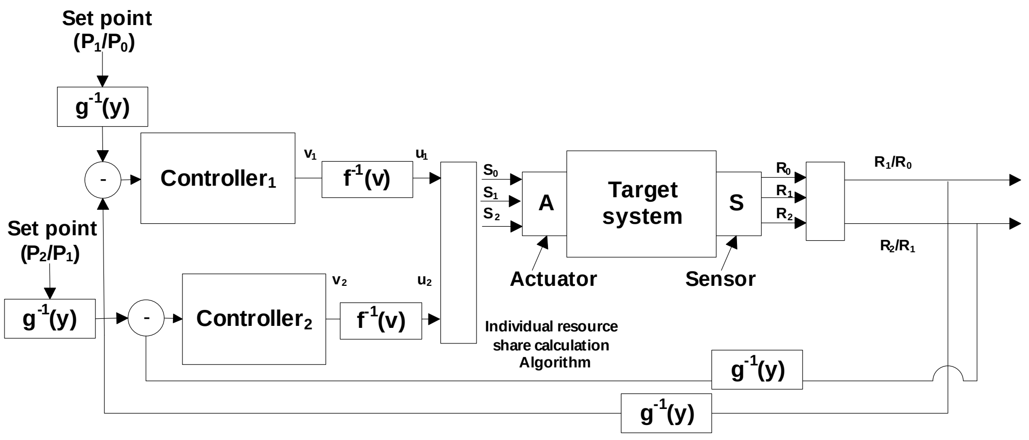


Figure F.4: The closed-loop control system for a system with three classes

F.1.2 Evaluation

This section evaluates the performance of the Hammerstein-Wiener nonlinear control system, which includes two controllers (namely, HWCS$_1$ and HWCS$_2$) under different operating conditions and business requirements. In order to compare the performance, we also implemented a Hammerstein model based control system, which includes two controllers (namely, HCS$_1$ and HCS$_2$) and a linear model based control system which also includes two controllers (namely, LCS$_1$ and LCS$_2$). The parameters of the controllers are listed in Table F.1.

Table F.1: Parameters of the control systems

Parameter	HWCS$_1$, HWCS$_2$	HCS$_1$, HCS$_2$	LCS$_1$, LCS$_2$
K_p	0.34	2.71	0.78
K_i	0.15	1.04	0.35
Min situation limit	-6	-6	0.33
Max situation limit	6	6	3

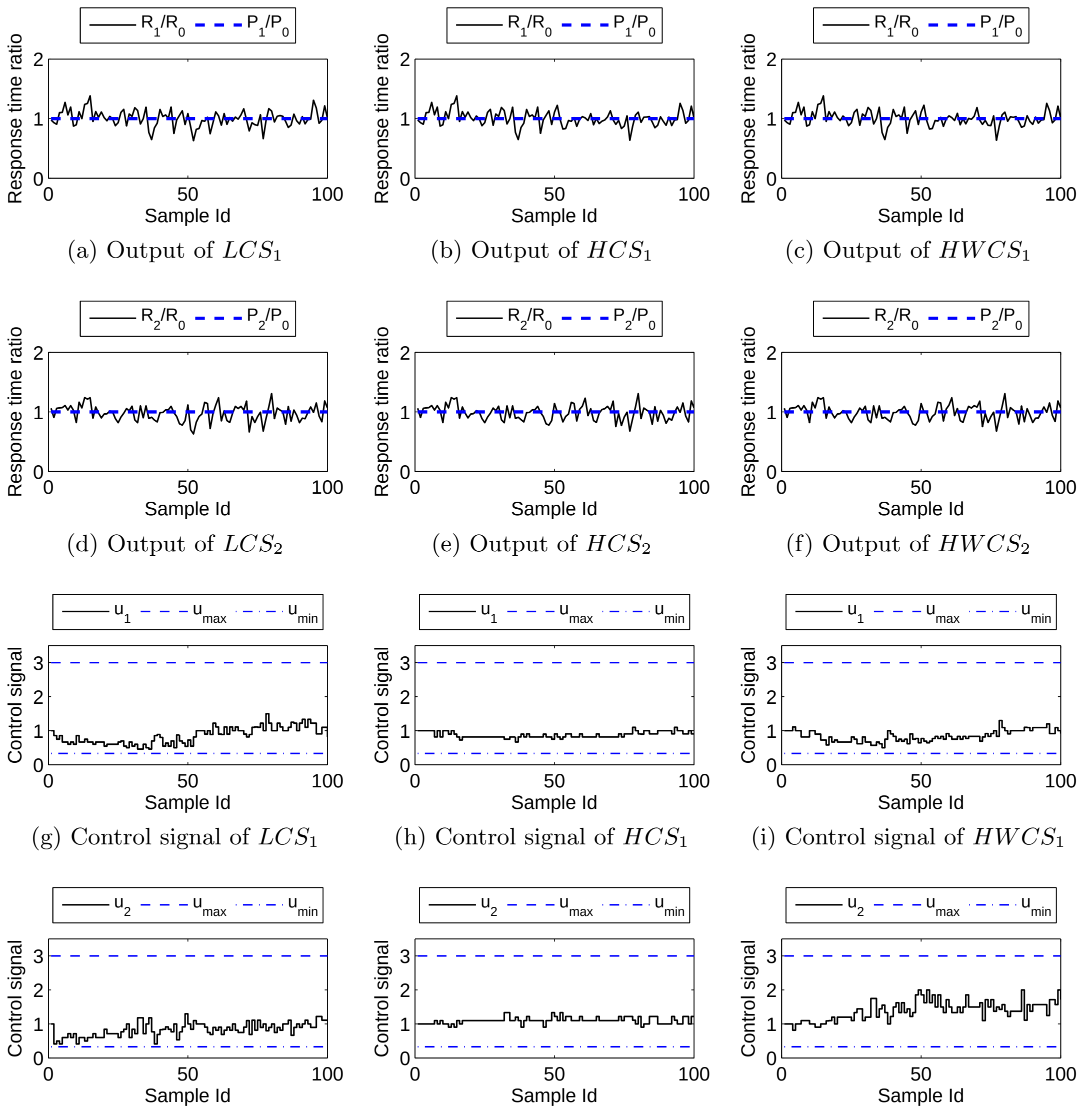
In the following subsections, the management capabilities of the above control systems are investigated under different conditions. These cases also corresponds to the operating conditions and objectives listed in Table 5.3 of Chapter 5, which includes workload condition near full capacity and workloads of extreme overloads.

F.1.3 Workloads of Full Capacity or Lower

Here, we evaluate the performance management of the designed control systems in different workload conditions and system objectives, while maintaining the workloads at the full capacity or lower.

***Case A**: Performance in the nominal region*

In this case, all the classes are assumed to be equally important, that is $P_0 : P_1 : P_2 = 1 : 1 : 1$. In order to maintain both controllers integrated to the control system in the nominal region (where the input and output nonlinearities are not prominent), similar workload intensities are maintained. Experiment starts with all three classes applying 10 requests/sec workload till the 50th sample, and then all increase their workload to 20 requests/sec. Figure F.5 shows the responses of the control systems.



(a) Output of LCS_1 (b) Output of HCS_1 (c) Output of $HWCS_1$

(d) Output of LCS_2 (e) Output of HCS_2 (f) Output of $HWCS_2$

(g) Control signal of LCS_1 (h) Control signal of HCS_1 (i) Control signal of $HWCS_1$

(j) Control signal of LCS_2 (k) Control signal of HCS_2 (l) Control signal of $HWCS_2$

Figure F.5: Performance management in the nominal region (Case A)

Table F.2 summarizes the statistics of the outputs of all control systems. There is not much difference in performance management in this case, because both controllers in each control system operate around the nominal region where input and output nonlinearities are not that severe. Therefore, the linear and nonlinear control systems provide similar performance in this condition.

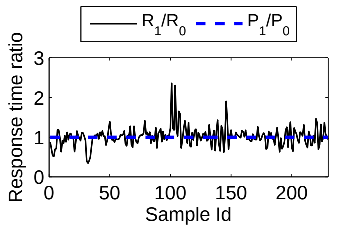
***Case B**: Performance away from the nominal region*

Again we set $P_0 : P_1 : P_2 = 1 : 1 : 1$ and make the system to operate in high workload demands of each class separately. The experiment starts off with each class$_0$, class$_1$ and class$_2$ applying 18 requests/sec workloads and remains at that level unless otherwise specified. During the 30th to 80th sample class$_0$ increases its workload to 30 requests/sec. Afterwards, during the 100th to 150th sample workload of the class$_1$ increases to 30 requests/sec. Finally at the 170th sample class$_2$ workload is increased to 30 requests/sec. These workload settings force the control systems to operate in the regions where input and output nonlinearities are severe. The performance of the control systems is shown in Figure F.6.

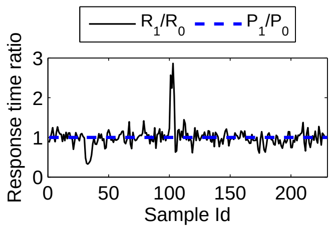
The management issues observed in Section 5.3.1 are also evident when the linear control system operates away from the nominal region. In addition, the performance at the startup is also significantly poor in LCS_1 and LCS_2. This is because, LCS_2 moves to the region where input nonlinearity is severe due to the startup disturbance affecting both control systems. Then, when the workload of class$_0$ is introduced into the system, the controllers take time to adjust the resource caps for all classes and settle to the appropriate resource caps. Overshooting in both outputs of the linear controllers are observed due to this reason. Then, again at the 100th and the 170th samples both LCS_1 and LCS_2 shows oscillatory behavior, because of the input nonlinearity. When the control signals of LCS_1 and LCS_2 are examined a highly oscillatory behavior is observed which led to the unstable behavior in both outputs y_1 and y_2. In contrast, the input nonlinearity compensated HCS_1 and HCS_2 shows much better steady state behavior with satisfactory performance after the workload disturbances of class$_1$ and class$_2$ at the 100th and the 170th sample instances respectively. However, the same controllers show less disturbance rejection capabilities at all the disturbances, due to the output nonlinearities, leading to large settling times and overshooting. The input and output nonlinearity compensated Hammerstein-Wiener control system shows significantly better steady state performance and disturbance rejection capabilities compared to both linear and Hammerstein control systems. See, Table F.2 for statistics.

***Case C**: When Class$_0$ is more important*

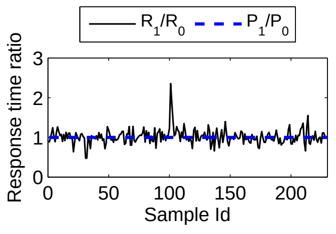
In this case, we implement performance differentiation factors as $P_0 : P_1 : P_2 = 1 : 1.5 : 2.25$, specifying the importance of the classes 0, 1 and 2 in the descending order. The experiment starts off with workload for class$_0$, class$_1$ and class$_2$ as 10, 20, 20 requests/sec respectively. However, at the 50th sample class$_2$ increases its workload to 30 requests/sec. Therefore, the performance of class$_0$ should not be affected by the high workloads of lower



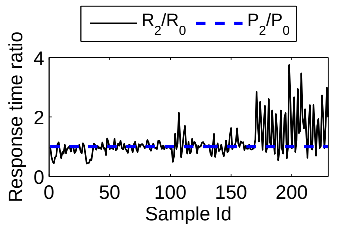
(a) Output of LCS_1



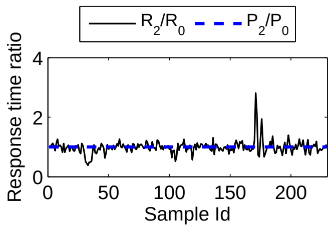
(b) Output of HCS_1



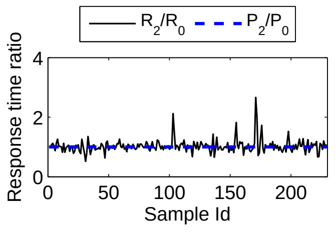
(c) Output of $HWCS_1$



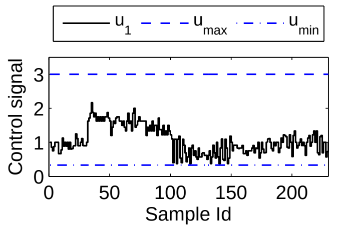
(d) Output of LCS_2



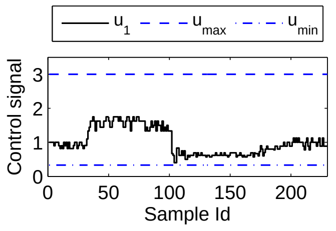
(e) Output of HCS_2



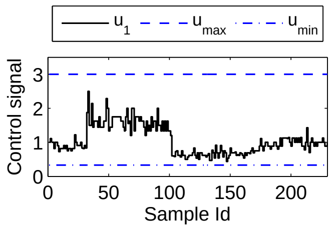
(f) Output of $HWCS_2$



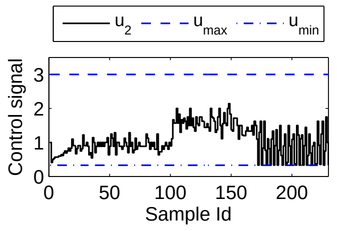
(g) Control signal of LCS_1



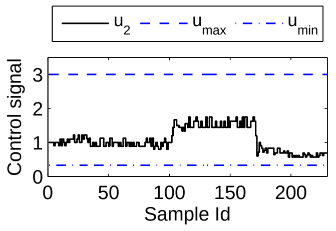
(h) Control signal of HCS_1



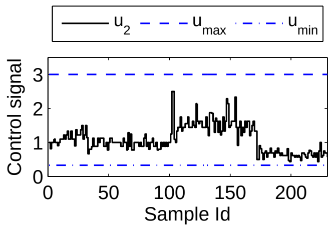
(i) Control signal of $HWCS_1$



(j) Control signal of LCS_2



(k) Control signal of HCS_2

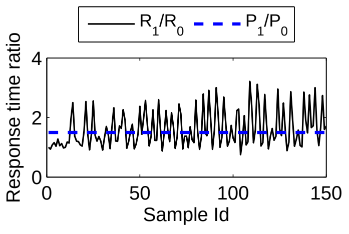


(l) Control signal of $HWCS_2$

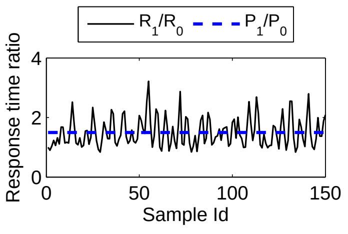
Figure F.6: Performance management away the nominal region (Case B)

priority classes. Figure F.7 shows the responses of the control systems.

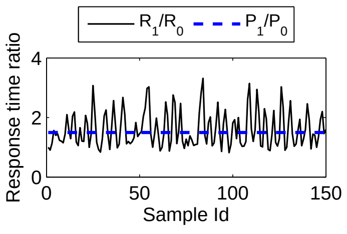
The LCS_1, HCS_1 and $HWCS_1$ achieves the set point with some steady state error. This is due to the queuing delays generated for the less priority class (class$_1$). Thus, the performance of the important class is maintained by these control systems. However, the LCS_2 shows significant performance issues for least priority class. The set point is not tracked at all after the workload disturbance of class$_2$ at the 50th sample, which leads to larger response time for class$_2$. This is again caused by the input and output nonlinearities exist in the system, which leads to highly oscillatory control signals in both controllers (see Figures F.7g and F.7j). In contrast, the nonlinear control systems show significantly better performance even under workload variations by tracking the reference signal of



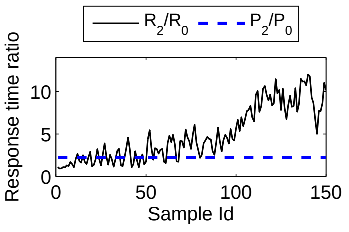
(a) Output of LCS_1



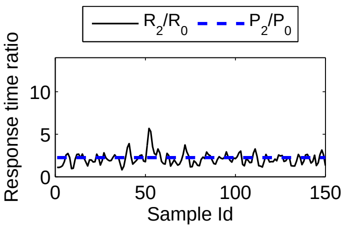
(b) Output of HCS_1



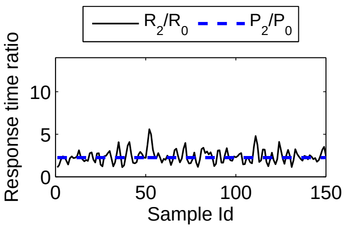
(c) Output of $HWCS_1$



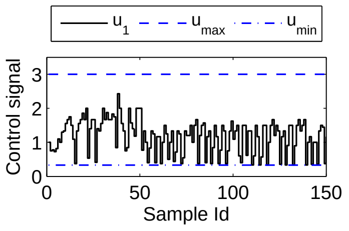
(d) Output of LCS_2



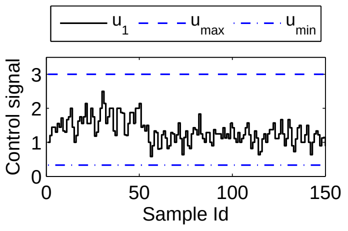
(e) Output of HCS_2



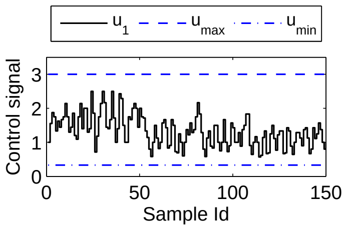
(f) Output of $HWCS_2$



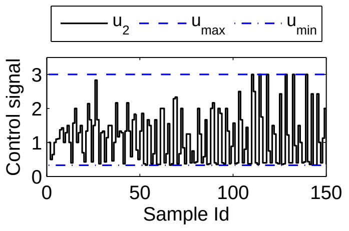
(g) Control signal of LCS_1



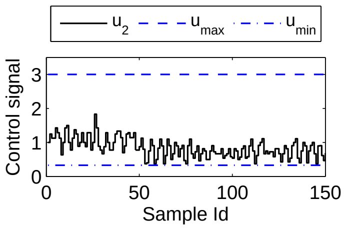
(h) Control signal of HCS_1



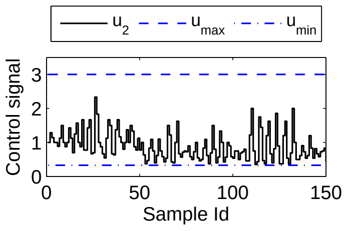
(i) Control signal of $HWCS_1$



(j) Control signal of LCS_2



(k) Control signal of HCS_2



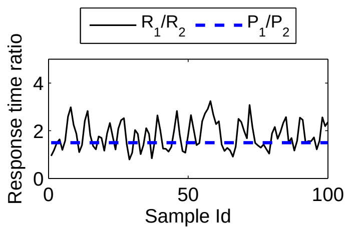
(l) Control signal of $HWCS_2$

Figure F.7: Performance management when Class$_0$ is more important (Case C)

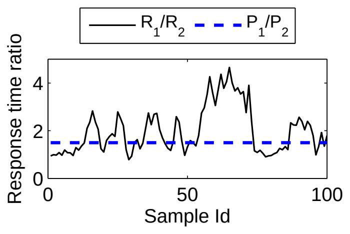
both outputs. However, HCS_1 and HCS_2 shows better performance since the output noise affect the nonlinear compensator in this condition (see Table F.2).

