The Pennsylvania State University
The Graduate School

FEEDBACK PERFORMANCE CONTROL FOR SELF-MANAGING COMPUTER SYSTEMS: AN LPV CONTROL THEORETIC APPROACH

A Thesis in
Mechanical Engineering
by
Wubi Qin

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2007

The thesis of Wubi Qin was reviewed and approved* by the following:

Qian Wang
Assistant Professor of Mechanical Engineering
Thesis Advisor, Chair of Committee

Asok Ray
Distinguished Professor of Mechanical Engineering

Kon-Well Wang
William E. Diefenderfer Chaired Professor of Mechanical Engineering

Anand Sivasubramaniam
Professor of Computer Science & Engineering

Karen A. Thole
Professor of Mechanical Engineering
Department Head of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School.

Abstract

High-performance server systems have been widely used in today's commercial and scientific server environments, for example, Web and Email applications, online trading and multimedia services, and many more. This research investigates the autonomic performance management of computer server systems in the context of Internet hosting centers. Two scenarios of the performance mangagement problem are studied: admission control and resource allocation. This research contributes to the problem in three major aspects: modeling of Internet services via new approaches, more realistic and advanced control designs, and intensive performance evaulations of various design solutions.

Modeling wise, it is learned that the linear methods may not be adequate for Internet services modeling when there are large variations in load conditions. Consequently, more sophisticated nonlinear system modeling is needed to improve system performance and robustness with respect to large variations of load conditions. A linear uncertain model and a Linear-Parameter-Varying (LPV) model of hosting center server systems are derived based on theoretical analysis of transient queueing dynamics, as well as LPV system identification, where workload characterizing parameters are utilized as scheduling variables.

Exisiting feedback control based performance management for Internet server systems relies on worst-case estimates of load and resource availability thus provisions resources to meet peak demands. Since the worst-case resource demand is likely to be significantly higher than its normal usage, these methods could be economically unfavorable. This work focuses on the development of an LPV control design for the performance management of server where time-varying effects of the system are explicitly considered by utilizing workload arrival and service parameters as scheduling variables. Further, it is widely noticed that workload parameters of Internet services are more or less unpredictable. This makes deterministic modeling and control design approaches either too conservative or unable

to meet performance service level agreements (SLA). Probabilistic modeling and control design methods are proposed to balance between the risk of missing performance SLA and the resource efficiency.

Request level approach to provide response time guarantee based on detailed analysis of requests response time components is studied. A request level scheduling algorithm is proposed that can be used with workload characterization and modeling techniques. Preliminary results in a response time guarantee problem show impressive request level performance over a wide range of sampling intervals.

Intensive simulations are conducted using real Web server workloads. Performances of the proposed LPV based modeling and control design approaches are benchmarked using queueing theory based provisioning results and existing linear control modeling & designs. It is shown that the proposed method outperforms linear control and conventional queueing-theory based designs. These evaluations provide guidelines on which method to use for a particular problem. The results from this research can be easily generalized to accommodate more complicated models in characterizing workloads and server environments to enhance system performance.

Table of Contents

List of Figures

List of Tables

List of Symbols

Λ	Arrival process, p. 11
λ	Arrival rate, p. 11
s	Service demand, p. 11
μ	Service rate, p. 11
K	Buffer size, p. 11
T	Response time, p. 12
q	Queue length, p. 12
ρ	Utilization ratio, p. 12
$u(t)$	Control input, p. 15
$y(t)$	System output variable, p. 15
$W(t)$	Waiting time, p. 15
$q(t)$	Queue length, p. 15
$\bar{T}$	Reference variable, p. 15
$A(\cdot), B(\cdot)$	System matrices, p. 21
$r(\cdot)$	Workload parameters, p. 22
J	Cost function, p. 24
W_e, W_u	Weighting functions in H_∞ control design, p. 24

$K(r), K(p)$	Parameter-dependent controller, p. 24
$\bar{(\cdot)}$	Equilibrium value of the corresponding variable, p. 41
$\hat{(\cdot)}$	Deviation value of the corresponding variable, p. 41
Δt	Sampling time interval, p. 36
ΔT	Server configuration (allocation) time interval, p. 36
$\Phi(\omega)$	Characteristic function of an α-stable distribution, p. 63
$S^{\alpha}_{\sigma,\chi,m}$	Self-similar process with parameters σ,χ,m, and α, p. 63
H	Level of self-similarity, p. 64
$N_{\alpha,H}$	A stationary process, p. 64
α	Stability index of an α-stable distribution, p. 64
β_A	Skew parameter, p. 65
σ	Scale parameter, p. 65
$p(k)$	Time varying exogenous parameter, p. 65
$w_s(n)$, $w_q(n)$	The service time and waiting time components of the response time of a request, p. 94
$t_{iA}(n,1)$	Inter-arrival time between the request $n(k)$, and the request been served when $n(k)$ enters the queue, p. 94
$L(n)$	Instantaneous request length for request n, p. 96
$t_{arrival}(n)$	Tagged arrival time of request n, p. 97
L_n	The number of requests arrived before the n^{th} request that are still in the queue, p. 97

Acknowledgments

This research is supported in part by NSF under Grants 0325056 and 0409184, support from the Pittsburgh Greenhouse Project. I would like to take this opportunity to thank my advisor Dr. Qian Wang. I am grateful for her advices and encouragements. It is impossible to achieve what has been done without her support. Special thanks should be given to my committee members, Dr Anand Sivasubramaniam, Dr. Asok Ray, and Dr. Kon-Well Wang, for their opinions and suggestions. I would also like to thank Dr Natarajan Gautam and my collaborators, now Dr. Amitayu Das and Dr Yiyu Chen, for their ideas and contributions.

Dedication

to my father, you are survived by mom, sons and daughters.
to my mother, I couldn't have gone this far in my career without your encouragement and support.
to my wife and son.

Chapter 1

Introduction

1.1 Motivation

High performance server systems have been widely used in today's commercial and scientific server environments, for example, Web and Email applications, online trading and multimedia services, and many more. With the fast-growing total cost of ownership (initial investment, management, and maintenance) and the increasing complexity of server systems, many enterprises are outsourcing their IT service needs by offloading their applications to third party hosting/data centers.

A hosting center often operates thousands of servers and provides services to multiple applications at the same time. A guaranteed level of performance, which is referred to as Quality of Service (QoS) delivered to end customers, is often part of a Service Level Agreement (SLA) specified between the hosting center and each application owner. The SLA may include different performance specifications (e.g., response time, bandwidth and throughput) in terms of the amount of money the application owner pays the hosting center. Consequently, a hosting center needs to allocate sufficient resources (CPU, memory and I/O bandwidth) to each individual application so that the SLA for each application is met; on the other hand, the hosting center would like to optimize the resource allocation to each application such that the profit can be maximized.

Traditional performance management and resource allocation for large clusters of server systems have been relied on queueing-theory based steady-state analysis and static optimization. Since the incoming traffic (request arrival and request

demand) for an individual application is essentially stochastic and time varying, for example, the workload/traffic pattern to a particular web site may be bursty and exhibit seasonal phenomena, static resource allocation is not favored. Static allocation could either cause the system to be over-utilized so that the SLA can not be met, or allocating for worst-case load conditions to meet the peak demand, which is obviously economically unattractive since the worst-case resource demand is likely to be significantly higher than its normal usage.

In order for the system to adapt to dynamically-varying workload and operating conditions, feedback based approaches are strongly motivated. There has been increasing research interest in applying control-theoretic approaches to the management of computer systems, though mainly in the area of communication systems. The research on the management and resource allocation for server systems is still in its infancy; most of existing work on performance control of Web, e-commerce and storage servers is limited to system-identification based linear ARX models and classical PID control designs.

This thesis is motivated by following two reasons. First, the inherent nonlinear dependence of system performance metrics with respect to resource allocation variables, as well as the demand on adaptation to stochastic, time-varying load conditions calls for the development of nonlinear modeling and design methodologies to improve system performance. Second, a workload is often characterized by two complimentary distributions: the request inter-arrival time and the service demand distributions, which capture the workload intensity and its variability. Consequently, probabilistic approaches to system modeling and control designs are in demand. Rather than over-provisioning for the worst-case load condition, performance management in today's Internet hosting center would limit rather than eliminate the risk of failing to meet request demand, allocating to each application the minimal server resources needed for acceptable service quality and leaving surplus resources to deploy elsewhere.

1.2 Research objectives

The general goal of this research is to develop systematic control-theoretic approaches to automating performance management for large clusters of Internet

server systems, in particular, hosting centers. Specific objectives of this thesis are as follows,

- Development of control-oriented nonlinear models for performance management of Internet hosting centers that directly characterize the system dependence on the stochastic, dynamically changing load and operating conditions;
- Development of control design methodologies for the performance management of hosting centers that explicitly utilize detail workload information (distributions) so that the performance specifications can be met in the presence of dynamically varying load conditions, based on the models derived in the first objective;
- Development of probabilistic approaches to robust control design to balance between the risk of missing performance SLA and the resource efficiency for the performance management of hosting centers.

To be more specific, a Internet service is modeled as a dynamical linear parameter varying (LPV) system, parameterized by incoming workload characteristics. These workload characteristics are then utilized in LPV control design in a way such that the synthesized controller can adapt to workload changes and control design performance can be improved.

1.3 Literature review

Control theory provides a promising foundation for modeling, analysis and design for feedback based performance management of large and complex computer systems. There has been increasing research effort in applying control-theoretic approaches to the performance management for information systems. Most applications though are in the area of communication network systems and congestion control [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. This dissertation considers the performance management for server systems (Web, E-mail, and storage servers in the context of data center, etc), where response time is the primary performance metric, which introduces nonlinearity and is much more difficult to model and control. Compared to communication networks, a significant difference lies in the modeling of request

service demand. Requests in a network system are just fixed-size packets (all packets have the same size); the service time of a request at a link (leaving a router) corresponds to the transmission time of the packet. Thus a request's service time is more or less constant (depending on the link bandwidth); an exponential service time distribution would be good enough for modeling communication networks. In contrast, service demand in a Web/storage server system is highly varied and poses serious modeling challenges. In the following we summarize the existing literature on control theoretic approaches to server systems.

1.3.1 Application

Existing literature has applied control-theoretic approaches to applications such as Web servers [11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21], E-mail servers [15], and Lotus Notes [22, 23]; storage systems in the aspects of data migration scheduling [24], performance isolation [25, 26], and I/O throughput regulation [27]; dynamic voltage scheduling (DVS) in energy conscious devices [28, 29]; load balancing in distributed computer systems [30]; and real-time CPU scheduling [31, 32, 33, 34].

1.3.2 Modeling

In order to solve the performance management problem by control theoretic approach, a dynamic model that describes the relation between control variables (e.g., resource allocated to each application) and performance metrics (e.g. response time, utilization ratio, etc.) is necessary. In general, the dynamic models can be obtained either through first-principles approaches or system identification algorithms using empirical data.

System identification: The majority of existing literatures for performance control of server systems adopts linear-time-invariant (LTI) models obtained by system-identification. Reference 17 models the dynamic relation between administrator tuning parameters such as the maximum number of users, memory usage, as well as CPU usage of an Apache Web server using both single-input-single-output (SISO) and multiple-input-multiple-output (MIMO) approaches. Standard Matlab identification toolbox is used to estimate the parameters offline in [12], where sinusoidal inputs are used as excitation signals in their identification experiments.

Similar LTI models have been derived using Matlab system identification toolbox in [28, 25, 14, 22, 23]. In these papers a server is treated as a black box and can be easy to implement. However designing proper experiments to obtain training data for identification would appreciate detail domain knowledge in order for the identified models to work well. Further, they are linear time-invariant models, no information on variations of workload and operating conditions have ever been taken into account in the system modeling.

In order to adapt to load changes recursive least squares algorithms have been applied in adaptive control design methods to build system models, which can update the models with online measurements. In this category, [35, 36, 37] adopts an ARMA difference equation to describe the input variable (caching buffer size) and system output (average time delay requests experienced in a sampling interval) relation of an Apache Web server. A first-order ARX model is used to approximate the transient behavior of a memory pool between allocated resource and measured response time every sampling interval in [30]. An advantage of these methods is that they do not require detailed a priori knowledge of the system. However they still need an initial kick-off model of the system for the algorithm to evolve. A "proper" initial model is usually necessary for the online identification algorithm to converge. In addition there are many parameters need to be configured properly to guarantee stability and performance, such as when the model should be updated, how to weight the contributions of the current as well as previously identified models, and the criteria to discard certain updated models, etc. Indeed implementation issues need to be taken care of when applying these methods.

First-principles: [13] proposed a first-principles approach to constructing transfer functions for admission control in a generic M/M/1/K queuing system, where it is linearised and approximated by a first order I/O model. A transfer function is constructed and parameterized using workload characteristics work load intensity. [38] models a Web server as a nonlinear system to capture the transient behavior of the application workloads. The parameters of the model are updated using prediction by time series analysis techniques.

1.3.3 Control design

The most applied control design technique in the surveyed literatures is PID control. PID control designs are adopted in CPU scheduling [28, 29], QoS management for email servers [36, 14, 15], groupware [22, 23], and storage systems [25, 24], etc. A fuzzy-logic control is used to optimize performance of an Apache Web server in [39], which is essentially an automatic parameter tuning using a fuzzy controller that employs rules incorporating qualitative knowledge of the effect of tuning parameters. In [26] the authors argue that it is impractical to devise off-line models of enterprise-scale storage systems, therefore based on a recursive least-squares model, an adaptive controller was presented. [40] designs a model predictive controller for a CPU utilization regulation problem. [14] proposes the combination of a PID control with a queueing model based feed-forward compensation for the control of Web servers, where a queueing theory model is used to provision resources that provides average delay guarantee while the PID overcomes uncertainties and disturbances.

1.4 Contributions

This disertation investigates the performance management of computer server systems in the context of Internet hosting centers. Compared to existing literatures this thesis has the following contributions to the problem.

Modeling: Though modeling the performance management as a linear dynamical system (which captures system transient behavior) has shown certain competency compared to the queueing-based steady-state analysis, it is widely aware of that linear models may not be adequate when there are large variations in load conditions. Consequently, more sophisticated nonlinear system modeling is needed to improve system performance and robustness with respect to large variations of load conditions. A nonlinear model of hosting center server systems is derived based on first-principles analysis of transient queueing dynamics, which is linearized and approximated by a linear parameter varying (LPV) model. Empirical counterpart models are obtained via system identification. In both modeling approaches workload characterizing parameters are utilized as scheduling variables. Using real

Web traces it is shown that the LPV model over-performs linear models. The first-principles models also make offline training not necessary.

Control design: Traditional performance management for server systems relies on worst-case estimates of load and resource availability thus provisions resources to meet peak demands. Since the worst-case resource demand is likely to be significantly higher than its normal usage, these methods could be economically unfavorable. This work focuses on the development of an LPV control design for the performance management of server where we explicitly consider time-varying effects of the system by utilizing workload arrival and service parameters as scheduling variables. It is widely accepted that Internet services are under self-similar, bursty, and stochastic workload. LPV control based on deterministic characterization of these workloads may not work very well. We model Web server workloads using α-stable distributions and characterize workload characteristics using stochastic envelopes. LPV control design based on stochastic envelopes provide more modeling flexibility and can tolerate larger workload variance. A probabilistic method is introduced to further reduce control design conservativeness by considering nonlinear (bilinear) scheduling parameter dependence.

Evaluation: An intensive comparative study is conducted in this dissertation for the proposed modeling and control design techniques using real Web workloads. Queueing theory based provisioning results are used as the benchmark to evaluate the proposed control theoretic approaches. Simulation results show that the proposed LPV based modeling and control design outperforms both linear control and conventional queueing-theory based designs. The presented pros and cons evaluation provides guidelines on selecting the right approach for Internet service performance control. Results from this dissertation can be generalized in a straight-forward way to accommodate more complicated models in characterizing workloads and server environments to enhance system performance.

1.5 Organization

The organization of this thesis is as follows. In Chapter 2, we present the description of a hosting center, workload characterization together with commonly used performance specifications. Traditional queueing theory results will also be

reviewed in this chapter. The proposed modeling and control design methodologies are illustrated by two applications: admission control and CPU resource management for web applications housed on a hosting center, which will be addressed in Chapter 3 and Chapter 4 respectively. In these two chapters, results on deterministic modeling and robust control designs are presented. Chapter 5 presents a combined solution of the LPV modeling & control framework with workload characterization using α-stable-model based stochastic envelopes, which parameterizes a control-oriented dynamic-system model and resulting controller using workload-distribution parameters. Chapter 6 takes a step further by modeling Internet services as stochastic LPV systems and applies probabilistic robust control design to the same performance management problem. Finally a conclusion of the dissertation and suggested future work are given in Chapter 8. Some preliminary ideas and results are presented, along this direction further research is needed.

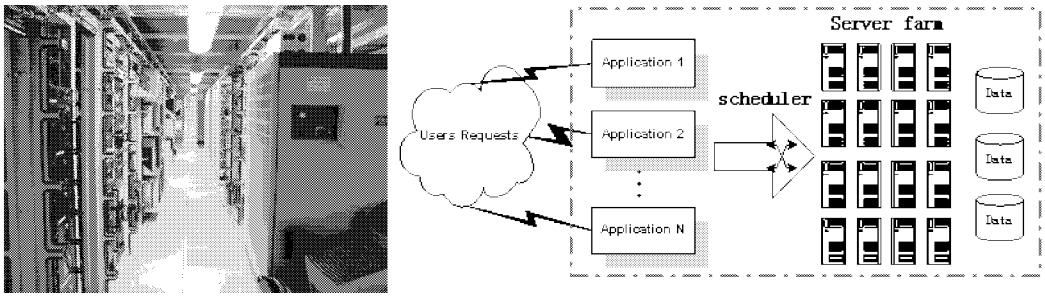
Chapter 2

Background and Foundation

In this chapter, we introduce the problem of performance management of server systems in the context of hosting centers. We first give an overall description of a hosting center, together with a review of workload characterization and commonly used performance specifications. We briefly introduce how to formulate the performance management of a hosting center as a feedback control problem, what are the possible control input, system output, and cost functions. Then we formularize the performance control as an optimal control problem. Finally we briefly review fundamentals of the queueing theory and related modeling results.

2.1 Overview

Generally there are multiple applications hosted in the center (Figure 2.1(a)). One application may span across a cluster of servers, while any server may be dedicated to one application only at any time. It is also possible that multiple small applications collocate on a single server. Each application provides a set of functionalities and services. Users of these applications are usually physically away from these servers. Remote users of these applications access the hosting center via communication networks, e.g. the Internet. Service requests initiated from remote users arrive at the gateway, where they are directed to requested services (applications).



(a) A production hosting center (b) Work flow schematics

Figure 2.1. Overview of hosting data centers

2.1.1 The work flow in a hosting center

The incoming request may demand resources such as CPU, memory, and I/O bandwidth. First for each incoming request, the hosting center may decide to admit or reject it, which is often referred to as admission control. Admission control is usually used to throttle the incoming traffic such that enough service for the admitted requests can be provided. After the requests for each application are admitted into the system, the hosting center needs to determine how much resource has to be allocated for each application dynamically in order to achieve the SLA, for example, how many servers need to be turn on/off, and allocated to each application, what is the corresponding CPU speed for each server? How much memory, I/O will be assigned to each hosted application? The function blocks of admission control and resource management are illustrated in Figure 2.1(b).

By performance management of a hosting center, we aim to maximize its operating profit in this thesis. So we need to consider both revenue and cost. The objective is to make an optimal balance between the two sides, subject to SLA requirements. In admission control, when a request arrives, associated with a particular application and its SLA, the controller decides whether to admit the request or not. For new incoming requests, if they are predicted to be unable to meet SLA target, or eventually causes already admitted requests to fail to meet their SLAs, they should not be admitted. The primary objective is to maximize requests admission subject to SLA constraints. In the resource management scenario, allocating more servers and/or running them at higher speed will definitely result in lower response time and more satisfied customers, and will thus generate

more revenue. However the operating cost also increases, due to the increase of items such as electricity bill, equipment purchases, etc.

2.1.2 A queueing system and related notations

A hosting center consists of clusters of servers. A server resembles a queueing system in that it receives requests, takes some time to serve each of them, and release them upon completion of service (Figure 2.2). Generally the following aspects together describe a queueing system,

Arrival process: customers' requests arrive one at a time and the times between two successive arrivals, which is often referred to as inter-arrival time, are non-negative random variables. The process that counts the number of arrivals up to time t is called a *renewal process*, and such a process is described by the common inter arrival time distribution. Some widely studied distributions are Exponential (M), Erlang (E), General (G), etc.

- Inter-arrival time A: the arrival time difference of any two consecutive requests of the workload as a sequence.
- Arrival rate λ: defined as the number of arriving request by the observation time.

Service demand: each arrival request claims a service that corresponds to certain service time demand. Some widely studied distributions for service demand are Exponential (M), General (G), etc.

- Service demand s: an attribute associated with each request indicating the amount of requested service, e.g. Read/Write access of a file, CPU time, etc.
- Service rate μ: average service rate as the number of served requests by the observation time. Service rate is related to service demand by the system's processing capability.

Buffer size: the maximum number of requests that can be in the system at any time, which sets the upper limit of queue length. If the buffer size is K, an arriving request that sees $K + 1$ requests in the system is rejected.

Performance metrics: some commonly used variables as performance metrics are,

- Response time T: the average time spent in the queue of all served requests.
- Queue length q: number of requests in the queue including the one being served, based on a reasonable number of observations. For a system with buffer size K, obviously $q \leq K$.
- Utilization ratio ρ: defined as the ratio of λ and μ.

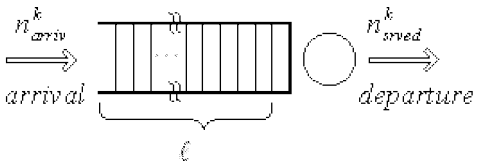


Figure 2.2. Schematics of a queueing system schematics

When a new request arrives, the server may be busy serving other requests, meaning it has to wait for its turn to be served. A scheduling policy that decides which, when, and how each request is served is necessary. Some widely used policies are: first-come-first-served (FCFS), last-in-first-out (LIFO), random, and processor-sharing (PS), where several requests can be served simultaneously, etc. For a given workload with certain inter-arrival pattern and request size distribution, the rate at which the server processes requests determines the number of requests in the queue, i.e. queue length.

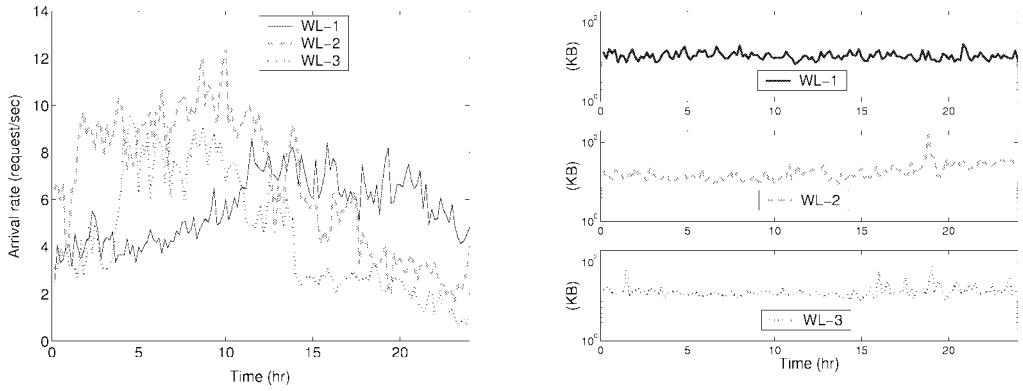
We shall follow the symbolic representation introduced by Kendall and represent the queueing system as Inter arrival time distribution / Service time distribution / Number of servers / Capacity / Service discipline. Thus for example $M/G/1$ represents a queueing system with Poisson arrivals, generally distributed service times, a single server, and infinite queue size, and a first-come-first-served queueing discipline.

2.2 Workload description

Workload consists of the sequence of all incoming customer requests. A workload is typically described by inter-arrival times between requests, service demand of each

request (which is referred to request size in certain applications); other workload characterizations could include seasonality, sequentiality/randomness in access, locality, etc. In this dissertation we will mainly be interested in inter-arrival times and request sizes, however, other workload characterization parameters could be incorporated without much difficulty.

In the queueing theory community probability distribution functions are used to approximate inter-arrival times and request sizes. The simplest workload has exponential distributions for both inter-arrival time and request size. Workloads with generally distributed arrival and/or request size are much more complicated. Unfortunately the majority of real word workloads fall into this category.



(a) Arrival rate (b) Service demand

Figure 2.3. Some example Web server workloads

A server hosting academic applications may have totally different workload characteristics from a server hosting a news website. Figure 2.3 shows the arrival rate (the reciprocal of inter arrival time) and service time (file size) of the three real Web server traces (WL-1/2/3) studied in this research during a 24 hours period, with 10-minute interval. Usually request sizes have much larger variance than arrival rates for these traces. Figure 2.4 plots the cumulative distribution function (CDF) of request file sizes of WL-2 at request level. It shows that the maximum request size is around $5000KB$, which is about 500 times of its average value. The 95-percentile value, which is larger than 95% of all the request sizes, is ten times more than the mean value. The heavy-tail of distribution indicates large variance. One can also do the same for the inter-arrival times distribution. Further, the large workload variance justifies the dynamic performance management proposed

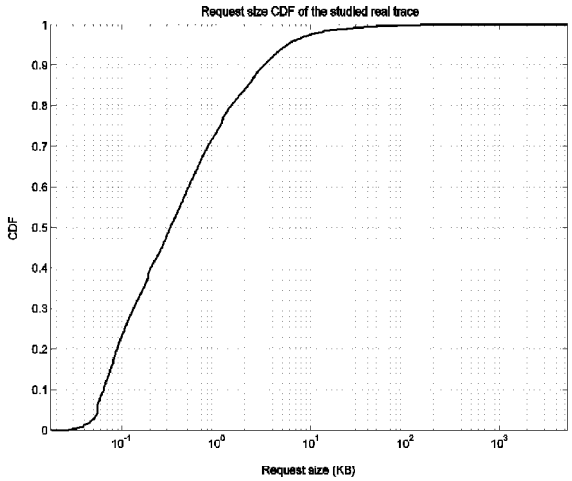


Figure 2.4. Cumulative distribution for request service demand

in this work. Designs based on the peak workload will be very conservative and economically unattractive if it is unable to dynamically adapt to changing workload condition.

2.3 Performance specifications

Hosting centers sell their services with SLAs to application owners. So defining the performance specifications is important since the SLAs are stated in terms of these specifications. There are different measures of performance specifications, depending on the particular application. In the context of hosting center, these specifications are generally related to measures that quantitize user experiences of the service provided by the hosting center, such as response time, throughput, and utilization, etc.

In general, the performance specification could be request-level or window-based. For example, in some real time scheduling applications the objective is to achieve a hard constraint on response time to guarantee certain worst case performance bound. In these applications request level specifications are considered. We use window-based performance specifications in this thesis in order to be compatible with the adopted control theoretic techniques. Our discussions on system modeling, controller design, and feedback loop implementation require a clear definition of sampling interval. It is true however that one can consider request level

response time in the discrete event control framework, which is out of the scope of this work.

2.4 The performance control problem

We consider a hosting center that operates multiple servers to support multiple applications (Figure 2.1(b)). The performance control problem is to provide performance guarantees for these applications to meet their SLA specifications by properly admitting incoming requests and/or allocating server resources. Achieving this brings revenue to the hosting center. On the other hand, operating cost is to be minimized such that the profit is maximized.

2.4.1 Control mechanisms

Performance control can be enabled via traffic shaping and/or resource management, or possibly some other mechanisms that are not the focus in this dissertation. In traffic shaping the decision is in the form of throttling server workload. In this research we focus on admission control as an example of traffic shaping where certain requests are adaptively rejected in order to meet SLA requirement of admitted requests, for example, rejection ratio is a possible choice of decision variable and can serve as the control action. Request rejection can be implemented in the scheduling policy. In a resource management framework, it needs to determine is in terms of the amount of resource the hosting center would provide to serve the incoming workload. The more resource being allocated the lower response can be achieved, given the workload characteristics. Different types of resources can be manipulated and allocated, such as CPU speed, network bandwidth, etc.

2.4.2 Feedback control elements

In order to formulate a feedback control loop, we need to identify and define the system elements such as control input, system output, sensor, actuator, etc. We also need to establish a dynamic model describing the relationship from input to output.

- Control input $u(t)$ is often defined in terms of the system variables that can be controlled. For example, in resource allocation problems, we can define the number of servers allocated and their speed; for admission control, we can define the ratio of incoming requests to be rejected.

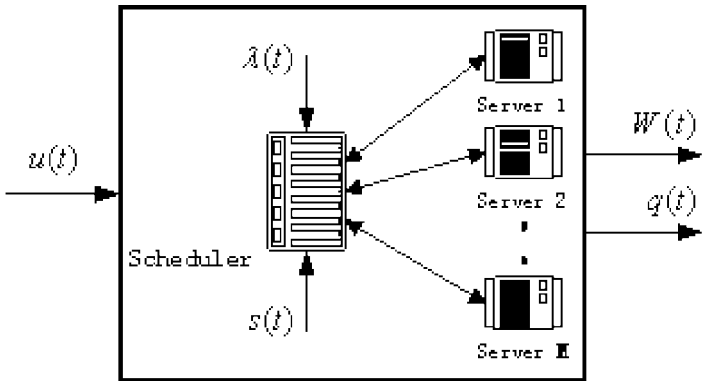
- System output variable $y(t)$ often consists of controlled performance measurement and possibly system states, such as response time $W(t)$, queue length $q(t)$, etc.

- The reference variable $\bar{T}(t)$ is usually SLA specifications, e.g., target response time.