***Case D**: When Class$_2$ is more important*

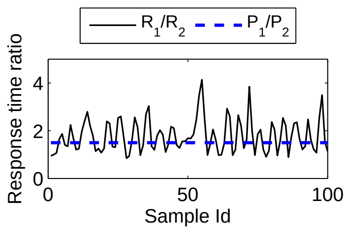
In this case, we implement performance differentiation factors as $P_2 : P_1 : P_0 = 1 : 1.5 : 2.25$, specifying the importance of the classes 0, 1 and 2 in the ascending order. These requirements are translated in to control systems as reference signals, where the reference signal for the controller managing class$_0$ and class$_1$ become $\frac{P_1}{P_0} = 0.67$ and the reference signal for the other controller is $\frac{P_2}{P_1} = 0.67$. The experiment starts off with workload for class$_0$, class$_1$ and class$_2$ as 20, 20, 10 requests/sec respectively. At the 50th sample class$_0$ increases its workload 30 requests/sec. Figure F.8 shows the performance of the control systems.



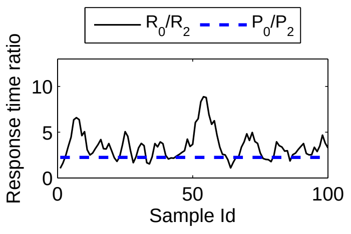
(a) Output of LCS_1



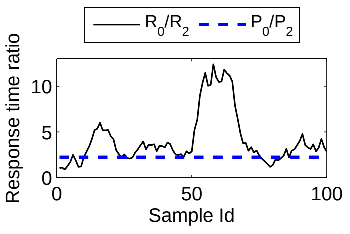
(b) Output of HCS_1



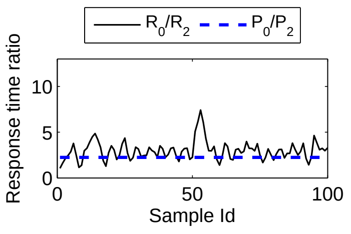
(c) Output of $HWCS_1$



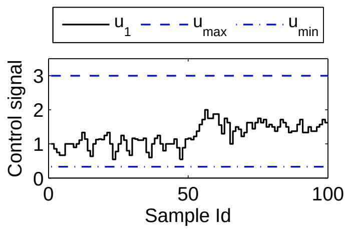
(d) Output of LCS_2



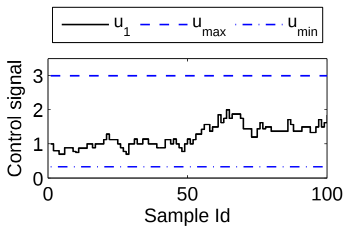
(e) Output of HCS_2



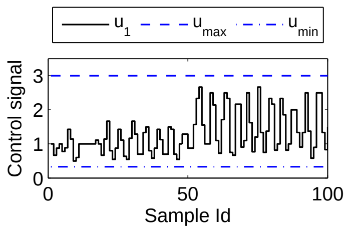
(f) Output of $HWCS_2$



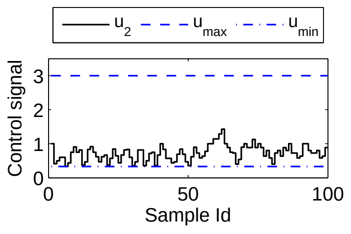
(g) Control signal of LCS_1



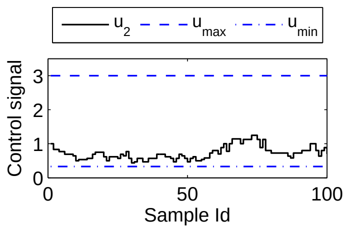
(h) Control signal of HCS_1



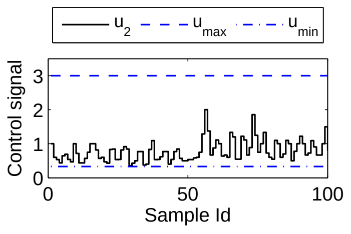
(i) Control signal of $HWCS_1$



(j) Control signal of LCS_2



(k) Control signal of HCS_2



(l) Control signal of $HWCS_2$

Figure F.8: Performance management when Class$_0$ is more important (Case D)

In this case, the reference signals are placed in the output region where variations damp out due to the output nonlinearity. Consequently, the control systems that do not compensate for the output nonlinearity perform poorly at the 50th sample due to the workload disturbance. For instance, Linear and Hammerstein control systems show large settling times because they take a large time to adjust the resource caps after the disturbance. This effect of the output nonlinearity is compensated by the $HWCS_1$ and $HWCS_2$ by tracking the reference signal much better, with significantly better reactions to the disturbances at the 50th sample (also see Table F.2).

F.1.4 Workloads of Extreme Overload

This section covers the cases when the workloads has persistently overloaded the system.

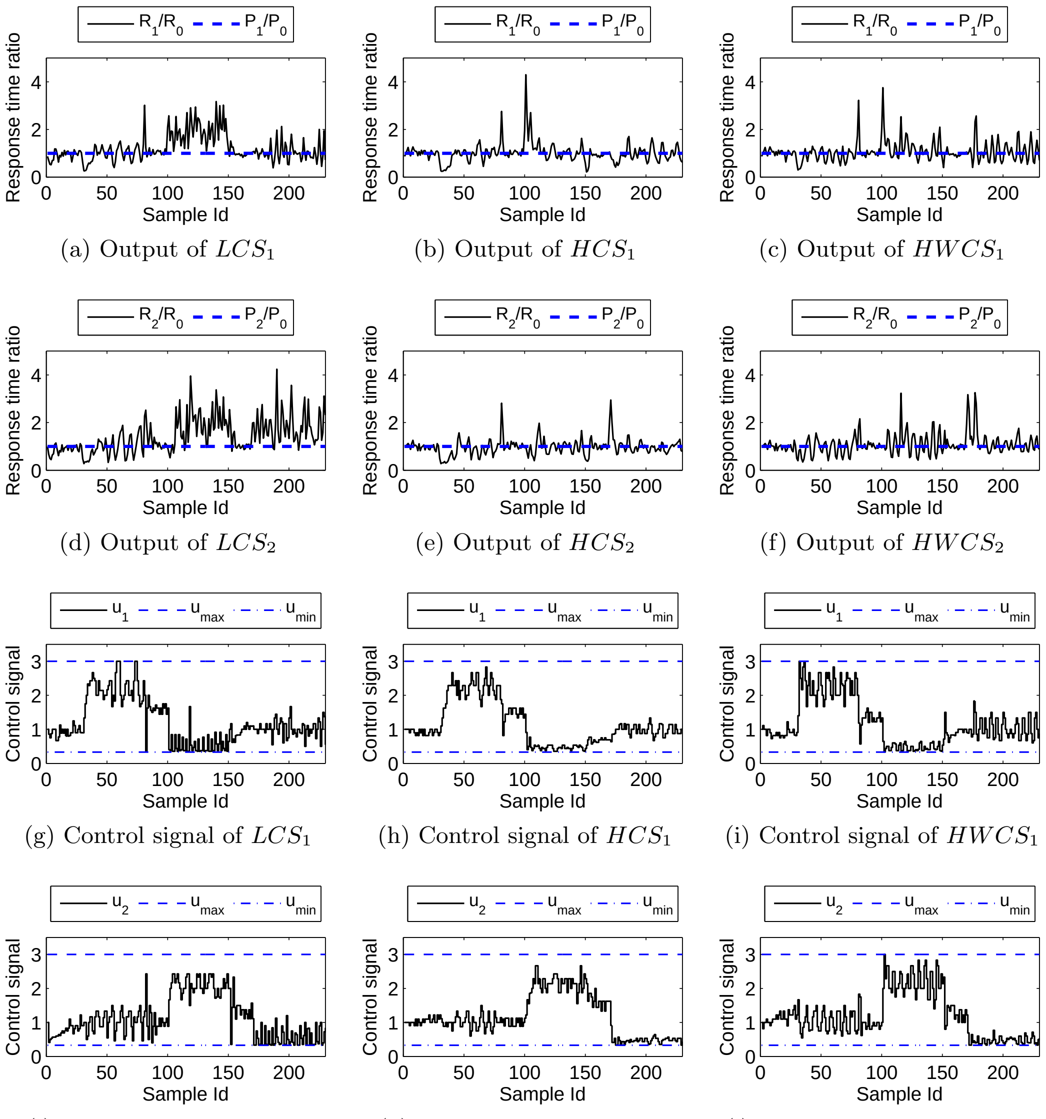
Case E: *Performance in away from the nominal region under overload*

This case is similar to Case B, where control systems are forced to operate in the regions that the nonlinearities are severe. In this case we maintained the same workload settings of Case B. However, in order to overload the system at the 30th sample workload of class_0 is increased to 60 requests/sec. Similarly, at the 100th and 170th samples class_1 and class_2 workloads are increased to 60 requests/sec, respectively to overload the system. Figure F.9 shows the outputs of the control systems.

The performance of LCS_1 and LCS_2 is again significantly poor in this case as well. When the large workloads of class_0 is applied on the system performance of the linear control system provides satisfactory performance because the system is operating in the region where input nonlinearities are not that severe (see behavior of the control signals of LCS_1 and LCS_2 till the 80th sample). However, with the disturbances of class_1 and class_2, the controllers are operating in the region where input nonlinearity affects the performance of the linear controllers, which leads to highly oscillatory control signals affecting the outputs subsequently. Due to the disturbances at the 100th and 170th sample, the reference signal was not maintained by both linear controllers leading to unstable behaviors. In contrast, the both nonlinear control systems achieve the required control objectives, however the oscillatory behavior in the outputs is because when a class overloads the system the performance of other classes have to be degraded as well to maintain equal response times for all classes. The disturbance rejection capabilities of HCS_1 and HCS_2 are poor compared to Hammerstein-Wiener control system because of the output nonlinearity. In particular, at the 30th and 170th samples larger overshooting were observed for the Hammerstein control system. The steady state performance of $HWCS_1$ and $HWCS_2$ is oscillatory because of the noisy output conditions compared to Hammerstein control systems. The same reason leads to large spikes at the outputs of Hammerstein control system because of the less reaction to the noisy conditions. The input and output nonlinearity compensated Hammerstein-Wiener control system provides much effective performance when the system is forced to operate in the regions where the nonlinearities are severe. Further, it provides less request rejection rates compared to both other control systems (see Table F.3).

Case F: *Performance under overload when* $\textit{Class}_0$ *is more important*

In this case, we implement performance differentiation factors as $P_0 : P_1 : P_2 = 1 : 1.5 : 2.25$, specifying the importance of the classes 0, 1 and 2 in the descending order. The experiment starts off with workload for class_0, class_1 and class_2 as 20, 50, 50 requests/sec respectively. That is the more important class sends less workload, while other two classes



(a) Output of LCS_1 (b) Output of HCS_1 (c) Output of $HWCS_1$

(d) Output of LCS_2 (e) Output of HCS_2 (f) Output of $HWCS_2$

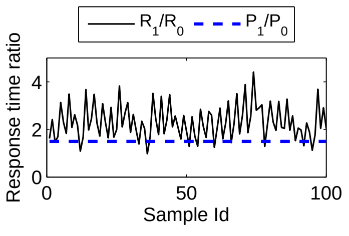
(g) Control signal of LCS_1 (h) Control signal of HCS_1 (i) Control signal of $HWCS_1$

(j) Control signal of LCS_2 (k) Control signal of HCS_2 (l) Control signal of $HWCS_2$

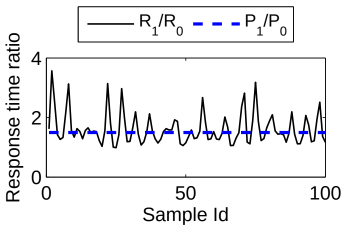
Figure F.9: Performance management when system is operating away from nominal region under overload (Case E)

have overloaded the system with high workloads. Figure F.10 shows the output and input signals of the control systems.

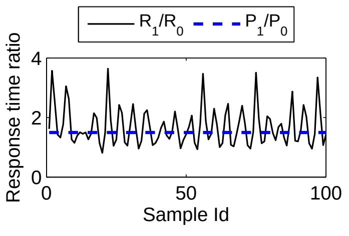
Again, the LCS_1 and LCS_2 do not achieve any of the specified control objectives. This is because, the LCS_1 has to deal with the resource caps which locate in the region where input nonlinearity is severe. This behavior ultimately affects the performance of the LCS_2 as well. However, both control systems with the input nonlinearity compensation provide significantly better performance management. The performance of Hammerstein-Wiener control system is slightly poor compared to Hammerstein control system, because of the noisy conditions affecting the performance of the output nonlinear compensation.



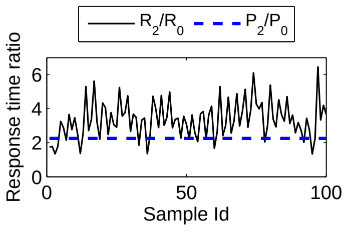
(a) Output of LCS_1



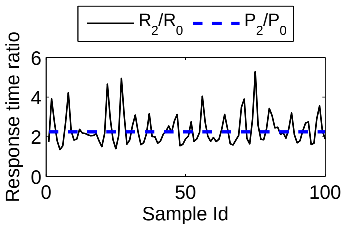
(b) Output of HCS_1



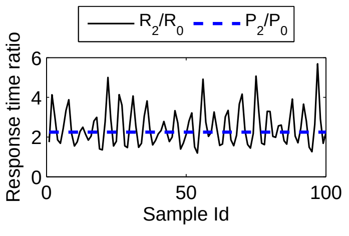
(c) Output of $HWCS_1$



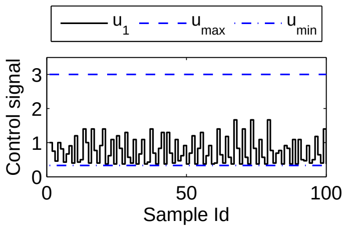
(d) Output of LCS_2



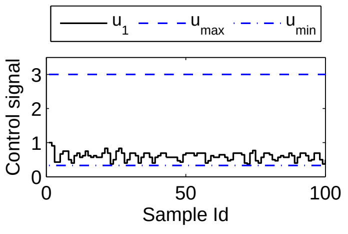
(e) Output of HCS_2



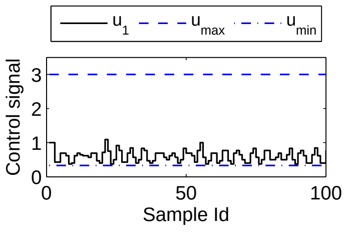
(f) Output of $HWCS_2$



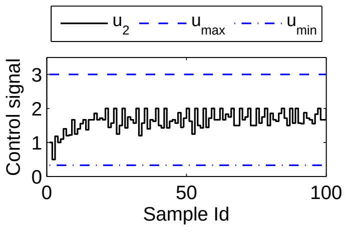
(g) Control signal of LCS_1



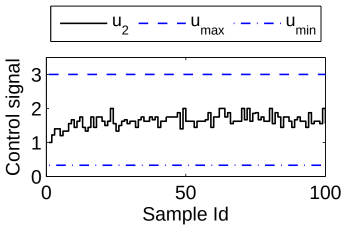
(h) Control signal of HCS_1



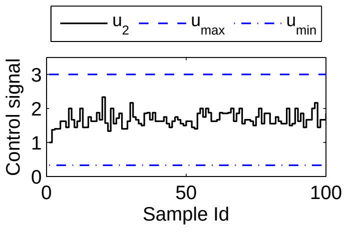
(i) Control signal of $HWCS_1$



(j) Control signal of LCS_2



(k) Control signal of HCS_2



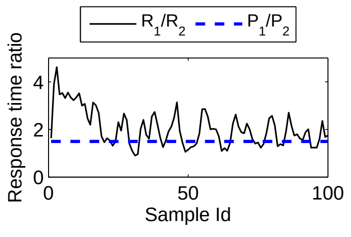
(l) Control signal of $HWCS_2$

Figure F.10: Performance management when Class_0 is more important (Case F)

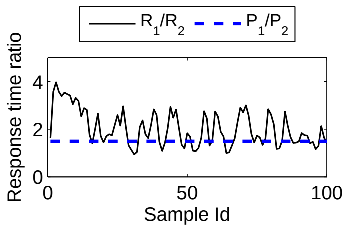
***Case G**: Performance under overload when Class_2 is more important*

In this case, we specify the importance of classes 0, 1 and 2 in the ascending order, implementing performance differentiation factors as $P_2 : P_1 : P_1 = 1 : 1.5 : 2.25$. The experiment starts off with workload for class_0, class_1 and class_2 as 50, 50, 20 requests/sec respectively. That is the more important class_2 sends less workload, while class_0 and class_1 have overloaded the system with high workloads. Figure F.11 shows the output and input signals of the control systems.