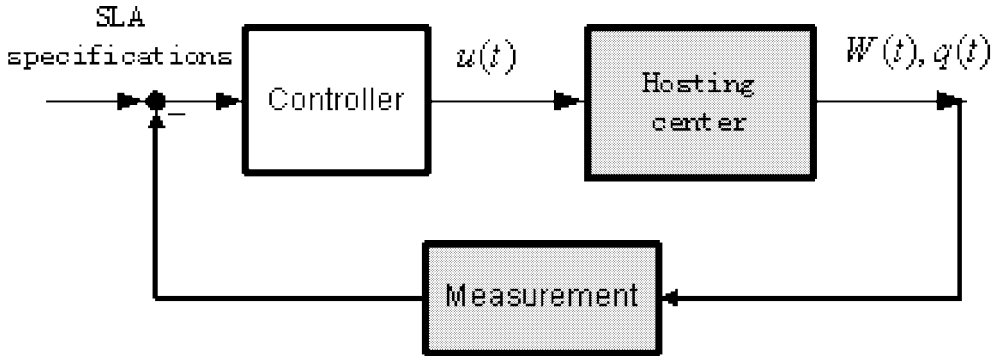
- Sensors that measure the concerned output performance metrics, and actuators that implement the control input $u(t)$ needed to be designed / implemented as well. In the context of computer system, an actuator may be a mechanism to adjust certain system parameters such as CPU frequency, stack size, storage volume, I/O rate, etc. A sensor usually means a functional algorithm that record/compute a set of interested information, such as the measurement of queue length, response time to customer requests.

The next step will naturally be building a control oriented dynamical model of the hosting center (a cluster of servers) before we can study the performance control problem. Workload characteristics, such as inter-arrival time and request sizes, are in general stochastic variables and are considered as system parameters. A hosting center is obviously a time varying system since inter-arrival times and request sizes change over time (Figure 2.5). Nevertheless one can always approximate the system by an LTI model considering the nominal inter-arrival time and request size.

As mentioned in Section 2.1 the main objective is to maximize operating profit in the performance management of hosting centers, which is usually to make a trade off between the cost side and the gain side, subject to SLA requirements. It can be achieved by formulating a properly defined optimization problem. Thus a well defined cost function is necessary which intuitively incorporates both control action (cost) and performance measures (gain). Several solutions are possible with a properly defined cost function. For example, we can approximate the system by



(a) Hosting center as an I/O model



(b) Feedback control block diagram

Figure 2.5. Hosting center from control system viewpoint

an LTI model and directly solve the optimization problem taking an LQ control design approach. We can also model the system as an LPV model and then apply H_∞ LPV control synthesis, where the cost function is optimized using loop shaping techniques. It is also possible to optimize the cost function by dynamic programming.

2.5 Queueing theory results review

Queueing theory will be used in developing the LPV models and implementing the performance benchmark. Queueing theory is a branch of applied probability theory. Historically the subject of queueing theory has been developed largely in the context of communication traffic engineering. Now it has been applied to numerous applications in telecommunications, manufacturing, transportation,

and information systems, etc. Simply put, queueing theory establishes steady state relations between arrival request, service rate, and output performances. Variables such as arrival rate, service rate, queue length, response time are of particular interest. A briefly summarizes commonly used queueing theory results.

Chapter 3

Admission control

This chapter presents a linear parameter varying (LPV) approach to the modeling and control of server systems to provide QoS guarantees. In particular we focus on the admission control mechanism to provide performance control. Workload intensity, which is the ratio of request arrival rate and service rate, is chosen as the scheduling parameter in the LPV formulation. Existing tools on LPV system identification and control synthesis are applied to this example and results are compared with that of linear designs. An advantage of the proposed method is that it does not require a priori knowledge of the workload as long as it is on-line measurable. In addition, the utilizing of information on load conditions and resource availability would allow better performance in QoS guarantees compared to linear control techniques.

3.1 Admission control and problem formulation

Admission control regulates customer requests in an effort to provide adequate resources to those that are already admitted so that their SLAs are met. It is an effective performance control mechanism especially when the server is under heavy workload, in which case dynamic resource allocation alone may not be sufficient since there is not too much resource left. In this case admitting a new request may result in deteriorated performance of other requests, leading to unsatisfactory customer requests. As a result the server will generate less profit. What admission control does is to reject certain customer requests so that the admitted requests

are delivered good performance. There are different mechanisms to implement the rejection while we adopt a fairly straightforward one: randomly reject a percentage of incoming requests during a time interval. For example, when a rejection ratio $u(k)$ in the k^{th} sampling period is determined, a request comes in this sampling period is denied to enter the system with probability $u(k)$ (or is admitted to the system with probability $1 - u(k)$). Thus the original arrival rate $\lambda(k)$ becomes $\lambda_e(k)$, the effective arrival rate, as shown in Figure 3.1.

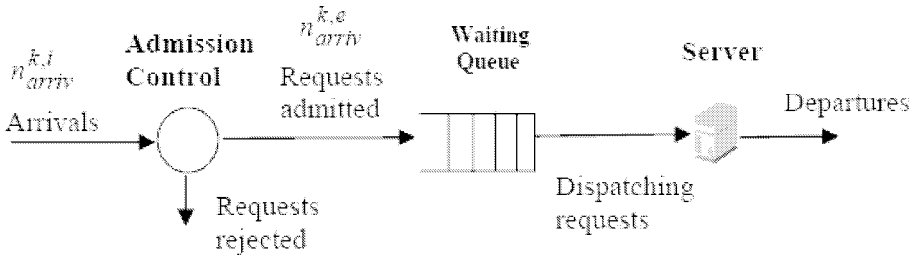


Figure 3.1. Admission control for a single queue

We define rejection ratio as control input and response time as system output. For a fixed setting of rejection ratio, the performance output, e.g. response time, depends on the incoming workload. We first derive a dynamic model describing the relation between rejection ratio and system performance, then we design a controller to dynamically set the rejection ratio so that the performance measure is within desired range.

Suppose that the server itself (the hardware) does not change over time, the performance output depends on incoming workload. So the controller should set the rejection ratio based on the workload. In order for the controller to adapt to dynamically varying load conditions, it would be desirable if the controller gain changes with respect to the changing workload so that the controller is more responsive to performance violations. This motivates the formulation of an LPV control system, where the workload parameters are used as scheduling variables.

Since the traffic load varies in a slower time scale (usually in minutes for a web server application) compared to the system dynamics (where the response time of a web server is usually in seconds), the LPV system is slow varying, which satisfies the conditions for a general LPV control synthesis.

3.2 Modeling for admission control

In this section, we briefly review how linear time invariant as well as LPV system identification algorithms can be applied to build the dynamic relation from input (rejection ratio) to output (performance metrics variable) of admission control for a hosting server, i.e. estimating the afore-mentioned coefficients based on experimental training data. For the LPV model, time-varying workload characterizing parameters are specified as scheduling variables.

3.2.1 Linear models

The input-output dynamic relation from rejection ratio $u(k)$ to system response time $T(k)$ is essentially nonlinear. A simplistic modeling solution is to construct a linear time-invariant empirical model using system identification techniques, which can be interpreted as the linearization of the original nonlinear dynamic system at a nominal operating condition. We consider an ARX model as follows,

$$A(q)T(k) = B(q)u(k) + e(k) \tag{3.1}$$

with

$$A(q) = 1 + a_1 q^{-1} + \cdots + a_{na} q^{-na} \tag{3.2}$$

$$B(q) = b_0 q^{-1} + \cdots + b_{nb} q^{-nb} \tag{3.3}$$

where q is the delay operator, na and nb determine the system order.

The constant coefficients a_i and b_i are computed through running system identification algorithms (e.g., standard tools in Matlab) on properly designed experimental data $(u(k), T(k))$. We denote this model as *LinearModel-ARX*.

3.2.2 LPV models

In order for the system to adapt to dynamically varying load conditions, we formulate a linear parameter varying system by defining workload parameters as scheduling variables. Assuming that coefficients a_i and b_i in Eqs. 3.1-3.3 are functions of load conditions, we specify the following LPV-ARX model,

$$A(q,r)T(k) = B(q,r)u(k) + e(k) \tag{3.4}$$

with

$$A(q,r) = 1 + a_1(r(k-1))q^{-1} + \cdots + a_{na}(r(k-na))q^{-na} \tag{3.5}$$

$$B(q,r) = b_0(r(k)) + b_1(r(k-1))q^{-1} + \cdots + b_{nb}(r(k-nb))q^{-nb} \tag{3.6}$$

where in general $r(k)$ represents the vector of workload parameters; it is also the scheduling variable for the LPV-ARX system.

The function relation of coefficients $a_i(r), i = 1, \cdots, na$ and $b_j(r), j = 1, \cdots, nb$ in terms of the load parameter r could be nonlinear in general. We can start by assuming that the plant has a Linear Fractional Transformation (LFT) dependence on the scheduling variable r. Alternatively, we assume that $r(k)$ enters Eqs. 3.4-3.6 in a polynomial manner, i.e., $a_i(r), i = 1, \cdots, na$ and $b_j(r), j = 1, \cdots, nb$ are polynomials in r of degree $N-1$,

$$a_i(r) = a_i^1 + a_i^2 r + \cdots + a_i^N r^{N-1} \tag{3.7}$$

$$a_j(r) = b_j^1 + b_j^2 r + \cdots + b_j^N r^{N-1} \tag{3.8}$$

Depending on the system under consideration, the vector r could include different parameters that characterize workload behavior, e.g., arrival rate, file size, locality, read/write ratio, etc. As a starting point, we use a scalar scheduling parameter $r(k)$ defined as the reciprocal of workload intensity,

$$r(k) = \frac{\lambda(k)}{\mu(k)} \tag{3.9}$$

which makes the implementation of the LPV system identification easier and more efficient. Here workload intensity is used slightly different from its convectional definition, which is the ratio of the arrival rate λ and service rate μ.

A naive approach for estimating the coefficients $a_i(r), i = 1, \cdots, na$ and $b_j(r)$, $j = 1, \cdots, nb$ can be conducted as follows: 1) identify a set of linear time-invariant ARX models as Eqs. 3.1-3.3 corresponding to a sequence of values of workload parameter r, 2) then derive $a_i(r), i = 1, \cdots, na$ and $b_j(r), j = 1, \cdots, nb$ by interpolating corresponding coefficients of the set of linear time-invariant ARX models.

We apply a Least-Mean-Squares based algorithm from [41]to directly identify the LPV system Eqs. 3.4-3.9, where polynomial dependence on the scheduling parameter r is assumed.

Define an $n \times N$ matrix Θ containing all the coefficients to be identified and define the extended regression operator Ψ containing the input/output data and the parameter trajectories,

$$\Theta = \begin{bmatrix} a_1^1 & \cdots & a_1^N \\ \cdots & \cdots & \cdots \\ a_{na}^1 & \cdots & a_{na}^N \\ b_0^1 & \cdots & b_0^N \\ \cdots & \cdots & \cdots \\ b_{nb}^1 & \cdots & b_{nb}^N \end{bmatrix}, \Psi_k = \begin{bmatrix} -T(k-1) \\ \cdots \\ -T(k-n_a) \\ \theta(k) \\ \cdots \\ \theta(k-n_b) \end{bmatrix} \begin{bmatrix} 1 r(k) \cdots r^{N-1}(k) \end{bmatrix} \tag{3.10}$$

Least-Mean-Squares Algorithm:

1) Initialize the estimated $\hat{\Theta}_0$,

2) $\varepsilon_k \leftarrow y(k) - trace(\hat{\Theta}_k^T \Psi_k)$

3) $\hat{\Theta}_{k+1} \leftarrow \hat{\Theta}_k + \alpha \varepsilon_k \Psi_k$

Figure 3.2. An LMS algorithm for identification of a polynomial parameter-dependent LPV system

The Least-Mean-Squares algorithm in Figure 3.2 is used to compute the estimate $\hat{\Theta}$ iteratively. This algorithm does not require the scheduling variable to be slow varying, but requires the persistence of excitation for the inputs and scheduling parameters. For different LPV system identification algorithms, refer to [41, 42, 43]for LFT dependence on scheduling variables and in [44]for nonlinear parameter dependence (where a neural network is used).

3.3 control synthesis

Corresponding to the different types of models developed, we may design either an LTI or an LPV controller for the performance control.

3.3.1 Linear control design

Corresponding to the linear ARX model (Eq. 3.1), we formulate a Linear Quadratic (LQ) control problem, which optimizes the cost function as follows:

$$J = \sum_{k=1}^{\infty} \left(r_T \cdot \hat{T}^2(k) + r_u \cdot \hat{u}^2(k) \right) \tag{3.11}$$

where r_T and r_u are penalty weighting matrices on off-target response time and deviation from nominal rejection ratio. Intuitively we would like the actual performance output to track its SLA, this justifies for the first component. On the control action side, we want to reject as few requests as possible, which is an absolute value instead of a deviation amount. Thus for the second part however it is a little different from what we actually want.

3.3.2 $LPV - H_\infty$ control synthesis

Corresponding to the LPV-ARX model (Eq. 3.4, we formulate an $LPV - H_\infty$ control synthesis as illustrated by Figure 3.3.

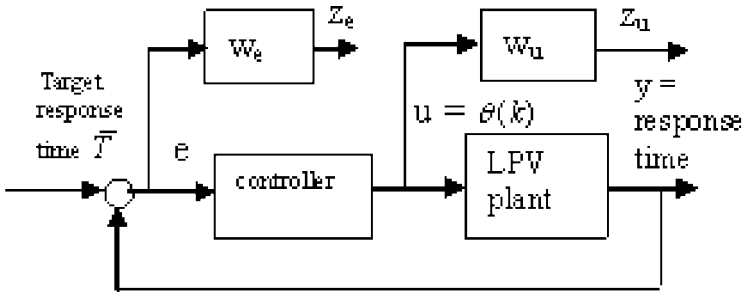


Figure 3.3. Robust control system interconnection

In order to apply the above $LPV - H_\infty$ control synthesis, low-pass filters are appended to input and output channels of the original LPV plant. The cutoff bandwidth of the low-pass filters is chosen to be much higher than the feedback sampling frequency so that the system performance would not be affected. With a bit abuse of notation, we let $P(r)$ denote the LPV model that includes the original plant as well as the input/output low-pass filters.

The performance specifications on minimizing tracking error of meeting target response time and reducing control action are addressed through the design of

weighting functions W_e and W_u, respectively. Define an augmented plant $P_{aug}(r)$ that includes the actual LPV system $P(r)$ to be controlled as well as auxiliary weighting functions W_e and W_u representing closed-loop performance criteria (Figure 3.3). The LPV control can be classified as a generalized gain-scheduling control. It designs a parameter-dependent controller $K(r)$ to stabilize an augmented LPV plant $P_{aug}(r)$ for all admissible parameter trajectories r, minimizing the effect of the exogenous inputs on the controlled variables in certain norm. In $LPV - H_\infty$ control, the controller $K(r)$ is designed such that the closed-loop system is stabilized and the H_∞ norm of the transfer function T_{zw} from the exogenous input w (the reference response time $\bar{T}$) to the controlled variable z (the weighted error signal $\tilde{e}$ and the weighted control signal $\tilde{u}$) is minimized, i.e.,

$$\|T_{zw}\| \leq \gamma \tag{3.12}$$

with performance level γ.

In general, for an affine parameter-dependent plant $P(r)$, the design of an affine parameter-dependent controller $K(r)$ is often reduced to solving a set of parameter-dependent Linear Matrix Inequalities (LMIs) [45]. Consider the affine parameter-dependent LPV system,

$$\sigma x = A(\delta)x + B(\delta)uy = C(\delta)x + D(\delta)u \tag{3.13}$$

where $\sigma := z$ is for discrete time. The LPV system has quadratic H_∞ performance γ if and only if there exists a single matrix $X > 0$ such that $B(X, \gamma) < 0$ for all admissible values of the parameter vector γ, where

$$B_{A(\delta),B(\delta),C(\delta),D(\delta)}(X, \gamma) := \begin{bmatrix} -X^{-1} & A & B & 0 \\ A^T & X & 0 & C^T \\ B^T & 0 & -\gamma I & D^T \\ 0 & C & D & -\gamma I \end{bmatrix} \tag{3.14}$$

Then the Lyapunov function $V(x) = x^T X x$ establishes the global asymptotic stability. For an LPV system with polynomial parameter dependence, a sum-of-squares based approach was presented by [46]for control synthesis.

By utilizing time-varying load parameters as scheduling variables, the LPV

control synthesis is expected to improve control performance with efficient control usage. Furthermore, the LPV design does not require a priori knowledge of load conditions as long as they are on-line measurable.

3.4 Simulation results

The proposed modeling and control design results are implemented and evaluated by a Web server simulator [47]. System identification is based on a set of synthetic workload running on computer simulation. Then we compare LPV synthesis results and the LQ results. Some common issues in modeling and control synthesis are discussed.

3.4.1 Model identification and validation

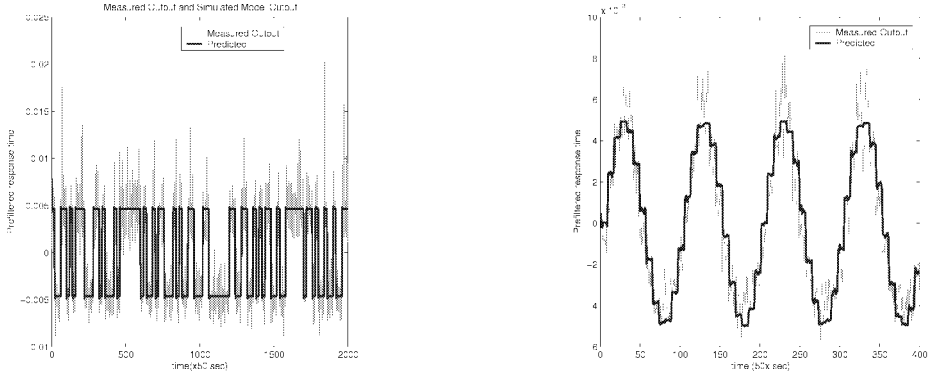
We assume that in a sampling period, the request arrivals follow an exponential distribution with mean rate $\lambda(k)$ (requests per second). After requests are admitted to the system, they are served in a first-come first-serve (FCFS) manner. We assume that service times are independent and the service rate follows an exponential distribution with mean rate $\mu(k)$ (requests per second). Size of the document requested is not included for simplicity in illustrating the proposed LPV approach. Note that although exponential distribution is used here the underlying approach does not preclude using any other distributions.

Before applying LPV system identification, we first examine Eq. 3.4 at a nominal load condition. It is intended to evaluate whether a linearized model at the nominal load condition is able to capture the major dynamics when the load variation is small. For r = 0.5 with mean service rate 100 requests/sec, we use a pseudo-random binary input for the rejection ratio $\theta(k)$; together with the resulting response time $T(k)$ we construct a second-order ARX model (data fitting does not get improved by increasing the order of the model),

$$T(k+2) = a_1 T(k+1) + a_2 T(k) + b\theta(k+1) + e(k+2) \tag{3.15}$$

with a1 = 0.01938, a2 = -0.01747, and b = -0.01027. Figure 3.4(a) shows the predicted versus the measured response time for a pseudo-random binary input

used in the system identification. Figure 3.4(b) shows the model validation on data obtained by using a multiple-step input for rejection ratio. From the results we can see that the linear approximation has captured the major system dynamics at a nominal load condition.



(a) Predicted vs measured data with pseudo-random binary rejection ratio

(b) Validation against a different set of data obtained using a multiple-step input as rejection ratio

Figure 3.4. Construct an ARX model at workload intensity r = 0.5 using system identification; the input/output data shown in the figure are prefiltered/detrended data.

The pseudo-random binary signal used to generate rejection ratio $\theta(k)$ and the random signal used to generate the scheduling parameter (workload intensity $r(k)$) in the LPV system identification are plotted in Figure 3.5(a) and Figure 3.5(b).

The resulting LPV model corresponding to Eq. 3.4 is

$$\begin{aligned} T(k+2) = \; & [0.3464 + 0.1313r(k+1)] * T(k+1) + [0.2527 + 0.1187r(k)] * T(k) \\ & + [-0.0007 + 0.0443r(k+1)]\theta(k+1) + e(k+2) \end{aligned} \tag{3.16}$$

Then we use a different set of rejection ratio input and workload intensity parameter trajectories (shown in Figure 3.5(c) and Figure 3.5(d)) to validate the identified model. Figure 3.6 plots the LPV predicted response time versus the measured response time in the model validation, which demonstrates the accuracy of the identified LPV model.

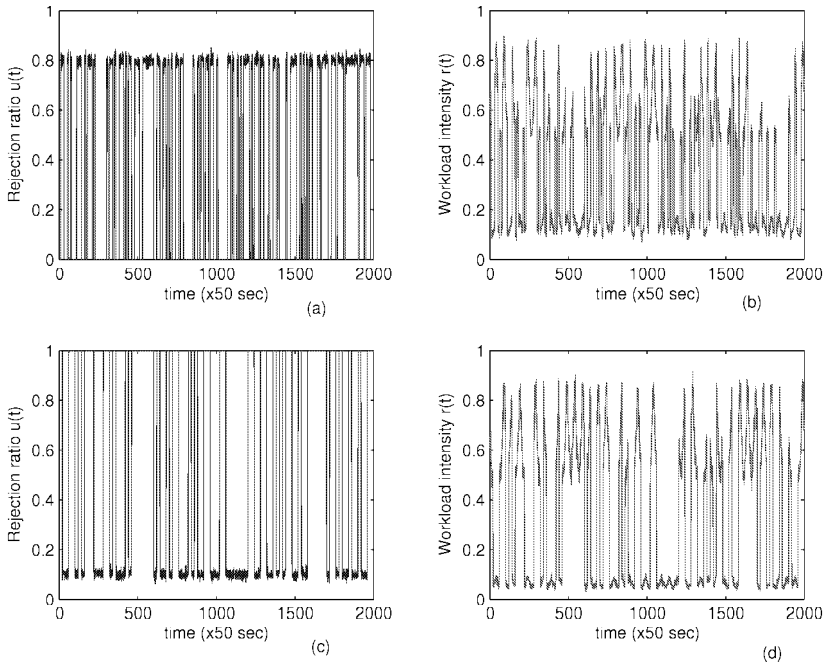


Figure 3.5. Input and scheduling parameter trajectories used in LPV system identification and model validation. (a) A pseudo-random binary signal used to generate rejection ratio in system identification; (b) A random signal used to generate workload intensity r(k) as scheduling parameter in system identification; (c) A pseudo-random binary signal used for rejection ratio in model validation; (d) A random signal used for workload intensity r(t) in model validation.

3.4.2 Control synthesis

The goal of the control synthesis is to achieve a target response time $\bar{T}$. The admission control has to balance between achieving the target response time and maintaining certain system throughput. Rejecting all requests would definitely put response time to zero, but the service provider would not make any money by serving requests either.

We specify the target response time $\bar{T}$ as 0.02 sec. We first design an LQ controller using the model (Eq. 3-15) that is identified at nominal workload intensity r = 0.5. In order to guarantee the target response time, the plant was augmented with an integrator at the control input. Figure 3-7 shows the performance of this LQ design operates at the design point (workload intensity r = 0.5) and at the off-design load condition (r = 0.8). It is noted that the LQ design is able to achieve

the 0.02 sec target response time at r = 0.5, but when the traffic load increases to r = 0.8, it does not meet the target response time.

Next we design a robust LPV controller based on the identified LPV model Eq. 3-16. The control synthesis block diagram is shown in Figure 3-3. The controlled variable includes tracking error and actuation effort, which are shaped by weighting functions We and Wu, respectively. A suitable set of weighting functions in s-domain is chosen as

$$W_e = \frac{0.1429s + 0.4}{s + 0.02}, W_u = \frac{0.1s^2 + s + 2.5}{0.2s^2 + 44.72s + 2500} \tag{3.17}$$

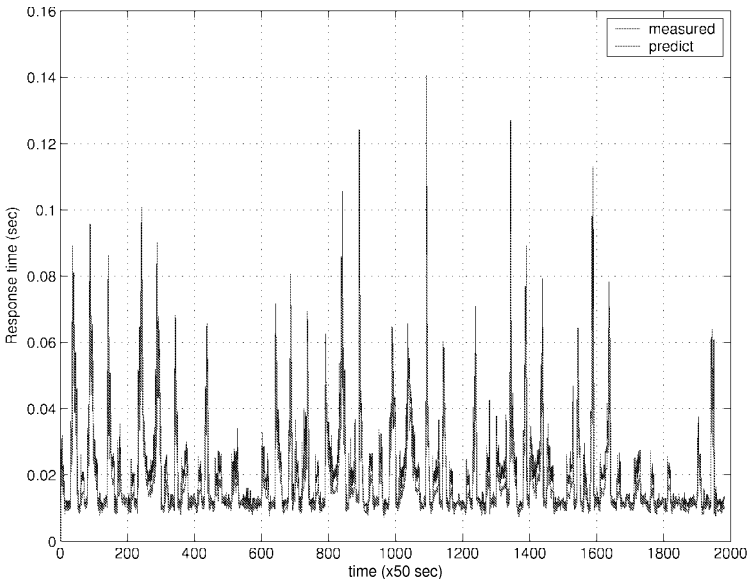
In Figure 3.8(a), the performance for the LPV controller to operate at workload intensity r = 0.8 is compared with that of the LQ controller running under the same load condition. The LPV controller is able to achieve the 0.02 sec target response time at the off-nominal load condition. Figure 3.8(b) compares the performance of the LPV controller against that of the LQ design for a time-varying load conditions. It is noted that the LQ design only provides response time guarantee for the nominal load or less intensive traffic; the response time increases dramatically for the heavy traffic. In comparison, the LPV design adapts to the change of workload intensity very well; it provides the response time guarantee despite of the dynamically changing load conditions.

3.5 Summary

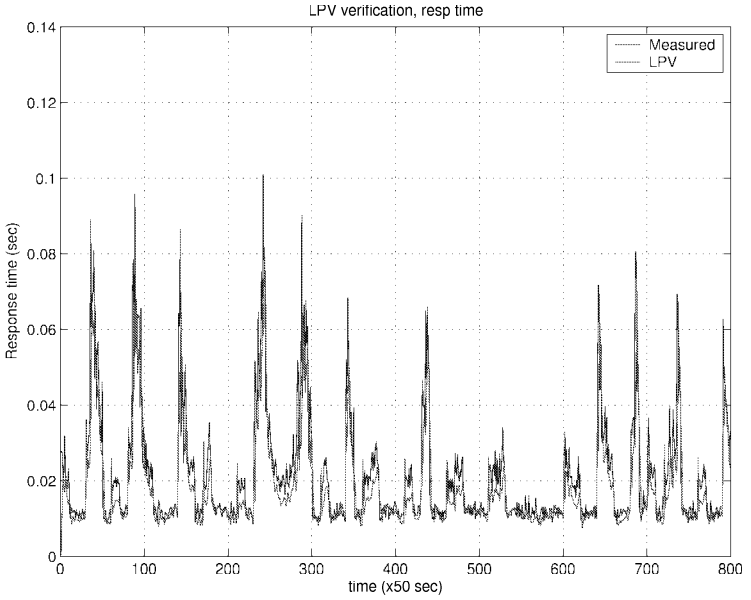
This chapter presents an LPV formulation for admission control. Workload parameters such as workload intensity are used as scheduling parameters in the LPV modeling and control, which allows system's fast adaptation to traffic changes. Identification and validation using synthetic workloads shows that the proposed LPV-ARX modeling out-performs LTI ARX models. LQ control design fails to meet response time requirement when operating condition changes, while the LPV controller can adapt to such changes and can still provide response time guarantees. Compared to queuing-theory based approaches that study steady-state behavior and require prediction of workload parameters, the LPV design does not require a priori knowledge of workload parameters provided they are on-line measurable.

Therefore, the proposed LPV modeling and control framework sets a promising direction for the performance control for today's hosting center server systems that operate in open environments with unpredicted load changes.

The results of this chapter are published in American Control Conference 2005 [48], and IFAC Journal of Control Engineering Practice, April 2007 [49].



(a) Predicted vs measured output for the input and scheduling parameter trajectories in Fig. 3.5(c) and (d)



(b) Zoom-in of (a)

Figure 3.6. Model validation on the identified LPV system model.