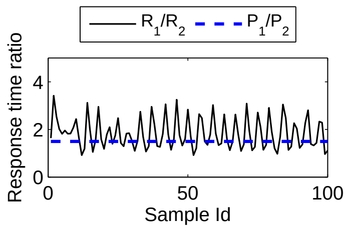
The first observation is that, the linear and Hammerstein control systems provides much poor disturbance rejection at the startup due to the output nonlinearity. In this case, the control systems have to operate in the region of output where the variations are



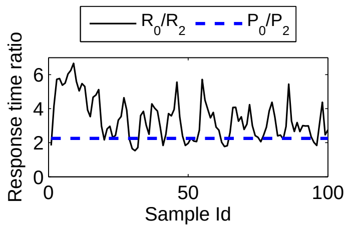
(a) Output of LCS_1



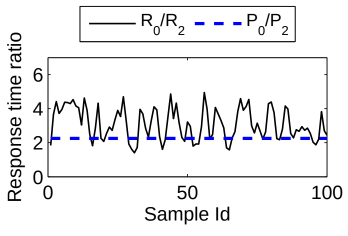
(b) Output of HCS_1



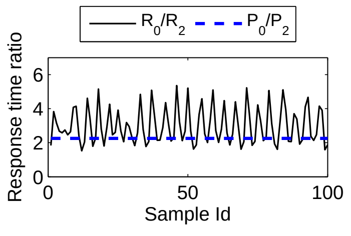
(c) Output of $HWCS_1$



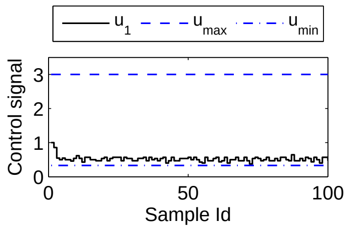
(d) Output of LCS_2



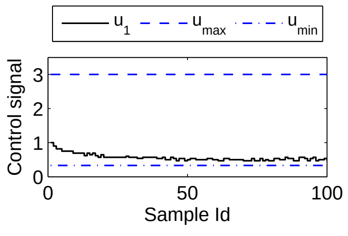
(e) Output of HCS_2



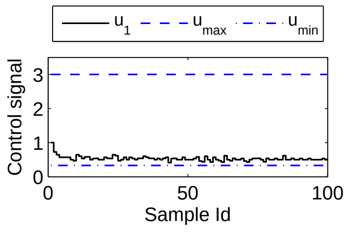
(f) Output of $HWCS_2$



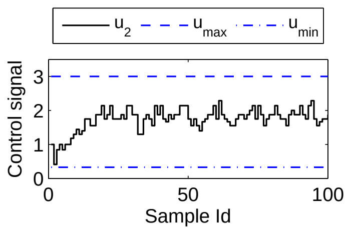
(g) Control signal of LCS_1



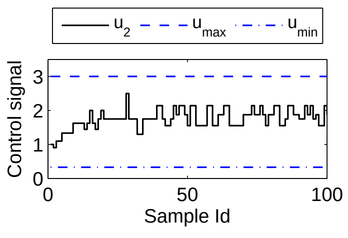
(h) Control signal of HCS_1



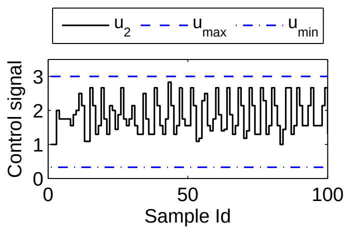
(i) Control signal of $HWCS_1$



(j) Control signal of LCS_2



(k) Control signal of HCS_2



(l) Control signal of $HWCS_2$

Figure F.11: Performance management of three class system when Class$_2$ is more important (Case G)

damped out, consequently the controllers take time to detect the variations at the outputs and adjust the resource caps. This is one of the contributing reasons for the LCS_1 to operate without oscillatory behavior even if it is operating in the region where input nonlinearity is severe. In contrast, the output nonlinearity compensated Hammerstein-Wiener control system provides much better performance at the steady state by reacting to the noisy conditions and adjusting the resource caps in order to achieve the specified control objectives.

Table F.2: Statistical summary of control systems managing three classes

Case	LCS			HCS			HWCS	
	y_1	y_2		y_1	y_2		y_1	y_2
A	1.888	1.884		1.65	1.551		1.64	1.543
B	13.858	70.78		15.602	11.582		8.021	11.108
C	34.895	92.152		31.156	107.811		40.932	74.042
D	57.386	2529.547		36.429	82.191		54.24	94.95
E	82.5	161.444		39.257	26.463		42.442	46.115
F	116.185	234.661		29.154	60.26		43.666	94.028
G	71.024	188.531		66.72	96.333		42.906	116.434

Table F.3: Summary of average loss rates of the control systems managing three class system

Case	LCS				HCS				HWCS		
Class	0	1	2		0	1	2		0	1	2
E	21	22	22		21	21	21		20	21	21
F	0	18	30		0	16	29	0	0	16	30
G	32	14	0		31	15	0	0	31	14	0

F.2 Absolute Performance Mmanagement With Three Classes Using MIMO Wiener Nonlinear Control

This section validates the scalability of the proposed MIMO Wiener model based control approach for absolute performance management by applying it on a shared resource environment with three classes described at the start of this appendix. The main control objective of the absolute management system is to maintain the response times $R_0(k)$, $R_1(k)$ and $R_2(k)$ of the workloads of $class_0$, $class_1$ and $class_2$ around $R_{SLA,0}(k)$, $R_{SLA,1}(k)$ and $R_{SLA,2}(k)$ respectively.

F.2.1 Model Identification and Control System Design

As illustrated in Section 4.3.4, a Wiener SISO experiment does not have to be conducted because the nonlinear behavior of these three classes was similar under high work load conditions. As a result the inverse output nonlinear compensator implemented in Section 4.3.7 can be used to compensate the nonlinearities of each class. A MIMO SID experiment is then conducted to capture the rest of the dynamics of inputs, outputs and their interactions. This MIMO linear model captures the relationships between u_0, u_1, u_2 and w_0, w_1, w_2. Similar to the case of two classes, a pseudo binary signal was designed selecting 8 and 10 as the resource caps for each of the input, with a switching frequency of 2 samples. 20 requests/sec workloads were applied for class$_0$, class$_1$, class$_2$ for 600 samples. The gathered data was used to design a first order MIMO ARX model as follows:

$$w(k+1) = \begin{bmatrix} 0.5698 & -0.0309 & -0.0284 \\ 0.0042 & 0.6037 & -0.0278 \\ -0.0075 & 0.0084 & 0.6326 \end{bmatrix} w(k) + \begin{bmatrix} 3.1353 & -1.1425 & -0.4338 \\ -0.9014 & 3.1412 & -0.9476 \\ -1.2425 & -1.9075 & 4.1422 \end{bmatrix} u(k), \quad \text{(F.4)}$$

where $w(k) = [w_0(k)\ w_1(k)\ w_2(k)]^T$ and $u(k) = [u_0(k)\ u_1(k)\ u_2(k)]^T$.

MIMO MPC design: Following the implementation process covered in Chapter 5, MPC (namely, WMPC) was developed using the model in equation (F.4) and relevant constraints. The parameters of the Laguerre network for the three input case were set at ($a_1 = 0.5$, a_2 =0.55 , a_2 =0.6 and $N_1, N_2, N_3 = 1$) and $N_p = 15$ and $u(0) = [10\ 10\ 10]^T$. R_w was set to $1000 \times I_{(3\times3)}$ after careful analysis because of the low model fit.

F.2.2 Evaluation

For the comparison purposes we designed a linear control system as well. The details are as follows:

MIMO linear model based control system: The linear model was estimated under the same settings used to construct the MIMO linear model of WMPC. Model fit was also low in this case as well (see equation (F.5).

$$y(k+1) = \begin{bmatrix} 0.4818 & -0.0189 & -0.0397 \\ -0.0058 & 0.5172 & -0.0494 \\ -0.0606 & -0.0052 & 0.5207 \end{bmatrix} y(k) + \begin{bmatrix} 0.1429 & -0.0131 & 0.0144 \\ -0.0011 & 0.1354 & -0.0018 \\ -0.0112 & -0.0385 & 0.1784 \end{bmatrix} u(k), \quad \text{(F.5)}$$

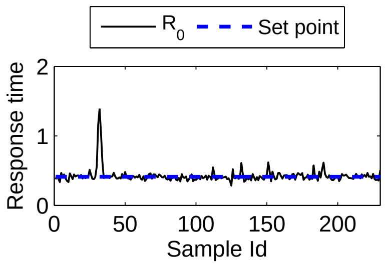
where $y(k) = [y_0(k)\ y_1(k)\ y_2(k)]^T$ and $u(k) = [u_0(k)\ u_1(k)\ u_2(k)]^T$.

All the parameters of the linear MPC were set to same values as WMPC. However, the parameters of the Laguerre network were set at ($a_1 = 0.45$, a_2 =0.45, a_2 =0.5). The R_w was set to $2 \times I_{(3\times3)}$.

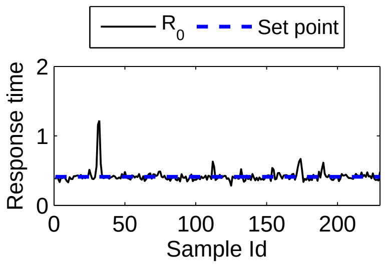
Case A*: High separate workloads from each class*

In this case, the references of all classes are set to 0.41 seconds. The experiment starts with class$_0$, class$_1$ and class$_2$ applying 15 requests/sec workloads each and remains at that level unless otherwise specified. During the 30th to 80th sample class$_0$ increases its workload to 35 requests/sec. Afterwards, during the 100th to 150th samples, the workload of the class$_1$ increases to 35 requests/sec. Finally at the 170th sample class$_2$ workload is increased to 35 requests/sec. With these disturbances the control systems are moved to sensitive region of the response time curve requiring effective disturbance rejection capabilities. Again the hypothesis is that WMPC provides much better performance with low overshooting without affecting the steady state behavior. The performance of the control systems is shown in Figure F.12.

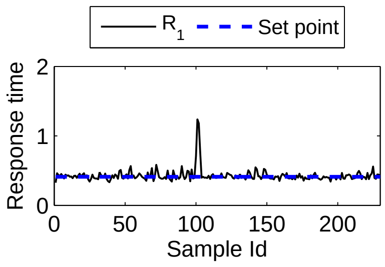
In this case as well we see the issues observed in Chapter 5. When the high disturbances of all three classes are applied on the system separately, the disturbance forces the control systems to operate in the sensitive region. The nonlinearity compensation done by the WMPC control system effectively rejects the disturbance faster than the LMPC without



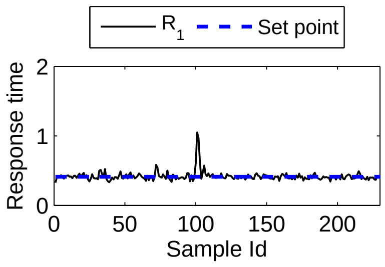
(a) Output R_0 of MIMO LMPC



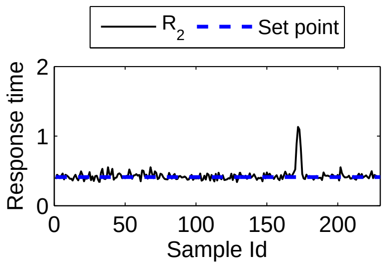
(b) Output R_0 of MIMO WMPC



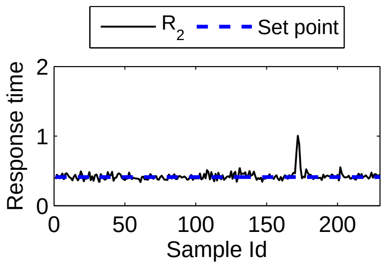
(c) Output R_1 of MIMO LMPC



(d) Output R_1 of MIMO WMPC



(e) Output R_2 of MIMO LMPC



(f) Output R_2 of MIMO WMPC

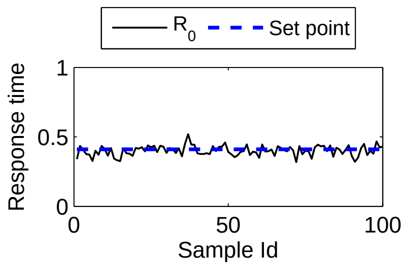
Figure F.12: Performance management under high separate workloads (Case A)

adversely affecting the steady state behavior.

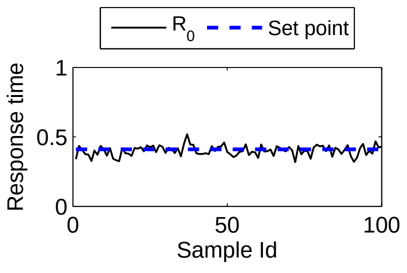
***Case B**: Different response time requirements*

In this case, we implement performance differentiation by setting $R_{SLA,0} = 0.41$, $R_{SLA,1} = 0.5$ and $R_{SLA,2} = 0.6$ seconds, specifying the importance of classes 0, 1 and 2 in the descending order. The reference signals are placed in the insensitive (class_0) and sensitive regions (class_1 and class_2) of the response time curve. The experiment starts off with workload for class_0, class_1 and class_2 as 10, 20, 20 requests/sec respectively. However, at the 50th sample class_2 increases its workload 30 requests/sec. Figure F.13 shows the responses of the control systems.

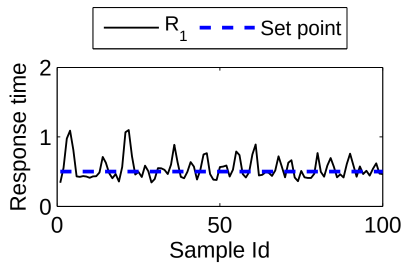
The behavior of the most important class (R_0) is similar in both control systems, which is placed in the insensitive region (see, Table F.4). However, again it is evident that the better performance of WMPC, in particular when the classes are placed in the insensitive region of the response time curve. It provides better disturbance rejection capabilities with lower steady state error for R_1 and R_2 compared to LMPC.



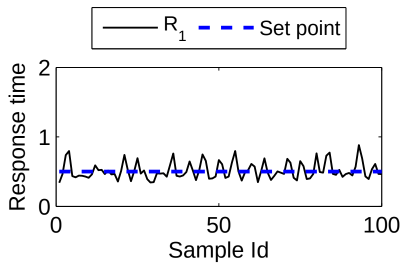
(a) Output R_0 of MIMO LMPC



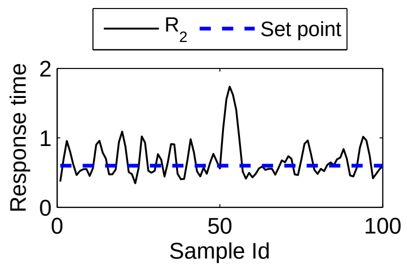
(b) Output R_0 of MIMO WMPC



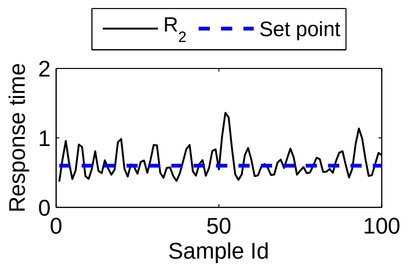
(c) Output R_1 of MIMO LMPC



(d) Output R_1 of MIMO WMPC



(e) Output R_2 of MIMO LMPC



(f) Output R_2 of MIMO WMPC

Figure F.13: Performance management under different response time requirements (Case B)

F.3 Relative Performance Management of Three Classes Using MMST Adaptive Control

In this section, we present the design details of MMST adaptive control scheme for relative performance management system of a system with three classes. The same design process covered in Section 6.2.3 is followed. That is a control system is designed for the first pair of classes, assuming that the minimal resource amounts are maintained for the other class. The control inputs for this control system are computed as $u = \frac{6}{18}, \frac{7}{17}, \dots, \frac{17}{7}, \frac{18}{6}$. Then, these operating points are divided into two regions similar to the case of system with two classes. Afterwards, two SID experiments are conducted and then models are estimated to represent region 0 and 1. Finally, two controllers, i.e., one aggressive controller and less aggressive controller are designed. The details of these controllers can be found in Table F.5. The next step is to decide the parameters of

Table F.4: Summary of statistics of absolute management system with three classes

		LMPC			WMPC			Dif(WMPC-LMPC)		
Output	Case	SSE	MIN	MAX	SSE	MIN	MAX	SSE	MIN	MAX
R_0	G	2.464	0.284	1.392	1.844	0.283	1.222	-0.62	0	-0.17
R_0	H	0.139	0.319	0.519	0.14	0.319	0.519	0	0	0
R_1	G	1.947	0.327	1.237	1.122	0.327	1.05	-0.824	0	-0.186
R_1	H	2.824	0.34	1.099	1.484	0.34	0.881	-1.34	0	-0.218
R_2	G	1.78	0.333	1.132	1.043	0.333	1.004	-0.737	0	-0.128
R_2	H	7.405	0.345	1.737	3.996	0.371	1.36	-3.409	0.02	-0.377

the MMST switching scheme and implement the control system and integrate it into the software system as in Figure F.14. The α and β was set to 0 and 1 and $T = 3$ same as in the MMST control system for two classes.

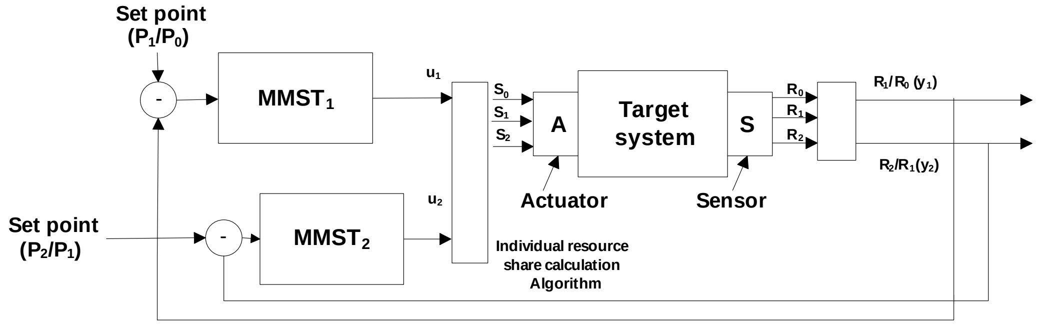


Figure F.14: The MMST control system for system with three classes

Table F.5: Parameters of the control systems for system with three classes

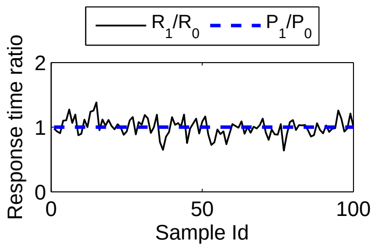
Parameter	controller -0	controller-1
K_p	0.80	0.38
K_i	0.39	0.08
Min situation limit	0.33	0.33
Max situation limit	3	3
Initial input	u(0) = 1	u(0) = 1

F.3.1 Evaluation

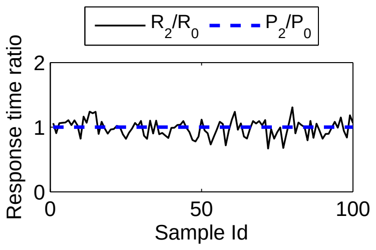
In this section, we show the performance and resource management capabilities of the MMST adaptive control scheme. In particular, here we only present the results of MMST-T2 scheme and compare it with the single linear model based control system designed in Section F.1.2. This is because of the poor performance of the MMST-T4 and Adaptive control systems. It is worth noting that the operating conditions and business requirements used in this section are same as to Section F.1.2.