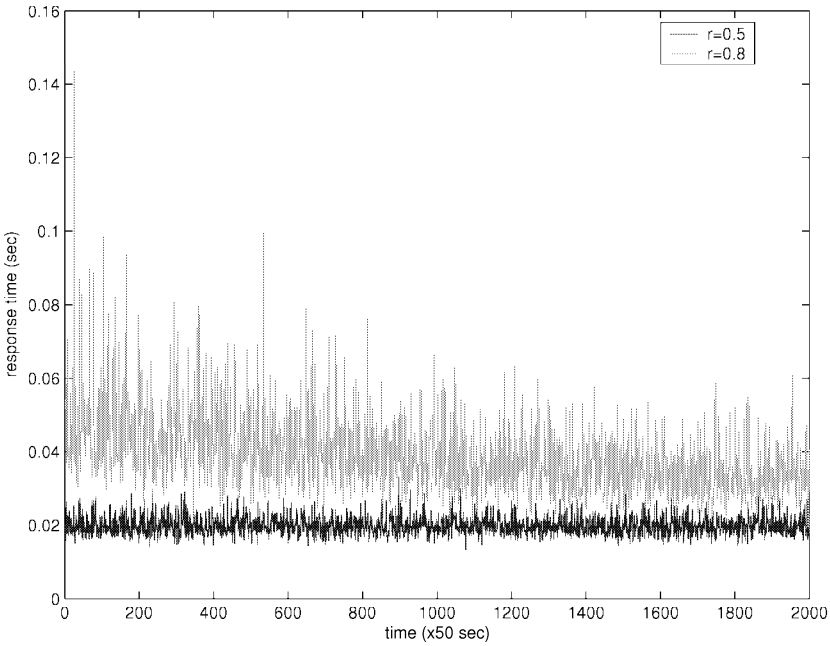
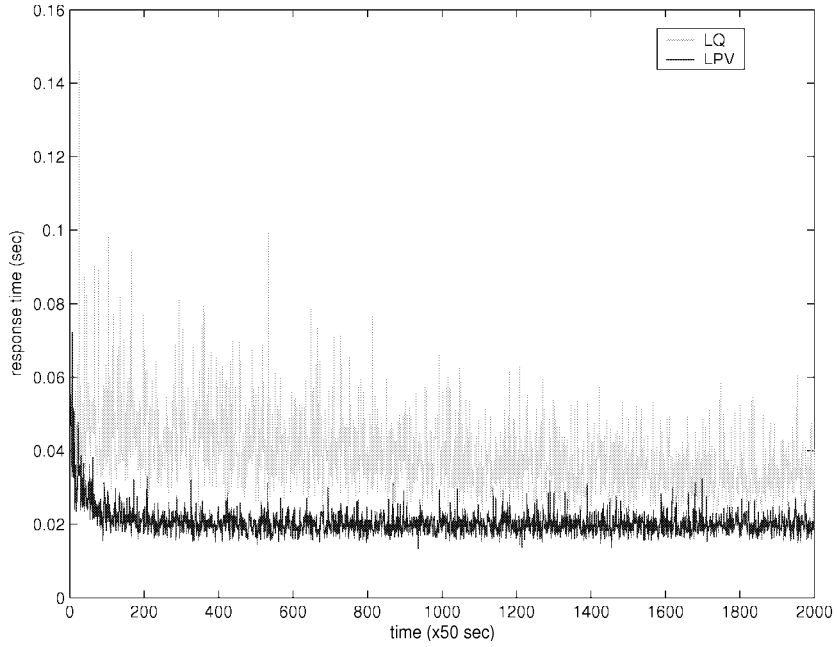
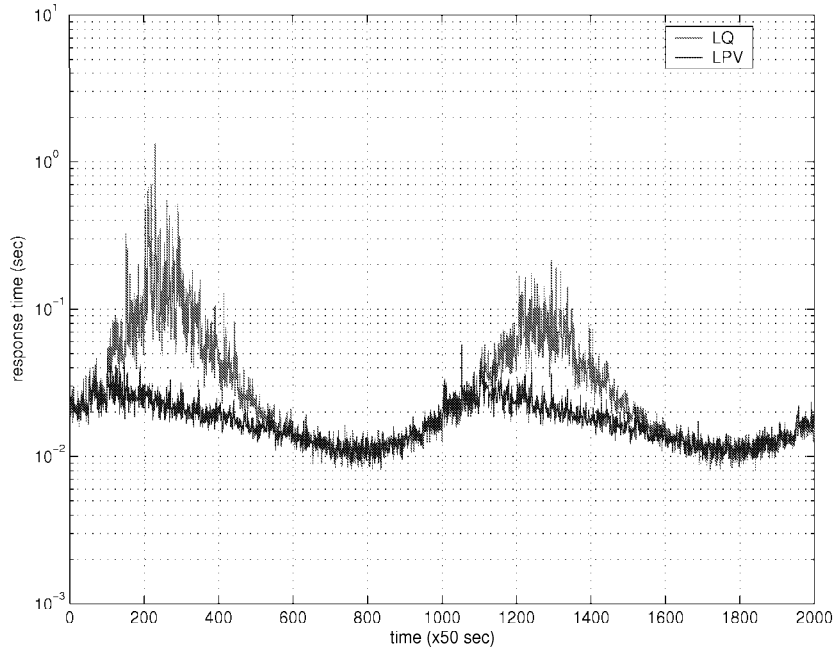


Figure 3.7. Simulation results for a LQ controller (designed based on the nominal model Eq. 3-15) to operate under the nominal load r = 0.5 and the off-design load r = 0.8.



(a) Both operate under the load r = 0.8



(b) Both operate for a workload with dynamically changing load conditions

Figure 3.8. Simulation results for an LQ controller versus an LPV controller.

Chapter 4

Resource management

In this chapter, a CPU frequency management problem is studied to provide response time guarantees with minimal energy cost. We have derived first-principles models based on analyzing transient queueing dynamics as well as empirical models using system identification algorithms. $LPV-H_\infty$ controllers are then designed for the derived LPV system models to meet performance SLA with minimal CPU usage. Using real Web server workloads, the performance of LPV control compares favorably to various linear control designs and a design based on conventional queueing theory.

4.1 The resource management problem

We consider a hosting center that operates M identical servers to support N different Web applications at any time. Each server is equally capable of running any application; it is devoted to a particular application while each application may span on multiple servers. As discussed in our coauthored paper [47], two mechanisms are available for power management of server clusters. One is to turn off the server when it is not in use, which will reduce power consumption but will take time and energy to reboot the machine; in addition, reboot will increase wear-and-tear of components (e.g., disks) and reduce their mean time between failure (MTBF). Another way to reduce power consumption is to use the dynamic voltage/frequency scaling (DVS) scheme since the power consumed in circuits is proportional to the cubic power of the operating clock frequency. In a blade server

system, CPU consumes a high percentage of the bulk power, e.g., an Intel Xeon consumes $\sim 75-100$ Watts at full speed, while the other blade components including disk only add about 15-30 Watts, thus DVS control becomes very important in power management [47].

In summary, the objective of the power management for a hosting center is to dynamically allocate an appropriate number of servers to each application and to control each server's CPU frequency so that the target response time of each application is met with minimal energy consumption. In [47], the power management is formulated to minimize the following operational cost (housing N applications over Z time units of duration Δt) while meeting target response time,

$$\begin{array}{l} \sum_{z=1}^{Z} \left(\sum_{i=1}^{N} K_{\mathrm{S}} \cdot n_i(z) \cdot (P_0 + P_f \cdot f_i^3(z)) \cdot \Delta t \right) \\ + \sum_{z=1}^{Z} \left(B \cdot \left(\sum_{i=1}^{N} n_i(z) - \sum n_i(z-1) \right)^{+} \right) \end{array} \tag{4.1}$$

The first term in Eq. 4.1 calculates the electricity cost of operating servers. The power consumed by one server node equals to the sum of the CPU power consumption, which is proportional to the cubic power of the server frequency f with coefficient P_f, and the power consumption of other components that does not scale with frequency. The energy consumption is then obtained by multiplying the power by time. For the i^{th} application, it is assumed that all servers allocated to this application run at the same frequency since this will consume less power than otherwise due to the dependence of power consumption on the cubic power of CPU frequency. Then total energy consumption for the i^{th} application is calculated by multiplying the energy for a single server with the number of servers allocated to this application. The electricity cost is obtained by further multiplying K_{S}, the dollars per unit of energy consumption (kilowatt hour (KWH)).

The second term in Eq. 4.1 computes the cost due to turning on new servers across successive time periods (in 4.1, $\{\cdot\}^{+} = max(\cdot, 0)$). The coefficient B denotes the dollar cost for a single server turn-on, which includes energy cost for bring up the machine (energy consumed during the machine boot time) and the dollar cost resulted from a smaller MTBF due to wear and tear in turning-on servers [47].

We decompose this performance/power management problem into two subproblems, as illustrated by Figure 4.1:

1. Dynamically determine a minimal aggregate CPU frequency for each application that can meet response time guarantee as though there is a single server per application running at this frequency, noting that the energy consumption is directly related to the frequency in Eq. 4.1.

2. Solve an online server allocation problem, which determines the number of servers allocated to each application and the CPU frequency of each individual server such that the aggregate frequency obtained from the first subproblem is provided. The objective for this subproblem is to seek the one with minimum energy consumption among different configuration choices: running more servers with a lower frequency or running less servers with a higher frequency.

In our previous work [47], the first subproblem was solved through a linear optimal control using a linear ARX model, which was derived by system identification algorithms; the second subproblem was solved through a greedy on-line algorithm, which is described in the next section. With server allocation the final energy consumption can be calculated. The possible approximation errors due to this 2-step decomposition have been addressed in [47]. In this chapter we aim to develop an LPV modeling and design for server performance management, thus we focus on the first subproblem - dynamical control of the aggregate CPU frequency to meet target response time with minimal CPU usage. The aggregate CPU frequency resulting from the LPV control will then be used to determine the number of servers and their operating CPU frequencies by applying the same greedy on-line algorithm as in [47](see Figure 4.1). In the sequel, we denote the average aggregate CPU frequency in the k^{th} time interval as $u(k)$ and refer it to be CPU in brief.

4.1.1 Online Server Allocation: a greedy algorithm

The decision on server allocation is updated every T time period, which is indexed by $z = 1, \cdots, Z$. In each T time period, the control decision on CPU frequency is updated every Δt time period, which is indexed by $p = 1, \cdots, P$, i.e., $T = P \cdot \Delta t$. Note that the server allocation time period T is not necessarily the same as the time period ΔT for updating operating condition in the LPV modeling and design.

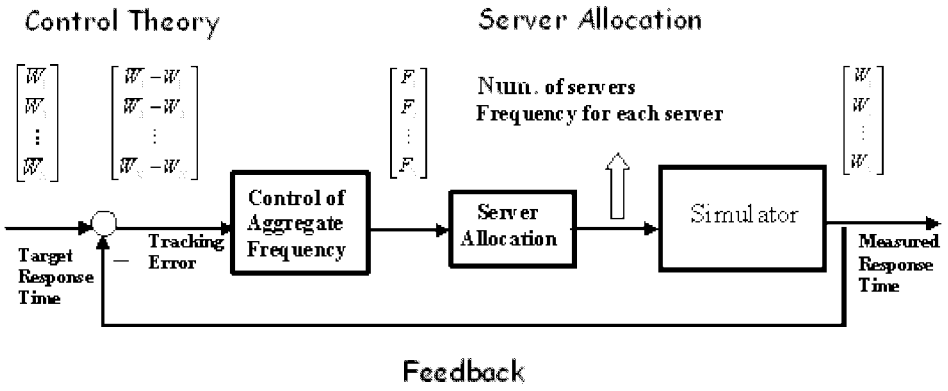


Figure 4.1. Feedback control loop for performance management

Let $u_i(z,p)$ denote the aggregate frequency allocated to the i^{th} application in the p^{th} Δt-time interval within the z^{th} T-time duration. The dynamic control of the aggregate frequency $u_i(z,p)$ at any time is obtained from the first subproblem thus $u_i(z,p)$ is assumed to be known for the second subproblem, where an online optimization is formulated to allocate servers to each application and to set the CPU frequency of each on-server with the goal of providing the aggregate CPU $u_i(z,p)$ with minimal energy cost.

At each time interval of T, define

$$n(z) = \sum_{i=1}^{N} n_i(z) \tag{4.2}$$

to denote the total number of servers being turned on for all applications, and define

$$u(z) = \sum_{i=1}^{N} (max_p(u_i(z,p))) \tag{4.3}$$

to denote the CPU capacity that the $n(z)$ servers should provide in the z^{th} T-time duration. We allocate the number of servers for the i^{th} application, $n_i(z)$, proportional to this application's aggregate frequency demand, i.e.,

$$n_i(z) = \left\lceil max_p(u_i(z,p)) \cdot \frac{n(z)}{u(z)} \right\rceil \tag{4.4}$$

where $\lceil \cdot \rceil$ denotes the ceiling value. As $f_i(z,p) = u_i(z,p)/n_i(z) \leq u(z)/n(z)$,

$i = 1, \cdots, N$, we replace Eq. 4.1 by the cost function as follows,

$$\sum_{z=1}^{Z} K_S \cdot n(z) \cdot T \cdot \left(P_0 + P_f \cdot (u(z)/n(z))^3\right) + \sum_{z=1}^{Z} B \cdot (n(z) - n(z-1))^+ \quad (4.5)$$

Since each server's frequency should be operated within (f_{min}, f_{max}), the total number of on-servers $n(z)$ should satisfy $\underline{n}(z) \leq n(z) \leq \bar{n}(z)$ with $\underline{n}(z) = \lceil u(z)/f_{max} \rceil$ and $\bar{n}(z) = min\left(\lfloor u(z)/f_{min} \rfloor, M\right)$, where $\lfloor \cdot \rfloor$ denotes the floor value.

Note that if we ignore the server turn-on cost which is the second term in Eq. 4.5, the optimal number of on-servers for minimizing (Eq. 4.5), denoted by $n^*(z)$, is achieved by setting the first derivative of (Eq. 4.5) to zero, which gives $n^*(z) = \left(\lfloor u(z)(2P_f/P_0)^{1/3} \rfloor\right)$. This $n^*(z)$ is the break-even number of servers between the cost increase due to server reboot and the power cost decrease due to server operating at a lower frequency. If we take into account the cost of turning on servers, a greedy algorithm is implemented:

1. If the number of on-servers in the previous time period $n(z-1)$ is higher than $min(n^*(z), \bar{n}(z))$, the number of servers $n(z)$ is set to $min(n^*(z), \bar{n}(z))$, and there is no associated boot cost for this;

2. Otherwise, additional servers beyond $n(z-1)$ could be turned on, but the total number of on-servers is still upper-bounded by $min(n^*(z), \bar{n}(z))$.

For the second case, the number of on-servers is determined iteratively. Assuming that the number of servers has been increased from $n(z-1)$ to q, turning on one more server beyond q will add a one-time boot cost B, while it will reduce power cost by $K_S \cdot (j - z + 1) \cdot T \cdot (-P_0 + 2P_f u^3(z)/q^3)$ (which is the derivative of the first term in (Eq. 4.5) at $n(z) = q$) only if no server will be turned off during time z to time j. The latter condition is checked by verifying if q is less than $min(n^*(z), \bar{n}(z))$ up to time j. Note that at current time period z, we do not have future aggregate frequency $u(j)$ for $j \leq z$ to calculate $\bar{n}(j)$. So we use an estimated value to restrict q, which could cause the algorithm even greedier. Based on this argument, the algorithm searches if there exists such j within the whole Z time periods, during which the saving in power cost is larger than the one-time boot cost for turning on one more server. Servers will be turned on when such j

exists until the number of on-servers touches its upper bound. After the aggregate frequency and number of servers for each application have been determined, the server frequency is calculated as $f_i(z,s) = D(u_i(z,s)/m_i(z))$, for $z = 1, \cdots, Z$ and $s = 1, \cdots, S$, where $D(\cdot)$ denotes rounding up the frequency to the different operating levels in F. Figure 4.1.1 shows the pseudo code for the greedy algorithm.

```
Server Allocation(u)
 1  /* Called at end of (u − 1) to set m_i(u) */
 2  /* Input: F_i(u − 1, s), s = 1, ··· , S; i = 1, ··· , N */
 3  /* Input: m_i(u − 1), i = 1, ··· , N */
 4  /* Input: m̄(j), j = u, ··· , U, which are estimated */
 5  F_i(u) — max_s F_i(u − 1, s); F(u) = Σ_{i=1}^{N} F_i(u)
 6  m(u − 1) = Σ_{i=1}^{N} m_i(u − 1)
 7  m̄(u) = min(⌊F(u)/f_min⌋, M)
 8  m̲(u) = ⌈F(u)/f_max⌉
 9  m*(u) = ⌊F(u)(2P_f/P_fixed)^{1/3}⌋
10  if m(u − 1) ≥ min(m*(u), m̄(u))
11    then
12          m(u) — max(min(m*, m̄(u)), m̲(u))
13    else
14          i — m(u − 1)
15          flag — 1
16          while (i ≤ min(m*(u), m̄(u)))
17              && (i ≥ m̲(u)) && (flag == 1)
18          do flag — 0
19             for j ← u to U
20             do if (K_§ T(j − u + 1)(2P_f F^3(u)/s^3 − P_fixed) > B_0)
21                        && (i + 1 ≤ m̄(j))
22                   then
23                        flag ← 1
24                        break/* get out of for j loop */
25
26
27             if flag == 1
28               then i — i + 1
29
30          m(u) ← max(i, m̲(u))
31
32  for i — 1 to N
33  do m_i(u) — ⌈m(u) * F_i(u)/F(u)⌉
34  return m_i(u), i = 1, ··· , N
```

Figure 4.2. Pseudo code of online server allocation

4.2 First-principles modeling

In this section, we focus on analyzing transient dynamics of the system in Figure 2.5 to develop control-oriented models. Over the sampling interval $[(k-1)\Delta t, k\Delta t]$ define $q(k)$ as the queue length at the end of the interval k, and $T(k)$ as the mean response time following the definition of q and T in Chapter 2. Choose allocated CPU $u(k)$ as the control input, $q(k)$ as the state variable, and $T(k)$ as the system output.

We consider the situation where the traffic load seen by the system is high and the utilization is high, i.e., the server is serving requests all the time. Since the number of requests queued up during the time period of $[k\Delta t, (k+1)\Delta t]$ is the sum of the initial queue length at $k\Delta t$ and the number of arrival requests minus the number of requests serviced in this duration, we have

$$q(k+1) = \left\{q(k) + n^k_{arriv} - n^k_{srved}\right\}^+ \tag{4.6}$$

where $\{\cdot\}^+ = max(\cdot, 0)$, enforcing queue length to be non-negative. When the traffic load is high and the system utilization is high, the number of arrivals and departures in a sampling period can be approximated by $\lambda(k)\Delta t$ and $\mu(k)\Delta t$, respectively; i.e.,

$$\lambda(k) = n^k_{arriv}/\Delta t, \mu(k) = n^k_{srved}/\Delta t \tag{4.7}$$

Also due to the high load and high utilization assumption for the system, the queue will never be empty thus the projection to the positive plane $\{\cdot\}^+$ in Eq. 4.6 can be removed, using A.2 and Eq. 4.7, Eq. 4.6 becomes,

$$q(k+1) = q(k) - \frac{\Delta t}{s(k)}u(k) + \lambda(k)\Delta t \tag{4.8}$$

Further, the average response time $T(k)$ in $[k\Delta t, (k+1)\Delta t]$ can be approximated by the sum of mean queueing delay $q^*(k)/X(k)$ and mean service time $1/\mu(k)$, where $q^*(k)$ denotes the average queue length in the k^{th} sampling period. That is,

$$T(k) = \frac{q^*(k)}{X(k)} + \frac{1}{\mu(k)} \tag{4.9}$$

For a stable open system, the throughput $X(k)$ can be approximated by the arrival rate $\lambda(k)$; while for a closed system, the throughput $X(k)$ is approximated by $\mu(k)$. By further approximating the average queue length $q^*(k)$ by $q(k)$ and absorbing the one request in service in the calculation of $q(k)$, we have for a open system,

$$T(k) = \frac{q(k)}{\lambda(k)} \tag{4.10}$$

or for a closed system,

$$T(k) = \frac{q(k)}{\mu(k)} = \frac{q(k)s(k)}{u(k)} \tag{4.11}$$

It should be noted that the system consisting of Eqs. 4.8 and 4.10 is already a linear parameter varying system with scheduling variables $s(k)$ and $\lambda(k)$. The system consisting of Eqs. 4.8 and 4.11 is a nonlinear time-varying system since the output variable $T(k)$ is inversely proportional to the control input $u(k)$.

In the rest of this section, we linearize the nonlinear equation 4.11 at specific operating points (trajectory) to derive linear time-invariant (LTI) models and linear parameter varying models. Given a sampling time period Δt, it is assumed that it might take multiple sampling periods for the system to achieve (quasi) steady state. We use index m to denote every time period ΔT when the operating condition would vary (Figure 4.3).

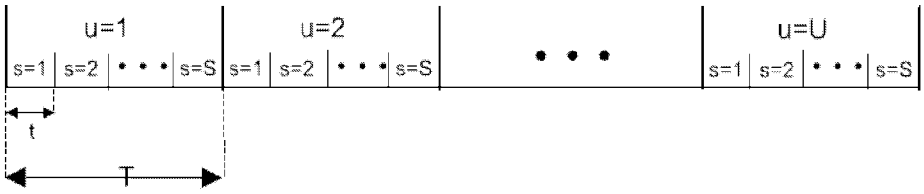


Figure 4.3. Sampling time is denoted by Δt and operating condition changes every $\Delta T = S \cdot \Delta t$

It is without loss of generality to assume that ΔT is an integer multiple of the sampling period Δt. Define

$$q(k) = \bar{q}(m) + \hat{q}(k), T(k) = \bar{T}(m) + \hat{T}(k), u(k) = \bar{u}(m) + \hat{u}(k) \tag{4.12}$$

where we let $(\bar{q}(m), \bar{T}, \bar{u}(m))$ denote the (steady-state) operating condition in the m^{th} ΔT-time period ("-" represents steady-state condition); note that $\bar{T}$ does not depend on m since we consider the same target response time $\bar{T}$ across all time periods.

There are two ways to determine operating condition $\big(\bar{q}(m), \bar{T}, \bar{u}(m)\big)$. First we calculate $\big(\bar{q}(m), \bar{T}, \bar{u}(m)\big)$ using G/G/1 queueing equations A.2 & A.7 for each ΔT, where ΔT is chosen sufficient long enough for G/G/1 modeling to be valid. That is, the linearization of nonlinear system 4.8 & 4.11 is performed around the G/G/1 steady-state condition. Alternatively, we choose the steady-state equilibrium of 4.8 & 4.11 as the operating condition $\big(\bar{q}(m), \bar{T}, \bar{u}(m)\big)$. By setting $q(k+1) = q(k)$ in Eq. 4.8 and using Eq. 4.11, the equilibrium condition in each ΔT is characterized by

$$\bar{\lambda}(m) = \bar{\mu}(m) = \frac{\bar{u}(m)}{\bar{s}(m)}, \bar{T} = \frac{\bar{q}(m)\bar{s}(m)}{\bar{u}(m)} \tag{4.13}$$

It should be noted that, when solving $(\bar{q}(m), \bar{u}(m))$ for a specified target response time $\bar{T}$ using either the G/G/1 queueing model (Eqs. A.2 & A.7) or the equilibrium condition (Eq. 4.13), the mean arrival rate $\bar{\lambda}(m)$ and service rate $\bar{s}(m)$ need to be calculated with respect to the time period ΔT (the choice of ΔT could be different for using G/G/1 and the equilibrium condition in (Eq. 4.13).

Linearizing 4.8 & 4.11 at $(\bar{q}(m), \bar{T}, \bar{u}(m))$ and using 4.12, we have

$$\begin{aligned} \hat{q}(k+1) = &\ \hat{q}(k) - \tfrac{\Delta t}{\bar{s}(m)}\hat{u}(k) + \eta(k) \\ \hat{T}(k) = &\ \tfrac{\bar{s}(m)}{\bar{u}(m)}\hat{q}(k) + \tfrac{\bar{s}(m)}{\bar{u}^2(m)}\bar{q}(m)\hat{u}(k) \end{aligned} \tag{4.14}$$

where

$$\eta(k) = \lambda(k)\Delta t - \frac{\Delta t}{s(k)}\bar{u}(m) \tag{4.15}$$

is modeled as a disturbance.

4.2.1 Derivation of LTI and LPV models

Based on Eqs. 4.12-4.15 or Eq. A.2, A.7, 4.12, and 4.14-4.15, we have derived a set of Linear Time-invariant and Linear Parameter-Varying Models as follows.

Linear Time-Invariant Models: If we choose ΔT as the entire duration of

the workload, i.e. $s = 1$ in Figure 4.3, then there is only one single operating condition $(\bar{q}, \bar{u})$ (which gives the target response time $\bar{T}$) in the whole duration of the workload, if we further approximate the $s(k)$ and $\lambda(k)$ in Eqs. 4.14-4.15 by constant values (e.g., the mean or high percentile value computed over the entire duration of the workload), the system Eqs. (4.14-4.15) becomes a linear time-invariant system. We define the following LTI models:

- *LinearModel-Equilbm*: the LTI model with operating condition $(\bar{q}, \bar{u})$ satisfying Eq. 4.13. Note that if we use the same and s and λ in Eq. 4.13 and 4.15, the disturbance η in 4.15 disappears.

- *LinearModel-G/G/1*: the LTI model with operating point $(\bar{q}, \bar{u})$ satisfying G/G/1 equations A.2 & A.7.

Linear Parameter Varying Models: If we consider an operating trajectory $(\bar{q}(m), \bar{u}(m))$, $m = 1$ to S, which is solved either from the G/G/1 queueing model A.2-A.7 or the equilibrium equation 4.13 for all the ΔT time periods, the linearized system Eqs. 4.14-4.15 become a linear parameter varying system derived based on Jacobian linearization. We define the following LPV models:

- *LPVModel-Equilbm*: the LPV system with $(\bar{q}(m), \bar{u}(m))$ satisfying Eq. 4.13. Essentially the scheduling variables are the workload arrival rate λ and service demand s.

- *LPVModel-G/G/1*: the LPV model with $(\bar{q}(m), \bar{u}(m))$ satisfying Eq. A.2 & Eq. A.7. Note that the operating trajectory depends on the workload arrival rate λ, service demand s, and squared coefficients of variation C_a^2 and C_s^2.

- *LPVModel-OpenS*: the LPV system Eqs. 4.8 & 4.10.

It should be noted that if the target response time $\bar{T}$ is chosen to be time-varying (e.g. piece-wise constant), $\bar{T}$ could be used as one scheduling variable for the LPV system, but we do not consider this case in this chapter.

4.2.2 Remarks on the modeling

The first-principles models developed in this section are essentially parameterized (scheduled) by the workload characterizing parameters; in particular, request ar-

rival rate λ and service demand s. One of the advantages for using these derived models for the performance management of Web servers is that same off-line control designs can be used to deal with different types of workloads/Web applications, only requiring on-line measurements of workload parameters. Therefore, the proposed modeling and control design in this chapter could lay a good foundation for building a middle ware for different applications. To the best of our knowledge, it is the first reported work to develop first-principles based LPV models for server performance control by specifying workload parameters as scheduling variables.

In [1], a similar form as Eq. 4.8 was used to describe queue dynamics for the congestion control of communication network systems, $q(k+1) = q(k)+\sum_{m-1}^{M}(r_m(k)-\mu(k))$; where r_m is the control variable, denoting the source rate from source m at the input of the bottleneck link. The service rate $\mu(k)$ in [1]was modeled by a p-dimensional stable Autoregressive (AR) process and a Linear Quadratic control was applied to determine r_m in order to achieve target queue length. We consider the performance management for Web server systems for which response time (other than queue length) is the primary performance metric, which introduces nonlinearity and is much more difficult to model and control. Another significant difference lies in the modeling of request service demand. Requests in a network system are just fixed-size packets (all packets have the same size); the service time of a request at a link (leaving a router) corresponds to the transmission time of the packet. Thus a request's service time is more or less constant (depending on the link bandwidth); an exponential service time distribution would be good enough for modeling communication networks. In contrast, service demand in a Web/storage server system is highly varied and poses serious modeling challenges. Also motivated by the complicated workload characterization and difficulty in deriving accurate performance models, system identification algorithms are applied in Section 4.3 to develop control-oriented empirical models.

4.3 Empirical modeling

We adopt similar linear and LPV system identification algorithms as that are used in Chapter 3. Here the control input is CPU speed instead of rejection ratio though. Again, depending on the system under consideration, the vector r could

include different parameters that characterize workload behavior, e.g. arrival rate, file size, locality, and read/write ratio etc (or the scheduling parameters used in the first-principles models in Section 4.2). As a starting point, we use a scalar scheduling parameter $r(k)$ defined as the reciprocal of workload intensity,

$$r(k) = \frac{1}{\lambda(k)s(k)} \tag{4.16}$$

which makes the implementation of the LPV system identification easier and more efficient. Here workload intensity is used slightly different from its convectional definition, which is the ratio of the arrival rate λ and service rate μ. We denote the model obtained as *LPVModel-ARX*.

Motivated by the analytical LPV model which is derived by linearization around G/G/1 steady-state operating conditions Eqs. A.2 & A.7, we have also investigated LPV models using additional scheduling parameters (in the system identification) such as the squared coefficients of variation of the interarrival time C_a^2 and service demand C_s^2. In particular, we specify $r = (\frac{1}{\lambda s}, \frac{1}{\lambda s C_a^2}, \frac{1}{\lambda^2 s^2 (C_a^2 + C_s^2)})$ and refer to the corresponding LPV model as *LPVModel-ARX-VAR*.

4.4 Control design tools

Similar control design tools in Chapter 3 are used here. Corresponding to each linear/LPV model derived in the previous section, we design control for the CPU frequency $u(k)$ so that the response time $T(k)$ will meet its target value $\bar{T}$ while reducing energy consumption. We classify the designs into linear and LPV and explain each of them in detail as follows.

4.4.1 Linear designs

We formulate a Linear Quadratic (LQ) control problem, which optimizes the cost function as follows:

$$J = \sum_{k=1}^{\infty} (r_T \cdot \hat{T}^2(k) + r_u \cdot \hat{u}^2(k)) \tag{4.17}$$

where r_T and r_u are weights for meeting the response time SLA and minimizing the control energy.

a) *LinearDesign-Equilbm*: linear design at high-percentile load conditions using the *LinearModel-Equilbm*.

Consider the LTI model *LinearModel-Equilbm*, where constant values of the arrival rate λ and service demand s will be used in Eqs. 4.12-4.15. In order to meet response time SLA for varying load conditions while reducing design conservativeness, we design the linear controller using different percentile values of λ and s calculated over entire workload, since the arrival rate and service demand could vary a lot in the entire workload. In particular, the following percentile $\alpha = 50-, 85-, 95-$percentile values for λ and s are used. Therefore, the linear optimal control is designed with objective function 4.17 subject to the following dynamic equation,

$$\begin{aligned} \hat{q}(k+1) = &\quad \hat{q}(k) - \tfrac{\Delta t}{s_{\alpha\%}}\hat{u}(k) \\ \hat{T}(k) = &\quad \tfrac{1}{\lambda_{\alpha\%}}\hat{q}(k) - \tfrac{T}{s_{\alpha\%}\lambda_{\alpha\%}}\hat{u}(k) \end{aligned} \tag{4.18}$$

where the subscript "$\alpha\%$" represents the α-percentile value. In order to reduce steady-error in meeting target response time, we augment the system Eq. 4.18 by adding an integrator. Define a new state variable $\hat{q}_1$,

$$\hat{q}_1(k) = \sum_{i=1}^{k-1} \hat{q}(i) \tag{4.19}$$

$$\hat{q}_1(k+1) = \hat{q}_1(k) + \hat{q}(k) \tag{4.20}$$

The state feedback control law for CPU frequency $u(k)$ is then computed as,

$$u(k) = \bar{u} + [k_1 k_2] \begin{bmatrix} \hat{q}_1(k) \\ \hat{q}(k) \end{bmatrix} = \lambda_{\alpha\%} s_{\alpha\%} + k_1 \hat{q}_1(k) + k_2 \hat{q}(k) \tag{4.21}$$

where the feedback gain k_1 and k_2 are computed by solving the LQ problem defined by Eqs. 4.17-4.20. Following a similar procedure, we have the following control designs corresponding to different linear models.

b) *LinearDesign-G/G/1*: linear design using the *LinearModel-G/G/1*.

c) *LinearDesign-ARX*: linear quadratic design using the *LinearModel-ARX* in

Section 3.2.1.

d) *LinearDesign-ARX + G/G/1-Feedforward*: add a feedforward controller using the G/G/1 queueing model to the design c).

It should be noted that the linear designs a) and b) presented above are not online implementable since the entire workload is not available a priori to compute the required information on arrival rate λ and service demand s. For real-time implementation, predicted values for λ and s would be used instead, which could lead to additional uncertainty in modeling thus degrade control performance. In order to use designs c) and d), we need historical workload data as training data to build the ARX model. Since we mainly use the above linear controllers as baseline designs to compare with LPV modeling and designs, we neglect the online implementation issues for the above linear controllers.

4.4.2 $LPV - H_\infty$ control designs

We design controllers for each LPV models derived in the modeling section.

a) *LPVDesign-OpenS*: $LPV - H_\infty$ control design for the *LPVModel-OpenS* consisting of Eqs. 4.8-4.10.

Note that for the open-system modeling, system (4.8-4.10) is already a linear parameter varying system. Define a (shifted) new control variable, $\hat{u}(k) = u(k) - \bar{u}(k)$ with $\bar{u}(k) = \lambda(k)s(k)$, Eqs. (4.8-4.10) become

$$q(k+1) = q(k) - \frac{\Delta t}{s(k)}\hat{u}(k), T(k) = \frac{q(k)}{\lambda(k)} \tag{4.22}$$

It is an affine parameter-dependent plant with scheduling variables $r_1 = 1/\lambda(k)$ and $r_2 = 1/s(k)$. In implementation the scheduling variables can be determined by measuring workload arrival rate and request size. In addition we need to specify the range of the scheduling variable. This can be done offline by estimating the extreme cases of the workload. Once we obtain the LPV model and the scheduling parameter, we can apply the similar LPV synthesis technique and system interconnection as used in Chapter 3.

Following a similar procedure, we have designed the following controllers:

b) *LPVDesign-Equilbm*: $LPV - H_\infty$ control design for the *LPVModel-Equilbm*.

Define scheduling parameters $r_1 := 1/\lambda(k)$, $r_2 := 1/s(k)$, and $r_3 :=$

$1/(\lambda(k)s(k))$. The *LPVModel-Equilbm* then becomes an affine parameter-dependent plant. Note that the scheduling parameters defined in this way are not independent, which may cause the resulting design more conservative, but it allows the direct application of affine parameter-dependent LPV control designs without resorting to the polynomial parameter-dependent LPV. The disturbance $\eta(k)$ in (4.15) is included in the exogenous input w in the H_∞ control design.

c) *LPVDesign-G/G/1*: $LPV - H_\infty$ control design using the *LPVModel-G/G/1*.

We specify scheduling parameters $r_1 = 1/s(k)$, $r_2 = s(k)/\bar{u}(m)$, and $r_3 = s(k)\bar{q}(m)/\bar{u}^2(m)$ with $(\bar{u}(m), \bar{q}(m))$ from (A.2) & (A.7).

d) *LPVDesign-LPVARX*: $LPV - H_\infty$ control design using the *LPVModel-ARX*.

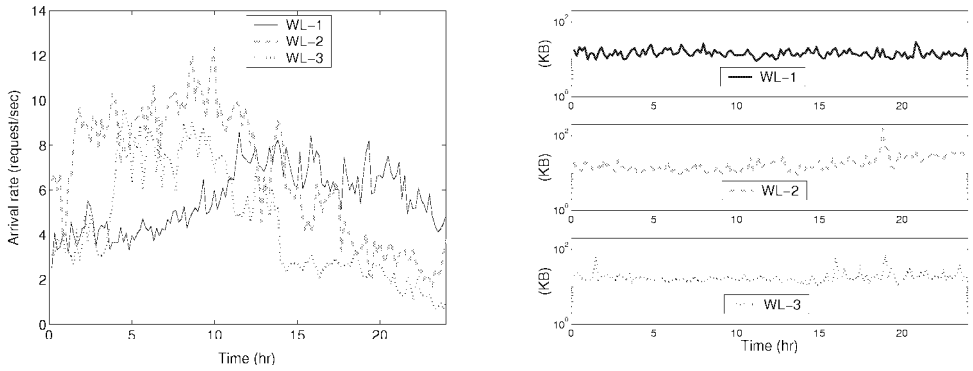
According to the trace used in this chapter (see next section), data fitting of the LPV model with polynomial dependence on scheduling variables does not improve much than the LPV model with affine dependence on scheduling variables; therefore, the $LPV - H_\infty$ control algorithm for affine LPV models is used, though the Sum-of-Squares based synthesis [46]could be applied for polynomial LPV-ARX models.

4.5 Simulation results and performance evaluation

Modeling and control designs have been evaluated using a simulator built on top of the commercial simulation library CSIM. We use real HTTP traces from [50], which were collected during October 2004. There are 3 traces in all, denoted as Workload 1 to 3 (or WL 1-3). The arrival pattern and service demand (file size) of the three (one-day) workloads are plotted in Figure 4.4. Note that the values of dynamic model coefficients highly depend on the choice of sampling period. We present results obtained for sampling period $\Delta t = 2$ minutes and $\Delta T = 10$ minutes ($\Delta T = 1$hr whenever G/G/1 queueing model is used since it gives the best result compared to other time granularities). Sensitivity of the modeling and control design to sampling interval is discussed as well. CDF plots of request sizes and inter-arrival times indicate that these traces have large variances.

4.5.1 Modeling and validation

We first study if the developed first-principles models capture major system dynamics of real traces; we only analyze LPV models here. Then training experiments are conducted to identify linear ARX and LPV-ARX models. In both cases, a trajectory of CPU frequency $u(k)$, which is proportional to the time-varying workload intensity (arrival rate λ times service demand s), is used to run the workload. The resulting time histories of queue length $q(k)$ and response time $T(k)$ are measured.



(a) Arrival rate (b) Service demand

Figure 4.4. Some example Web server workloads

4.5.2 Results from first-principles approach

Figure 4.5 plots the measured response time $T(k)$ versus the one-step prediction by the *LPVModel-Equilbm* and the *LPVModel-G/G/1* for all three workloads. Except for prediction given by the *LPVModel-G/G/1* running Workload 2, we can see that both LPV models give reasonable accurate predictions; in particular, they are able to capture the spikes in response time. The predictions are relatively more conservative when response time is low (both models tend to predict higher response time than the measured values), e.g., hour 15-24 of Workloads 1&3 by the *LPVModel-Equilbm*, first 15 hrs of Workload 3 by *LPVModel-G/G/1*. This deterioration in prediction corresponds to the situation when the server is experiencing light load so that the modeling assumptions might not be well satisfied.

It is also noted that the *LPVModel-G/G/1* provides slightly better prediction in certain regions, e.g., after Hour 20 for both Workload 1 and Workload 3. The

poor prediction of the *LPVModel-G/G/1* for Workload 2 at Hour 18-20 is due to the significantly high variation of request arrival rate C_a^2 for which Eq. A.7 is a bad approximation for queue length; the several miss-prediction points after hour 20 are due to the high variance of service demand C_s^2.

4.5.3 Identification results

The same system identification algorithms are applied to each of the three workloads (WL-1/2/3). For Workload 2, the sampling time Δt is set to 5min in order for the system identification algorithm to converge while $\Delta t = 2$min for both Workload 1 & 3. In model validation, similarly to the previous section, a trajectory of CPU frequency $u(k)$ that is proportional to the time-varying workload intensity is used to run the workload.

Figure 4.6 shows the model predicted response time versus the measured values using the same CPU frequency as in Figure 4.5. We can see that the *LPVModel-ARX* fits data quite nicely (in particular for Workload 1), improving significantly compared to the *LinearModel-ARX*. By comparing the *LPVModel-ARX* in Figure 4.6 with the analytical *LPVModel-Equilbm* in Figure 4.5, it seems that for Workload which is less varied (e.g., Workload 1), the system-identified model gives slightly better predictions while for Workloads with large variations/sudden jumps (e.g. Workload 2 & 3), the two models are quite comparable.

We have also evaluated the *LPVModel-ARX-VAR*, which utilizes variance (besides mean) of load conditions as additional scheduling parameters in the system identification. It is observed that there is marginal improvement in prediction by taking into account the variance in workload arrival and service distributions in the system identification.

4.5.4 Control design

Each LQ and LPV design described in Section 4.4 has been implemented in the simulation package CSIM. Table 4.1 lists values of design parameters for each controller: feedback gains in Eq. 4.21 for LQ designs and weighting functions for $LPV - H_\infty$ controllers (same weighting functions are used across different LPV designs). Linear designs using 85−percentile load conditions are presented

in Table 4.1 since they achieve the best trade off in meeting target response time and reducing CPU usage compared to designs using 50− and 95−percentile load conditions. The target response time $\bar{T}$ is set to 20 sec for all the designs.

Design Methods	Design parameters		
	Workload 1	Workload 2	Workload 3
LinearDesign-Equilbm	$k_1 = -0.02$ $k_2 = -0.032$	$k_1 = -0.0044$ $k_2 = -0.0008$	$k_1 = -0.0435$ $k_2 = -0.0214$
LinearDesign-G/G/1	$k_1 = -0.0415$ $k_2 = -0.0205$	$k_1 = -0.0170$ $k_2 = -0.0083$	$k_1 = -0.0432$ $k_2 = -0.0209$
LinearDesign-ARX	$k_1 = -0.1666$ $k_2 = 0.0037$	$k_1 = -0.0294$ $k_2 = -0.0295$	$k_1 = -0.0538$ $k_2 = -0.0045$
LPVDesign (systemID models)	$W_e = \frac{0.18z^2-0.34z+0.16}{z^2-1.95z+0.95}$ $W_u = \frac{4.71z^2-9.19z+4.49}{z^2-1.77z+0.83}$	$W_e = \frac{0.31z^2-0.47z+0.20}{z^2-1.88z+0.89}$ $W_u = \frac{9.27z^2-17.43z+8.22}{z^2-1.51z+0.68}$	$W_e = \frac{0.07z^2-0.13z+0.06}{z^2-1.95z+0.95}$ $W_u = \frac{4.71z^2-9.19z+4.49}{z^2-1.77z+0.83}$
LPVDesign (analytical models)	$W_e = \frac{3.66z^2-6.74z+3.19}{100(z^2-1.95z+0.95)}$, $W_u = \frac{0.94z^2-1.84z+0.90}{z^2-1.77z+0.83}$ $\overline{d}$		

Table 4.1. Control design parameters

Model Type	Design Method	Workload 1		Workload 2		Workload 3	
		RT	CPU	RT	CPU	RT	CPU
SystemID	ARX	19.88	17.98	34.00	71.59	18.15	20.49
	ARX+G/G/1	21.16	18.41	29.87	72.83	19.81	20.23
	LPVARX	19.90	14.48	20.16	37.79	17.83	13.44
Analytical	Linear-Equilbm	21.21	22.50	25.99	38.18	17.93	26.16
	Linear-G/G/1	22.49	24.41	28.52	39.49	19.59	24.78
	LPV-OpenS	19.89	14.02	20.06	19.87	17.83	13.42
	LPV-Equilbm	19.89	14.03	20.06	19.86	17.83	13.42
	LPV-G/G/1	19.63	19.93	18.49	27.87	18.22	22.12
Pure-G/G/1		3.23	31.98	8.44	45.13	33.4	22.48

Table 4.2. Performances results of different design methods

4.5.5 Simulation results

In this section, we present simulation results for each control design running real traces plotted in Figure 4.4. Table 4.2 lists the average response time and CPU frequency for each design, where for the *LinearDesign-Equilbm* and *LinearDesign-G/G/1*, the results designed at 85−percentile of load conditions are presented here. Average values for response time and CPU are calculated with respect to the whole 24 hr duration of the workload. The unit of the CPU frequency has been transformed from MHz to the processing capacity MB/sec. For comparison purpose, we also list the result produced by using $u(m)$ (ΔT = 1hr), which is computed directly from $G/G/1$ queueing model A.2 & A.7 by setting target response time $\bar{T}$ = 20sec; in the computation of $u(m)$, we assume full knowledge

of the workload thus no prediction is involved. We denote this baseline design as *Pure-G/G/1* in the rest of the chapter.

Note that the target response time $\bar{T}$ is set to 20sec, for performance evaluation, we take 10% slackness and specify that if the average response time for a design is less than 22 secs, the response time SLA is met and the design is a feasible solution. We only compare the magnitude of CPU frequency among feasible designs. Except the *Pure-G/G/1* for Workload 3, we can see that almost all designs satisfy the response time SLA for Workload 1 and Workload 3 (the *LinearDesign-G/G/1* has slightly higher response time for Workload 1); while for Workload 2, only LPV designs and the *Pure-G/G/1* design meet the target response time.

Linear vs. LPV: From Table 4.2, we can see that compared to linear designs, the LPV designs save up to 30% CPU in average for Workload 1 and Workload 3. For Workload 2, LPV designs are able to meet target response time while use significantly less CPU than linear designs. This demonstrates that the utilization of detailed time-varying information on workload improves the performance of control design substantially. By examining the time-varying behavior in Figure 4.7, where response time histories of the *LPVDesign-Equilbm* and the *LinearDesign-Equilbm* are plotted, we can see that the LPV design outperforms linear designs significantly. In particular, during the transient overload situation (e.g., ~hour 21 of Workload 1, ~hour 19 of Workload 2 and ~hour 16 of Workload 3 by observing both arrival and service demand shown in Figure 4.4), the LPV design is able to allocate the right amount CPU thus to maintain reasonable response time while the LQ design is not able to handle this, which causes significant oscillations in CPU allocation, dramatic spikes in response time and getting stuck in overload for much longer time.

W/ vs. W/O G/G/1-Feedforward: Both *LinearDesign-G/G/1* and *LPVDesign-G/G/1* can be viewed as G/G/1 feedforward control combined with feedback control. Table 4.2 shows that G/G/1-feedforward based linear designs provide no or marginal improvement in reducing response time and CPU usage compared to other linear designs. It is a clear trend that G/G/1-feedforward based LPV designs consume much higher CPU than other LPV designs, which is very possibly due to the conservativeness of the G/G/1 model in response time prediction.

Design Based on First-Principles vs SystemID Model: For Workload 1 and Workload 3, linear designs using system-identification based models have slightly better response time (with lower CPU usage) than the linear designs based on analytical models; the results of LPV designs based on both models are very close to each other (except the *LPVDesign-G/G/1*). While for Workload 2, whose variability in arrival pattern and service demand is very high thus is quite challenging for the system identification algorithms used in this chapter, the designs based on first-principles models are more competent, showing significant reduction in CPU usage while meeting target response time. In terms of easiness for implementation, analytical-model based LPV designs are not trace dependent - all designs can be done offline and only require on-line measurement of load parameters thus effort in building models using historical training data is saved. On the other hand, system-identification based modeling is more generic and applicable to broader types of systems and traces, which is particular useful when analytical performance models are hard to derive for complicated systems (e.g. storage systems, multiple-queue networks and other complex queueing systems).

Control Theoretic vs Pure-G/G/1: Note that the control of CPU for the *Pure-G/G/1* is updated every 1hr since it is the time granularity for which queueing predication has the best accuracy for the traces used, while feedback control design is implemented with Δt = 2min (Δt = 5min for the *LPVDesign-LPVARX*) and Δt = 10min. In general, it is expected to see that queueing-based approaches perform better in coarse time granularity while feedback control would be more responsive for finer time granularity. Here we focus on comparing performance of the *LPVDesign-Equilbm* vs the *Pure-G/G/1*. From Table 4.2, we can see that, in general (except for Workload 3), the *Pure-G/G/1* provides very low response time (far less than the target value) but with almost double CPU usage than the *LPVDesign-Equilbm*. This shows that the result from the G/G/1 model is quite conservative, and there is no flexibility in trading off meeting target response time and reducing CPU usage. For Workload 3, the squared coefficient of variation for arrival rate C_a^2 is around $2.5 \sim 5.0$ for certain periods, which makes Eq. A.7 a poor approximation thus the *Pure-G/G/1* can not meet target response time. We do not present the time-varying behavior between the LPV design vs. the *Pure-G/G/1* here due to the dramatic difference in response time and CPU

usage.

Energy Consumption: With these allocation designs we evaluate the energy consumption of each design, which is the result of the combination of each linear/LPV control with the same on-line server allocation algorithm listed in the Appendix. We will also compare our designs with several naive heuristics that are static, rather than dynamic, control of performance/power: 1) turning on all servers in the pool (M = 30 servers) all the time, either at the minimum frequency (denoted as All-server/min-freq) or at the maximum frequency (denoted as All-server/max-freq), where the number M = 30 is determined in terms of the peak load; 2) operating servers at the minimum/maximum frequency with minimal necessary number of servers to meet the target response time (denoting them as Min-server/min-freq and Min-server/max-freq, respectively). Note that for the last two schemes, it needs an exhaustive search of number of servers to find the least necessary number to meet response time SLA, thus they are not online implementable. As listed in Table 4.2, we found that it takes at least 8/13/8 servers to operate at the maximum frequency to meet response time SLA for Workload 1/2/3 respectively, while it takes at least 13/21/14 servers to run at the minimum frequency for meeting target response time. For the comparison of energy consumption across different designs, we evaluate the energy consumption of each design expressed as a percentage of the energy (Energy %) that would be consumed by All-server/max-freq.

Remark 1: We have also evaluated the modeling and control designs based on using a single LPV model, which is identified from either one of the three traces. In particular, 1) applying the LPV model and weighting functions of LPV design derived from WL1, WL2 has average response time 136.56 secs with average aggregate CPU 42.53 GHz and WL3 has average response time 22.04 with aggregate CPU 22.47; 2) applying the LPV model and design parameters derived from WL2, WL1 has average response time 24.97 with aggregate CPU 28.31 and WL3 has response time 18.94 with CPU 39.42; 3) applying the LPV model and design parameters derived from WL3, WL1 has average response time 25.55 with aggregate CPU 18.16 and WL2 has response time 32.51 with CPU 33.18. Considering that the target response time is 20sec (plus the 10% slackness, it is considered to satisfy the response time SLA if the response time is less than 22sec), the above

performance results based on a single LPV model are reasonably good. The only exceptional case comes from applying the model derived from WL1 to WL2, which has several times longer response time than the target value. However, noting that WL2 changes significantly from WL1 in that it has substantially higher variance in both arrival and service demand, any system-identification based approach would have difficulty in dealing with this since in order for any system-identification based models to work well, it is not uncommon to assume that there are no dramatic changes between the evaluation data and the training data. Alternatively, the first-principles based approach could be a better option since it does not rely on any historical data training. Noting that in this paper, we have investigated both first-principles based and system-identification based modeling and control designs, we do not intend to advocate one approach over another, but rather evaluate the pros and cons of each approach and view them as complementary approaches.

Considering that WL2 seems to have the highest variance in both arrival and file size among the three workloads, future work to improve current LPV system identification results may include incorporating variance information of the workload into scheduling parameters, or exploring higher order (or even nonlinear) dependence on scheduling variables.

Remark 2: Note that the system identification related results presented in the previous and current sections essentially used the same data for both identifying models and evaluation. Evaluating an identified model on the original training data (based on which the model is built) is the first step to validate a system identification model since it is not necessary that any derived model would fit the training data well - appropriate function form and system order would be required to provide a good fit. One major purpose for presenting this type of system-identification results is to demonstrate that an LPV model would fit the data; the identified LPV models capture the right functional relationship with respect to load parameters and sufficient system order to characterize system dynamics. Another purpose is to serve as a comparison basis for the first-principles based approach. We would like to investigate how the analytical models (which are developed based on certain approximations) would compare to the "perfectly"-identified models. The second step for model validation often requires evaluating the identified model on a new set of data. The results from Remark 1 have provided sufficient information on

this aspect. In particular, if we append WL2/WL3 to WL1, append WL3/WL1 to WL2, or append WL1/WL2 to WL3, then each "new" workload, which is a combination of any two original workloads, could be viewed as a 2-day trace, and the first half is used to build the model while the other half is used for evaluation.

4.5.6 Sensitivity to sampling interval

The choice of sampling period may affect system modeling and the resulting control designs. There is a trade off in choosing an appropriate sampling period. Shorter sampling interval tends to capture system dynamics more accurately and make the system more responsive, but it could cause the model to be more sensitive to noise, further, the modeling assumptions in Section 4.2 for a queueing system may not be satisfied for an overly short sampling period. We have examined the sensitivity of modeling and control design to different time granularities such as $\Delta t = 2-$, $5-$, $10-$, $20-$ and $30-$ minutes sampling periods. We found that results for $\Delta t = 2-$ and $5-$ minutes intervals are acceptable but the results for $10-$, $20-$, and $30-$ minutes sampling time have larger response time in several orders of magnitude.

4.6 Summary

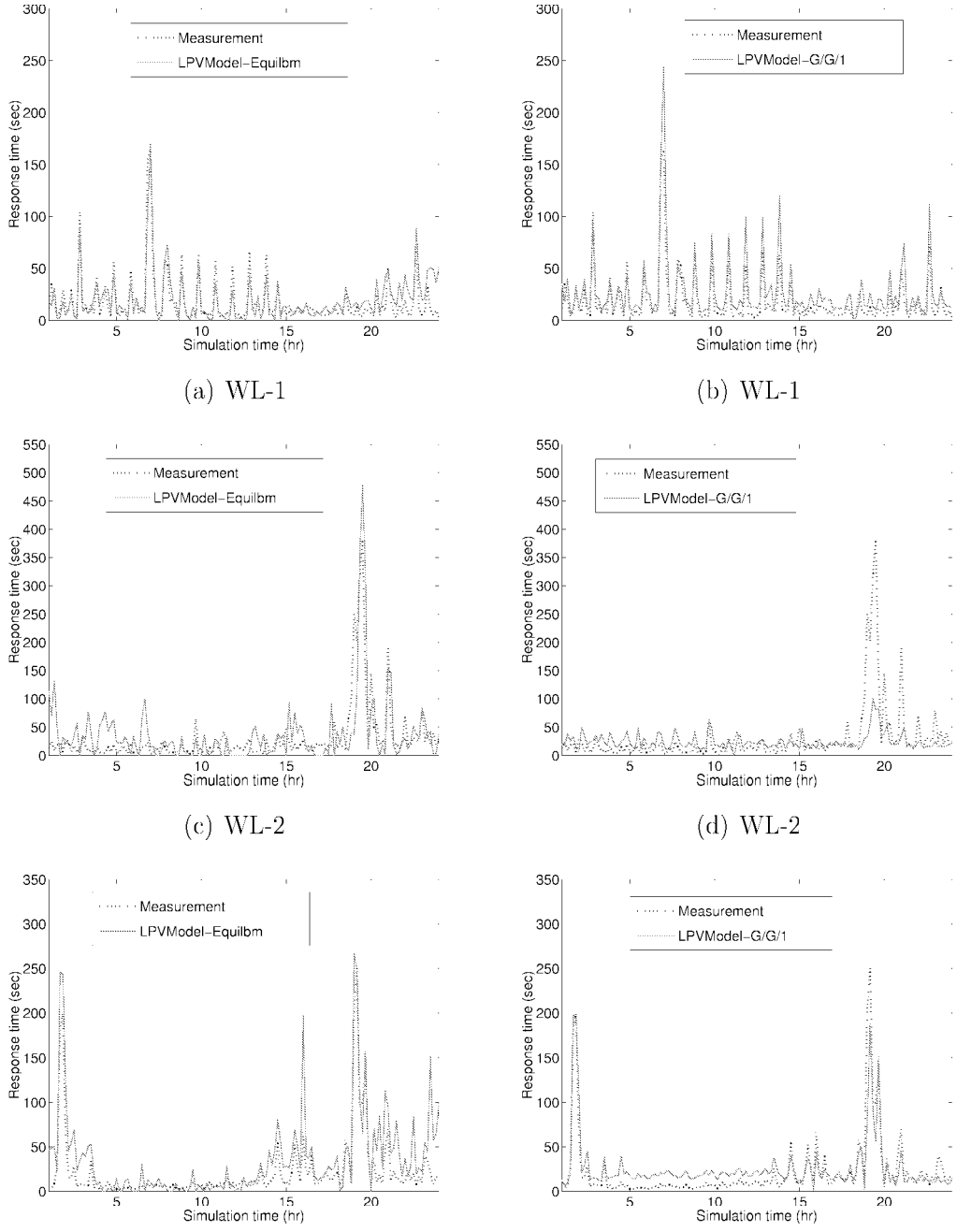
This chapter has presented a comprehensive framework for control-oriented modeling and design for performance management of Web servers in achieving response time SLA. We have developed system identification models as well as first-principles models, which are based on analyzing transient queueing dynamics for a hosting server in the presence of time varying load conditions. Both LTI and LPV control designs are studied and queueing theory based optimization results are presented as a benchmark. Results show that LPV designs outperform both LTI and queueing optimization, in terms of responsiveness and conservativeness. LPV control design achieves the best trade off between cost and revenue and thus the optimal profit among the three approaches.

As a significant contribution of the proposed approach, we model the performance control of a hosting server as a LPV system with the time-varying load conditions as scheduling variables. Various LPV models have been evaluated, and

corresponding controllers are designed. Through simulations using real Web traces, it is demonstrated that LPV designs outperform linear controls and a conventional G/G/1-based approach in terms of mean response time, mean control effort, and time histories of response time.

It is worth pointing out that due to the inherent LPV nature of server system performance with respect to workload variations, the presented framework in this chapter provides the versatility in dealing with different types of workload and operating environment without much modifying the implementation of control algorithms. The general framework of this LPV modeling and control is applicable to a variety of resource and performance management problems for server systems.

The results of this chapter are published in SIGMETRICS 2005 [47], American Control Conference 2006 [51], and IEEE Transaction of Control System Technology 2007 [52].

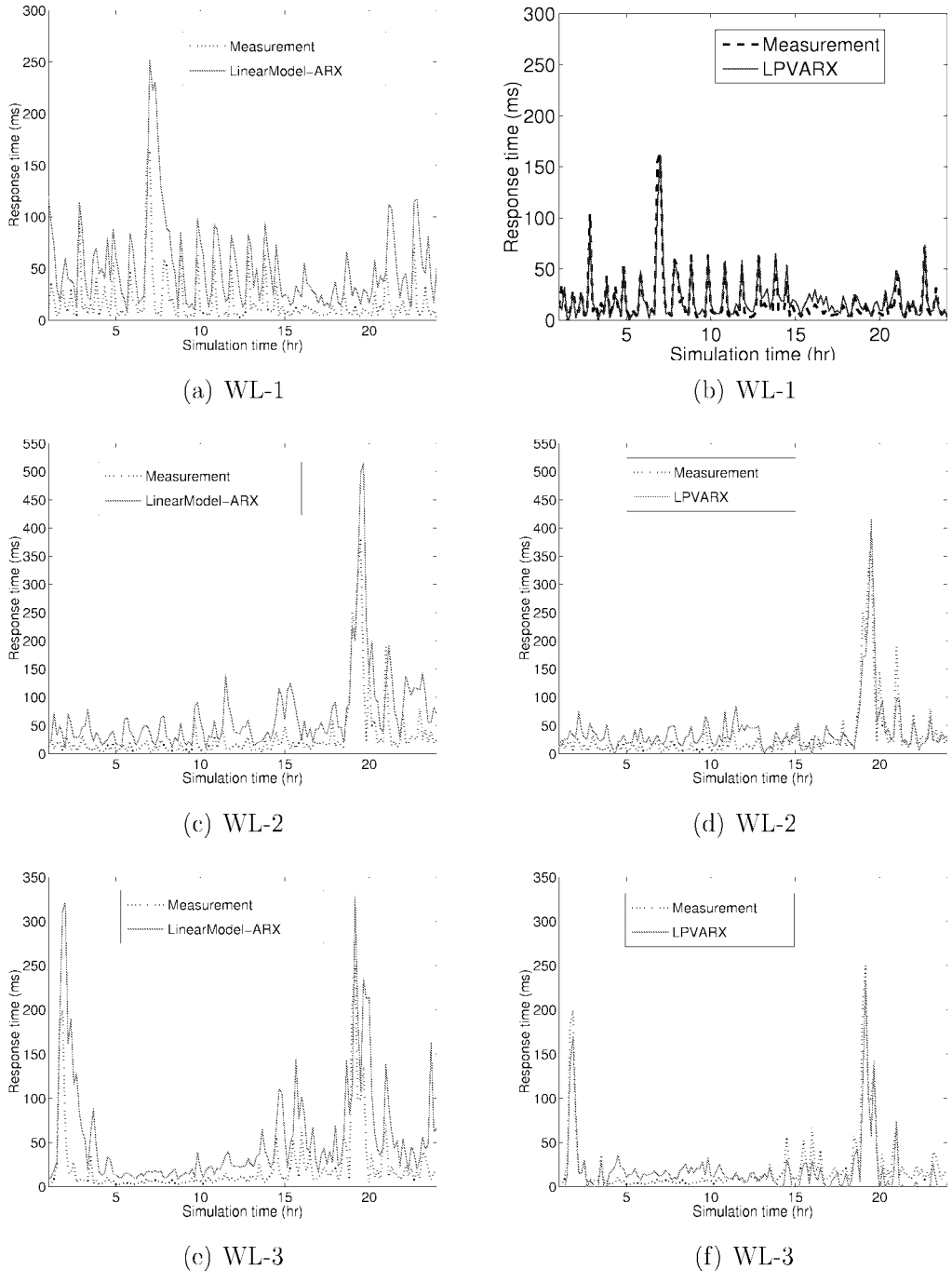


(a) WL-1 (b) WL-1

(c) WL-2 (d) WL-2

(e) WL-3 (f) WL-3

Figure 4.5. Model validation for analytical LPV models, where results for the *LPVModel-Equilbm* and the *LPVModel-G/G/1* are shown.