F.3.2 Workloads of full capacity or lower

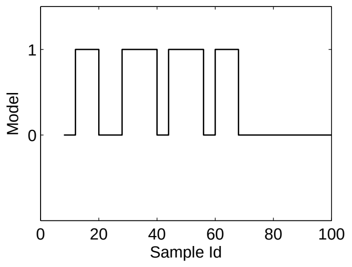
Case A: *Performance in the nominal region*



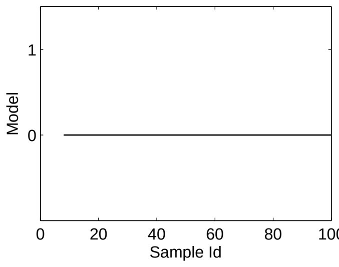
(a) Output of MMST-T2_1



(b) Output of MMST-T2_2



(c) Model switching signal of MMST-T2_1



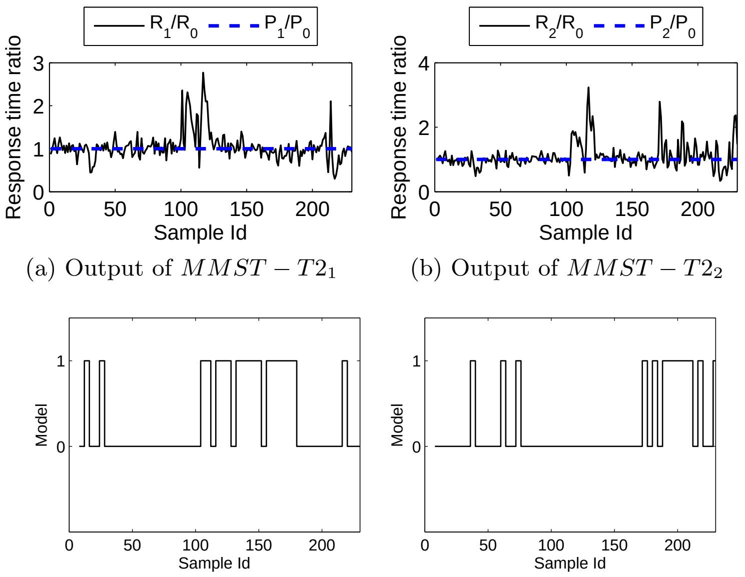
(d) Model switching signal of MMST-T2_2

Figure F.15: Performance management in nominal region (Case A)

In the nominal condition MMST-T2 control system indicates similar performance to a single linear model based control system (see Figure F.15 and Table F.6). The model switching signals shows stable behavior in both MMST-T2_1 and MMST-T2_2 control systems. Some chattering was observed in MMST-T2_1 because there is no explicit model to represent the dynamics in the nominal region. One of the other contributing factors for stable performance under chattering is the bump-less transfers implemented by the MMST-T2 control system.

Case B: *Performance away from the nominal region*

The result of this case is shown in Figure F.16. It illustrates that the models and controllers integrated to the MMST-T2 scheme have combined to provide better performance than LCSs (see Table F.6). In particular, when the workloads of class$_1$ and class$_2$ are high the control systems have to operate in the region where the input nonlinearities are severe. However, the model switching enables selecting the controller-1, which provides much better performance in that region. For instance, MMST-T2_1 control system has selected controller-1 between the 104th to 150th samples. The operation of controller-0 during the 100th to 104th samples led to high overshooting similar to LCS$_1$ but recovered after 27 samples due to the selection of controller-1. Similarly, in the case of MMST-T2_2 control system selected controller-0 before the 100th sample, while controller-1 was selected between the 180th to 225th samples most of the time, providing better performance under



(a) Output of $MMST - T2_1$ (b) Output of $MMST - T2_2$

(c) Model switching signal of $MMST - T2_1$ (d) Model switching signal of $MMST - T2_2$

Figure F.16: Performance management of three class system away the nominal region (Case B)

large disturbance in region-0. As a consequence, settling time has improved compared to LCSs. The sudden deviation after the 200th sample settled down in 15 samples, when the experiment is run further time periods. This is because of the chattering occurred in MMST-T2_2 control system.

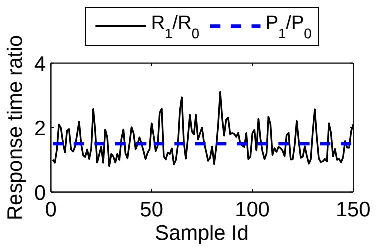
***Case C**: When Class$_0$ is more important*

In this case, LCS showed significant performance management issues. In contrast, the MMST-T2 control system which combines two controllers provide significantly better performance with stable behavior. The model switching signals shown in Figures F.17c and F.17d indicates that MMST control systems have used both controllers during the operations, in particular controller-1. Therefore, the performance issues caused by the nonlinearities have been reduced by the combined performance of two controllers with switching capabilities. Although some chattering was observed in MMST-T2_1 control system because of the noisy output signals, it did not lead to significant performance degradation (see Table F.6). However, the suitable model and control have been used to deliver the better results in the case of MMST-T2_2 reducing the unstable behavior of LCS.

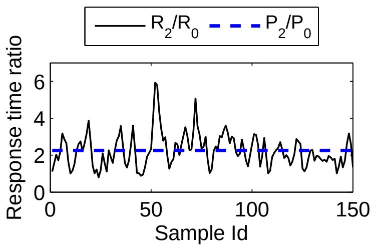
F.3.2.1 Workloads of Extreme Overload

***Case F**: Performance under overload when Class$_0$ is more important*

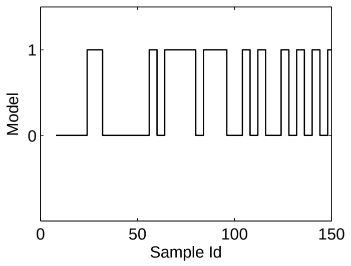
In this case, as well the LCSs indicated significant performance issues due to input and output nonlinearities. Again, the MMST-T2 control systems show significant per-



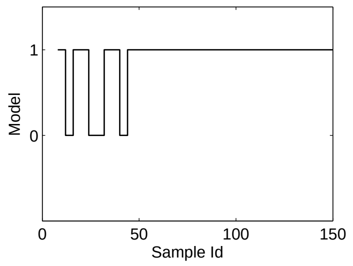
(a) Output of $MMST-T2_1$



(b) Output of $MMST-T2_2$

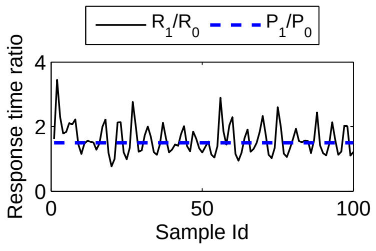


(c) Model switching signal of $MMST-T2_1$

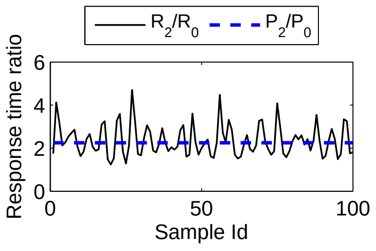


(d) Model switching signal of $MMST-T2_2$

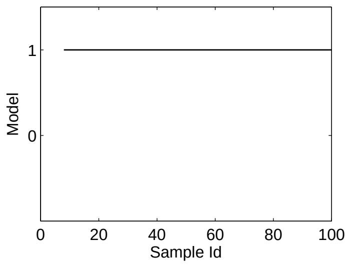
Figure F.17: Performance management when Class_0 is more important (Case C)



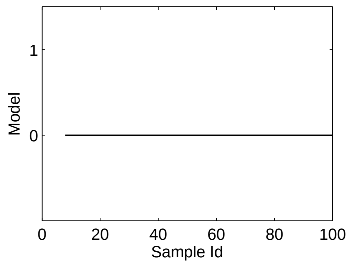
(a) Output of $MMST-T2_1$



(b) Output of $MMST-T2_2$



(c) Model switching signal of $MMST-T2_1$



(d) Model switching signal of $MMST-T2_2$

Figure F.18: Performance management of three class system when Class_0 is more important (Case F)

formance improvements by combining the performance of two controllers with different gains. MMST-T2_1 has operated with controller-1 during the entire operations without any chattering (See Figure F.18c). Similarly, Figure F.18d shows that MMST-T2_2 has operated with controller-0, providing much better performance compared to LCS. That is two MMST control systems have autonomously selected different models and controllers with significantly different gains. This behavior indicates that designing and tuning a single controller to provide control is significantly different for this condition.

Apart from these conditions presented in this section, results of the other cases are summarized in Tables F.6 and F.7. In all other cases, MMST-T2 control systems have provided similar performance to LCS indicating that chattering has not led to significant performance degradations.

Table F.6: Statistical summary of MMST control systems managing three classes

	y1			y2	
Case	MMST2	LCS		MMST2	LCS
A	1.791	1.888		1.55	1.558
B	26.925	13.858		22.336	68.017
C	6.057	4.581		5.411	5.007
D	19.992	28.219		75.09	114.306
E	70.739	82.5		41.31	99.162
F	38.266	60.268		30.167	34.368
G	1.806	1.705		4.052	3.94

Table F.7: Summary of average loss rates of MMST control systems managing three classes

	MMST2			LCS		
Case	$Class_0$	$Class_1$	$Class_2$	$Class_0$	$Class_1$	$Class_2$
E	9	10	11	9	10	12
F	0	31	58	0	35	59
G	62	28	0	63	28	0

Appendix G

Effects of Simulation Parameters

In Chapters 4, 5 and 6, the simulations were conducted in the simulation environment with configuration parameter set out in Chapter 3. In Chapter 3, we fixed the seeds of different classes to different values as well as the parameters of the processing time distribution to fixed values. In this appendix, further simulations are carried out to investigate the effects of these configuration variables of the simulation environment on the performance of the nonlinear control techniques. Here, we conduct Monte-Carlo simulations on the same system with two classes used in previous chapters, in order to statistically validate the impact of the configuration parameters of the simulation environment for each nonlinear control technique proposed for relative and absolute management. If these results do not invalidate the results presented in previous chapters, then we can conclude that those results were not generator due to pure chance, which also indicate the robustness and wide applicability of the proposed approaches.

G.1 Effects on Hammerstein-Wiener Nonlinear Control

In this section, we investigate the impact of the configuration parameters on the relative performance management system.

G.1.1 Effects of Seeds

The seeds selected for a particular class affects the processing times generated randomly during the simulations. This may have affected the behavior of the control systems. In particular, the behavior under sudden workload disturbances. In this section, we use 30 randomly selected seeds for each class to generate the processing time delays. All three control systems, which include LCS, HCS and HWCS are executed for 30 times with these seed configurations for each case listed in Section 5.3 and Appendix E. The gathered experiment results of the three control systems are analyzed under the *Kruskal-Wallis* non-parametric statistical test[1]. The Kruskal-Wallis statistical test provides us a technique to check whether there are significant differences between the outputs produced

[1]it is noteworthy that the data was initially evaluated with the *Kolmogorov-Smirnov test* to identify whether data is normally distributed.

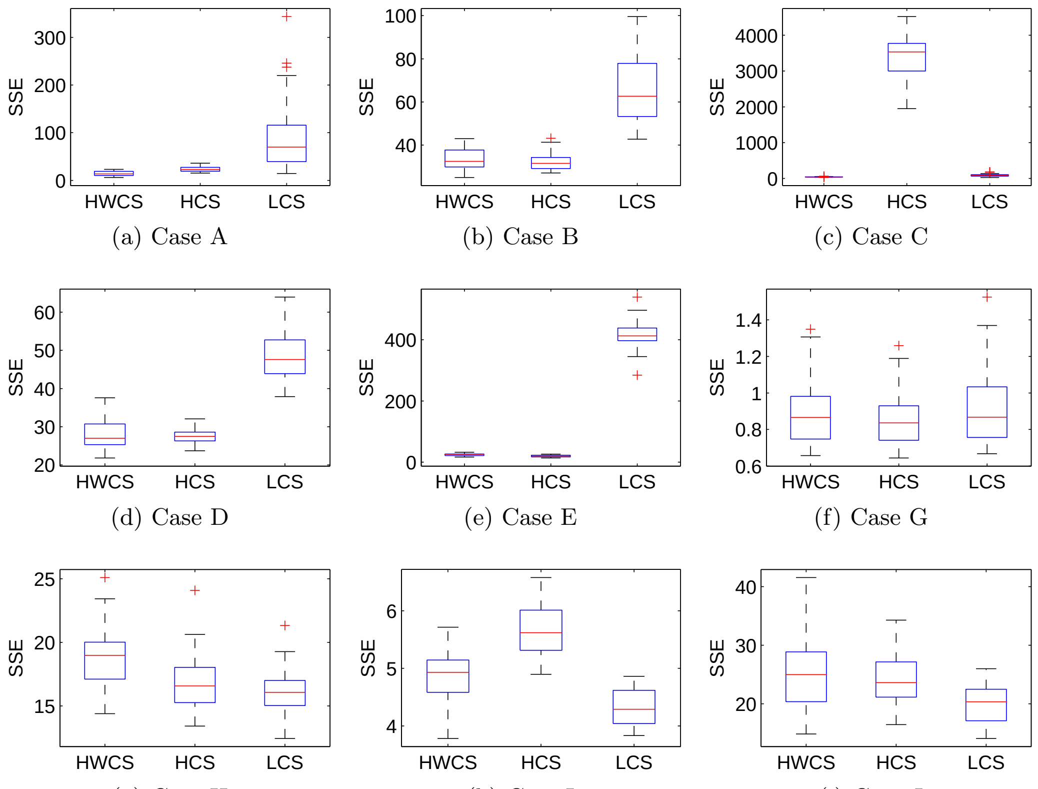
by three control systems. The null hypothesis is that all control systems provide equal performance management based on the SSE of the experiments. The statistical result of Kruskal-Wallis test is called the *p-value*. If p-value is greater than 0.01, the performance of the control systems has no significant difference. Therefore, we compare the p-value for each case and if p-value is less than 0.01, then we conclude that the outputs of three control systems are different. Finally, in order to investigate which control system is the best for a particular case, the SSE values are compared for all 30 seed configurations.

Table G.1: P-values of relative management control systems for 30 runs with different seeds

Case	p-value
A	1.90E-14
B	2.63E-05
C	1.31E-16
D	1.29E-13
E	1.65E-15
G	0.420946
H	1.12E-13
I	9.49E-14
J	0.000473

The results of these simulations are listed in Table G.1. From the p-values in Table G.1, apart from the *Case G*, the performance of all other cases of three control systems are significantly different from each other. In order to investigate which control system provides the best management in each case, Figure G.1 illustrates the box-plots of the statistical results.

From the box-plots it is evident that there is an impact on the performance management based on the selected seeds. The SSE statistics are largely affected by how the disturbance rejection behavior changes with the seeds. The same observations to the results in Table 5.4 have been produced in these experiments as well. In fact, the SSE statistics listed in Table 5.4 fall inside the bounds of the SSE statistics shown in the box-plots produced by the 30 runs. In particular, the cases where LCS showed significant performance degradation and instabilities in Section 5.3, have shown similar behavior irrespective of the seeds used (e.g., Case A, D and E). In summary, HWCS outperforms LCS in the same cases, where either the input and output nonlinearities were severe. The HCS and HWCS show similar performance and SSE statistics for many cases. However, HCS is significantly outperformed by HWCS in the cases where the output nonlinearities affect the performance (e.g., Case C and I). The HCS outperforms HWCS in Case H, when the noisy conditions affect the performance of the output compensator. From these observations, it is evident that the selected seeds for the random processing time distributions have not invalidate the result presented in Chapter 5.



(a) Case A (b) Case B (c) Case C

(d) Case D (e) Case E (f) Case G

(g) Case H (h) Case I (i) Case J

Figure G.1: Box-plots of SSE of 30 runs for different cases of relative management

G.1.2 Effects of Limits of the Processing Time Distributions

In Chapter 5, the processing times were generated using a uniform random distribution with parameters of $r_{min} = 100$ and $r_{max} = 700$ ticks or seconds (see Chapter 3). In this section, we investigate the behavior of the control systems when the range of the processing time distributions is smaller and larger than the aforementioned range. In the former we set the processing time distribution to be $\mathcal{U}(100, 300)$ (a range of 0.2 sec), in the later we set it to be $\mathcal{U}(100, 1100)$ (a range of 1 sec). When the range is high, there will be larger variations at the system outputs. For instances, a multi-class application providing versatile business functionalities may show such behaviors. In contrast, small ranges indicate it's a small scale system with a limited and similar set of functionalities. These settings enable us to examine whether the conclusions made in Chapter 5 are valid for different ranges of processing time. Although for a statistical validation, simulations for at least 30 different such parameter ranges have to be carried out. Such a large scale validation is hard to conduct because for each setting all three control systems (90 in total) have to be designed and workload conditions have to be determined for each case after profiling the system. As a consequence, such a validation is hard to automate, requiring a significant amount of manual efforts. In order to mitigate this limitation we have conducted simulations for three such ranges and experiments are also conducted in real-world experimental case-study systems.

It is important to emphasize that with these significantly different settings the behavior and dynamics of a multi-class shared resource environment changes drastically. As a consequence, all the control systems have to be built from scratch for each setting. This includes the design of the input and output inverse nonlinear compensators, linear model and subsequently the controller for the case of HWCS. A similar process is carried out for the design of HCS and LCS as well. In addition, the workload settings of these experiments have to be altered depending on the total workload capacities.