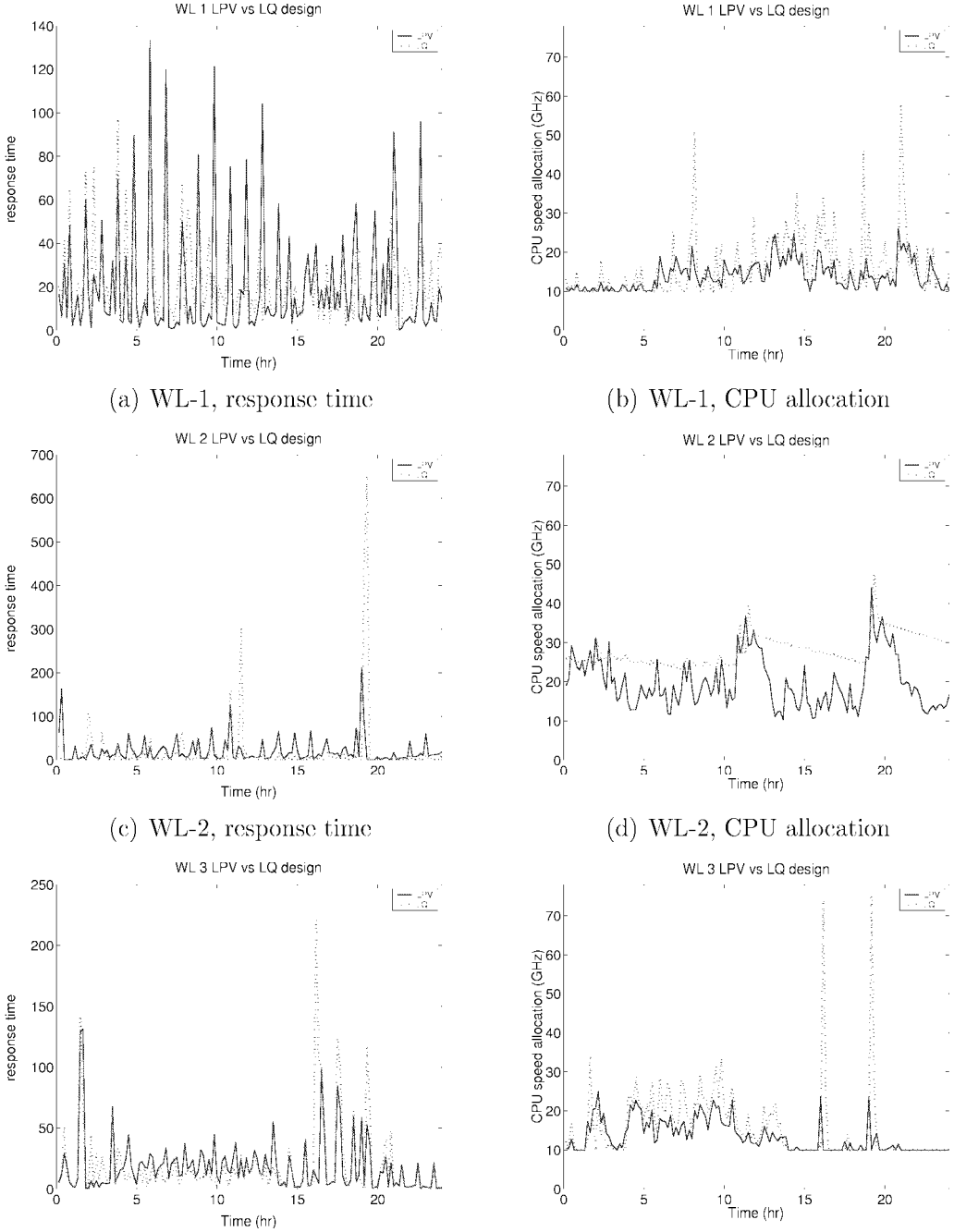


(a) WL-1 (b) WL-1

(c) WL-2 (d) WL-2

(e) WL-3 (f) WL-3

Figure 4.6. Validation for system-identification models, where the *LinearModel-ARX* and the *LPVModel-ARX* are shown.



(a) WL-1, response time

(b) WL-1, CPU allocation

(c) WL-2, response time

(d) WL-2, CPU allocation

(e) WL-3, response time

(f) WL-3, CPU allocation

Figure 4.7. Time histories of response time for the *LPVDesign-Equilbm* vs the *LinearDesign-Equilbm*.

Chapter 5

Managing Server Performance Under Self-Similar Workloads

This chapter presents a novel control-theoretic approach based on LPV techniques and workload characterization using α-stable-model based stochastic envelopes. The proposed approach parameterizes a control-oriented dynamic-system model and resulting controller using workload-distribution parameters. By further incorporating the α-stable modeling into the LPV control approach, the presented solutions not only allow system to adapt to workload changes, but also show great promise in handling *self-similar* workloads. The proposed system modeling and control design approach is applied to a CPU management problem for web servers, which performs Dynamic Voltage/Frequency Scaling to achieve response time guarantee. Simulations using real web-server traces are conducted to show the strength of the proposed approach.

5.1 Introduction

Beside feedback control, another body of work in the resource management of Internet service has focused on understanding and characterizing the load imposed on these servers by studying real world traces from different environments. Here we do not intend to review the numerous references in this area but try to concentrate on the literature on characterizing the self-similar traffic, for which [53] gives an extensive bibliographical guide to research in self-similar network traffic and perfor-

mance modeling. Note that incoming traffic to web servers may exhibit self-similar behavior [54, 55] thus cannot be modeled by Poisson and Markovian processes due to long-range dependence, which refers to correlations that decay very slowly [56]. In addition, such self-similar traffic also tends to exhibit burstiness in a wide range of time scales. While heavy-tailed Pareto and stable distributions can be used to model burstiness [54, 56], they cannot directly model long-range correlations. At the other end, Gaussian self-similar models can be used to capture long-range correlations and the self-similarity at different time scales [57], but they fall short in their ability to model burstiness. There is also research which applies multi fractal (multiplicative processes or cascade models), in contrast to a mono fractal scaling (single Hurst parameter) characterization, to model more complex scaling behavior in networks [58]. In this chapter, we are particularly interested in recent work [59, 60] on using α-stable stochastic processes to model the self-similarity of incoming traffic. Over and beyond the flexibility of fitting different levels of burstiness and correlation in data, it provides a small set of parameters which have clear physical meaning with manageable on-line computational cost.

In this chapter we investigate (i) how well do purely predictive (using sophisticated workload models such as the α-stable stochastic processes), and purely feedback-based (without a sophisticated workload model) strategies compare under real workload? and (ii) can we integrate the sophisticated prediction models with feedback control to perform better than the existing two individual options? We present a novel framework that combines the LPV control with the α-stable workload modeling techniques. We use a set of traces from the real world for a proxy server to evaluate the proposed method. Results show that the proposed solution can (i) meet response time bounds without over provisioning; and (ii) outperform both the predictive-model-based provisioning and the feedback control that adopts a simple workload model.

5.2 Problem Formulation

We consider managing CPU frequency to provide response time guarantees for requests in the context of Internet servers. As a first step, we consider a single queue served by a single server, whose CPU frequency can be tuned dynamically, e.g.,

using the dynamic voltage/frequency scaling (DVS) mechanism which is allowed by most processors today. Noting that the CPU frequency is closely related to power consumption and electricity cost associated with each server, the DVS scheme allows energy saving while still serving requests at a lower CPU speed thus leading to no or less performance degradation. The design goal is to dynamically determine a minimal CPU frequency that can meet target response time.

In the previous Chapter 4 a solution to CPU allocation for multiple servers multiple applications is presented, where the problem was decomposed into two subproblems. The first subproblem is to determine a minimal *aggregate CPU frequency* for each application that can meet response time guarantee as though there is a single server per application running at this frequency. Then the second subproblem is to solve an online server allocation problem, which determines the number of servers allocated to each application and the CPU frequency of each individual server such that the aggregate frequency obtained from the first subproblem is provided. This chapter focuses on providing new modeling and control design framework to solve the first subproblem. Rather than the aggregate CPU frequency, we use CPU frequency directly in the sequel.

5.3 Control-Oriented Web Server Modeling

In this section, we integrate the $\alpha-$stable self-similar workload characterization with the linear parameter varying technique to build a control-oriented model. The review of α-stable modeling in Section 5.3.1 and Section 5.3.2 is mainly in terms of recent results from [59, 60].

5.3.1 An Alpha-Stable Model for Self-similar Workloads

Definition α-stable distribution [61]: A random variable X is called to follow an α-stable distribution, referred to as $X \sim S^{\alpha}_{\sigma,\chi,m}$, if there are parameters, $0 < \alpha \leq 2$, $\sigma \geq 0$, $-1 \leq \chi \leq 1$, and $m \in \mathbb{R}$ such that its characteristic function has the following form:

$$\Phi(\omega) = E(e^{j\omega X}) = \exp\{jm\omega - |\sigma\omega|^{\alpha}[1 - j\chi sgn(\omega)\theta(\omega, \alpha)]\} \tag{5.1}$$

with $\theta(\omega, \alpha)$ defined as $\theta(\omega, \alpha) = \tan(\alpha\pi/2)$ if $\alpha \neq 1$, otherwise $\theta(\omega, \alpha) = -2/\pi \ln|\omega|$.

Note that the characteristic function in Eq.5.1 is determined by four parameters. The exponent α specifies which family the distribution belongs to, and it is an indicator of the bursty level of the distribution. The distribution for $\alpha = 2$ is the Gaussian distribution. The parameter χ denotes the skewness, and m and σ denote the mean (location) and the scale (dispersion) of the distribution. There are various algorithms [62, 63, 64] on parameter estimation for α-stable modeling of web traffic. In this chapter, we are particularly interested in one research direction based on the linear fractional stable noise (LFSN) [59]. An LFSN $N_{\alpha,H}$ is a stationary process dependent on the characteristic exponent α of the α-stable distribution and the Hurst parameter H, which indicates the level of self-similarity and satisfies $0 < H < 1$. The LFSN has long-range dependence if the Hurst parameter $H > 1/\alpha$. In addition, an LFSN sequence $N_{\alpha,H}$ corresponds to an α-stable measure with zero mean, scale parameter $\sigma = 1$ and some fixed skewness χ satisfying $-1 \leq \chi \leq 1$ ($\chi = 1$ is often used to represent a totally positively skewed distribution) [61]. That is, the LFSN sequence $N(i) \sim S^{\alpha}_{1,\chi,0}$. Further, the LFSN is the incremental process of a linear fractional stable motion (LFSM) process. It is worth pointing out that the fractional Brownian motion is a special case of the LFSM with the index $\alpha = 2$, and the fractional Gaussian noise is a special case of the LFSN with $\alpha = 2$.

Consider applying the $\alpha-$stable self-similar model to a self-similar arrival process, then the number of requests arrived in the kth sampling period (with duration Δt) can be computed as follows,

$$\Lambda(k) = (c \cdot N_{\alpha,H}(k) + \lambda) \cdot \Delta t \tag{5.2}$$

where the parameter λ denotes the mean arrival rate. The scaling factor c is defined as the ratio of the scale parameter of the traffic process to the scale parameter of the LFSN. Noting that the scale parameter of the LFSN $N_{\alpha,H}$ is 1, the scaling factor c is actually the scale/dispersion around the mean of the traffic. In summary, the α-stable model in Eq.5.2 for request arrivals depends on four parameters: α denoting the level of burstiness, H denoting the level of self-similarity, λ representing the mean arrival rate, and c denoting the scale.

5.3.2 Stochastic Envelope for an Alpha-Stable Self-similar Distribution

The notion of stochastic envelope provides the flexibility for capacity allocation to satisfy different percentiles of service demand. A stochastic envelope for the arrivals in Eq.5.2 is computed as [60],

$$\hat{\Lambda} = (\lambda + \beta_\Lambda \cdot \sigma_\Lambda) \cdot \Delta t \tag{5.3}$$

The skew parameter β_A is uniquely determined by the probability that the number of arrivals Λ surpasses its approximation $\hat{\Lambda}$, where the probability is set to a pre-specified small risk ϵ, i.e.,

$$P\left\{\Lambda > \hat{\Lambda}\right\} = P\left\{\frac{\Lambda - \lambda \cdot \Delta t}{\sigma_\Lambda \cdot \Delta t} > \beta_\Lambda\right\} = \epsilon \tag{5.4}$$

In Eq.5.3, the parameters λ and σ_A denote the mean and scale parameters of the arrival process. Comparing Eq.5.2 and Eq.5.3, and denoting the scale parameter of the LFSN $N_{\alpha,H}$ by σ_N, then $\sigma_A = c \cdot \sigma_N$, where c is the scaling factor in Eq.5.2. Therefore, the arrival parameters λ and σ_A can be obtained from the α-stable modeling of the request arrival process. Note that there is no closed-form mathematical formula to give the direct relation from the risk ϵ to the parameter β_A in Eq.5.4, in terms of the α-stable distribution. However, given a set of ϵ's, the corresponding β_A's can be numerically generated and tabulated using the α-stable distribution model. If we also model the service demand (or file size) of requests using the α-stable self-similar model, similar results as Eq.5.2 and Eq.5.3 can be obtained as well, and the details are given in the next section.

5.3.3 A Linear Parameter Varying Fluid Model

A general discrete-time linear parameter varying system, denoted by $\Gamma(p)$, can be represented by a difference equation as follows:

$$x(k+1) = F(p(k))x(k) + G(p(k))u(k) \tag{5.5}$$

where $x(k)$ and $x(k+1)$ are often referred to as state variables in control systems theory, and $u(k)$ is called the control variable. $p(k)$ is a time-varying exogenous parameter, which is often referred to as the scheduling variable.

Consider sampling intervals with sampling time Δt, and let $N(k)$ denote the number of jobs in the system at the beginning of the kth sampling interval, then the number of jobs in the system at the beginning of the $(k+1)th$ sampling interval can be calculated as,

$$N(k+1) = \{N(k) + \Lambda(k) - D(k)\}^+ \tag{5.6}$$

where $\Lambda(k)$ denotes the number of arrivals in the kth period, and $D(k)$ denotes the number of departures (finished requests) in the kth period. The notation $\{\cdot\}^+ = \max(\cdot, 0)$. Eq. 5.6 indicates that the number of jobs in the system at the end of Δt is the sum of the initial number of jobs in the system and the number of arrivals minus the number of requests serviced in this duration.

Considering a self-similar arrival process, by Eq.5.3, the number of arrivals is approximated by its ϵ-stochastic envelope,

$$\Lambda(k) \approx (\lambda(k) + \beta_\Lambda(k) \cdot \sigma_\Lambda(k)) \cdot \Delta t \tag{5.7}$$

with β_Λ defined in Eq.5.4. We further consider that the request file size (service demand), denoted by $s(k)$, is a random variable approximated by the α-stable self-similar model,

$$s(k) \approx \nu(k) + \beta_s(k) \cdot \sigma_s(k) \tag{5.8}$$

where $\nu(k)$ and $\sigma_s(k)$ are the mean and scale parameters of $s(k)$ respectively. The parameter β_s is determined similarly as in Eq.5.4,

$$P\{s(k) > (\nu(k) + \beta_s(k) \cdot \sigma_s(k))\} = P\left\{\frac{s(k) - \nu(k)}{\sigma_s(k)} > \beta_s(k)\right\} = \epsilon \tag{5.9}$$

Note that different risk parameters ϵ can be chosen for approximation of $\Lambda(k)$ and $s(k)$. Let $u(k)$ denote the allocated capacity and assume that the service time is inversely proportional to the allocated capacity, then the number of requests

being served is approximated as,

$$D(k) = \frac{\Delta t}{s(k)/u(k)} \approx \frac{u(k) \cdot \Delta t}{\nu(k) + \beta_s(k) \cdot \sigma_s(k)} \tag{5.10}$$

Consequently, by plugging Eq.5.7 and Eq.5.10 into Eq.5.6, and linearizing the $\{\cdot\}^+$ (the linearization is just used in the control design but the simulation will include the $\{\cdot\}^+$), we have,

$$N(k+1) = N(k) + \left\{ (\lambda(k) + \beta_\Lambda(k) \cdot \sigma_\Lambda(k)) - \frac{u(k)}{\nu(k) + \beta_s(k) \cdot \sigma_s(k)} \right\} \cdot \Delta t \tag{5.11}$$

By the Little's law, the average response time $T(k)$ in the kth sampling time interval can be approximated by

$$T(k) = \frac{N(k)}{\lambda(k)} \tag{5.12}$$

Note that for a given pre-specified risk level ϵ for stochastic envelope, Eqs. (5.11-5.12) are parameterized by workload-characterizing parameters $\lambda(k)$, $\sigma_\Lambda(k)$, $\nu(k)$, and $\sigma_s(k)$ (where the parameters β_Λ and β_s are determined once ϵ is fixed), which can be obtained through α-stable self-similar models for the request arrival and service demand. Define $p = \{\lambda, \sigma_\Lambda, \nu, \sigma_s\}$, and compare Eqs. 5.11-5.12 with Eq. 5.5, we see that the system model consists of Eqs. 5.11-5.12 is a linear parameter varying system with parameters of arrival and service demand defined as scheduling variables. It should also be noted that the scheduling variables ($\lambda(k)$, $\sigma_\Lambda(k)$, $\nu(k)$, and $\sigma_s(k)$) may not be updated in the same time-granularity as the control $u(k)$ and mean response time $T(k)$, due to the minimum time required for on-line identifying distribution parameters of the $\alpha-$stable models.

5.4 LPV Controller Design

Corresponding to the LPV model in Eqs. 5.11-5.12 which is scheduled by the α-stable distribution parameters, this section designs an load-parameter scheduled LPV controller that generates time-varying aggregate capacity $u(k)$ such that the response time $T(k)$ will meet its target value $\bar{T}$ with minimal allocated capacity.

The LPV control is often classified as a generalized gain-scheduling control, but it provides provable stability guarantee for the closed-loop system, which can not be provided by traditional gain-scheduling control. Another advantage of the LPV control is that it does not require *a priori* knowledge of the scheduling parameters but their online measurements.

A block diagram of a general LPV control is illustrated in Fig.5.1, and we briefly summarize the LPV control as follows [45]. An LPV model $\Gamma_{aug}(p)$ can be viewed as a linear-time-invariant model Γ_{aug} (the subscript aug will be explained in next paragraph), which is scheduled in a specific way by the parameter p . The right figure in Fig.5.1(b) shows that the scheduling parameter p enters the model $\Gamma_{aug}(p)$ in a so-called linear-fractional-transformation (LFT) way [45]. An LPV model $\Gamma_{aug}(p)$ which is affine dependent on the scheduling variable p is a special case of the LPV models that have LFT dependence on the scheduling variables. For an LPV model $\Gamma_{aug}(p)$, an LPV control designs a scheduling-variable parameterized controller $K(p)$ such that the closed-loop system is stabilized for all admissible parameter trajectories p [45]. Note that in Fig.5.1(b), $K(p)$ is a linear-time-invariant controller K scheduled by p in the exactly same way as the LPV model $\Gamma_{aug}(p)$ is scheduled by p. Besides stability, the controller also assures that the norm of the transfer function from w to z, denoted by T_{zw} (which is an input-output mapping in frequency domain) is less than a specified level γ, that is $\|T_{zw}\| \leq \gamma$. Commonly used norms $\|\cdot\|$ include H_∞, H_2, and L_1 norms. Off-the-shelf LPV control algorithms can be applied to generate the control law K, and the online controller $K(p)$ is implemented by scheduling K with parameters p online.

Next we explain how the augmented model Γ_{aug} relates to the original system model Γ and what the input/output variables w, u, z, y in Fig.5.1(b) mean in English. The augmented model Γ_{aug} is a combination of Γ and a set of filters, which are often referred to as weighting functions. There are two input vectors and two output vectors of the augmented model Γ_{aug}: 1) the input w denotes any external input signals to the system, e.g., the reference signal, disturbance and sensor noise; 2) the input u denotes the control input to the system model, which is also the output of the controller K; 3) the output y is the output variable of the system model, which is measured and fed back into the controller K; 4) the controlled

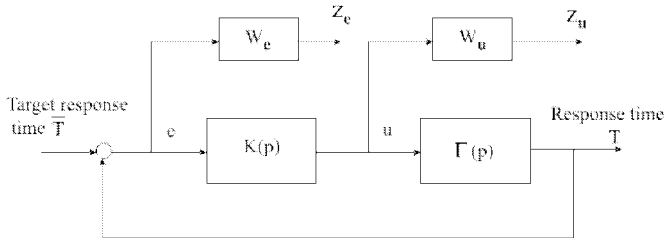
variable z, which consists of the set of variables of interest, e.g., the tracking error and control effort.

Given a set of external inputs w to the system, performance specifications of the controller can be fulfilled by specifying appropriate controlled variables z and designing the weighting functions. For example, in Fig.5.1(a), given a target response time $\bar{T}$ (which is an external input included in w), reducing the tracking error can be achieved by 1) including the weighted tracking error signal Z_e in the controlled variable z, 2) designing a filter W_e that specifies the frequency range where reducing tracking error should be emphasized, and 3) minimizing (or restricting) the norm of the transfer function from the input w to the output z. In Fig.5.1(a), we also include the weighted control input signal $Z_u = W_u \cdot u$ in the controlled output z, then limiting the control effort can be achieved by minimizing or reducing the norm of the transfer function from $\bar{T}$ to Z_u. By comparing Fig.5.1(a) and the left figure of Fig.5.1(b), the augmented Γ_{aug} is formed by combining the original model Γ and weighting functions W_e and W_u, with appropriate block-diagram transformations.

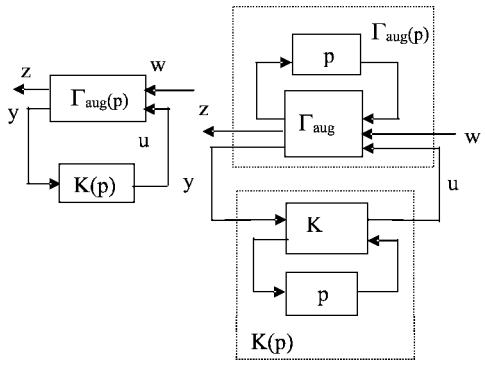
For the LPV model in Eqs. (5.11-5.12), we formulate an LPV-H_∞ control design problem as shown in Fig.5.1(a). Performance specifications on minimizing tracking error of meeting target response time and reducing control action are addressed by minimizing the H_∞ norm of the transfer function T_{zw} from the input signal w, which includes the target response time $\bar{T}$, to the controlled output variable z, which includes the frequency-weighted tracking error Z_e and the frequency-weighted control action Z_u. The algorithms for a general LPV-H_∞ can be found in [45], and off-the-shelf MATLAB LPV Robust Control toolbox is used in our controller design. The design parameters are the weighting functions W_e for the tracking error and W_u for the control input.

5.5 Simulation Evaluation

Our simulation uses real HTTP traces from the Web Caching project [50]. There are three traces in all and each corresponds to an individual web application for one-day duration. We denote the traces as Workload 1 to 3 (or WL 1-3) respectively. The arrival rates and file sizes of the three workloads can be found in Fig. 5.3. The



(a) H_∞ control design interconnection



(b) LFT parameter dependence structure

Figure 5.1. Block diagram for a general LPV control

modeling and control designs have been evaluated using a simulator built on top of the CSIM simulation package. It is assumed that the static http requests in WL 1-3 hit in the cache. Micro benchmarks have been run for requests with different file sizes to obtain request service times, and it is verified that the service time of a request is more or less proportional to the file size and inversely proportional to the operating CPU frequency [47]. In Sec. 5.5.1, we first study how the α-stable models fit the arrival and service demand of the three workloads and how the fitness relates to the self-similarity and burstiness of the workloads.

5.5.1 α-Stable Modeling of Arrival and Service Demand

The α-stable model for a workload is determined by four parameters: stability index, scale, mean and skewness (or the Hurst parameter). We have evaluated several popular α-stable-modeling algorithms in [62, 59, 63] to investigate their accuracy in identifying these parameters and easiness/complexity in implementation.

Here we use the quantile-based algorithm in [62, 59] for identifying parameters.

Evaluating α-stable self-similar modeling using CDFs Note that the notion of stochastic envelope originates from the cumulative distribution function (CDF) for a random variable. Consequently, we first would like to compare the CDFs constructed from measured request arrivals and service demands and the CDFs predicted by the identified α-stable models. The CDF plots given in Fig. 5.2 are obtained by applying a single (stationary) α-stable self-similar model to either the arrivals or file sizes of each workload of the 24hr duration, where the x-axis is normalized by the mean arrival rate or mean file size of each workload. We have the following observations:

- Majority region of each CDF curve corresponding to the α-stable model is located to the right of the CDF curve from the measurements. This implies that the α-stable-model estimated arrivals (or service demands) are more conservative than the actual measurements.

- The α-stable estimated CDFs and actual measurement based CDFs match quite well for high-percentiles (around $85\% - 95\%$), which are exactly the percentile range used for the stochastic-envelope based CPU allocation.

- There exist relatively large mismatches in CDFs between the α-stable models and the actual measurements corresponding to low-percentile values. Observations from workloads' time histories reveal that the mismatches could be due to that the request arrivals and file sizes exhibit multi modal probability density functions, especially there are a few requests that have much larger file sizes than the rest of requests, while the α-stable models have unimodal probability density functions. By trying to fit a multi modal distribution using a unimodal distribution, it is expected to see a bias (shift) of the distribution, and relatively larger mismatches at the low percentile region or extremely high percentile region (probability approaching one). However, since our CPU allocations target high-percentile range ($85\% - 95\%$), thus estimation errors in the low-percentile or extremely high-percentile ranges would not affect much our capacity allocation anyway.

Table 5.1. Correlation of load parameters and estimation errors between the α-stable-model predicted CDF and real-workload CDF

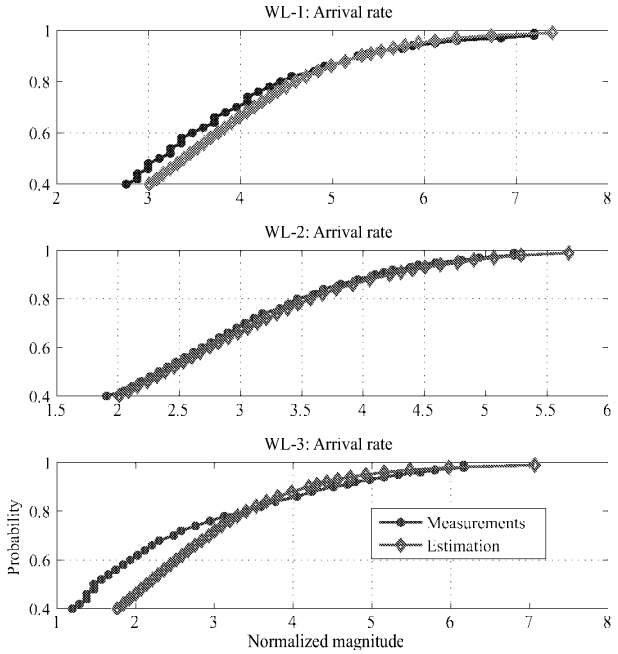
	Arrival					File size				
	Est. error	α	$1/\alpha$	σ	H	Est. error	α	$1/\alpha$	σ	H
WL-1	0.1184	1.9240	0.5196	1.6685	0.64	1.1490	1.2243	0.8168	0.3126	0.52
WL-2	0.0511	2	0.5	2.8302	0.66	0.5282	1.4986	0.6673	0.6738	0.5
WL-3	0.2543	1.8561	0.5388	2.1735	0.65	1.3505	1.1665	0.8573	0.3858	0.55

To further understand the fitness of α-stable models with respect to the real traces, in Table 5.1, we list for each workload, load parameters and the estimation (mismatch) error, which is calculated as the summation of the squared error over all sample points between the α-stable model predicted CDF and the measurement based CDF. We have observed that

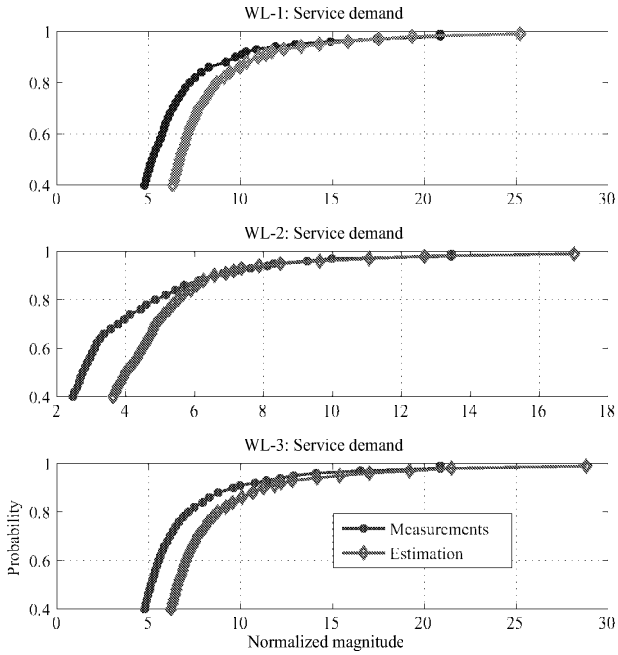
- In all workloads, both arrivals and file sizes have Hurst parameters $H \geq 0.5$ thus indicating self-similarity.
- Since $H > 1/\alpha$ indicates long-range dependence (5.3.1), the arrival processes of all workloads exhibit stronger long range dependence than their service demands since $H > 1/\alpha$ for all arrivals while $H < 1/\alpha$ for all file sizes. This could also explain why the estimation error for file sizes are much higher than that for the arrivals.
- Smaller estimation errors always correspond to higher α values (higher burstiness), especially for the arrival process of WL2 where α=2 implying that the arrival process follows a Guassian self-similar model. This is partially due to that the accuracy of α-stable modeling increases when the stability index α increases.
- Among the three workloads, WL-2 has the highest scale parameter in both arrivals and file sizes but has the lowest estimation error from α-stable modeling.

In summary, we speculate that with higher stability index (for burstiness) and higher Hurst parameter (for self-similarity), the α-stable self-similar models would provide better and less conservative estimations for workloads.

Evaluating the time-varying α-stable models We further study if a time-varying α-stable model would capture the time-varying behavior of load conditions.