Table G.2: Statistics of the relative management control systems when the processing time limits are small $\mathcal{U}(100, 300)$

	LCS				HCS				HWCS		
Case	SSE	MIN	MAX		SSE	MIN	MAX		SSE	MIN	MAX
A	4715.93	0.34	24.32		9.34	0.37	3.09		6.86	0.54	3.09
B	5587.05	0.91	24.4		44.04	0.51	4.44		41.21	0.88	5.83
C	267.32	0.89	8.58		1935.35	0.74	16.31		46.86	0.88	5.83
D	77.22	0.36	3.11		16.69	0.38	2.75		32.97	0.39	3.02
E	273.63	0.43	4.87		10.93	0.98	3.12		10.93	0.98	3.12
G	0.2	0.88	1.13		0.19	0.88	1.13		0.2	0.88	1.13
I	156.16	0.66	4.19		71	0.94	3.43		30.38	1.1	3.23

Table G.3: Statistics of the relative management control systems when the processing time limits are large $\mathcal{U}(100, 1100)$

	LCS				HCS				HWCS		
Case	SSE	MIN	MAX		SSE	MIN	MAX		SSE	MIN	MAX
A	105.01	0.26	4.98		17.28	0.2	2.64		11.25	0.25	2.71
B	78.04	0.6	3.74		32.47	0.52	3.2		25.44	0.87	3.37
C	37.67	0.95	3.78		85.47	0.84	5.39		26.98	0.85	3.58
D	142.93	0.33	5.89		10.36	0.3	2.26		14.92	0.32	2.39
E	364.12	1.54	6.1		56.15	1.5	6.36		32.81	1.03	3.99
G	1.94	0.68	1.44		1.64	0.65	1.39		1.73	0.63	1.39
I	6.78	0.96	2.47		13.64	0.96	2.83		8.32	0.96	3.03

Tables G.2 and G.3 summarize the statistics for the two selected settings. The results of the low and high processing time distribution ranges reemphasis that the conclusions made in the Chapter 5 are still valid. That is apart from the nominal region (Case G), LCS performs poorly when the input and output nonlinearities are severe. Further, when the limits of the processing times are smaller, the performance of the LCS is significantly poor (see Table G.2, cases A, B). This is because in order to force the control system to operate in the nonlinear regions, larger workload disturbances had to be applied. Consequently, the oscillatory behavior in the control signal affects the output significantly, leading to unstable and larger output variations. In contrast, HCS and HWCS shows much better performance even under such settings. Furthermore, HCS shows poor performance in the same cases (e.g., Case C) compared to HWCS as observed in Chapter 5. Therefore, these configuration settings of the simulation environment have not invalidated the conclusions and results presented in Section 5.3.

G.2 Effects on MIMO-Wiener Control

In this section, we investigate the impact of the configuration parameters on the absolute performance management system.

G.2.1 Effects of Seeds

In this section, we use 30 randomly selected seeds for each class to generate the processing time delays. The LMPC and WMPC are executed for 30 times with these seed configurations for each case listed in Section 5.5. The gathered experiment data of the two control systems is analyzed using the Kruskal-Wallis non-parametric statistical test.

Table G.4: P-values of absolute management control systems for 30 runs with different seeds

Case	R_0	R_1
Case A	1.21E-05	0.045945
Case B	0.002561	0.008875
Case C	1.94E-09	4.73E-11

From the p-values in Table G.4, all the cases show significantly different output data with the exception of Case A. The box-plots in Figure G.2 indicates that LMPC has shown highly variable performance in all cases compared to WMPC. In particular, in Case C, where the control systems have to operate in the sensitive region, WMPC has significantly outperformed the linear counterpart. Furthermore, the performance provided by LMPC for the important class (R_0) has affected significantly as well in the same case, which is consistent with the observations of Section 5.5. In Case A and B, the upper bounds of SSE and MAX are higher for the cases of LMPC compared to WMPC. Although there is some overlapping between the box plots for the some cases, we can conclude that the average performance of WMPC is better than LMPC. Consequently, these results are consistent with the results presented in Section 5.5.

G.2.2 Effects of Limits of the Processing Time Distributions

In this section, we investigate the effect on the performance management of the control systems when the range of processing time distributions is smaller and larger as defined in Section G.1.2. Table G.5 and G.6 shows the results of the control systems.

When the range of the processing time is small the variability of the average response time is significantly low (see Table G.5). Therefore, there is no significant difference between the control systems in Case A and B. However, there are improvements for Case C, in WMPC compared LMPC.

Table G.5: Statistics of the absolute management control systems when the processing time limits are small $\mathcal{U}(100, 300)$

Case	LMPC			WMPC		
	SSE	MIN	MAX	SSE	MIN	MAX
R_0						
A	6.511	0.185	0.832	6.323	0.186	0.781
B	4.344	0.186	0.311	4.28	0.186	0.262
C	4.319	0.184	0.256	4.004	0.186	0.617
R_1						
A	9.304	0.183	1.432	7.939	0.183	1.284
B	4.326	0.183	0.253	4.346	0.183	0.286
C	15.578	0.194	2.165	8.322	0.19	1.669

Table G.6: Statistics of the absolute management control systems when the processing time limits are large $\mathcal{U}(100, 1100)$

Case	LMPC			WMPC		
	SSE	MIN	MAX	SSE	MIN	MAX
R_0						
A	12.948	0.41	1.89	8.177	0.41	1.331
B	4.31	0.503	0.929	4.322	0.503	0.929
C	4.404	0.472	0.929	4.321	0.471	0.929
R_1						
A	237.634	0.464	5.509	166.989	0.45	4.798
B	5.312	0.447	0.916	7.083	0.447	1.04
C	76.115	0.447	3.435	22.849	0.447	2.421

When the range of processing time is large, the variability in output is higher (see Table G.6). As a consequence, SSE statistics of both linear and nonlinear control systems are higher than the statists we observed in Section 5.5. However, the results are consistent with the observations and conclusions of earlier settings, which include the significant improvements for Case A and C for the case of WMPC.

G.3 Effects on MMST Adaptive Control

This section investigates the effects of simulation parameters on the relative management performed by the multi-model control approach proposed in Chapter 6.

G.3.1 Effects of Seeds

In this section, we compare the performance of MMST-T2 control system and a single model based control system to investigate the effects of different seeds.

From the results of Kruskal-Wallis non-parametric statistical test in Table G.7, it is evident apart from the cases marked with $\star$, other cases show differences in performance management. Figure G.3 shows that LCS has slightly outperformed MMST-T2 scheme only in Case C. In all other cases, which include Case A, B, D and I, MMST-T2 control

Table G.7: P-values of MMST control system for 30 runs with different seeds

Case	p-value
A	1.33E-06
B	1.05E-05
C	1.32E-01
D	1.69E-10
E	4.27E-06
G	0.505859 ⋆
H	0.859184 ⋆
I	1.79E-01
J	0.524952⋆

system has shown better performance. Therefore, selected seeds do not invalidate the results presented in Chapter 6.

G.3.2 Effects of Limits of the Processing Time Distributions

In this section, we compare the performance of MMST-T2 and LCS to investigate the effects of the selected range of the processing time distribution. Tables G.8 and G.9 summarize the statistics of control systems designed using the same design methodology presented in Chapter 6.

Table G.8: Statistics of the MMST control system when the processing time limits are small $\mathcal{U}(100, 300)$

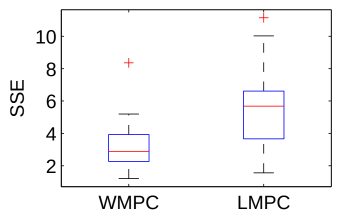
	MMST-T2				LCS		
Case	SSE	MIN	MAX		SSE	MIN	MAX
A	17.225	0.275	4.393		4715.93	0.34	24.32
B	139.783	0.226	9.054		5587.05	0.91	24.4
C	127.604	0.109	1.126		267.32	0.89	8.58
D	16.694	0.38	2.752		77.22	0.36	3.11
E	70.996	0.942	3.432		273.63	0.43	4.87
G	0.195	0.884	1.127		0.2	0.88	1.13
I	14.485	0.952	3.049		156.16	0.66	4.19

In the case where the rage is small, significant performance improvements have been made by the MMST-T2 control system for all cases. In the case of high processing time range, Case H which produced better results in other settings has indicated poorer performance for the MMST-T2 control system compared to LCS. This is because due to noisy conditions, controller-0 was selected for short period of time leading to oscillatory behavior at the output.

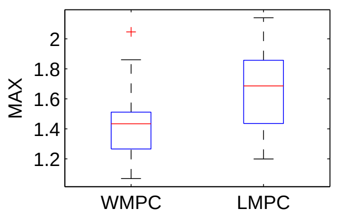
In summary, the settings of processing time distribution selected have not invalidated the results presented in Chapter 6.

Table G.9: Statistics of the MMST control system when the processing time limits are large $\mathcal{U}(100, 1100)$

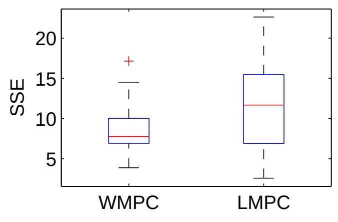
	MMST-T2				LCS		
Case	SSE	MIN	MAX		SSE	MIN	MAX
A	16.98	0.26	2.726		105.013	0.26	4.983
B	159.135	0.586	6.598		78.037	0.597	3.739
C	103.739	0.302	1.28		106.722	0.264	1.053
D	16.454	0.335	2.806		142.931	0.335	5.885
E	97.581	1.035	6.427		364.122	1.543	6.1
G	2.347	0.648	1.454		1.938	0.677	1.442
I	5.86	0.962	2.736		6.784	0.962	2.468



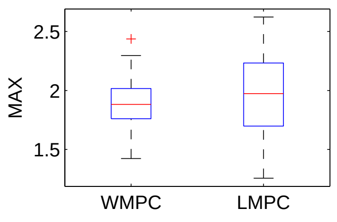
(a) SSE Case A, R_0



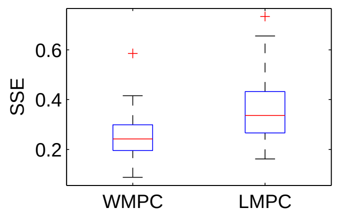
(b) MAX Case A, R_0



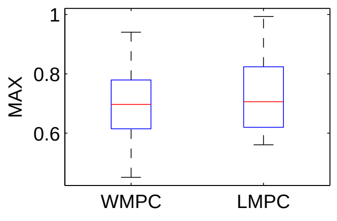
(c) SSE Case A, R_1



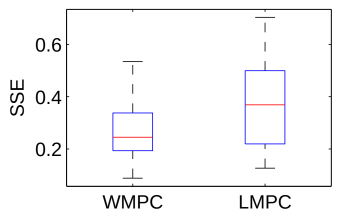
(d) MAX Case A, R_1



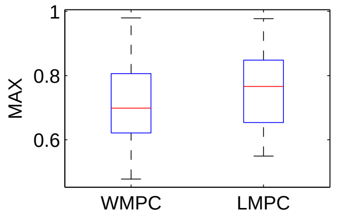
(e) SSE Case B, R_0



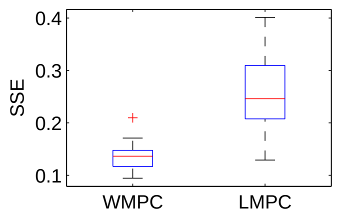
(f) MAX Case B, R_0



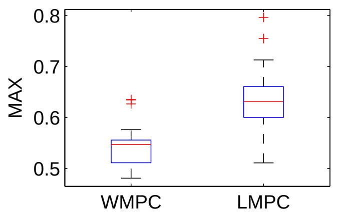
(g) SSE Case B, R_1



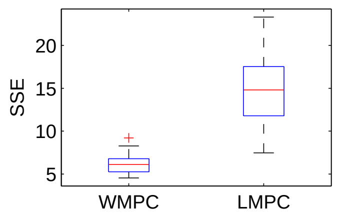
(h) MAX Case B, R_1



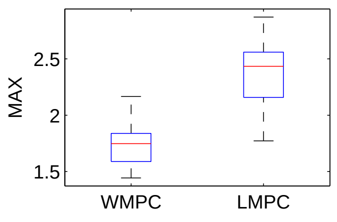
(i) SSE Case C, R_0



(j) MAX Case C, R_0

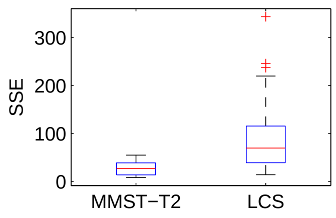


(k) SSE Case C, R_1

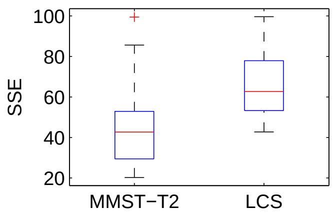


(l) MAX Case C, R_1

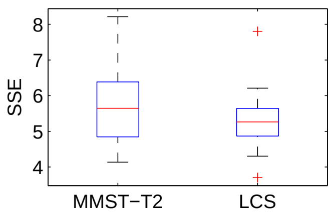
Figure G.2: Box-plots of SSE and MAX statistics of 30 runs for different absolute management Cases



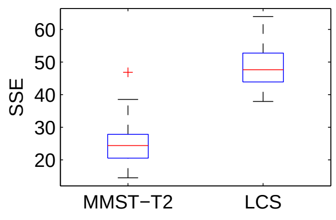
(a) Case A



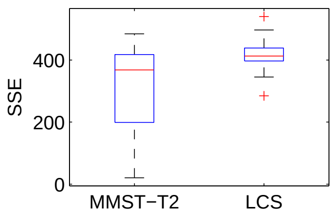
(b) Case B



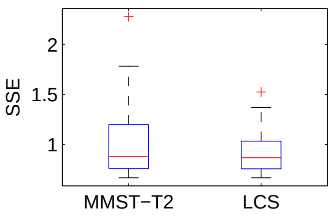
(c) Case C



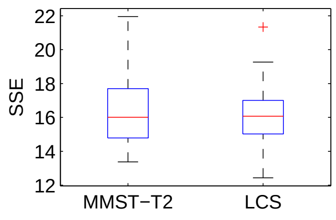
(d) Case D



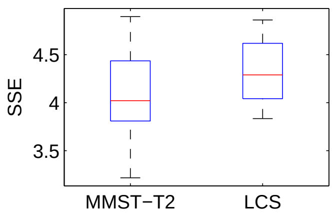
(e) Case E



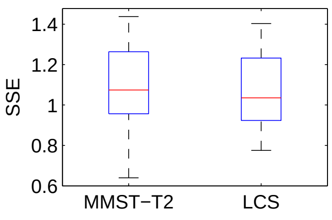
(f) Case G



(g) Case H



(h) Case I



(i) Case J

Figure G.3: Box-plots of SSE of 30 runs for different MMST control cases

Appendix H

Relative Management in a System With a Large Amount of Resources

In this appendix, a simulation environment with $S_{total} = 100$, $S_{0,min}$ and $S_{1,min} = 10$ is used. This means that the system has a large amount of resources compared to the settings used in Section 5.3. With these settings, the control input u has 80 operating points, enabling a mechanism to examine the benefits of the Hammerstein-Wiener nonlinear control approach in a system with a larger amount of resources.

Following the procedure in Chapter 4 and 5, we designed a HWCS, which include estimating the inverse nonlinear functions and linear model, followed by the design of a controller. Figure H.1 shows the input and output nonlinear mappings used to design and implement the input and output compensators respectively. The operating points layout in Figure H.1a shows severe nonlinearities around the nominal operating point, i.e., $\frac{S_0}{S_1} = 1(\frac{50}{50})$. In order to design the input compensator we selected $v_{min} = -20$ and $v_{max} = 20$ and a 7th order polynomial, which yielded a model fit $R^2 = 0.99$. Equation (H.1) shows the polynomial of the inverse input nonlinearity.

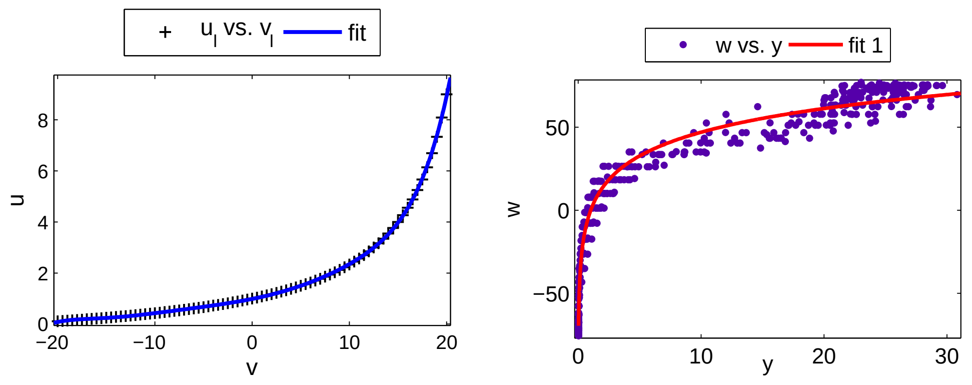
A system identification experiments were conduced following the design procedure in Section 4.2.6, in order to estimate the inverse output nonlinear component and the linear component of the Hammerstein-Wiener model. It is worth noting that, in a system with a large number of operating points the operating points and the workload conditions have to be carefully selected to capture the behavior around the required region of operations. Figure H.1b illustrates the model fit $R^2 = 0.95$ and equation (H.2) shows coefficients of the inverse nonlinear output function derived from the experiment data.

Afterwards, a linear model (see equation (H.3)) was estimated and the controller was designed using the pole-placement method (see Table H.1). Similarly, HCS and LCS were designed as well. Table H.1 lists the parameters of all three control systems. In this

Table H.1: Parameters of the control systems

Parameter	HWCS	HCS	LCS
K_p	0.31	4.70	0.76
K_i	0.10	1.30	0.28
Min situation limit	-20	-20	0.11 (10/90)
Max situation limit	20	20	9(90/10)

section, we use the conditions of Case A in Section 5.3.1, which gives useful conditions to examine the effects of input and output nonlinearities together. However, since there are a large number of resources we apply 185 and 50 requests/sec for $class_0$ and $class_1$ at the 30th sample in order to force the system to operate away from the nominal region. Similarly, 50 and 185 requests/sec workloads are applied at the 100th sample.

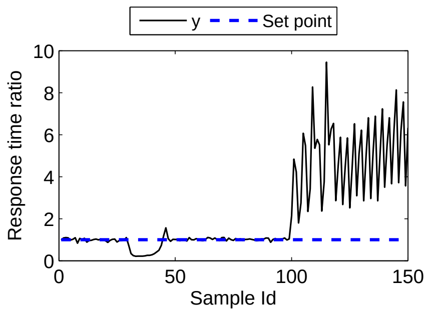


(a) Inverse input nonlinear model fit (b) Inverse output nonlinear model fit

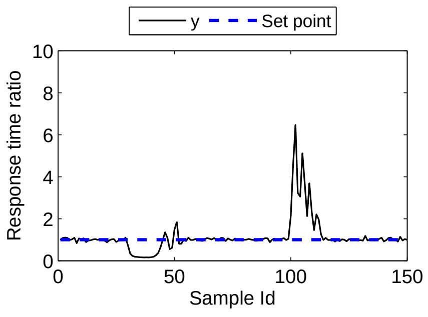
Figure H.1: The model fit of the inverse input and output when there are a large number of resources

$$
\begin{aligned}
u(k) &= f^{-1}(v) \\
&= 2.377 \times 10^{-9} v(k)^7 + 4.725 \times 10^{-8} v(k)^6 - 6.328 \times 10^{-7} \\
&\quad v(k)^5 - 6.328 \times 10^{-7} v(k)^4 - 7.776 \times 10^{-6} v(k)^3 + 0.0002 v(k)^4 \\
&\quad + 0.004 v(k)^2 + 0.077 v(k) + 0.986 \qquad \text{(H.1)} \\
w &= g^{-1}(y) \\
&= 22.00 log(y) - 2.10 \qquad \text{(H.2)} \\
w(t+1) &= 0.86 w(t) + 1.60 v(t) \qquad \text{(H.3)}
\end{aligned}
$$

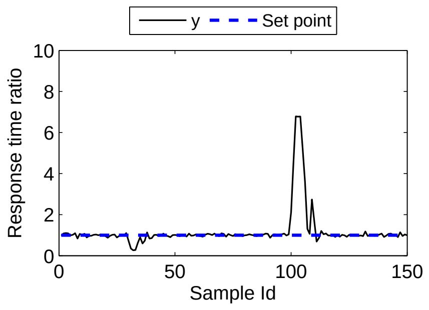
The performance of the control systems shown in Figure H.2 indicates the same observations to Section 5.3. LCS shows highly oscillatory behavior in *region 1* after the high workload disturbance at the 100th sample, where $class_1$ demands large amount of resources leading to unstable performance. This is caused by the input nonlinearity. In



(a) Output of LCS



(b) Output of HCS



(c) Output of HWCS

Figure H.2: Performance in system with a large amount of resources

addition, the overshooting and settling time at the 30th sample are significantly poor compared to HWCS, which is caused by the output nonlinearity. In contrast, both nonlinear control systems show satisfactory performance, effectively rejecting the disturbances and settling down to the required differentiation level without large steady state errors. However, when HCS and HWCS are compared, HCS shows significantly high overshooting and settling time due to the large workload disturbance at the 30th and 100th sample. This is because of the output nonlinearity. The performance of HWCS, which compensates input and output nonlinearities, is satisfactory in a large scale shared resource environment as well providing better performance and resource management with performance isolation.

Appendix I

Effects of the Range V (v_{max}, v_{min})

The design process presented in Section 4.2.4 includes the selection of v_{min}, v_{max} and δv variables. However, these variables were selected arbitrarily, in particular, we chose v_{max} = -9, v_{min} = 9 in Section 5.3. In this appendix, we compare the impact of these design parameters on the performance of the control system using a simulation environment of two classes similar to Section 5.3. The v_{max}, v_{min} are the design parameters of the input nonlinear component, in other words the Hammerstein component. Therefore, in order to isolate just the input nonlinearity, we use a Hammerstein model based control system to investigate the impact of the selected range for parameter v without compensating the output nonlinearities. Here, three types of parameter configurations will be examined as follows:

- Type I. v_{min} = -90, v_{max} = 90 and δv =10,
- Type II. v_{min}= -2.25, v_{max} = 2.25and δv =0.25,
- Type III. v_{min}= 1, v_{max}= 19 and δv =1,

In Type I and II, we investigate the impact of v_{max}, v_{min} and δv in large and small ranges. The Type III investigates the placement of the v_{max}, v_{min} entirely in the positive side, instead of equally spreading it around zero. Using these settings three Hammerstein model based control systems are designed and their performance is investigated in the experiment settings of Case A in Section 5.3.1. As mentioned Case A conditions force the control systems to operate in regions where input and output nonlinearities are severe, consequently providing us a basis to compare the performance of these control systems. The inverse nonlinear function of the input compensators for Type I, II and III are shown in equations (I.1), (I.2) and (I.3) respectively. Similarly, the linear ARX models for the linear components of the Hammerstein model are shown in equations (I.4),(I.5) and (I.6) respectively. Afterwards, the controllers were designed as well for each configuration. Their parameters are shown in Table I.1. The output signals of control systems for this experiment are shown in Figure I.1.

$$\begin{aligned} u(k) &= f^{-1}_{TypeI}(v) \\ &= 7.468 \times 10^9 v(k)^4 + 1.003 \times 10^6 v(k)^3 + 7.698 \times 10^5 v(k)^2 + 0.01244 v(k) + 1.005 \end{aligned} \tag{I.1}$$

$$\begin{aligned} u(k) &= f^{-1}_{TypeII}(v) \\ &= 0.01912 v(k)^4 + 0.06417 v(k)^3 + 0.1232 v(k)^2 + 0.4978 v(k) + 1.005 \end{aligned} \tag{I.2}$$

$$\begin{aligned} u(k) &= f^{-1}_{TypeIII}(v) \\ &= 7.468 \times 10^5 v(k)^4 - 0.001984 v(k)^3 + 0.02242 \times 10^5 v(k)^2 - 0.0274 v(k) + 0.2747 \end{aligned} \tag{I.3}$$

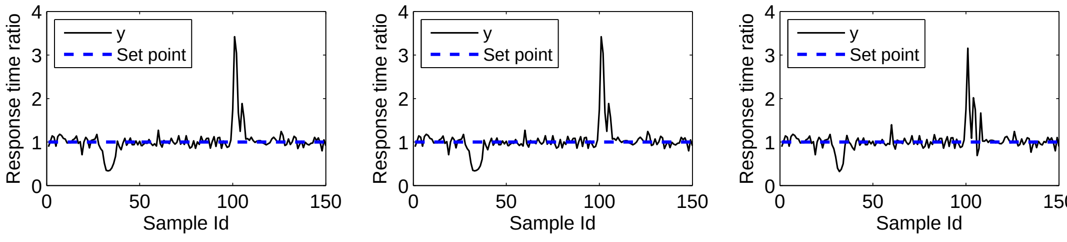
$$y(t+1) = 0.90y(t) + 0.02v(t) \tag{I.4}$$

$$y(t+1) = 0.90y(t) + 0.80v(t) \tag{I.5}$$

$$y(t+1) = 0.80y(t) + 0.10v(t) \tag{I.6}$$

Table I.1: The controller gains for different types of configuration parameters v_{min} and v_{max}

	K_p	K_i
Type I ($\delta v = 10$)	32.50	12.50
Type II ($\delta v = 0.25$)	0.81	0.31
Type III ($\delta v = 1$)	5.50	2.50
Section 5.3.1 Case A ($\delta v = 1$)	2.70	1.04



(a) Type I (b) Type II (c) Type III

Figure I.1: Effect of design parameter v_{min} and v_{max}

The main observation is that none of the control systems designed from these parameter configurations illustrate the performance issues of the linear model based control system (see Case A Figure 5.5). The Type I and II configurations show almost identical output signals. This similarity of the performance may have yielded because there is no difference how the controllers behave in the closed-loop system. An interesting observation here is that the gains of the controllers in Type I and II have scaled according to δv (see Table I.1). For instance, the δv parameter of Type I is 40 times larger compared to Type II. This has also scaled the gains of these two control systems 40 times. The same conclusion can be made when the Type I and Type II gains are compared with the gains used in Section 5.3. Such scaling occurs because the autoregressive coefficient of the ARX model has remained constant while the exogenous input coefficient has scaled with respect to δv (see equations (I.4),(I.5)). Consequently, other than small differences, the dynamics of the system and performance management provided by the control systems have remained almost the same for the Type I and II configurations. However, in the case of Type III, the same conclusions cannot be made. Although the performance was satisfactory, it is different from other control systems. This is because, the range of v lies on the positive side, consequently affecting the performance of the control system. In the design of the negative feedback control systems it is recommended to place the control input around zero (that is in the positive and negative sides) in order to improve the model estimation and runtime control due to the integrator embedded in the controller. This experiment setting did not provide us with conclusive results, but theoretically the design parameter selections similar to Type I and II can be recommended. Furthermore, we can conclude that there is little or no impact on the performance of the control system, depending on the selected δv. As a result, arbitrary values v_{min} and v_{max} can be selected, but adhering to the above recommendation.

Bibliography

[1] S. Abdelwahed, N. Kandasamy, and S. Neema. Online control for self-management in computing systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 368 – 375, 2004.

[2] S. Abdelwahed, S. Neema, J. Loyall, and R. Shapiro. A hybrid control design for QoS management. In *24th IEEE Real-Time Systems Symposium*, pages 366 – 369, 2003.

[3] T. Abdelzaher, K. Shin, and N. Bhatti. Performance guarantees for web server end-systems: a control-theoretical approach. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):80 –96, 2002.

[4] T. F. Abdelzaher and N. Bhatti. Web server QoS management by adaptive content delivery. In *International Workshop on Quality of Service*, pages 216–225, 1999.

[5] R. B. Abdennour, M. Ksouri, and F. M'Sahli. Nonlinear model-based predictive control using a generalised Hammerstein model and its application to a semi-batch reactor. *The International Journal of Advanced Manufacturing Technology*, 20:844–852, 2002.

[6] L. Abeni, L. Palopoli, and G. Buttazzo. On adaptive control techniques in real-time resource allocation. In *Euromicro Conference on Real-Time Systems*, pages 129 –136, 2000.

[7] J. Almeida, V. Almeida, D. Ardagna, ı. Cunha, C. Francalanci, and M. Trubian. Joint admission control and resource allocation in virtualized servers. *J. Parallel Distrib. Comput.*, 70:344–362, 2010.

[8] W. Aly and H. Lutfiyya. Using feedback control to manage QoS for clusters of servers providing service differentiation. In *IEEE Global Telecommunications Conference*, volume 2, pages 960–964, 2005.

[9] W. H. F. Aly and H. Lutfiyya. Dynamic adaptation of policies in data center management. In *IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 266–272, 2007.

[10] M. Amirijoo, P. Brannstrom, J. Hansson, S. Gunnarsson, and S. H. Son. Toward adaptive control of QoS-importance decoupled real-time systems. In *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, FEBID, 2007.

[11] M. Amirijoo, J. Hansson, S. Gunnarsson, and S. Son. Enhancing feedback control scheduling performance by on-line quantification and suppression of measurement disturbance. In *IEEE Real Time and Embedded Technology and Applications Symposium*, pages 2 – 11, 2005.

[12] M. Armbrust, A. Fox, G. Rean, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. *Magazine : Communications of the ACM*, 2008.

[13] K. J. Astrom and B. Wittenmark. *Adaptive Control.* Addison-Wesley Publishing Company, 1995.

[14] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne, S. Weerawarana, and P. Fremantle. Multi-tenant soa middleware for cloud computing. In *International Conference on Cloud Computing*, CLOUD '10, pages 458–465, 2010.

[15] E. W. Bai. Decoupling the linear and nonlinear parts in Hammerstein model identification. *Automatica*, 40:671–676, 2004.

[16] J. BAI. *A model integrated framework for designing and optimization of self-managing computing systems.* PhD thesis, 2008.

[17] S. A. A. Banawan and G. Radhakrishnan. An adaptive threshold based scheduling policy for atm networks. In *IEEE Conference on Computers and Communications*, pages 439–445, 1996.

[18] J. Banks, J. Carson, B. L. Nelson, and D. Nicol. *Discrete-Event System Simulation (4th Edition).* Prentice Hall, 2004.

[19] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *ACM symposium on Operating systems principles*, SOSP'03, pages 164–177, 2003.

[20] A. Beloglazov and R. Buyya. Energy efficient resource management in virtualized cloud data centers. In *International Conference on Cluster, Cloud and Grid Computing*, pages 826 –831, 2010.

[21] A. Bemporad and M. Morari. Robust model predictive control: A survey. *Springer-Verlag, Lecture Notes in Control and Information Sciences*, 245:207–226, 1999.

[22] L. Bertini, J. Leite, and D. Mosse. SISO PIDF controller in an energy efficient multi-tier web server cluster for e-commerce. In *Workshop on Feedback Control Impl. and Design in Computing Systems and Networks*, FeBID'07, 2007.

[23] V. Bhat, M. Parashar, H. Liu, M. Khandekar, N. Kandasamy, and S. Abdelwahed. Enabling self-managing applications using model-based online control strategies. In *IEEE International Conference on Autonomic Computing*, pages 15 – 24, 2006.

[24] N. Bhatti and R. Friedrich. Web server support for tiered services. *IEEE Network*, 13(5):64–71, Sept. 1999.

[25] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated service. 1998.

[26] A. Block, B. Brandenburg, J. H. Anderson, and S. Quint. An adaptive framework for multiprocessor real-time system. In *Euromicro Conference on Real-Time Systems*, ECRTS'08, pages 23–33, 2008.

[27] H. H. J. Bloemen, C. T. Chou, T. J. J. van den Boom, V. Verdult, M. Verhaegen, and T. C. Backx. Wiener model identification and predictive control for dual composition control of a distillation column. *Journal of Process Control*, 11:601 – 620, 2001.

[28] R. Braden, D. Clark, and S. Shenker. Integrated services in the internet architecture: an overview. 1994.

[29] Y. Brun, G. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Muller, M. Pezz'e, and M. Shaw. Engineering self-adaptive systems through feedback loops. *Software Engineering for Self-Adaptive Systems*, pages 48–70, 2009.

[30] M. Burgess and G. Undheim. Predictable scaling behaviour in the data centre with multiple application servers. In *IFIP/IEEE Distributed Systems: Operations and Management*, DSOM 2006, pages 49–60, 2006.

[31] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[32] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *Joint Conference of the IEEE Computer and Communications Societies*, INFOCOM'02, pages 506 – 515, 2002.

[33] A. L. Cervantes, O. E. Agamennoni, and J. L. Figueroa. A nonlinear model predictive control system based on Wiener piecewise linear models. *Journal of Process Control*, 13:655–666, 2003.

[34] B. Chen, X. Peng, and W. Zhao. Towards runtime optimization of software quality based on feedback control theory. In *Asia-Pacific Symposium on Internetware*, Internetware'09, pages 10:1–10:8, 2009.

[35] I.-R. Chen and T.-H. Hsi. Threshold-based dynamic admission control algorithms for real-time multimedia servers. In *ACM symposium on Applied Computing*, SAC '96, pages 224–229, 1996.

[36] M. Chen, X. Wang, R. Gunasekaran, H. Qi, and M. Shankar. Control-based real-time metadata matching for information dissemination. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 133–142, 2008.

[37] M. Chen, X. Wang, and X. Li. Coordinating processor and main memory for efficient server power control. In *International conference on Supercomputing*, ICS '11, pages 130–140, 2011.

[38] M. Chen, X. Wang, and B. Taylor. Integrated control of matching delay and CPU utilization in information dissemination systems. In *International Workshop on Quality of Service*, pages 1 –9, 2009.

[39] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev.*, 33:303–314, 2005.

[40] S.-W. Cheng. *Rainbow: cost-effective software architecture-based self-adaptation.* PhD thesis, Carnegie Mellon University, 2008.

[41] L. Chenyang, T. F. Abdelzaber, J. A. Stankovic, and S. H. Son. A feedback control approach for guaranteeing relative delays in web servers. In *IEEE Real-Time Technology and Applications Symposium*, pages 51–62, 2001.

[42] L. Chenyang, J. Stankovic, G. Tao, and S. Son. Design and evaluation of a feedback control EDF scheduling algorithm. In *Real-Time Systems Symposium*, pages 56 –67, 1999.

[43] F. Chong and G. Carraro. Architecture strategies for catching the long tail. *MSDN*, 2006.

[44] G. W. Corder and D. I. Foreman. *Nonparametric Statistics for Non-Statisticians: A Step-by-Step Approach.* Wiley, 2009.

[45] F. Dan and L. Kueiming. Identification for disturbed MIMO Wiener systems. *Nonlinear Dynamic*, 55, 2009.

[46] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Using MIMO feedback control to enforce policies for interrelated metrics with application to the Apache web server. In *IEEE/IFIP Network Operations and Management Symposium*, pages 219 – 234, 2002.

[47] Y. Diao, J. L. Hellerstein, S. Parekh, and J. P. Bigus. Managing web server performance with autotune agents. *IBM Systems Journal*, pages 136–149, 2003.

[48] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Parekh, and C. Garcia-Arellano. Incorporating cost of control into the design of a load balancing controller. *IEEE Real-Time and Embedded Technology and Applications Symposium*, page 376, 2004.

[49] Y. Diao, X. Hu, A. Tantawi, and H. Wu. An adaptive feedback controller for SIP server memory overload protection. In *International conference on Autonomic computing*, ICAC'09, pages 23–32, 2009.

[50] B. Du and C. Ruan. Modeling and robust control for trusted web server. In *IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, EUC '10, pages 659–665, 2010.

[51] L. Eggert and J. Heidemann. Application-level differentiated services for web servers. *World Wide Web*, 2:133–142, 1999.

[52] M. Eiblmaier, R. Mao, and X. Wang. Power management for main memory with access latency control. In *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2009.

[53] J. Entrialgo, D. F. García, J. García, M. García, P. Valledor, and M. S. Obaidat. Dynamic adaptation of response-time models for QoS management in autonomic systems. *J. Syst. Softw.*, 84:810–820, May 2011.

[54] Z. Fangling and W. Jinbiao. LP based MPC algorithm in distributed real-time systems with end-to-end tasks. In *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, volume 2, pages 1049–1054, 2005.

[55] C. Fehling, F. Leymann, and R. Mietzner. A framework for optimized distribution of tenants in cloud applications. In *International Conference on Cloud Computing*, pages 252 –259, 2010.

[56] N. Fescioglu-Unver and M. Kokar. Application of self controlling software approach to reactive tabu search. In *IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 297 –305, 2008.

[57] R. Fontaine, P. Laurencot, and A. Aussem. Mixed neural and feedback controller for Apache web server. *ICGST International Journal on Computer Network and Internet Research*, 09:25–30, 2009.

[58] X. Fu, X. Wang, and E. Puster. Dynamic thermal and timeliness guarantees for distributed real-time embedded systems. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 403 –412, 2009.

[59] Y. Fu, N. Kottenstette, Y. Chen, C. Lu, X. Koutsoukos, and H. Wang. Feedback thermal control for real-time systems. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 111 –120, 2010.

[60] Y. Fu, C. Lu, and H. Wang. Robust control-theoretic thermal balancing for server clusters. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1 –11, 2010.

[61] N. Gandhi, D. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh. MIMO control of an Apache web server: modeling and controller design. In *American Control Conference*, pages 4922–4927, 2002.

[62] A. Gao, D. Mu, H. Su, and W. Pan. Proportional hit rate in caching service: A feedback control approach. In *International Symposium on Computer Network and Multimedia Technology*, pages 1 –4, 2009.