(a) Arrival rate



(b) Service demand

Figure 5.2. CDF of α-stable model predicted workload

We divide the whole trace of each workload into period segments and let ΔT denote the duration of each such segment with k as period index. Then we identify the workload parameters $\lambda(k)$, $\sigma_\Lambda(k)$, $\nu(k)$, and $\sigma_s(k)$ corresponding to certain ϵ-stochastic envelopes, and generate the α-stable model predicted arrivals and file sizes in each period ΔT. Fig.5.3 shows the measured vs. α-stable model predicted arrivals and service demands of each workload. In these figures, *Mean Value* is computed using the measurements, while *A-S* corresponds to α-stable model predicted workload statistics for 85% envelope ($\epsilon = 0.15$). Both the *Mean Value* and A-S are calculated using $\Delta T = 1hr$. It can be seen that the time-varying α-stable models capture the load variations very well.

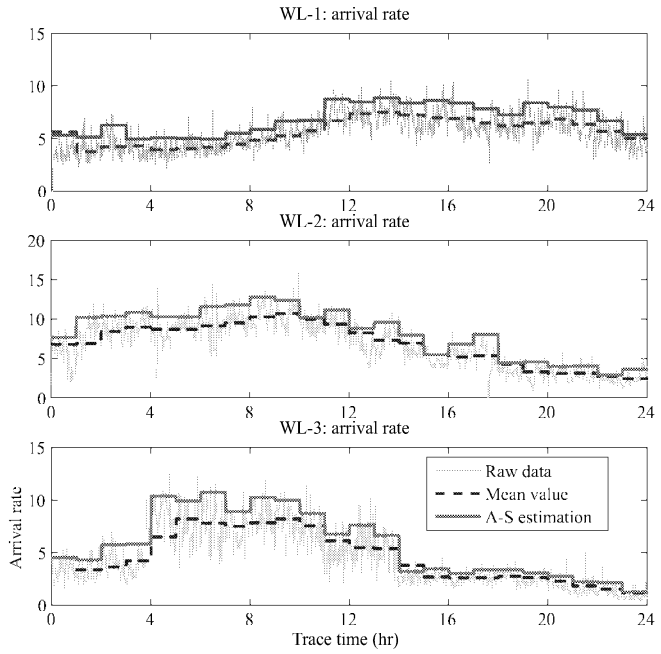
5.5.2 Control Design Results

In this section, we compare our α-stable-parameter scheduled LPV control design (with $\epsilon = 0.15$ stochastic envelopes for both arrivals and file sizes) to three other design approaches:

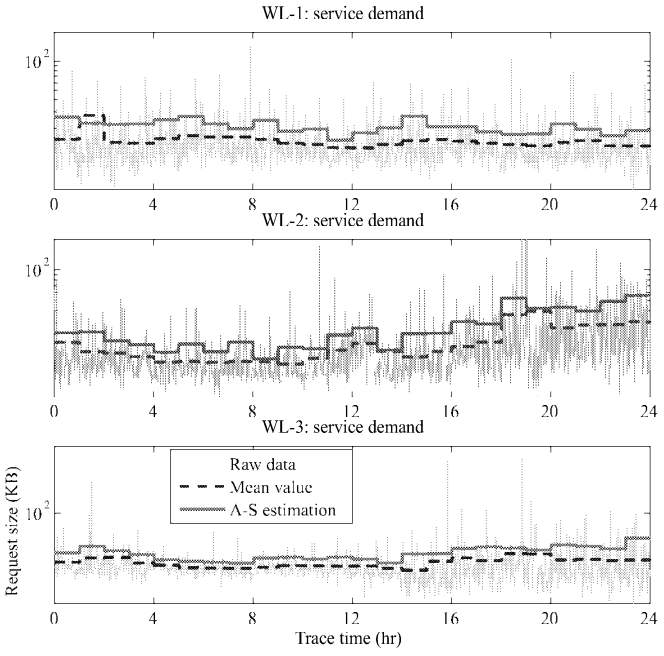
1. Alloc-by-request (Req): a nobrainer scheme that allocates CPU demanded by the total requests, which is the product of the arrival rate and file size in a sampling period.

2. OpenLoop-α-stable (OpLp): a pure α-stable model based CPU allocation scheme, which does not use any feedback thus can be considered as an open-loop design. The stochastic envelope used in this approach is chosen as $\epsilon = 0.15$. The algorithm is a modified version of that from [60] and is described below.

   An α-stable stochastic envoloping method is applied to solve the CPU allocation problem in the afore-mentiond OpLp experiments. The algorithm is based on [60], which considers a resource allocation problem with both buffer size and delay constraints (T). While the original problem considers a single-server queueing system of constant-service rate (C), and assumes a service policy where the requests exceeding the buffer size (B) are discarded, our formulation is different in several aspects: we study the infinite buffer size case; service rate is time varying, which makes the problem more chal-



(a) Arrival rate



(b) Service demand

Figure 5.3. Time-varying α-stable model predicted workload

lenging; we only consider the delay constraint. The following approximations are made in order to apply the reported resource allocation algorithm,

- Derives a buffer size constraint from time delay constraint using the Little's law to obtain its two-constraint equivalent. More specificly, we characterize the arrival process over each ΔT using α-stable models to get λ, then $B = \lambda * T$;
- Approximates the time varying service rate by its α-stable envolope value.

With these approximations the CPU allocation algorithm in the OpLp experiments is a straight-forward implementation of "Resource allocation with loss and delay constraints" in [60]. See the pseudo code below.

define *mean-arrival-rate* as λ
define *mean-service-demand* as S
define *buffer-size* as B
define *resource-allocation* as R
define *aggregate-CPU-allocation* as uk
B $= target\text{-}response\text{-}time * \lambda$
for **each** *allocation-interval*
get H, K, $a\sigma_1$ from the arrival process
get $S^{85\%}$ as the 85% service demand
R $= m + B^{(H-1)/H}(Ka)^{1/H}H(1-H)^{(1-H)/H}$
uk $= R * S^{85\%}$
end

3. LQ: a linear quadratic (LQ) controller which minimizes the weighted quadratic sum of response-time tracking error and CPU frequency. An ARX system-identification model is used.

Essentially, we compare performances among a design that allocates by demand with no modeling on either workload or dynamics (from CPU to response time), a design that uses detail workload modeling but no feedback, a design that uses

Table 5.2. Design results

WL1																
$(\Delta T, \Delta t)$	mean resp. time (sec)				resp. time var.				mean CPU (MHz)				CPU var.			
(min, sec)	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req
(10,10)	20.43	20.00	21.44	110.1	1610	1589	1587	3.6e4	14.14	16.10	15.68	12.93	88	94	13	87
(30,30)	20.08	19.79	34.84	74.62	1050	1532	3553	3.4e4	13.01	14.96	14.73	12.78	58	48	12	70
(60,60)	20.09	19.72	39.34	76.14	1011	1707	4818	2.5e4	13.22	16.01	14.58	12.55	42	38	11	44
(240,300)	20.25	22.62	61.04	78.61	1117	3730	9127	7907	14.23	20.62	14.43	12.07	17	89	12	13
WL2																
$(\Delta T, \Delta t)$	mean resp. time (sec)				resp. time var.				mean CPU (MHz)				CPU var.			
(min, sec)	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req
(10,10)	21.39	22.80	418.3	1962	1734	4254	7.7e5	3.4e6	18.50	30.56	21.77	15.51	213	264	64	162
(30,30)	21.13	25.01	372.7	592.3	1563	7654	5.7e5	5.6e5	18.25	19.82	24.35	15.68	150	65	81	117
(60,60)	21.17	25.54	211.9	205.1	1990	9023	2.0e5	6.9e4	19.69	21.23	26.90	15.57	140	59	100	85
(240,300)	26.03	411.9	31.29	292.5	6884	4.5e5	2.1e4	4.1e4	25.04	23.61	29.47	15.43	235	298	45	54
WL3																
$(\Delta T, \Delta t))$	mean resp. time (sec)				resp. time var.				mean CPU (MHz)				CPU var.			
(min, sec)	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req	LPV	LQ	OpLp	Req
(10,10)	18.40	18.17	210.8	24.21	4267	4303	1.1e5	7810	14.27	16.59	14.54	13.71	92	121	62	72
(30,30)	18.08	35.39	104.0	31.73	3482	1.0e4	5.1e4	1.1e4	15.70	16.03	15.61	13.27	101	119	68	47
(60,60)	18.13	104.3	120.2	40.33	4151	2.9e4	6.4e4	1.2e4	16.23	15.24	15.19	13.05	88	223	63	36
(240,300)	20.33	26.02	883.7	85.78	4462	1.2e4	2.1e6	1.9e4	16.02	19.99	14.71	12.81	27	74	33	23

feedback but no sophisticated workload modeling, and the proposed design that integrates LPV control and α-stable modeling.

The main metrics used for comparison here are average response time and average CPU frequency. The response time SLA is set to be 20s; with 10% slack, the design is considered to meet the response time SLA thus feasible if the mean response time is less than 22s. The mean CPU frequency will then be compared among feasible designs. Besides the mean statistics, Table 5.2 also lists variances of response time and CPU frequency to give an indication on reacting to transient overloads.

Note that the α-stable self-similar modeling and all design schemes depend on the time granularity of the sampling periods (denoted by Δt) during which the control is implemented and the response time and workload statistics are measured, and possibly the time intervals where the α-stable models are updated (denoted by ΔT). In Table 5.2, we include the results for different time granularities: $\Delta t =$ 10s, 30s, 60s, 300s, and for each Δt, the corresponding $\Delta T =$ 10min, 30min, 1hr, 4hr. All results in the Table are based on online implementation.

From Table 5.2, one can see that the LPV controller consistently meets the response time SLA (except for WL2 at $\Delta t = 300s$) and has lower mean CPU frequency than other feasible designs. The LPV controller also has much lower

variances of response time, and lower variances of CPU in most cases. Further, the LPV controller is very robust to different time granularities. Detail comparisons are given as follows:

- *LPV vs. LQ*: In terms of the mean response time, the best LQ controllers can meet target response time for both WL1 and WL3, and have slightly higher response time than the target value for WL2. Though the mean CPUs of the LQ controllers are only slightly higher than the LPV controllers, the best LQ design has at least 50% higher (and up to ten-times) variance of response time than the best LPV controller for each workload. This indicates that the LPV controllers adapt to load variations and handle transient overloads much better than the linear controllers.

- *LPV vs. OpLp*: The best design of OpLp meets the response time SLA for WL1, but its response time is far from the target value for both WL2 and WL3, while it uses comparable mean CPU frequency as the LPV controller. It is also observed that OpLp has much higher variance of response time ($\sim$100 times higher than the LPV) and is very sensitive to the sampling time. There is no consistent trend that smaller time granularities would provide better results or vice versa based on the simulations. The OpLp designs allocate CPU corresponding to certain stochastic envelope with no feedback, thus they ignore and do not react to spikes in workloads, while the LPV designs smooth out the effect of workload spikes on response time by using feedback.

- *LPV vs. Req*: The best Alloc-by-request design is close to meeting target response time for WL3, but none of them can satisfy the response time SLA for WL1 and WL2. We can see that the Alloc-by-request designs are not able to allocate enough CPU by just utilizing mean statistics of workloads. In addition, among all designs for all workloads, the Req designs have the highest (or 2nd highest) variances of response time.

5.6 Summary

It is widely aware of that Internet services are under self-similar workloads, which exhibit large variance in a wide range of time scales. The large variance in arrival rate and service demand have significant negative impact on Server's performance. As a result QoS specifications such as response time may not be able to guaranteed under these workloads using the workload characterization methods discussed in Chapter 4, due to under estimation. α-stable distributions are adopted to model Web server workloads and their characteristics such as arrival rate and service demand are estimated using stochastic envelopes based on α-stable modeling.

The results of this chapter has been published on the IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks 2007 [65], and submitted to Journal of Performance Evaluation [66].

Chapter 6

Stochastic Linear Parameter Varying Control for CPU Management of Internet Servers

In this chapter, we continue with the same application on CPU allocation, but adopt a probabilistic robust control approach. In particular, we present a stochastic LPV control that uses an LPV fluid model scheduled by randomly-distributed workload parameters and provides control solutions via solving a stochastic semi-definite program.

In the sequel we first give the motivations of introducing stochastic control design in Section 6.1. Stochastic LPV modeling of Internet servers is presented in Section 6.2. In Section 6.3, we present the stochastic robust LPV control, where the randomly distributed workload parameters are used as scheduling parameters, and the resulting probabilistic LPV control is solved via stochastic semi-definite programming. Simulation results using real Web traces are given in Section 6.4. Conclusions are drawn in the end.

6.1 Motivation for Stochastic Robust LPV Control

The stochastic approach to Internet Web-server dynamics modeling and performance control proposed in this chapter is mainly motivated by two reasons.

First, a workload is often characterized by two complementary distributions: (1) the request inter-arrival time and (2) the service demand distributions, which capture the workload intensity and its variability. Performance metrics are commonly specified in a statistical way as well. Consequently, probabilistic approaches to system modeling and control designs are needed.

Second, traditional performance management for server systems relies on worst-case estimates of load and resource availability; thus, server resources are often provisioned to meet peak demands. However, the worst-case resource demand is likely to be significantly higher than its normal usage. Thus provisioning for peak demands often implies underutilization of server resources most of time. Consequently, rather than over-provisioning for the worst-case load, performance management in today's Internet hosting center would limit rather than eliminate the risk of failing to meet demand, allocating to each application the minimal server resources needed for acceptable service quality, and leaving surplus resources to deploy elsewhere. This also motivates the need for probabilistic approaches for modeling system uncertainty and operating conditions.

Thirdly, the deterministic LPV control design employed in the previous chapters (Chapter 3-Chapter 5) requires affine dependence on scheduling parameters. As a result a third scheduling parameter has to be introduced to transform a nonlinear (bilinear) LPV model into an affine LPV model, which leads to design conservativeness. Gridding techniques are available for nonlinear deterministic LPV control design, however they are usually computationally intensive. The probabilistic LPV control design method introduced in this chapter is able to handle nonlinear parameter dependence and is computationally more effective.

6.2 Derivation of an LPV Web Server Model

In this section, we focus on analyzing transient dynamics of the system in Figure 2.5 to develop control-oriented models. Over the sampling interval $[(k-1)\Delta t, k\Delta t]$ define $N_s(k)$ as the queue length at the end of the interval k, and $T(k)$ as the mean response time following the definition of N_s and T in Chapter 2. Choose allocated CPU $u(k)$ as the control input, $N_s(k)$ as the state variable, and $T(k)$ as the system output.

6.2.1 A Nonlinear Time-Varying Model

We consider the situation where the traffic load seen by the system is high and the utilization is high, i.e., the server is serving requests all the time. Since the number of requests queued up during the time period of $[k\Delta t, (k+1)\Delta t]$ is the sum of the initial queue length at $k\Delta t$ and the number of arrival requests minus the number of requests serviced in this duration, we have

$$N_s(k+1) = \left\{ N_s(k) + n^k_{arriv} - n^k_{srved} \right\}^+ \tag{6.1}$$

where $\{\cdot\}^+ = max(\cdot, 0)$, enforcing queue length to be non-negative. When the traffic load is high and the system utilization is high, the number of arrivals and departures in a sampling period can be approximated by $\lambda(k)\Delta t$ and $\mu(k)\Delta t$, respectively; i.e.,

$$\lambda(k) = n^k_{arriv}/\Delta t, \mu(k) = n^k_{srved}/\Delta t \tag{6.2}$$

Also due to the high load and high utilization assumption for the system, the queue will never be empty thus the projection to the positive plane $\{\cdot\}^+$ in Eq. 6.1 can be removed, using A.2 and Eq. 6.2, Eq. 6.1 becomes,

$$N_s(k+1) = N_s(k) - \frac{\Delta t}{s(k)} u(k) + \lambda(k)\Delta t \tag{6.3}$$

Further, the average response time $T(k)$ in $[k\Delta t, (k+1)\Delta t]$ can be approximated by the sum of mean queueing delay $q^*(k)/X(k)$ and mean service time $1/\mu(k)$, where $q^*(k)$ denotes the average queue length in the k^{th} sampling period. That is,

$$T(k) = \frac{q^*(k)}{X(k)} + \frac{1}{\mu(k)} \tag{6.4}$$

For a stable open system, the throughput $X(k)$ can be approximated by the arrival rate $\lambda(k)$; while for a closed system, the throughput $X(k)$ is approximated by $\mu(k)$. By further approximating the average queue length $q^*(k)$ by $N_s(k)$ and absorbing the one request in service in the calculation of $N_s(k)$, we have for a open system,

$$T(k) = \frac{N_s(k)}{\lambda(k)} \tag{6.5}$$

or for a closed system,

$$T(k) = \frac{N_s(k)}{\mu(k)} = \frac{N_s(k)s(k)}{u(k)} \tag{6.6}$$

It should be noted that the system consisting of Eqs. 6.3 and 6.5 is a nonlinear time-varying system since the output variable $T(k)$ is inversely proportional to the control input $u(k)$.

6.2.2 A Linear Parameter Varying Model Derived Using Jacobian Linearization

An LPV model can be derived based on Jacobian linearization, i.e., linearizing Eqs. 6.3 & 6.6 around an equilibrium trajectory. By setting $N_s(k+1) = N_s(k)$ in 6.3 and using 6.6, the equilibrium trajectory is characterized by

$$\begin{aligned} \bar{\lambda}(m) = \quad & \bar{\mu}(m) = \frac{\bar{u}(m)}{\bar{s}(m)} \\ \bar{T} = \quad & \frac{N_s(m)\bar{s}(m)}{\bar{u}(m)} \end{aligned} \tag{6.7}$$

Define

$$N_s(k) = \bar{N}_s(m) + \hat{N}_s(k)T(k) = \bar{T}(m) + \hat{T}(k)u(k) = \bar{u}(m) + \hat{u}(k) \tag{6.8}$$

where we let $(\bar{N}_s(m), \bar{T}, \bar{u}(m))$ denote the (steady-state) operating condition in the m^{th} ΔT-time period ("-" represents steady-state condition); note that $\bar{T}$ does not depend on m since we consider the same target response time $\bar{T}$ across all time

periods.

Note that from Eq. 6.7, we have $\bar{u}(k) = \bar{\lambda}(k)\bar{s}(k)$ and $\bar{N}_s = \bar{T}\bar{u}(k)/\bar{s}(k) = \bar{T}\bar{\lambda}(k)$, where the "-" and index k are used to denote "steady-state" values in the k^{th} sampling interval. The target response time T is the same across all intervals. By 6.8, we derive the LPV model from Eqs. 6.3 & 6.6 as follows,

$$\begin{aligned} \hat{N}_s(k+1) = & \hat{N}_s(k) - \frac{\Delta t}{s(m)}\hat{u}(k) \\ \hat{T}(k) = & \frac{1}{\bar{\lambda}(m)}\hat{N}_s(k) - \frac{\bar{T}}{s(m)\bar{\lambda}(m)}\hat{u}(k) \end{aligned} \tag{6.9}$$

Equation 6.9 is a linear parameter varying system with scheduling variables $\delta_1(m) := 1/\bar{\lambda}(m)$ and $\delta_2(m) := 1/\bar{s}(m)$, where $\bar{\lambda}(m)$ and $\bar{s}(m)$ can be approximated by the mean arrival rate $\lambda(k)$ and file size $s(m)$ in the m^{th} ΔT interval. ΔT in general does not have to be equal to the sampling interval. Since the traffic load varies in a much slower time scale (usually in minutes for a Web server application) compared to the system dynamics (where the response time of a Web server is expected to be seconds), the LPV system 6.9 is slow varying, which satisfies the conditions for LPV control design.

Meanwhile note that Eq. 6.9 has nonlinear (bilinear in particular) dependence on scheduling variables. Existing deterministic LPV control designs have to use a gridding technique to partition the scheduling parameter space in order to solve any nonlinear-parameter-dependent LPV systems. Alternatively, approximations have to be made to transform such LPV systems into ones that have either affine or linear-fractional-transformation (LFT) dependence on scheduling variables. The probabilistic robust LPV control used in this chapter (details are given in Section 6.3) applies a stochastic gradient algorithm to solve an LPV control for the model 6.9 scheduled by workload arrival rates and service demand. Advantages of this design method is that it does not need the gridding technique, neither approximations of the functional dependence with respect to scheduling parameters.

6.3 Probabilistic Robust LPV Control

We formulate an $LPV - L_2$ control design problem as illustrated by Figure 6.1, where the performance specifications on minimizing tracking error of meeting target response time and reducing control action are addressed through the design

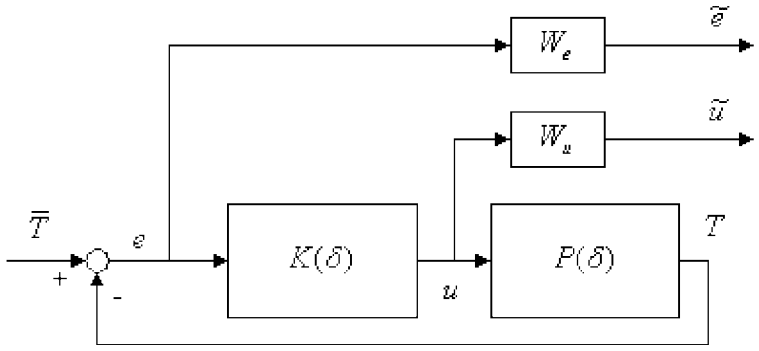


Figure 6.1. Robust control system interconnection

of weighting functions W_e and W_u, respectively. Low-pass filters are appended to both input and output channels of the original LPV plant. The cutoff bandwidth of the low-pass filters should be much higher than the feedback sampling frequency so that the system performance would not be affected. With a bit abuse of notion, let $P(\delta)$ denote the LPV model that includes the original plant Eq. 6.9 as well as the input/output low-pass filters. The controller $K(\delta)$ is designed such that the closed-loop system is stabilized and the L_2 norm of the transfer function from the exogenous input (the reference response time $\bar{T}$) to the controlled variables (the weighted error signal $\tilde{e}$ and the weighted control signal $\tilde{u}$) is minimized, i.e.,

$$\left\| \begin{array}{c} W_e S \\ W_u K S \end{array} \right\|_2 \leq \kappa \tag{6.10}$$

with performance level κ, where S denotes the sensitivity transfer function.

Note that in the LPV system Eq. 6.9, scheduling parameters are defined as $\delta_1(k) := 1/\lambda(k)$, $\delta_2(k) := 1/s(k)$, i.e., the LPV model Eq. 6.9 has nonlinear (bilinear) dependence with respect to scheduling parameters. In next section, we apply the probabilistic robust LPV control from [67] to solve Eq. 6.10.

6.3.1 A Probabilistic Robust LPV Control design

The control design of a LPV system is often reduced to solving a set of parameter-dependent Linear Matrix Inequalities (LMI) [45]. In this section, we illustrate the probabilistic robust LPV control. The LPV system Eq. 6.9 together with input-

output low-pass filters and performance weighting functions W_e and W_u (as in Eq. 6.10 and Figure 6.1) forms an augmented LPV plant; by abuse of notation, we still refer the augmented LPV system as the LPV system. Further, for the case of implementation using the Matlab toolbox, we derive a continuous LPV model from the original discrete-time augmented LPV system and put it into a standard LPV plant form as follows:

$$\begin{bmatrix} \dot{N}_s \\ e \\ T \end{bmatrix} = \begin{bmatrix} A(\delta) & B_1(\delta) & B_2(\delta) \\ C_1(\delta) & D_{11}(\delta) & D_{12}(\delta) \\ C_2(\delta) & D_{21}(\delta) & 0 \end{bmatrix} \begin{bmatrix} N_s \\ d \\ u \end{bmatrix} \tag{6.11}$$

where d denotes the external input vector (in Figure 6.1, it includes the target response time $\bar{T}$), $e(k)$ is the controlled output vector (in Figure 6.1, it consists of the weighted tracking error between the measured response time and its reference value, and the weighted control signal). The system matrices $(A, B_1, B_2, C_1, C_2, D_{11}, D_{12}, D_{21})$ are defined accordingly.

The $LPV - L_2$ control problem (with L_2 norm $< K$) is solvable if and only if there exist $X = X^T$, $Y = Y^T$ that satisfy the following QMI/LMI, for $\gamma > 0$ and for all $\delta \in \Theta$ [67]:

$$\begin{aligned} P(X,\delta) = \; & A(\delta)X + XA^T(\delta) + XC_1^T(\delta)C_1(\delta)X \\ & +\kappa^{-2}B_1(\delta)B_1^T(\delta) - B_2(\delta)B_2^T(\delta) + \gamma I \\ \leq \; & 0 \end{aligned} \tag{6.12}$$

$$\begin{aligned} Q(Y,\delta) = \; & A^T(\delta)Y + YA(\delta) + YB_1(\delta)B_1^T(\delta)Y \\ & +\kappa^{-2}C_1^T(\delta)C_1(\delta) - C_2^T(\delta)C_2(\delta) + \gamma I \\ \leq \; & 0 \end{aligned} \tag{6.13}$$

$$R(X,Y) = -\begin{bmatrix} X & \kappa^{-1}I \\ \kappa^{-1}I & Y \end{bmatrix} \leq 0 \tag{6.14}$$

If Eqs. 6.12-6.14 hold, an LPV controller $(A_c(\delta), B_c(\delta), C_c(\delta))$ can be constructed in terms of the solution (X, Y) and online measurements of the scheduling parameter $\delta(k)$, i.e.,

$$\begin{aligned} A_c(\delta) = \;& A(\delta) - Y^{-1}C_2^T(\delta)C_2(\delta) - B_2(\delta)B_2^T(\delta)(X - \kappa^{-2}Y^{-1})^{-1} \\ & \kappa^{-2}Y^{-1}C_1^T(\delta)C_1(\delta) + \kappa^{-2}Y^{-1}(Q(Y,\delta) - \gamma I)(XY - \kappa^{-2}I)^{-1} \\ B_c(\delta) = \;& Y^{-1}C_2^T(\delta) \\ C_c(\delta) = \;& -B_2^T(\delta)(X - \kappa^{-2}Y^{-1})^{-1} \end{aligned} \tag{6.15}$$

Consequently, the LPV control design for Eq. 6.9 is reduced to solving the parameter-dependent QMI/LMI in Eqs. 6.12-6.14.

Define a matrix-valued function,

$$V(X,Y,\delta) = \begin{bmatrix} P(X,\delta) & 00 & \\ 0 & Q(Y,\delta) & 0 \\ 0 & 0 & R(X,Y) \end{bmatrix} \tag{6.16}$$

and a scalar function

$$\begin{aligned} J(X,Y,\delta) &= \|V^+(X,Y,\delta)\| \\ &= (\|P^+(X,\delta)\|^2 + \|Q^+(X,\delta)\|^2 + \|R^+(X,\delta)\|^2)^{1/2} \end{aligned} \tag{6.17}$$

where $(\cdot)^+$ denotes the projection to the semi-definite cone, e.g., $P^+ = U\Lambda^+U^T$, where U is the eigenvector of P and $\Lambda = diag\lambda_1, \lambda_2, \cdots$ contains all the eigenvalues of P, with $\Lambda^+ = diag\lambda_1^+, \lambda_2^+, \cdots$ and $\lambda_i^+ = max(\lambda_i, 0)$.

By treating the scheduling parameter δ as random samples from $\delta \in \Theta$ following certain statistical distributions (note that for the LPV Web server model Eq. 6.9, the scheduling parameter δ is $\delta = [\delta_1\delta_2]^T$, where $\delta_1(k) := 1/\lambda(k)$ and $\delta_2(k) := 1/s(k)$), we define the following stochastic optimization problem

$$\min_{X,Y} E_\delta(J(X,Y,\delta)) \tag{6.18}$$

where $E_\delta(\cdot)$ denotes the expected value with respect to random samples $\delta \in \Theta$. There are different approaches for optimizing the probabilistic cost function defined in Eq. 6.18. One approach is to first approximate the expected value $E_\delta(\cdot)$ by finite number of Monte Carlo simulations, e.g., the estimated expected value based on n_s Monte Carlo simulations is given by $\hat{E}_\delta(J) = 1/n_s \sum_{j=1}^{n_s} J(X,Y,\delta^j)$, where

$\delta^1, \delta^2, \cdots, \delta^{n_s}$ are random samples drawn from $\delta \in \Theta$. It is then followed by solving a deterministic optimization problem to minimize the estimated expected value $\hat{E}_\delta(J)$ for which a gradient algorithm can be applied since the expected value of J (which is defined in terms of QMI/LMI and their projections to the semi-definite cone) is a convex function. For a given approximation error bound η and confidence level α, the required number of Monte Carlo samples, which is a function of η and α can be explicitly derived using the statistical learning theory [68], guarantees that the minimizing solution to $\hat{E}_\delta(J)$ approaches the optimal solution to $E_\delta(J)$ with approximation error less than δ with probability $1-\alpha$.

In this chapter, we adopt an iterative stochastic gradient algorithm from [67] to search (X, Y) for minimizing $E_\delta(J(X, Y, \delta))$. In this algorithm, rather than approximating the expected value by its Monte Carlo estimate and then solving a deterministic minimization problem, the j^{th} iteration of (X, Y) is directly updated using the gradient $\partial_{X,Y} J$ evaluated at a random sample $\delta^j \in \Theta$ - the so-called stochastic gradient. The details of the stochastic gradient algorithm are given below.