[63] A. Gao, H. Zhou, Y. Hu, D. Mu, and W. Hu. Proportional delay differentiation service and load balancing in web cluster systems. In *IEEE Conference on Computer Communications Workshops INFOCOM*, pages 1 –2, 2010.

[64] D. F. Garcia, J. Garcia, J. Entrialgo, M. Garcia, P. Valledor, R. Garcia, and A. M. Campos. A QoS control mechanism to provide service differentiation and overload protection to internet scalable servers. *IEEE Transactions on Services Computing*, 2:3–16, 2009.

[65] S. Gerksic, D. Juricic, and S. Strmcnik. Adaptive implementation of Wiener model based nonlinear predictive control. In *IEEE International Symposium on Industrial Electronics, ISIE '99*, volume 3, pages 1159 –1164, 1999.

[66] S. Gerksic, D. Juricic, S. Strmcnik, and D. Matko. Wiener model based nonlinear predictive control. *International Journal of Systems Science*, 31:189 – 202, 2000.

[67] M. Gevers, L. Miskovic, D. Bonvin, and A. Karimi. Identification of multi-input systems: variance analysis and input design issues. *Automatica*, pages 559–572, 2006.

[68] F. Giri, F. Chaoui, M. Haloua, Y. Rochdi, and A. Naitali. Hammerstein model identification. In *Mediterranean Conference on Control and Automation*, MED'02, 2002.

[69] J. C. Gomez, A. Jutan, and E. Baeyens. Hammerstein an Wiener model identification using rational orthonormal bases. In *Latin American applied research*, 2003.

[70] J. C. Gomez, A. Jutan, and E. Baeyens. Wiener model identification and predictive control of a pH neutralisation process. *IEEE Proceedings-Control Theory and Applications*, 151:329–338, 2004.

[71] J. Gong and C.-Z. Xu. A gray-box feedback control approach for system-level peak power management. In *International Conference on Parallel Processing (ICPP)*, pages 555 –564, 2010.

[72] M. Gong and C. Williamson. Revisiting unfairness in web server scheduling. *Comput. Netw.*, 50(13):2183–2203, 2006.

[73] G. Goodwin, S. Graebe, and M. Salgado. *Control System Design.* Prentice Hall, 2001.

[74] A. Gounaris, C. Yfoulis, and N. Paton. An efficient load balancing LQR controller in parallel database queries under random perturbations. In *IEEE Control Applications, (CCA) Intelligent Control*, pages 794 –799, 2009.

[75] C. J. Guo, W. Sun, Y. Huang, Z. H. Wang, and B. Gao. A framework for native multi-tenancy application development and management. In *IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services*, pages 551 –558, 2007.

[76] A. Hagenblad. *Aspects of the Identication of Wiener Models.* PhD thesis, 2001.

[77] M. Harchol-Balter, B. Schroeder, N. Bansal, and M. Agrawal. Size-based scheduling to improve web performance. *ACM Trans. Comput. Syst.*, 21(2):207–233, May 2003.

[78] T. Heinis and C. Pautasso. Automatic configuration of an autonomic controller: An experimental study with zero-configuration policies. In *International Conference on Autonomic Computing*, pages 67 –76, 2008.

[79] J. Hellerstein, Y. Diao, and S. Parekh. A first-principles approach to constructing transfer functions for admission control in computing systems. In *IEEE Conference on Decision and Control*, volume 3, pages 2906 – 2912, 2002.

[80] J. L. Hellerstein. Self-managing systems: A control theory foundation. *IEEE Conference on Local Computer Networks*, pages 708–708, 2004.

[81] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley and Sons, 2004.

[82] D. Henriksson, Y. Lu, and T. Abdelzaher. Improved prediction for web server delay control. In *Euromicro Conference on Real-Time Systems, 2004*, pages 61 – 68, 2004.

[83] J. Heo, P. Jayachandran, I. Shin, D. Wang, T. Abdelzaher, and X. Liu. Optituner: On performance composition and server farm energy minimization application. *IEEE Transactions on Parallel and Distributed Systems*, 22(11):1871 –1878, 2011.

[84] J. Heo, X. Zhu, P. Padala, and Z. Wang. Memory overbooking and dynamic control of Xen virtual machines in consolidated environments. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 630 –637, 2009.

[85] H. Huang and A. Grimshaw. Automated performance control in a virtual distributed storage system. In *IEEE/ACM International Conference on Grid Computing*, pages 242 –249, 2008.

[86] W. Iqbal, M. Dailey, and D. Carrera. SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud. In *International Conference on Cloud Computing*, pages 243–253, 2009.

[87] B. G. Jeong, K. Yoo, and H. Rhee. Nonlinear model predictive control using a Wiener model of a continuous methyl methacrylate polymerization reactor. *Industrial and Engineering Chemistry Research*, 40:5968–5977, 2001.

[88] Y. Jiang, D. Meng, J. Zhan, and D. Liu. Adaptive mechanisms for managing the high performance web-based applications. In *International Conference on High-Performance Computing in Asia-Pacific Region*, pages 6 pp. –397, 2005.

[89] F. Jurado. Hammerstein-model-based predictive control of micro-turbines. *International Journal of Energy Research*, 30(7):511–521, 2006.

[90] A. Kalafatis, N. Arifin, L. Wang, and W. R. Cluett. A new approach to the identification of pH processes based on the Wiener model. *Chemical Engineering Science*, 50:3693 – 3701, 1995.

[91] A. D. Kalafatis, L. Wang, and W. R. Cluett. Identification of Wiener-type nonlinear systems in a noisy environment. *International Journal of Control*, 66:923 – 941, 1997.

[92] A. Kamra, V. Misra, and E. Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered web sites. In *IEEE International Workshop on Quality of Service*, pages 47 – 56, 2004.

[93] N. Kandasamy, S. Abdelwahed, and J. Hayes. Self-optimization in computer systems via online control: application to power management. In *International Conference on Autonomic Computing*, pages 54 – 61, may 2004.

[94] N. Kandasamy, S. Abdelwahed, and M. Khandekar. A hierarchical optimization framework for autonomic performance management of distributed computing systems. In *IEEE International Conference on Distributed Computing Systems*, page 9, 2006.

[95] K. D. Kang, J. Oh, and S. Son. Chronos: Feedback control of a real database system performance. In *IEEE International Real-Time Systems Symposium*, pages 267 –276, 2007.

[96] K. D. Kang, J. Oh, and Y. Zhou. Backlog estimation and management for real-time data services. In *Euromicro Conference on Real-Time Systems*, pages 289 –298, 2008.

[97] K.-D. Kang, S. Son, and J. Stankovic. Managing deadline miss ratio and sensor data freshness in real-time databases. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1200 – 1216, 2004.

[98] W. Kang, S. Son, and J. Stankovic. Design, implementation, and evaluation of a QoS-aware real-time embedded database. *IEEE Transactions on Computers*, (99):1, 2010.

[99] W. Kang, S. Son, J. Stankovic, and M. Amirijoo. I/O-aware deadline miss ratio management in real-time embedded databases. In *IEEE International Real-Time Systems Symposium*, pages 277 –287, 2007.

[100] W. Kang, S. H. Son, and J. A. Stankovic. DRACON: QoS management for large-scale distributed real-time databases. *Journal of Software*, 4(7):747–757, 2009.

[101] V. Kanodia and E. Knightly. Ensuring latency targets in multiclass web servers. *IEEE Transactions on Parallel and Distributed Systems*, 14(1):84 – 93, 2003.

[102] K. Kant and Y. Won. Server capacity planning for web traffic workload. *IEEE Trans. on Knowl. and Data Eng.*, 11:731–747, 1999.

[103] M. Karlsson. Maximizing the utility of a computer service using adaptive optimal control. In *International Conference on Networking, Sensing and Control*, pages 89 –94, 0-0 2006.

[104] M. Karlsson and M. Covell. Dynamic black-box performance model estimation for self-tuning regulators. In *International Conference on Automatic Computing*, pages 172–182, 2005.

[105] M. Karlsson and C. Karamanolis. Non-intrusive performance management for computer services. In *Middleware 2006*, Lecture Notes in Computer Science, pages 22–41. Springer Berlin / Heidelberg, 2006.

[106] M. Karlsson and C. Karamanolis. Non-intrusive performance management for computer services. In *Middleware*, volume 4290 of *Lecture Notes in Computer Science*, pages 22–41. Springer Berlin / Heidelberg, 2006. 10.1007/119250712.

[107] M. Karlsson, C. Karamanolis, and X. Zhu. Triage: Performance differentiation for storage systems using adaptive control. *Trans. Storage*, 1:457–480, 2005.

[108] M. Karlsson, X. Zhu, and C. Karamanolis. An adaptive optimal controller for non-intrusive performance differentiation in computing services. In *IEEE Conference on Control and Automation*, volume 2, pages 709 –714, 2005.

[109] T. Kelly. Utility-directed allocation. In *First Workshop on Algorithms and Architectures for Self-Managing Systems*, 2003.

[110] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Network Operations and Management Symposium*, pages 373 –381, 2006.

[111] M. Kihl, A. Robertsson, and B. Wittenmark. Analysis of admission control mechanisms using non-linear control theory. In *IEEE International Symposium on Computers and Communication*, volume 2, pages 1306 – 1311, 2003.

[112] M. Kjaer, M. Kihl, and A. Robertsson. Resource allocation and disturbance rejection in web servers using SLAs and virtualized servers. *IEEE Transactions on Network and Service Management*, 6(4):226 –239, 2009.

[113] M. Kjaer and A. Robertsson. Analysis of buffer delay in web-server control. In *American Control Conference*, ACC, pages 1047 –1052, 2010.

[114] M. Kjser, M. Kihl, and A. Robertsson. Response-time control of a single server queue. In *IEEE Conference on Decision and Control*, pages 3812 –3817, 2007.

[115] B. J. Ko, K. W. Lee, K. Amiri, and S. Calo. Scalable service differentiation in a shared storage cache. In *International Conference on Distributed Computing Systems*, pages 184 – 193, may 2003.

[116] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu. Hybrid supervisory utilization control of real-time systems. In *IEEE Real Time and Embedded Technology and Applications Symposium*, pages 12 – 21, 2005.

[117] N. K.S. and D. O.A. Adaptive control using multiple models, switching, and tuning. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 20000*, pages 159 –164, 2000.

[118] D. Kusic and N. Kandasamy. Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems. In *IEEE International Conference on Autonomic Computing*, pages 74 – 83, 2006.

[119] D. Kusic, N. Kandasamy, and G. Jiang. Approximation modeling for the online performance management of distributed computing systems. In *International Conference on Autonomic Computing*, page 23, 2007.

[120] D. Kusic, N. Kandasamy, and G. Jiang. Combined power and performance management of virtualized computing environments serving session-based workloads. *IEEE Transactions on Network and Service Management*, 8(3):245 –258, 2011.

[121] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. In *International Conference on Autonomic Computing*, pages 3 –12, 2008.

[122] T. Kwok and A. Mohindra. Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications. In *International Conference on Service-Oriented Computing, ICSOC'08*, pages 633–648. Springer-Verlag, 2008.

[123] D. Lawrence, J. Guan, S. Mehta, and L. Welch. Adaptive scheduling via feedback control for dynamic real-time systems. In *International Conference on Performance, Computing, and Communications*, pages 373 –378, 2001.

[124] C. Lefurgy, X. Wang, and M. Ware. Server-level power control. In *International Conference on Autonomic Computing*, page 4, 2007.

[125] L. Lennart. *System identification: theory for the user.* Prentice-Hall, Inc., 1997.

[126] N. Leontiou, D. Dechouniotis, and S. Denazis. Adaptive admission control of distributed cloud services. In *International Conference on Network and Service Management*, pages 318 –321, 2010.

[127] R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based web services. In *International Symposium on Integrated Network Management*, pages 247 – 261, 2003.

[128] Q. Li, Q. F. Hao, L. M. Xiao, and Z. J. Li. An integrated approach to automatic management of virtualized resources in cloud environments. *Comput. J.*, 54:905–919, 2011.

[129] X. H. Li, T. C. Liu, Y. Li, and Y. Chen. SPIN: Service performance isolation infrastructure in multi-tenancy environment. In *International Conference on Service-Oriented Computing*, ICSOC'08, pages 649–663, 2008.

[130] Z. Li, D. Levy, S. Chen, and J. Zic. Auto-tune design and evaluation on staged event-driven architecture. In *Model Driven Development for Middleware*, MODDM'06, pages 1–6, 2006.

[131] K. Liang, X. Zhou, K. Zhang, and R. Sheng. An adaptive performance management method for failure detection. In *ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, pages 51–56, 2008.

[132] H. C. Lim, S. Babu, and J. S. Chase. Automated control for elastic storage. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 1–10, 2010.

[133] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh. Automated control in cloud computing: challenges and opportunities. In *Workshop on Automated control for datacenters and clouds*, ACDC'09, pages 13–18, 2009.

[134] H. Lin, K. Sun, S. Zhao, and Y. Han. Feedback-control-based performance regulation for multi-tenant applications. In *International Conference on Parallel and Distributed Systems*, pages 134 –141, 2009.

[135] S. Lin and G. Manimaran. Double-loop feedback-based scheduling approach for distributed real-time systems. In *High Performance Computing*, volume 2913 of *Lecture Notes in Computer Science*, pages 268–278. Springer Berlin / Heidelberg, 2003.

[136] S. Lin and G. Manimaran. A feedback-based adaptive algorithm for combined scheduling with fault-tolerance in real-time systems. In *High Performance Computing*, volume 3296 of *Lecture Notes in Computer Science*, pages 101–110. Springer Berlin / Heidelberg, 2005.

[137] Y.-D. Lin, C.-M. Tien, S.-C. Tsao, R.-H. Feng, and Y.-C. Lai. Multiple-resource request scheduling for differentiated QoS at website gateway. *Comput. Commun.*, 31:1993–2004, 2008.

[138] H. Liu and S. Wee. Web server farm in the cloud: Performance evaluation and dynamic architecture. *Lecture Notes in Computer Science*, 5931:369–380, 2009.

[139] X. Liu, J. Heo, L. Sha, and X. Zhu. Queueing-model-based adaptive control of multi-tiered web applications. *IEEE Transactions on Network and Service Management*, 5(3):157 –167, 2008.

[140] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal multivariate control for differentiated services on a shared hosting platform. In *IEEE Conference on Decision and Control*, pages 3792 –3799, 2007.

[141] X. Liu, X. Zhu, S. Singhal, and M. Arlitt. Adaptive entitlement control of resource containers on shared servers. In *IFIP/IEEE International Symposium on Integrated Network Management*, pages 163 – 176, may 2005.

[142] Z. Liu and D. Mu. Coordinating power and performance in virtualized environments. In *IEEE International Conference on Computer Science and Automation Engineering*, volume 3, pages 705 –709, 2011.

[143] C. Lu. *Feedback Control Real-Time Scheduling.* PhD thesis, University of Virginia, 2001.

[144] C. Lu, G. A. Alvarez, and J. Wilkes. Aqueduct: Online data migration with performance guarantees. In *USENIX Conference on File and Storage Technologies*, FAST'02, 2002.

[145] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Transactions on Parallel and Distributed Systems*, 17(9):1014 –1027, 2006.

[146] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst.*, 23:85–126, 2002.

[147] C. Lu, X. Wang, and C. Gill. Feedback control real-time scheduling in orb middleware. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 37 – 48, may 2003.

[148] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Transactions on Parallel and Distributed Systems*, 16(6):550 – 561, 2005.

[149] C. Lu, X. Wang, and X. Koutsoukos. Feedback utilization control in distributed real-time systems with end-to-end tasks. *IEEE Trans. Parallel Distrib. Syst.*, 16:550–561, 2005.

[150] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, and X. Liu. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 208 – 217, may 2003.

[151] Y. Lu, T. Abdelzaher, C. Lu, and G. Tao. An adaptive control framework for QoS guarantees and its application to differentiated caching. In *IEEE International Workshop on Quality of Service*, pages 23 – 32, 2002.

[152] Y. Lu, T. Abdelzaher, and G. Tao. Direct adaptive control of a web cache system. In *American Control Conference*, volume 2, pages 1625 – 1630, 4-6, 2003.

[153] Y. Lu, A. Saxena, and T. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *International Conference on Distributed Computing Systems*, pages 615 –622, 2001.

[154] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *International conference on Compilers, architecture, and synthesis for embedded systems*, CASES'02, pages 156–163, 2002.

[155] W. C. L.Wang. *From Plant Data to Process Control:Ideas for Process Identification and PID Design.* Taylor and Francis, 2000.

[156] K. M. M, K. Baclawski, and Y. A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14(3):37–45, 1999.

[157] M. Maggio and A. Leva. Toward a deeper use of feedback control in the design of critical computing system components. In *IEEE Conference on Decision and Control*, pages 5985 –5990, 2010.

[158] V. Mathur, P. Patil, V. Apte, and K. Moudgalya. Adaptive admission control for web applications with variable capacity. In *International Workshop on Quality of Service, 2009*, pages 1 –5, 2009.

[159] Matlab. *Matlab System Identification Toolbox.* MATHWORKS, 2009a.

[160] J. Matthews, T. Garfinkel, C. Hoff, and J. Wheeler. Virtual machine contracts for datacenter and cloud computing environments. In *Workshop on Automated control for datacenters and clouds*, ACDC'09, pages 25–30, 2009.

[161] A. Merchant, M. Uysal, P. Padala, X. Zhu, S. Singhal, and K. Shin. Maestro: quality-of-service in large disk arrays. In *International conference on Autonomic computing*, ICAC'11, pages 245–254, 2011.

[162] R. Mietzner, T. Unger, R. Titze, and F. Leymann. Combining different multi-tenancy patterns in service-oriented applications. In *Enterprise Distributed Object Computing Conference*, EDOC'09, pages 131 –140, 2009.

[163] H. Naccache, G. C. Gannod, and K. A. Gary. A self-healing web server using differentiated services. In *International Conference Service-Oriented Computing*, ICSOC'06, pages 203–214, 2006.

[164] K. Narendra and J. Balakrishnan. Adaptive control using multiple models. *IEEE transactions on automatic control*, 42:171–187, 1997.

[165] K. Narendra and C. Xiang. Adaptive control of discrete-time systems using multiple models. *IEEE Transactions on Automatic Control*, 45:1669 –1686, 2000.

[166] K. S. Narendra and J. Balakrishnan. Improving transient response of adaptive control systems using multiple models and switching. In *Conference on Decision and Control, 1993*, pages 1067–1072, 1993.