6.3.1.1 Stochastic Gradient Algorithm

The subgradients of the convex function $J(X, Y, \delta)$ are derived as follows,

If $J(X, Y, \delta) > 0$,

$$\begin{aligned} \partial_X J(X,Y,\delta) = \; & \tfrac{P^\dagger(X,\delta)}{J(X,Y,\delta)} \left(A(\delta) + X C_1^T(\delta) C_1(\delta)\right) \\ & + \left(A^T(\delta) + C_1^T(\delta) C_1(\delta) X\right) \tfrac{P^+(X,\delta)}{J(X,Y,\delta)} - [I0] \tfrac{R^+(X,Y)}{J(X,Y,\delta)} [I0]^T \end{aligned} \tag{6.19}$$

$$\begin{aligned} \partial_Y J(X,Y,\delta) = \; & \tfrac{Q^\dagger(Y,\delta)}{J(X,Y,\delta)} \left(A^T(\delta) + Y B_1(\delta) B_1^T(\delta)\right) \\ & + \left(A(\delta) + B_1(\delta) B_1^T(\delta) Y\right) \tfrac{Q^\dagger(Y,\delta)}{J(X,Y,\delta)} - [0I] \tfrac{R^\dagger(X,Y)}{J(X,Y,\delta)} [0I]^T \end{aligned} \tag{6.20}$$

Otherwise,

$$\partial_X J(X, Y, \delta) = \partial_Y J(X, Y, \delta) = 0 \tag{6.21}$$

The iterative stochastic gradient algorithm starts with an initial condition X^0 and Y^0, and then implements the following in the j^{th} iteration:

If $J(X^j, Y^j, \delta^j) > 0$, define

$$\nu(X^j, Y^j, \delta^j) = \sqrt{\|\partial_X J(X^j, Y^j, \delta^j)\|^2} \tag{6.22}$$

$$\mu^j = \frac{J(X^j, Y^j, \delta^j)}{\nu(X^j, Y^j, \delta^j)} + \epsilon, \tag{6.23}$$

with ϵ as a design parameter for improving convergence of the algorithm, then

$$X^{j+1} = X^j - \mu^j \frac{\partial_X J(X^j, Y^j, \delta^j)}{\nu(X^j, Y^j, \delta^j)} \tag{6.24}$$

$$Y^{j+1} = Y^j - \mu^j \frac{\partial_Y J(X^j, Y^j, \delta^j)}{\nu(X^j, Y^j, \delta^j)} \tag{6.25}$$

Otherwise ($J(X^j, Y^j, \delta^j) = 0$),

$$X^{j+1} = X^j, Y^{j+1} = Y^j. \tag{6.26}$$

6.4 Simulation Results & Performance Analysis

Modeling and control designs in this chapter are evaluated by a simulator built on top of the CSIM, which is a commonly used commercial simulation library. The same real HTTP traces from the Web Caching Project group [50] used in Chapter 4 are studied in this Chapter (see Figure 4.4). The target response time $\bar{T}$ is set to be 20 seconds. We choose a 2-minute sampling period for implementing the LPV model in Section 6.2. However, for the purpose of better visibility of presentation in a limited space, simulation results in the rest of the chapter are plotted using a larger sampling period.

The stochastic robust LPV control law has been implemented in the simulator package CSIM. Then the control performance is evaluated using the real traces. Table 6.1 shows average response time and average CPU frequency corresponding to each of the three workloads, and the time-varying histories of the response time and CPU are plotted in Figures 6.2-6.4. For comparison purpose, we also list the results of a linear-quadratic (LQ) controller in Table 6.1 and Figures 6.2-6.4. The linear-quadratic controller is designed using a linear model, which is obtained by linearizing Eqs. 6.3 & 6.6 at 85-percentile of workload arrival and service demand, and a quadratic cost function which penalizes the response time tracking error and

Table 6.1. Mean performance statistics for three workloads

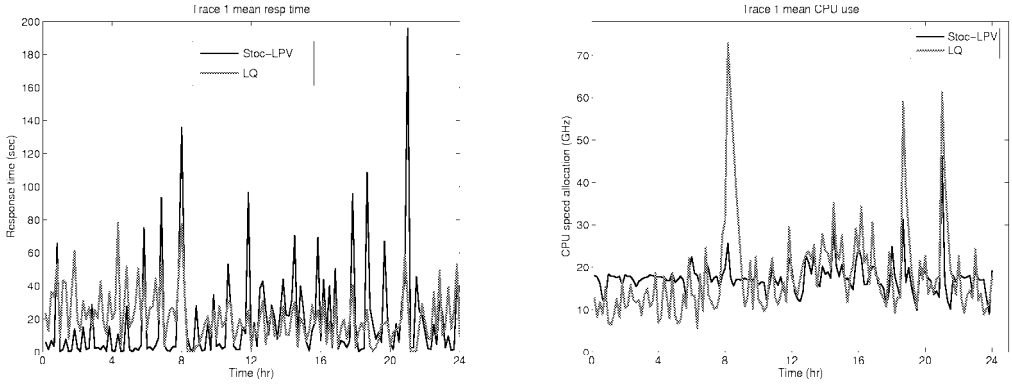
	Stochastic LPV		LQ	
	Resp.time (sec)	CPU freq. (GHz)	Resp.time (sec)	CPU freq. (GHz)
WL-1	18.89	17.45	20.82	18.23
WL-2	20.56	24.73	19.88	28.04
WL-3	21.68	15.95	16.85	37.49

CPU frequency.

Note that the target response time $\bar{T}$ is set to 20s, for performance evaluation, we take 10% slackness and specify that if the average response time for a design is less than 22s, the response time SLA is met and the design is feasible. It is observed that the proposed LPV control designs meet the response time SLAs for all three workloads. Among the three workloads, the mean CPU frequency for the WL2 is higher than the other two workloads, which is consistent with the observation that WL2 has substantially higher variance in both arrival and service demand. It is worth pointing out that a single LPV model and a single LPV control law can adapt to different workloads (WL2 changes significantly from WL1 and WL3) by scheduling the model and controller design with workload parameters online. For the LQ designs however we have to tune the controller gains for each workload in order to meet the target response time.

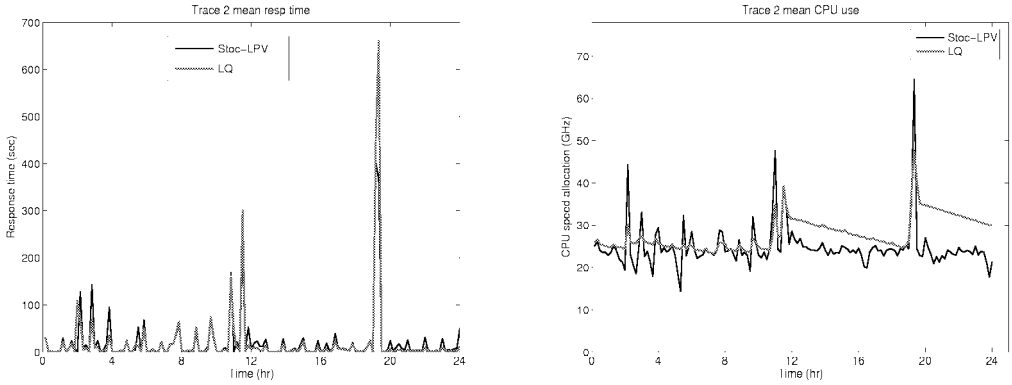
By comparing the mean performance statistics (mean response time and CPU frequency of the whole 24hr duration) between the probabilistic LPV control and those of the LQ control (designed at 85-percentile load condition), we can see that LPV control designs allocate much less CPU resources than LQ counterparts for all the workloads. The significant difference in CPU allocation for WL-3 is due to saturation, which we believe can be reduced by ad-hoc anti-windup techniques. For WL-2, probabilistic LPV design saves about 15% CPU and keeps a much smaller worst-case response time and less variance of response time as well. For WL-1, peak response time of LQ control is less than $100sec$, compared to $180sec$ of probabilistic LPV control. Notice however the peak response time of LQ for Wl-2 is almost $700sec$ and above $400sec$ for WL-3. This indicates that deterministic LQ control often allocates excessive CPU resources to try to keep response time as small as possible (WL-1), in some cases leading to controller saturation (WL-3). In contrast, probabilistic LPV control seems to have achieved a balance between violation of target response time and CPU allocation, with worst-case response time ranges

from $200 - 400sec$ accross workloads (note that WL-1/2/3 are independent real traces).



(a) Response time (b) CPU frequency

Figure 6.2. Time history results: Stochastic LPV versus LQ, WL-1

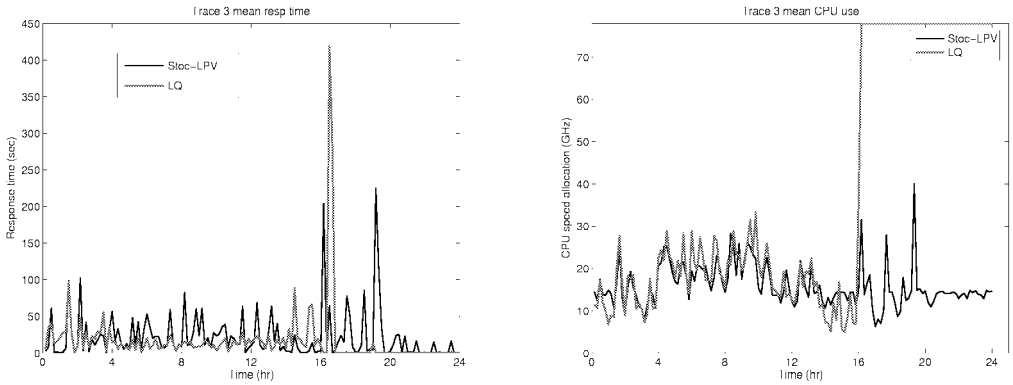


(a) Response time (b) CPU frequency

Figure 6.3. Time history results: Stochastic LPV versus LQ, WL-2

6.5 Summary

In this chapter, we present a stochastic LPV controller for the CPU management for an Internet Web server to meet response time SLA. Essentially, based on an LPV Web server model which is scheduled by randomly distributed workload arrival and service-demand parameters, an LPV control is obtained by applying a stochastic gradient algorithm which utilizes the sample distributions of workload



(a) Response time (b) CPU frequency

Figure 6.4. Time history results: Stochastic LPV versus LQ, WL-3

parameters. Through evaluation using real Web traces, the resulting stochastic LPV control shows better performance results than a linear quadratic controller designed at high-percentile load conditions. Future work will include investigating different formulations of the stochastic programming problem in deriving the stochastic LPV controller to achieve an explicit trade off between the risk of not meeting performance SLA and efficient resource usage. The results of this chapter have been accepted by the Conference on Decision and Control 2007 [69].

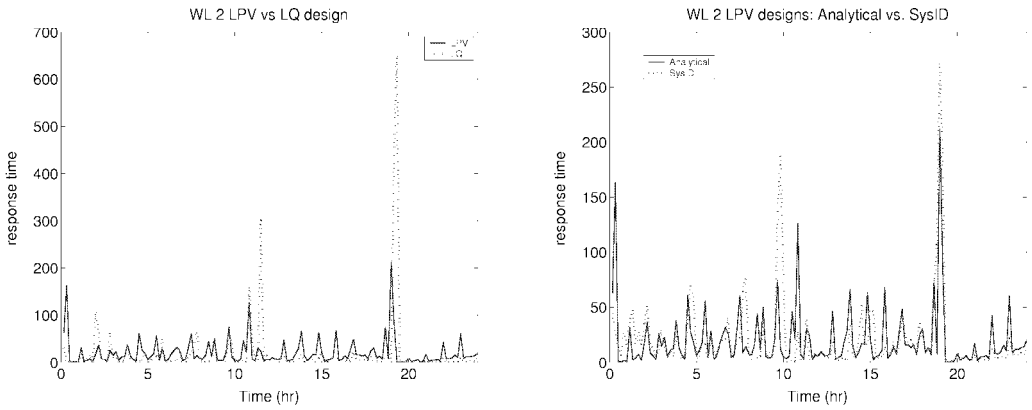
Chapter 7

A Request Level Approach: Preliminary Results and Direction for Future Research

7.1 Known Issues

It has been shown in Chapters 3∼6 that the control theory based approaches proposed in this thesis have advantages over open loop provisioning approaches based on queueing theory and on α-stable modeling. However these approaches are still subject to ineffectiveness in terms of request level response time. Note that the system models used for control designs are built on interval measurements. Thus inherently one can only enforces interval-wise performance measures, and request level response time violations are inevitable. Examine the experiment results Table 4.2, take the results of LPVARX, LPV-Equilbm, and Linear-Equilbm for Workload 2 for example. The average response times of these experiments are all very close to the target value (20sec). In the average sense, the response time meets the target value "perfectly". If one plots the time domain response times over 2-minute intervals as shown in Figure 7.1, it is observed that the response time oscillates around the target value wildly.

Further Figure 7.2 shows the request level response time of the same *LPV-Equilbm* experiment in fig:chapter-future-lpv-vs-lpvarx-ts. It shows that at the



(a) LPV-Equilbm vs. Linear-Equilbm (b) LPV-Equilbm vs. LPVARX

Figure 7.1. Response time histograms plotted at t_s=2min (source: Table 4.2)

request level the response time exhibits huge variance, with a maximum value of 525.7 seconds and minimum of 0.0004 seconds. In fact the output almost never settles at or approaches the target value. If the controlled system was a "conventional" mechanical system, it had lost control and became unstable already. In the field of response time control of Internet services, this is an existing phenomenon and has never been addressed before. Continued research effort is needed to solve this problem.

In the following sections a new direction for Internet services performance management that is based on request level optimization is discussed, and some preliminary results are presented.

7.2 A closer look of the FIFO queueing system at request level

Consider requests arrive at a single-server queue (Figure 7.2) at times A_1,A_2,A_3,$\cdots$, where A_n is the arrival time of the n^{th} request. D_1, D_2, D_3, $\cdots$, are the departure times of corresponding requests. The parameters $\lambda_i, i \geq 0$ are the arrival (birth) parameters, and $\mu_i, i \geq 0$ are the departure (death) parameters. The arrival and departure parameters follow some distributions $P_1(\lambda)$ and $P_2(\mu)$.

Define, $w_s(n)$, $w_q(n)$: the service time and waiting time components of the response time of a request,

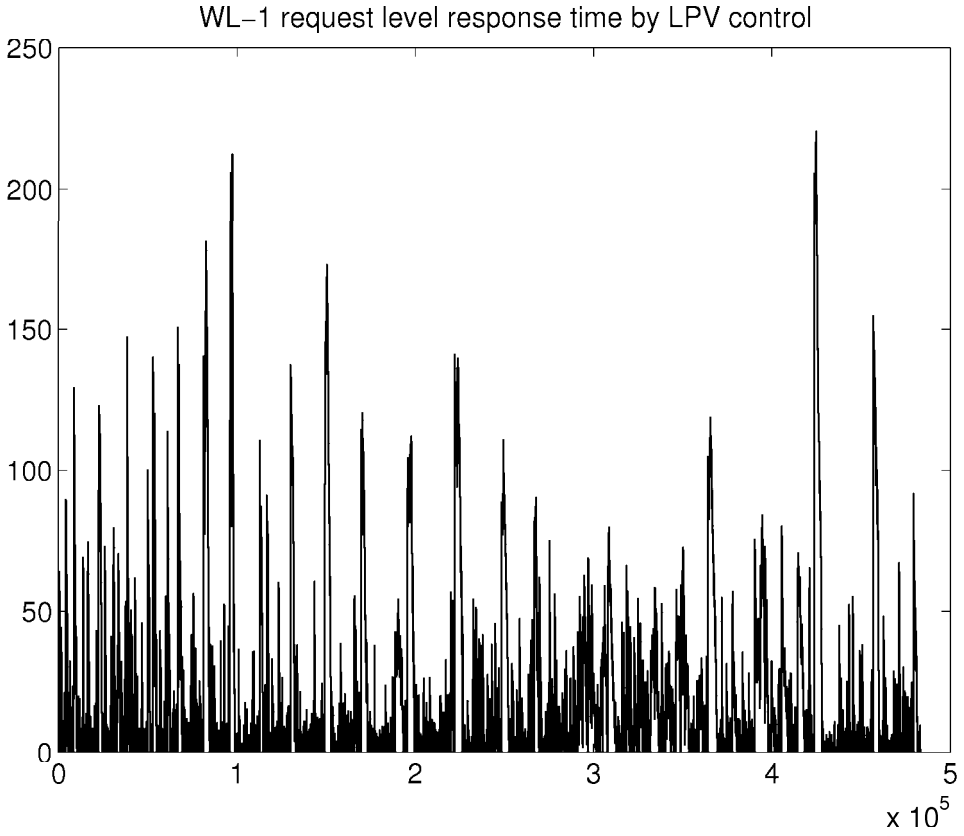


Figure 7.2. Big bursts of request level response time (control results from Table 4.2)

w: with a bit abuse of annotations, response time of individual requests,

k: sampling interval index,

$W(k)$: with a bit abuse of annotations, the mean response time of all served requests during interval k,

$N(k)$: total number of arrival requests during interval k,

$n(k)$: request arrival n during the interval k,

s: service demand of a request,

$u(k)$: CPU resource allocation,

$t_{iA}(n,1)$: inter-arrival time between the request n(k), and the request been served when n(k) enters the queue.

With these notations we now give definitions of queue length, service time, waiting time, and response time for each request, and define a mean response time measure over the studied interval.

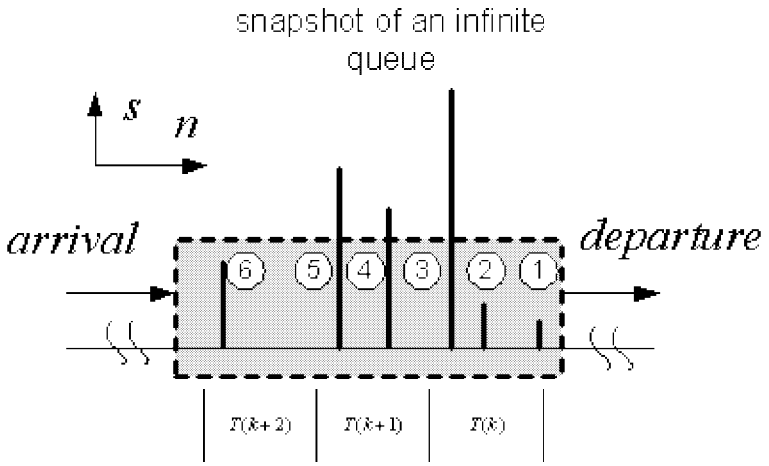


Figure 7.3. Schematics or a first-come-first-served queuing system

7.2.1 Instantaneous queue length $L(n)$

Instantaneous queue length of the n^{th} request is the number of requests in the queue station it observes at the instant it enters into the queue including itself, with the 1^{st} in the queue being the front-most one that is being serviced, and the L^{th} being the entering request n. The instantaneous request length is given inexplicitly as,

$$L(n) = \{L < n : D(n-L) > A(n) \textbf{ AND } D(n-L-1) <= A(n)\} \qquad (7.1)$$

Note that the instantaneous queue length is different from queue length in the usual sense, which is a statistical measure over a period of time based on a sufficiently large number of observations. $L(n)$ can be measured, or observed, for each request n upon arrival.

7.2.2 Completion interval of a request $k(t)$

For any given request, we try to predict its service completion time immediately when it enters the queue. The expected response time of each request can be determined using the above formulas, provided that the CPU allocation sequence $u(k)$ is known. It is straight forward to calculate the expected completion time

with the expected response time and the current clock time,

$$k_n = roof\left\{\left(t_{arrival} + w(n)\right)/T_s\right\} \tag{7.2}$$

where $t_{arrival}(n)$ is the tagged arrival time for each request.

7.2.3 Waiting time, service time, and response time

Consider the response time $w(n)$of request n that consists of two parts: a waiting time and a service time,

$$w(n) = w_q(n) + w_s(n) \tag{7.3}$$

while the service time part of each request is given as

$$w_s(n) := t_2 - t_1 \tag{7.4}$$

where t_1, t_2 are defined for each request as its service start time and departure time respectively, and should satisfy,

$$\int_{t_1}^{t_2} u\left(k(t)\right) dt = s(n) \tag{7.5}$$

where with a little abuse of notations, $k(t)$ is a function mapping continuous time t to the sampling interval index, and $u(k)$ the CPU resource allocation which is a piecewise constant function.

Let L_n denote the number of requests arrived before the n^{th} request that are still in the queue, and $(n,i), i = 1, \cdots, L_n$ the i^{th} request in the queue upon the arrival of request n. Denote $\hat{w}_s(n,i)$ as the service time of request (n,i) that contributes to $w_q(n)$. The sum of service time of all these terms uniquely determines $w_q(n)$, i.e.,

$$w_q(n) = \sum_{i=1}^{L_n - 1} \hat{w}_s(n,i) \tag{7.6}$$

Note that when request n enters the queue, the first request may have been partially served. This makes $\hat{w}_s(n,1)$ different from other terms in that it is the

remaining service time of the first request,

$$\begin{aligned} \hat{w}_s(n,i) &:= \quad w_s(n-L+i-1); \qquad i \neq 1; \\ \hat{w}_s(n,1) &:= \quad t_2 - t_1 = D_{n-L} - A_n; \quad i = 1. \end{aligned} \tag{7.7}$$

For each request its service time is determined by its size and the CPU allocation. The waiting time is more entangled which involves the service times of a finite number of previous requests, its arrival time, and the waiting time of a particular request (the currently served request). Essentially waiting time depends on the birth parameters, request sizes, and CPU allocation.

A web server queueing system can be considered as a dynamic system with arrival parameters and request sizes as exogenous input, CPU allocation as control action, and response time as output. The above formulation symbolically describes such queueing systems as time varying nonlinear dynamic models. Generally one needs to have the perfect a priori knowledge of the birth and death parameters to quantitatively study such a queueing system.

7.2.4 Mean response time

Denote $N(k)$ as the number requests served during interval k, the average response time during interval k is defined as,

$$W(k) := \frac{\sum_{n=1}^{N(k)} w(n)}{N(k)} \tag{7.8}$$

substitute 7.3 into 7.8,

$$W(k) := \frac{\sum_{n=1}^{N(k)} \{w_q(n) + w_s(n)\}}{N(k)} \tag{7.9}$$

According to 7.6,

$$W(k) := \frac{\sum_{n=1}^{N(k)} \left\{ \sum_{i=1}^{L(n)-1} \hat{w}_s(n,i) \right\}}{N(k)} + \frac{\sum_{n=1}^{N(k)} w_s(n)}{N(k)} \tag{7.10}$$

Substitute 7.7 into 7.10,

$$W(k) = \frac{\sum_{n=1}^{N(k)} \left\{ \sum_{i=2}^{L(n)-1} w_s(n-L+i-1) \right\} + D_{n-L} - A_n}{N(k)} + \frac{\sum_{n=1}^{N(k)} w_s(n)}{N(k)} \quad (7.11)$$

The response time during an interval k is defined as the average service time and waiting time of all served requests during that interval. The waiting time of a request n can be calculated from the service time of some requests, as well as the arrival time of the request itself, and departure time of the request being served when request n arrives. Because the request size, arrival time, and departure time can all be measured, the response time of all requests in the queue can be determined given the service rate.

When target response time is much longer than sampling interval, in the ideal situation the actual response time will be consistently much longer than sampling time. This means that at any time instance, it will take multiple sampling intervals to finish the requests in the queue, i.e. all finished requests during the upcoming several intervals are already in the queue at the given time instance. Thus from Eq. 7.11 the mean response time of several future intervals are "accurately" predictable, given the service rate trajectory.

7.3 Performance Management of Internet Service

The fact that future response time is predictable (over limited horizon) when target response time is much longer than sampling time indicates that Eq. 7.11 can be used to allocate CPU resources optimally to meet target response time. Another advantage is that it is possible to avoid excessive response time due to arrival bursts and extremely large requests, when these requests become foreseeable when they are in the queue.

7.3.1 Response Time Guarantee

In a response time guarantee problem the only concern is to enforce mean response time to the target value. The idea is that at the end of each interval, start with an

initial service rate and calculate the mean response time during the next interval using Eq. 7.11. The predicted mean response time is used to adjust the service rate iteratively, until the predicted mean response time is close enough to the target value. The desirable service rate during the next interval can then be set accordingly. In the ideal situation there are always enough waiting requests such that the actual response time during the next interval can be accurately predicted.

7.3.2 Resource Management Optimization

In the resource management problem the objective of performance control is to provide response time guarantee, while keep resource allocation at minimum. Define the following cost function for a typical performance control problem,

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \phi \left(W(t+j|t) - \bar{W}(t+j)\right)^2 + \sum_{j=1}^{N_u} \theta u(t+j-1)^2 \qquad (7.12)$$

where N_1 and N_2 are the minimum and maximum cost horizons and N_u is the control horizon, which does not necessarily have to coincide with the maximum horizon. The meaning of N_1 and N_2 are rather intuitive. They mark the limits of the instants in which it is desirable for the output to follow the reference. The objective is to compute the future control effort sequence $u(k)$ in such a way that the future response time $W(k)$ sequence is driven close to the target response time, while maintaining a minimum resource allocation. This is achieved by designing the control sequence that minimizes the above cost function.

7.4 Preliminary Results

To illustrate the effectiveness of the proposed concept a response time guarantee algorithm is implemented in this section. In this algorithm CPU frequency is provisioned based on Eq. 7.11. The only purpose is to maintain the target response time for all the requests in the queue, based on the snapshot at each T_s instance.

The same workloads WL-1/2/3 and simulator used in earlier chapters are used again in this study, and three different scenarios are investigated for comparison,

- **With Perfect Knowledge**: with perfect request level information of the workload, the response time of the next interval can be *accurately* predicted. This is the ideal case and indicates the upper bound of possible results.
- **With Pure Observation**: predicts the next interval mean response time simply based on observation of the queue. Knowledge of incoming requests beyond the current time is unavailable. This shows the performance simply based on the proposed method without any workload modeling.
- **With α-stable Modeling**: an extension based on the second scenario. When there are not enough waiting requests in the queue to predict the mean response time during the next interval, *virtual* requests generated from an α-stable model of the workload are used to compensate (stuff) the queue.

Figure 7.4 shows the request level response time of WL-1 in comparison with that of the results as in Chapter 4. The scheduling results is observation based and the sampling interval is 5 seconds. The mean response time of the scheduling method result is 17.71 seconds, which is quite close to that of the control based results. At the request level however the scheduling results are much more consistent and almost always settles at the target value (20 seconds). In this experiment only 427 out of 483838 requests (0.09%) have response times bigger than 50 seconds, compared to 12.43% of the LPV control based results.

Table 7.1$\sim$7.3 show performance measures in terms of mean response time and mean CPU allocation of the three scenarios of the request level scheduling algorithm. Note that as mentioned before the objective in the implemented scheduler is to guarantee that mean response time over the next interval is no larger than the target value (20 seconds) subject to CPU resource limitation. Thus the scheduler is inevitably conservative. It should be mentioned that it is straight-forward to implement more complex scheduler so that it is less greedy.

It can be observed that for all workloads (WL-1/2/3) the target response time are met regardless of the sampling interval used. As sampling interval shrinks, the performance improves in terms of CPU allocation. In general observation based scheduling performs better than α-stable model based scheduling for smaller sampling interval. When sampling interval is much smaller than target response time, e.g. 5 seconds, observation based scheduling almost achieves the same performance

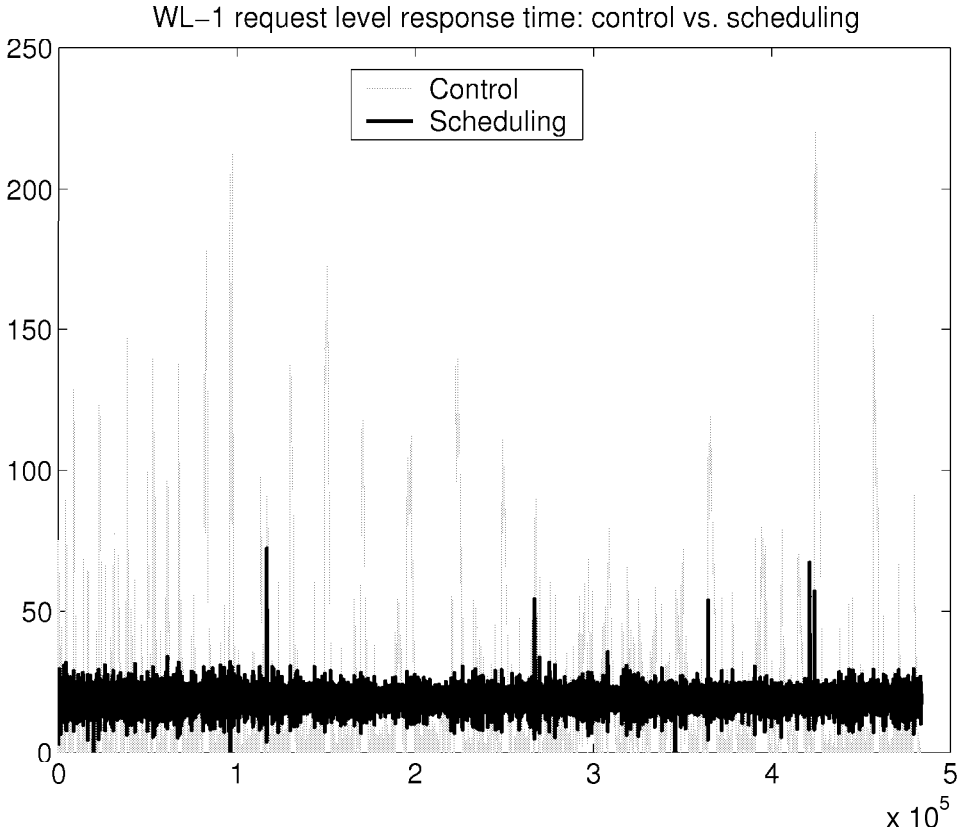


Figure 7.4. Request level response time: Control vs. Scheduling (control results from Table 4.2)

as the scheduling assuming full knowledge. This is quite intuitive, as when sampling interval shrinks the queue is long enough for the server to service during the coming interval, which essentially equals full knowledge.