[167] K. S. Narendra, J. Balakrishnan, and M. K. Ciliz. Adaptation and learning using multiple models, switching, and tuning. *IEEE Control Systems Magazine*, 15(3):37–51, 1995.

[168] K. S. Narendra, O. A. Driollet, M. Feiler, and K. George. Adaptive control using multiple models, switching, and tuning. *International journal of adaptive control and signal processing*, pages 1–16, 2003.

[169] R. Nathuji, P. England, P. Sharma, and A. Singh. *Feedback driven QoS-aware power budgeting for virtualized servers.* 2009.

[170] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud. Autonomic virtual resource management for service hosting platforms. In *Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD'09, pages 1–8, 2009.

[171] B. Nidhi and R. D. K. Continuous-time multiple-input, multiple-output Wiener modeling method. *Industrial and Engineering Chemistry Research*, 42:5583–5595, 2003.

[172] Y. Niu and G. Dai. Reservation-based state feedback scheduler for hybrid real-time systems. In *IEEE International Conference on High Performance Computing and Communications*, pages 198 –204, 2008.

[173] S. J. Norquay, A. Palazoglu, and J. Romagnoli. Model predictive control based on Wiener models. *Chemical Engineering Science*, 53:75–84, 1998.

[174] S. J. Norquay, A. Palazoglu, and J. A. Romagnoli. Application of Wiener model predictive control (WMPC) to an industrial C2-splitter. *Journal of Process Control*, 9:461 – 473, 1999.

[175] K. Ogata. *Modern Control Engineering.* Prentice Hall, 4th edition, 2001.

[176] J. Oh and K. D. Kang. An approach for real-time database modeling and performance management. In *Real Time and Embedded Technology and Applications Symposium*, pages 326–336, 2007.

[177] P. Padala. *Automated Management of Virtualized Data Centers.* PhD thesis, University of Michigan, 2010.

[178] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *European conference on Computer systems*, EuroSys'09, pages 13–26, 2009.

[179] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *SIGOPS/EuroSys European Conference on Computer Systems*, EuroSys'07, pages 289–302, 2007.

[180] W. Pan, D. Mu, H. Wu, and L. Yao. Feedback control-based QoS guarantees in web application servers. In *International Conference on High Performance Computing and Communications*, pages 328 –334, 2008.

[181] W. Pan, D. Mu, H. Wu, X. Zhang, and L. Yao. Feedback control-based database connection management for proportional delay differentiation-enabled web application servers. In *Network and Parallel Computing*, volume 5245 of *Lecture Notes in Computer Science*, pages 74–85. 2008.

[182] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23:127–141, 2002.

[183] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus. Using control theory to achieve service level objectives in performance management. *Real-Time Syst.*, 23:127–141, 2002.

[184] S.-M. Park and M. Humphrey. Feedback-controlled resource sharing for predictable escience. In *ACM/IEEE conference on Supercomputing*, SC'08, pages 13:1–13:11, 2008.

[185] S.-M. Park and M. Humphrey. Self-tuning virtual machines for predictable escience. In *International Symposium on Cluster Computing and the Grid*, CCGRID'09, pages 356–363, 2009.

[186] S.-M. Park and M. Humphrey. Predictable high-performance computing using feedback control and admission control. *IEEE Transactions on Parallel and Distributed Systems*, 22(3):396 –411, 2011.

[187] M. Pathirage, S. Perera, I. Kumara, and S. Weerawarana. A multi-tenant architecture for business process executions. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 121 –128, 2011.

[188] T. Patikirikorala. A simulation model to implement multiple client class server-client software architecture. Technical report. http://www.ict.swin.edu.au/personal/tpatikirikorala/Research.htm.

[189] T. Patikirikorala, A. Colman, J. Han, and L. Wang. A multi-model framework to implement self-managing control systems for QoS management. In *International symposium on Software engineering for adaptive and self-managing systems*, pages 218–227, 2011.

[190] T. Patikirikorala, A. Colman, J. Han, and L. Wang. An evaluation of multi-model self-managing control schemes for adaptive performance management of software systems. *Journal of Systems and Software*, 85(12):2678 – 2696, 2012.

[191] T. Patikirikorala, A. Colman, J. Han, and L. Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 33–42, 2012.

[192] T. Patikirikorala, A. Colman, J. Han, and L. Wang. Differentiated performance management in virtualized environments using nonlinear control. Technical report, Swinburne university of technology, 2013.

[193] T. Patikirikorala, L. Wang, and A. Colman. Towards optimal performance and resource management in web systems via model predictive control. In *Australian Control Conference*, AUCC'11, pages 469–474, 2011.

[194] T. Patikirikorala, L. Wang, A. Colman, and J. Han. Hammerstein-Wiener nonlinear model based predictive control for relative QoS performance and resource management of software systems. *Control Engineering Practice*, 20(1):49 – 61, 2011.

[195] R. Pearson and M. Pottmann. Gray-box identification of block-oriented nonlinear models. *Journal of Process Control*, 10:301–315, 2000.

[196] E. Peymani, A. Fatehi, and A. K. Sedigh. Automatic learning in multiple model adaptive control. In *UKACC Control Conference*, 2008.

[197] C. Poussot-Vassal, M. Tanelli, and M. Lovera. Linear parametrically varying MPC for combined quality of service and energy management in web service systems. In *American Control Conference*, pages 3106 –3111, 2010.

[198] W. Qin and Q. Wang. Feedback performance control for computer systems: an LPV approach. In *American Control Conference*, pages 4760 – 4765, 2005.

[199] W. Qin and Q. Wang. Using stochastic linear-parameter-varying control for CPU management of internet servers. In *IEEE Conference on Decision and Control*, pages 3824 –3829, 2007.

[200] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu. No power struggles: coordinated multi-level power management for the data center. In *International conference on Architectural support for programming languages and operating systems*, ASPLOS XIII, pages 48–59, 2008.

[201] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Design and evaluation of load control in web server systems. In *American Control Conference*, volume 3, pages 1980 –1985, 2004.

[202] J. Rolia, L. Cherkasova, M. Arlitt, and A. Andrzejak. A capacity management service for resource pools. In *International workshop on Software and performance*, WOSP'05, pages 229–237, 2005.

[203] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4:1–42, 2009.

[204] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. *ACM Trans. Internet Technol.*, 6(1):20–52, 2006.

[205] N. Shankaran, X. D. Koutsoukos, D. C. Schmidt, Y. Xue, and C. Lu. Hierarchical control of multiple resources in distributed real-time and embedded systems. In *Euromicro Conference on Real-Time Systems*, pages 151–160, 2006.

[206] P. Shao-Liang, L. Shan-Shan, L. Xiang-Ke, P. Yu-Xing, and Y. Hui. Feedback control with prediction for thread allocation in pipeline architecture web server. In *Distributed Computing and Networking*, volume 4308 of *Lecture Notes in Computer Science*, pages 454–465. Springer Berlin / Heidelberg, 2006.

[207] Y. Shiotani and Y. Kobayashi. Identification of multi-input multi-output Wiener-type nonlinear systems. In *ICROS-SICE International Joint Conference*, 2009.

[208] B. Solomon, D. Ionescu, M. Litoiu, and M. Mihaescu. A real-time adaptive control of autonomic computing environments. In *Conference of the center for advanced studies on Collaborative research*, CASCON'07, pages 124–136, 2007.

[209] A. Soria-Lopez, P. Mejia-Alvarez, and J. Cornejo. Feedback scheduling of power-aware soft real-time tasks. In *International Conference on Computer Science*, pages 266 – 273, 2005.

[210] J. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu. Feedback control scheduling in distributed real-time systems. In *IEEE Real-Time Systems Symposium*, pages 59 – 70, 2001.

[211] G. Starnberger, L. Froihofer, and K. M. Goeschka. Adaptive run-time performance optimization through scalable client request rate control. In *Joint WOSP/SIPEW international conference on Performance engineering*, ICPE, pages 167–178, 2011.

[212] A. J. Storm, C. Garcia-Arellano, S. S. Lightstone, Y. Diao, and M. Surendra. Adaptive self-tuning memory in DB2. In *International conference on Very large data bases*, VLDB'06, pages 1081–1092, 2006.

[213] W. Sun, X. Zhang, C. J. Guo, P. Sun, and H. Su. Software as a service: Configuration and customization perspectives. In *Congress on Services Part II*, pages 18 –25, 2008.

[214] Y. Tang, X. Luo, Q. Hui, and R. Chang. On generalized low-rate denial-of-quality attack against internet services. In *International Workshop on Quality of Service*, pages 1 –5, 2009.

[215] A. Tesanovic, M. Amirijoo, M. Björk, and J. Hansson. Empowering configurable QoS management in real-time systems. In *International conference on Aspect-oriented software development*, AOSD'05, pages 39–50, 2005.

[216] F. Tian, W. Xu, and J. Sun. Web QoS control using fuzzy adaptive PI controller. In *Ninth International Symposium on Distributed Computing and Applications to Business Engineering and Science (DCABES)*, pages 72 –75, 2010.

[217] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao. Load shedding in stream databases: a control-based approach. In *International conference on Very large data bases*, VLDB'06, pages 787–798. VLDB Endowment, 2006.

[218] B. Urgaonkar and P. Shenoy. Cataclysm: policing extreme overloads in internet applications. In *International conference on World Wide Web*, WWW'05, pages 740–749, 2005.

[219] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal. Dynamic provisioning of multi-tier internet applications. In *International Conference on Autonomic Computing*, ICAC'05, pages 217 –228, 2005.

[220] H. N. Van, F. Tran, and J.-M. Menaud. Performance and power management for cloud infrastructures. In *International Conference on Cloud Computing*, pages 329 –336, 2010.

[221] J. Voros. An iterative method for Hammerstein-Wiener systems parameter identification. *Journal of electrical engineering*, pages 328–331, 2004.

[222] B. Wahlberg, M. Jansson, T. Matsko, and M. A. Molander. Experiences from subspace system identification - comments from process industry users and researchers. *Modeling, Estimation and Control*, pages 315–327, 2007.

[223] L. Wang. *Model Predictive Control System Design and Implementation Using MATLAB*. Springer Publishing Company, Incorporated, 2009.

[224] M. Wang, N. Kandasamy, A. Guez, and M. Kam. Distributed cooperative control for adaptive performance management. *IEEE Internet Computing*, 11(1):31 –39, 2007.

[225] R. Wang, D. M. Kusic, and N. Kandasamy. A distributed control framework for performance management of virtualized computing environments. In *International conference on Autonomic computing*, ICAC '10, pages 89–98, 2010.

[226] X. Wang, M. Chen, and X. Fu. MIMO power control for high-density servers in an enclosure. *IEEE Transactions on Parallel and Distributed Systems*, 21(10):1412 –1426, 2010.

[227] X. Wang, M. Chen, C. Lefurgy, and T. Keller. SHIP: Scalable hierarchical power control for large-scale data centers. In *International Conference on Parallel Architectures and Compilation Techniques*, pages 91 –100, 2009.

[228] X. Wang, Y. Chen, C. Lu, and X. Koutsoukos. Towards controllable distributed real-time systems with feasible utilization control. *IEEE Transactions on Computers*, 58(8):1095 – 1110, 2009.

[229] X. Wang, X. Fu, X. Liu, and Z. Gu. PAUC: Power-aware utilization control in distributed real-time systems. *IEEE Transactions on Industrial Informatics*, 6(3):302 –315, 2010.

[230] X. Wang, D. Jia, C. Lu, and X. Koutsoukos. DEUCON: Decentralized end-to-end utilization control for distributed real-time systems. *IEEE Trans. on Parallel and Distributed Systems*, 18:2007, 2007.

[231] X. Wang and Y. Wang. Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions o Parallel and Distributed Systems*, 22(2):245 –259, 2011.

[232] Y. Wang, R. Deaver, and X. Wang. Virtual batching: Request batching for energ conservation in virtualized servers. In *International Workshop on Quality of Service*, pages 1 –9, 2010.

[233] Y. Wang and X. Wang. Power optimization with performance assurance for multi-tier applications in virtualized data centers. In *International Conference on Parallel Processing Workshops*, pages 512 –519, 2010.

[234] Y. Wang, X. Wang, M. Chen, and X. Zhu. Partic: Power-aware response time control for virtualized web servers. *IEEE Transactions on Parallel and Distributed Systems*, 22(2):323 –336, 2011.

[235] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. Watson, W. Rivera, X. Zhu, and C. Hyser. Appraise: application-level performance management in virtualized server environments. *IEEE Transactions on Network and Service Management*, 6(4):240 –254, 2009.

[236] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, and T. Kelly. Autoparam: Automated control of application-level performance in virtualized server environments. In *Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2007.

[237] Z. Wang, C. McCarthy, X. Zhu, P. Ranganathan, and V. Talwar. Feedback control algorithms for power management of servers. In *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, 2008.

[238] Z. Wang, X. Zhu, and S. Singhal. Utilization and SLO-based control for dynamic sizing of resource partitions. In *Distributed Systems, Operations and Management*, pages 133–144, 2005.

[239] Z. H. Wang, C. J. Guo, B. Gao, W. Sun, Z. Zhang, and W. H. An. A study and performance evaluation of the multi-tenant data tier design patterns for service oriented computing. In *IEEE International Conference on e-Business Engineering*, pages 94–101, 2008.

[240] J. Wei and C.-Z. Xu. Feedback control approaches for quality of service guarantees in web servers. In *American Fuzzy Information Processing Society*, pages 700 – 705, 2005.

[241] J. Wei and C.-Z. Xu. eQoS: Provisioning of client-perceived end-to-end QoS guarantees in web servers. *IEEE Transactions on Computers*, 55(12):1543 –1556, 2006.

[242] J. Wei, X. Zhou, and C.-Z. Xu. Robust processing rate allocation for proportional slowdown differentiation on internet servers. *IEEE Transactions on Computers*, 54(8):964 – 977, 2005.

[243] L. Wei and H. Yu. Research on a soft real-time scheduling algorithm based on hybrid adaptive control architecture. In *American Control Conference*, volume 5, pages 4022 – 4027, 2003.

[244] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD international conference on Management of data*, SIGMOD'09, pages 889–896, 2009.

[245] D. Westwick and M. Verhaegen. Identifying MIMO Wiener systems using subspace model identification methods. *Signal Processing*, 52:235 – 258, 1996.

[246] M. Woodside, T. Zheng, and M. Litoiu. Service system resource management based on a tracked layered performance model. In *IEEE International Conference on Autonomic Computing*, pages 175 – 184, 2006.

[247] K. Wu, D. Lilja, and H. Bai. The applicability of adaptive control theory to QoS design: limitations and solutions. In *IEEE International Parallel and Distributed Processing Symposium*, pages 4–8, 2005.

[248] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu. Economical and robust provisioning of n-tier cloud workloads a multi-level control approach. In *International Conference on Distributed Computing Systems*, pages 571 –580, 2011.

[249] C.-Z. Xu, B. Liu, and J. Wei. Model predictive feedback control for QoS assurance in webservers. *Computer*, 41(3):66 –72, 2008.

[250] W. Xu, Z. Tan, A. Fox, and D. Patterson. Regulating workload in J2EE application servers. In *International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, FeBID, 2006.

[251] W. Xu, X. Zhu, S. Singhal, and Z. Wang. Predictive control for dynamic resource allocation in enterprise data centers. In *IEEE/IFIP Network Operations and Management Symposium*, pages 115 –126, 2006.

[252] C. Xu-Dong, Z. Qing-Xin, L. Yong, and X. Guang Ze. End-to-end deadline control for aperiodic tasks in distributed real-time systems. *J. Supercomput.*, 43:225–240, 2008.

[253] H. Yansu, D. Guanzhong, G. Ang, and P. Wenping. A self-tuning control for web QoS. In *International Conference on Information Engineering and Computer Science*, ICIECS, pages 1 –4, 2009.

[254] J. Yao, X. Liu, X. Chen, X. Wang, and J. Li. Online decentralized adaptive optimal controller design of CPU utilization for distributed real-time embedded systems. In *American Control Conference*, pages 283 –288, 2010.

[255] J. Yao, X. Liu, and X. Zhu. Reduced dimension control based on online recursive principal component analysis. In *American Control Conference*, pages 5713 –5718, 2009.

[256] N. Ye, E. S. Gel, X. Li, T. Farley, and Y.-C. Lai. Web server QoS models: applying scheduling rules from production planning. *Comput. Oper. Res.*, 32:1147–1164, 2005.

[257] G. You and Y. Zhao. Dynamic requests scheduling model in multi-core web server. In *International Conference on Grid and Cloud Computing*, GCC '10, pages 201–206, 2010.

[258] P. Youbin, D. Vrancic, and R. Hanus. Anti-windup, bumpless, and conditioned transfer techniques for PID controllers. *Control Systems Magazine, IEEE*, 16(4):48–57, 1996.

[259] C. Yu and D. Qionghai. A improved elastic scheduling algorithm based on feedback control theory. In *International Conference on Signal*, ICSP, pages 1330 – 1339, 2004.

[260] J. Zhang and Y. Zou. Predictive control for performance guarantees in soft real-time scheduling systems. In *Congress on Intelligent Control and Automation*, volume 2, pages 6944 –6948, 2006.

[261] R. Zhang, C. Lu, T. Abdelzaher, and J. Stankovic. Controlware: a middleware architecture for feedback control of software performance. In *International Conference on Distributed Computing Systems*, pages 301 – 310, 2002.

[262] X. Zhou, Y. Cai, and E. Chow. An integrated approach with feedback control for robust web QoS design. *Comput. Commun.*, 29:3158–3169, 2006.

[263] X. Zhou, J. Wei, and C.-Z. Xu. Quality-of-service differentiation on the internet: A taxonomy. *Journal of Network and Computer Applications*, 30(1):354 – 383, 2007.

[264] Y. Zhou and K. D. Kang. Deadline assignment and tardiness control for real-time data services. In *Euromicro Conference on Real-Time Systems*, ECRTS, pages 100 –109, 2010.

[265] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. What does control theory bring to systems research? *ACM SIGOPS Operating Systems Review*, 43:62–69, 2009.

[266] X. Zhu, Z. Wang, and S. Singhal. Utility-driven workload management using nested control design. In *American Control Conference*, pages 14–16, 2006.

[267] X. Zhu, D. Young, B. Watson, Z. Wang, J. Rolia, S. Singhal, B. McKee, C. Hyser, D. Gmach, R. Gardner, T. Christian, and L. Cherkasova. 1000 islands: Integrated capacity and workload management for the next generation data center. In *International Conference on Autonomic Computing*, pages 172 –181, 2008.