Figures 7.5∼7.7 plot the response time trajectories of the three scenarios at sampling intervals (5, 20, 120) seconds over an 10 minutes window for workload WL-1/2/3 respectively. For all workloads it can be observed that over the time observation based scheduling matches full knowledge scheduling very well when sampling interval is very small compared to target response time, for example Δt = 5 seconds. As sampling interval becomes bigger, observation degrades. When $\Delta t = 120$ seconds no response time guarantee can be achieved for any of the three workloads. On the other hand α-stable modeling based scheduling achieves same level performance as observation based scheduling at short sampling intervals; at longer sampling intervals, it is still able to provide response time guarantee. Thus

Δt (sec)	Full Knowledge		With Observation		With α-stable	
	RT	CPU	RT	CPU	RT	CPU
5	17.48	10.93	17.71	10.93	17.03	11.01
10	18.30	10.95	18.51	10.95	17.23	11.29
20	18.59	10.98	17.87	11.43	15.43	12.14
30	18.42	11.06	15.20	13.68	13.91	13.31
60	18.10	11.39	18.27	38.81	12.2	19.72
120	17.49	12.04	36.82	39.49	15.17	26.58

Table 7.1. Response time guarantee using request level scheduling, WL-1

Δt (sec)	Full Knowledge		With Observation		With α-stable	
	RT	CPU	RT	CPU	RT	CPU
5	17.23	15.05	17.52	15.05	16.48	15.17
10	18.15	15.06	18.57	15.08	16.49	15.56
20	18.56	15.16	18.10	15.56	14.20	16.81
30	18.51	15.27	15.47	18.55	12.44	19.13
60	18.29	15.79	19.86	38.16	10.96	25.70
120	17.82	17.12	37.91	39.11	13.84	31.58

Table 7.2. Response time guarantee using request level scheduling, WL-2

the α-stable prediction based scheduling alone or an integrated scheduling based on both observation and α-stable modeling is able to provide response time guarantee for a wide range of sampling intervals. This shows the effectiveness of the proposed scheduling algorithm.

Two points need to be made. In real Internet applications it is not always possible to change CPU allocation very frequently, i.e. to use very short sampling intervals. One of the reason is that changing resource allocation configurations too frequently is not desirable due to hardware limitations. Thus an effective

Δt (sec)	Full Knowledge		With Observation		With α-stable	
	RT	CPU	RT	CPU	RT	CPU
5	16.83	10.43	17.02	10.43	16.21	10.86
10	17.68	10.45	17.77	10.44	16.58	11.17
20	18.08	10.48	17.06	10.75	15.64	11.64
30	17.89	10.57	14.78	13.71	14.59	12.63
60	17.70	10.92	17.59	34.29	14.30	18.58
120	17.31	11.44	34.87	38.94	21.71	25.72

Table 7.3. Response time guarantee using request level scheduling, WL-3

scheduling algorithm acting in a longer time window is necessary. Secondly, variant α-stable modeling algorithms can be deployed in the scheduling to adapt to different levels of modeling conservativeness. In these experiments, it is observed that the α-stable based scheduling results are more conservative than observation based schedulings, in terms of actual response time. We believe that the conservativeness can be reduced or eliminated by applying some other α-stable modeling techniques. In fact this may prove to be a desirable characteristics of the proposed α-stable prediction based scheduling, as it provides the flexibility and capability to be able to adapt to different workload characteristics.

7.5 Suggested Research Directions

Based on the preliminary results presented in the previous section, it is our belief that the proposed scheduling algorithm is of great potential and worth further investigation. The fact that it works so well with the simulator with real traces motivates future research along this direction to make it applicable to real Internet services. Performance degradation is expected when the scheduling is applied to real servers. In this case we believe that feedback control techniques can be integrated into the scheduling to adapt to inaccuracy of the underlying assumptions of the simulator.

Note that one of the fundamental assumption for validity of the simulator itself is that service time is reciprocally proportional to request size, which may not be valid for many applications. The justification for the assumption is that the requests hit the cache. The rate that a request hit the cache depends on size of the cache (compared to that of the file set) and the caching algorithm. In production servers the rate can be anywhere from several percent to about 50 percent. Performance degradation is inevitable when the solutions proposed in Chapters 3$\sim$6 are applied to real servers.

One of the most appreciated future research effort in Internet service performance management is probably to achieve similar performances shown in earlier chapters in terms of response time and resource allocation on real Internet servers. The fact that the request level scheduling based on Eqs. 7.1$\sim$7.11 works so well given the relation from request size to service time indicates great potential of a

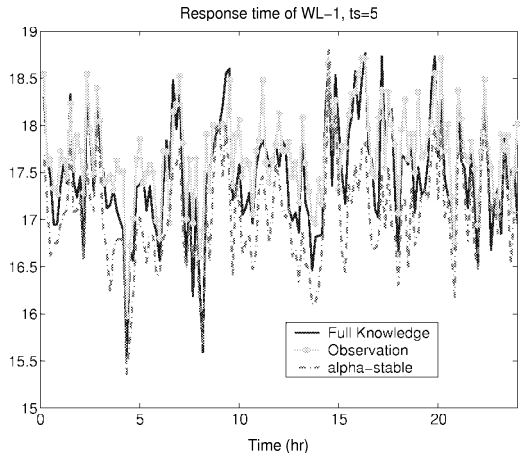
combined approach of the request level scheduling and a feedback mechanism that can adjust the estimation of the relation from request size to service time and/or the underlying α-stable model of the scheduler. The inherent nature of the feedback loop will provide certain robustness against estimation inaccuracy of such relationship.

In this framework there are at least two structures that can be considered, which are listed using the response time guarantee problem as an example,

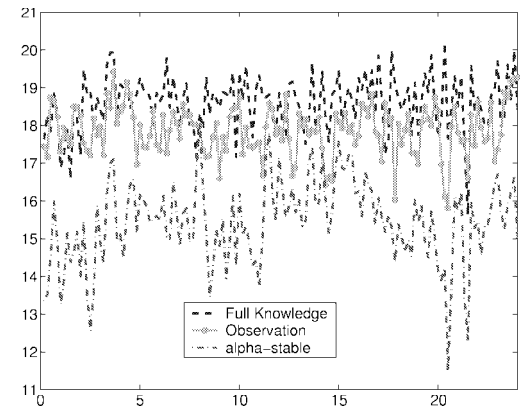
1. *Parallel Loops*: An inner loop as the request level scheduler that attempts to provide response time guarantee, which is compensated by an LPV feedback control outer loop to get the aggregate CPU allocation. See Figure 7.8(a).

2. *Complimentary Loops*: The request level scheduler depends on the feedback loop for the relation from request size to service time. Performance feedback information can also be used to improve the α-stable modeling algorithm. See Figure 7.8(b).

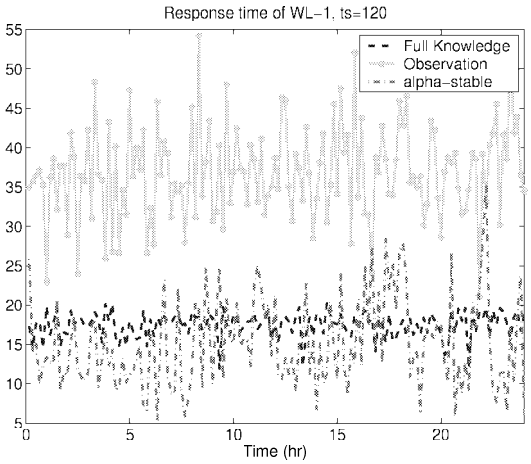
Note that for both structures the scheduler loop and the feedback loop can be working in a unsynchronized manner, meaning they may have different sampling interval. In a typical application the feedback loop kicks in less frequently.



(a) Sampling interval $\Delta t = 5$sec

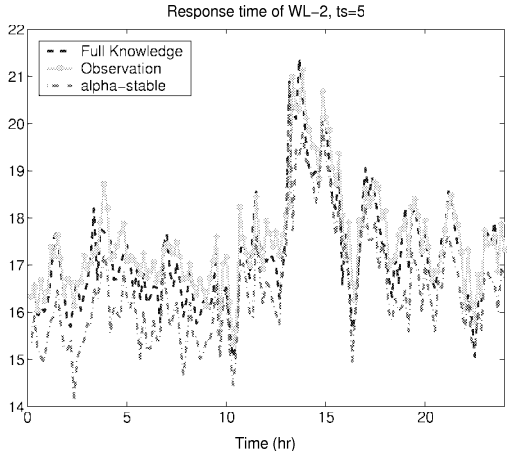


(b) Sampling interval $\Delta t = 20$sec

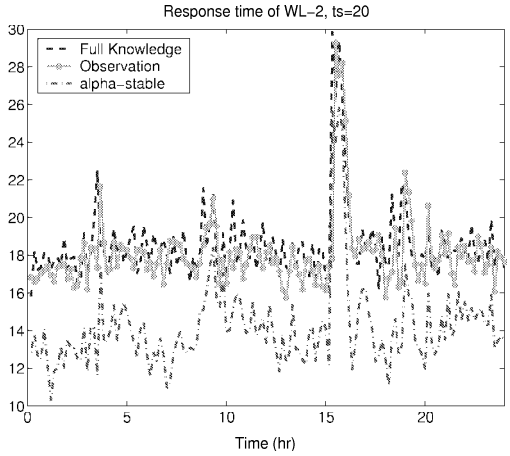


(c) Sampling interval $\Delta t = 120$sec

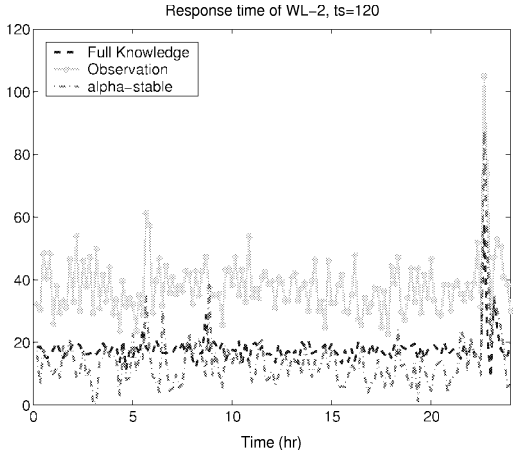
Figure 7.5. WL-1 time domain response time of the three scenarios



(a) Sampling interval $\Delta t = 5$sec

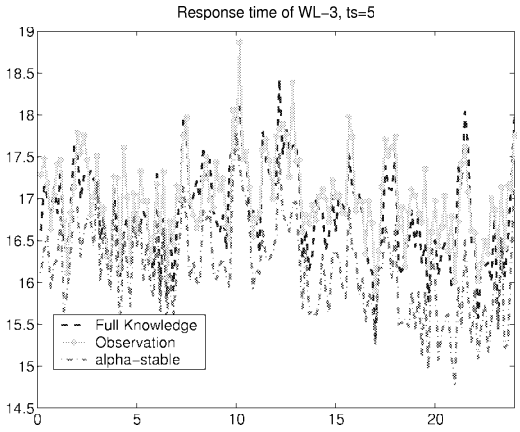


(b) Sampling interval $\Delta t = 20$sec

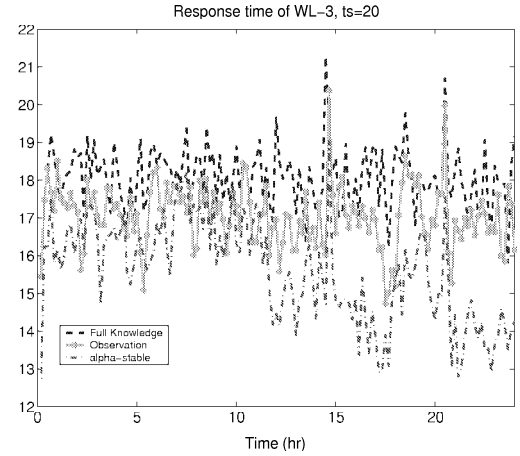


(c) Sampling interval $\Delta t = 120$sec

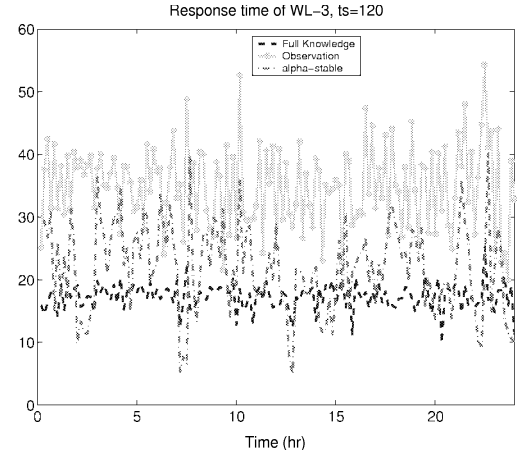
Figure 7.6. WL-2 time domain response time of the three scenarios



(a) Sampling interval $\Delta t = 5$sec

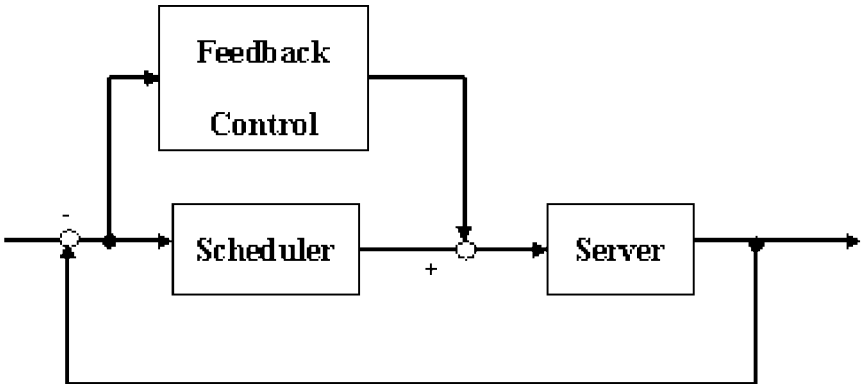


(b) Sampling interval $\Delta t = 20$sec

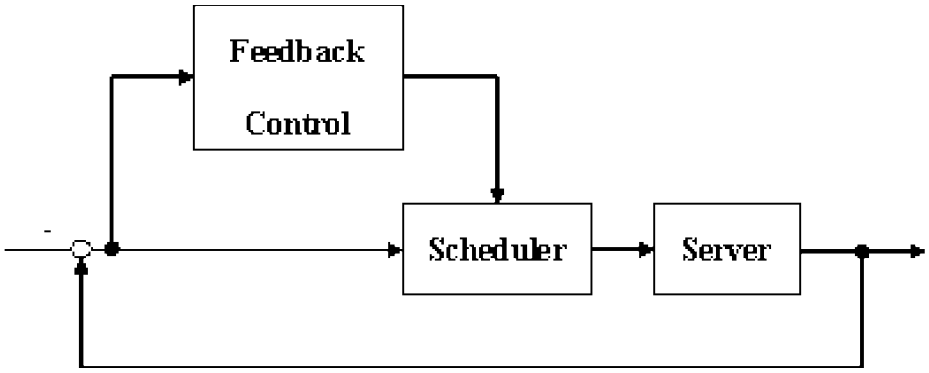


(c) Sampling interval $\Delta t = 120$sec

Figure 7.7. WL-3 time domain response time of the three scenarios



(a) Parallel Loop



(b) Complimentary Loops

Figure 7.8. Integrated feedback control and request level scheduling

Chapter 8

Summary and Conclusion

High performance server systems are now widely deployed as Internet has become part of our daily life. Performance management, which provides performance requirements guarantees while minimizing resource usage, becomes a critical issue considering the scale of deployment of these server systems these days. This dissertation is concerned with a linear parameter varying (LPV) modeling and control design framework for this problem.

In Chapter 3, we study linear time invariant (LTI) and LPV modeling and control design in admission control, where incoming service request can be rejected in order to meet specified target response time. For LPV control design we establish empirical Web server LPV models parameterized by the server's workload characteristics, i.e. workload intensity. Evaluations show that the identified LPV model is able to capture system dynamics better than ARX models. Based on the LPV model, controllers designed using standard LPV synthesis tools can adapt to workload changes by adjusting feedback gains. Simulation results show that target response time can be better enforced when the server is working in off-design workload conditions.

Despite the success of LPV control design based on empirical modeling techniques, a first principles LPV model is desirable for a more systematic modeling and control design framework. Based on a fluid single-queue-single-server model, Chapter 4 derives an analytical LPV model for Web servers, with queue length being the state variable, CPU frequency the input, and response time being the system output. The LPV model is explicitly parameterized by workload char-

acteristics: arrival rate $\frac{1}{\lambda}$, service demand $\frac{1}{s}$, and their multiplication $\frac{1}{\lambda \cdot s}$. The model is evaluated in simulation with real Web server traces and it captures the dynamic behavior of response time with respect to server CPU frequency, under three independent workloads WL-1/2/3. LPV control designs based on this models outperform those based on the same empirical system identification LPV models studied in Chapter 3.

In both chapters 3 and 4 straight forward workload characterizations to estimate the scheduling parameters of the LPV models. Either the mean value or a percentile value during the previous sampling interval is used as an estimation of the scheduling parameter during the current interval. When the workload has large variance however the estimation may not be accurate and fail to provide response time guarantee. Unfortunately it is widely agreed that workloads of Internet services are self-similar and has large variances in different scales, i.e. under-estimation may still occur even we take the mean values over a long sampling window. Chapter 5 presents a novel control-theoretic approach based on LPV techniques and workload characterization using α-stable-model based stochastic envelopes. The proposed approach parameterizes a control-oriented dynamic-system model and resulting controller using workload-distribution parameters. By further incorporating the α-stable modeling into the LPV control approach, the presented solutions not only allow system to adapt to workload changes, but also show great promise in handling workloads with large variances. The same CPU management problem as in Chapter 4 is revisited using the proposed method. Simulations show the strength of the proposed approach.

In Chapter 6 revisits the same CPU allocation problem in Chapter 4 with an emphasis on introducing a probabilistic approach to LPV control design. A major advantage of this control design is its capability of considering nonlinear parameter dependence of the analytical LPV model on $\frac{1}{\lambda}$ and $\frac{1}{s}$, without having to introduce an extra non-independent nonlinear (bilinear) parameter $\frac{1}{\lambda \cdot s}$. This reduces conservativeness of the LPV control design compared to the method used in Chapter 4.

The work of Chapters 3$\sim$6 of this thesis is solely conducted at the "aggregation" level, i.e. dynamical behaviors of the server system from input (CPU frequency or rejection ratio) to output (response time), and system state variable (queue

length) are only modeled with window based average values. It is perfectly fine if the QoS specifications are in terms of mean values as well. In practice many QoS requirements are specified in more detailed statistical language, e.g, *response time is less than 20 seconds for 85% of all requests.* It is difficult if not impossible at all to provide request level response time guarantee, since the relationship between mean response time and the response times of of individual requests during a given time window is not straightforward and is solely dependent on the distribution of workload characteristics. Investigations at the request level becomes neccessary in this case, which is recommended for future work.

Chapter 7 takes a request level approach to provide response time guarantee based on detailed analysis of requests response time components. A request level scheduling algorithm is proposed that can be used with α-stable modeling. Preliminary results in a response time guarantee problem show impressive request level performance over a wide range of sampling intervals. Potential future research directions are given in Chapter 7.

In summary, a major contribution of this dissertation is the formation of a self-sufficient LPV modeling and control design framework for performance management of Internet services. Compared to reported researches based on conventional Queueing analysis and linear modeling and control designs, the proposed method outperforms in terms of response time guarantee, variance of response time, and use of CPU resource allocation. The limitations of feedback control approaches are briefly discussed, which lead to the proposed request level scheduling algorithm. Preliminary results indicates great potential in performance management for Internet services.

Appendix A

Queueing Theory: A Brief Review

A.1 The Little's law

First we shall introduce an important theorem of queueing theory, which relates queue length with arrival rate, and average waiting time. Consider a queueing system where requests arrive at random time instants, get served according to a specified service discipline, receive a random amount of service and depart. Each request, depending on its size, consumes a certain amount of processing time and left the system (Figure 2.2). Then we have the following theorem,

Suppose that for a fixed sample path the limits λ, T exist and are finite. Then the limit of queue length q exists and is given by,

$$q = \lambda T, or T = \frac{q}{\lambda} \tag{A.1}$$

This is the well known Little's Law in queueing theory. It is essentially a deterministic result. Although the response times of individual requests depend on the service discipline, Eq. A.1 implies that the average response time is independent of service discipline since the quantities q, λ are independent of it. In fact the Little's law holds under very general conditions, regardless of service discipline, inter-arrival time distribution, and request size distributions, provided that the server is work conserving, i.e. there is no internally generated traffic.

A.2 Calculations of response time and queue length

M/M/1 queue: For a stable $M/M/1$ queue with workload intensity less than 1, i.e. $\rho < 1$, the expected number of requests in the system is given by

$$q = \frac{\rho}{1-\rho} \tag{A.2}$$

Thus as $\rho \to 1$ the expected number in the queue explodes to ∞. This is a manifestation of increasing congestion as $\rho \to 1$.

The steady state waiting time is given by

$$T = \frac{1}{\mu - \lambda} = \frac{1}{\lambda} \cdot \frac{\lambda}{\mu - \lambda} = \frac{1}{\lambda} \cdot q \tag{A.3}$$

G/G/1 queue: For a stable $G/G/1$ queue, i.e. $\rho < 1$, establishing a similar relation as Eq.A.2 is much more challenging. Many empirical formulas are proposed in existing literatures, and the following lists some commonly used ones [70],

$$q = \left(\frac{\rho^2(1+C_s^2)}{2-\rho+\rho C_s^2}\right)\left(\frac{\rho(2-\rho)C_a^2+\rho^2C_s^2}{2(1-\rho)}\right)+\rho \tag{A.4}$$

$$q = \left(\frac{\rho^2(1+C_s^2)}{1+\rho^2C_s^2}\right)\left(\frac{C_a^2+\rho^2C_s^2}{2(1-\rho)}\right)+\rho \tag{A.5}$$

$$q = \frac{\rho^2(C_a^2+C_s^2)}{2(1-\rho)} + \frac{(1-C_a^2)C_a^2\rho}{2} + \rho \tag{A.6}$$

where C_a^2 and C_s^2 denote squared coefficients of variation for inter-arrival time and service demand respectively. Formula Eq.A.6 can be rewritten in terms of arrival rate and service rate as,

$$q = \frac{\lambda}{\mu} + \frac{\lambda C_a^2(1-C_a^2)}{2\mu} + \frac{\lambda^2(C_a^2+C_s^2)}{2(1-\lambda/\mu)*\mu^2} \tag{A.7}$$

which provides a good approximation for queue length when the squared coefficient of variation $C_a^2 \leq 2$ while the approximation could get poor when C_a^2 is very large (the workload is highly varied). It is not very sensitive to server utilization, and it

provides quite accurate approximation even when the server utilization gets close to 1. Response time (T) can be obtained by applying the Little's law (Eq.A.1).

Bibliography

[1] Altman, E., T. Basar, and R. Srikant (1999) "Congestion control as a stochastic control problem with action delays," *Automatica*, **35**, pp. 1937 1950.

[2] Kousalya, G. and P. Narayanasamy (2005) "Dynamic Resource Management Framework For Wireless LAN," in *2nd IFIP International Conference on Wireless and Optical Communications Networks (WOCN)*, pp. 561 567.

[3] Hollot, C., V. Misiira, D. Towsley, and W. Gong (2001) "A control theoretic analysis of RED," in *Proceedings of INFOCOM*, pp. 1510–1519.

[4] Perkins, J. and R. Srikant (1999) "The Role of Queue Length Information in Congestion Control and Resource Pricing," in *Proceedings of the 38th IEEE Conference on Decision and Control*.

[5] Huang, M., P. Caines, and R. Malhame (2004) "Uplink Power Adjustment in Wireless Communication Systems: A Stochastic Control Analysis," *IEEE Transaction on Automatic Control*, **49**(10).

[6] Huang, M., P. Caines, and R. Malhamé (2001) "On a class of singular stochastic control problems arising in communications and thier viscosity solutions," in *Proceedings of IEEE Conference on Decision and Control*, pp. 1031–1036.

[7] Huang, M., R. Malhame, and P. Caines (2002) "Quality of Service Control for Wireless Systems: Minimum Power and Minimum Energy Solutions," in *Proceedings of American Control Conference*.

[8] Chaporkar, P. and S. Sarkar (2004) "Stochastic Control Techniques for Throughput Optimal Wireless Multicast," in *Proceedings of the 42nd IEEE Conference on Decision and Control*.

[9] Cucinotta, T., L. Palopoli, and L. Marzario (2004) "Stochastic Feedback-based Control of QoS in Soft Real-time Systems," .

[10] Misra, V., W. Gong, and D. Towsley (2000) "Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED," in *Proceedings of ACM/SIGCOMM*.

[11] Robertsson, A., B. Wittenmark, M. Kihl, and M. Andersson (2004) "Design and evaluation of load control in web server systems," in *Proceedings of American Control Conference*, pp. 1980 1985.

[12] Diao, Y., N. Gandhi, J. Hellerstein, S. Parekh, and D. M. Tilbury (2002) "Using MIMO feedback control to enforce policies for interrelated metrics with applications to the Apache web servers," in *Proceedings of Network Operations and Management*.

[13] Hellerstein, J., Y. Diao, and S. Parekh (2002) "A First Principles Approach to Constructing Transfer Functions for Admission Control in Computer Systems," in *Proceedings of IEEE Conference on Decision and Control*.

[14] Sha, L., X. Liu, Y. Lu, , and T. Abdelzaher (2002) "Queueing model based network server performance control," .

[15] Parekh, S., N. Gandhi, J. Hellerstein, D. M. Tilbury, and J. P. Bigus (2001) "Using control theory to achieve service level objectives in performance management," in *Proceedings of IEEE/IFIP Symposium on Integrated Network Management*, pp. 841 854.

[16] Abdelzaher, T., Y. Lu, R. Zhang, and D. Henriksson (2004) "Practical application of control theory to web service," in *Proceedings of American Control Conference*, pp. 1992–1997.

[17] Abdelzaher, T. and N. Bhatti (1999) "Web server QoS management by adaptive content delivery," in *Proceedings of Intl. Workshop on Quality of Service*, pp. 216–225.

[18] Abdelzaher, T. (2000) "An automated profiling subsystem for QoS-aware services," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, pp. 208–217.

[19] Abdelzaher, T., K. Shin, and N. Bhatti (2002) "Performance guarantees for Web server end-systems: A control theoretical approach," *IEEE Transaction on Parallel Distributed Systems*, pp. 80–96.

[20] Diao, Y., J. Hellerstein, and S. Parekh (2002) "Using fuzzy control to maximize profits in service level management," *IBM System Journal*, **41**(3), pp. 403 420.

[21] Diao, Y., N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury (2002) "MIMO control of an Apache Web server: Modeling and controller design," in *Proceedings of American Control Conference*, pp. 4922 4927.

[22] Gandhi, N., J. Hellerstein, S. Parekh, and D. Tibury (2001) "Managing the performance of Lotus Notes: a control theoretic approach," in *Proceedings of the Computer Measurement Group (CMG) International Conference.*

[23] Diao, Y., J. Hellerstein, and S. Parekh (2001) "A business-oriented approach to the design of feedback loops for performance management," in *Proceedings of the 12th Intl. Workshop on Distributed Systems: Operations & management.*

[24] Lu, C., A. Guillermo, and J. Wilks (2002) "Aqueduct: online data migration with performance guarantees," in *Proceedings of the USENIX Conference on File and Storage Technologies (FAST)*, pp. 219 230.

[25] Ko, B., K. Lee, K. Amiri, and S. Calo (2003) "Scalable Service Differentiation in a Shared Storage Cache," in *23rd International Conference on Distributed Computing Systems (ICDCS'03).*

[26] et. Al., M. K. (2004) *Triage: performance isolation and differentiation for storage systems, Tech. rep.*

[27] Lee, H., Y. Nam, K. Jung, S. Jung, and C. Park (2004) "Regulating I/O Performance of Shared Storage with a Control Theoretic Approach," .

[28] et. al., A. V. (2003) "A control-theoretic approach to dynamic voltage scheduling," in *Proceedings of the International conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES).*

[29] Joseph, R., D. Brooks, and M. Martonosi (2003) "Control techniques to eliminate voltage emergencies in high performance processors," in *Proceedings of the 9th Intl. Symposium on High-Performance Computer Architecture*, pp. 79–90.

[30] Diao, Y., C. Wu, J. Hellerstein, A. Storm, M. Surendra, S. Lightstone, S. Parekh, C. Garcia-Arellano, M. Carroll, L. Chiu, and J. Colaco (2005) "Comparative studies of load balancing with control and optimization techniques," in *Proceedings of American Control Conference*, pp. 1484–1490.

[31] Lu, C., J. Stankovic, G. Tao, and S. Son (1999) "The design and evaluation of a feedback EDF control scheduling algorithm," in *Proceedings of IEEE Real-Time Systems Symposium*, pp. 56–67.

[32] Steere, D., A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J.Walpole (1999) "A feedback-driven proportion allocator for real-rate scheduling," *Operating Systems Design and Implementation*, pp. 145–158.

[33] Stankovic, J., T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu (2001) "Feedback control scheduling in distributed systems," in *Proceedings of IEEE Real-Time Systems Symposium*, pp. 59–70.

[34] Lu, C., J. Stankovic, G. Tao, and S. Son (2002) "Feedback control real-time scheduling: framework, modeling, and algorithms," *Journal of Real-Time Systems, Special Issues on Control-Theoretical Approaches to Real-Time Computing*, **23**, pp. 85–126.

[35] Abdelzaher, T., J. A. Stankovic, C. Lu, R. Zhang, and Y. Lu (2003) "Feedback performance control in software services," *IEEE Control Systems Magazine*, **23**(3), pp. 74 90.

[36] Lu, C., T. Abdelzaher, J. Stankovic, and S. Son (2001) "A feedback control approach for guaranteeing relative delays in Web servers," in *IEEE Real-Time Technology and Applications Symposium (RTAS)*.

[37] Lu, Y., T. Abdelzaher, C. Lu, and G. Tao (2002) "An adaptive control framework for QoS guarantees and its application to differentiated caching services," in *Proceedings of Tenth International Workshop on Quality of Service*.

[38] Chandra, A., W. Gong, and P. Shenoy (2003) "Dynamic resource allocation for shared data centers using online measurements," in *IWQoS Proceedings*.

[39] Diao, Y., J. Hellerstein, and S. Parekh (2002) "Optimizing quality of service using fuzzy control," in *Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems Operations and Management*.

[40] Lu, C., X. Wang, and X. Koutsoukos (2004) "End-to-End utilization control in distributed real-time systems," in *Proceedings of 24th International Conference on Distributed Computing Systems (ICDCS'04)*.

[41] Bamieh, B. and L. Giarre (2002) "Identification of linear parameter varying models," *International Journal of Robust and Nonlinear Control*, (12), pp. 841–853.

[42] Mazzaro, C., B. Movsichoff, and P. Sanchez (1999) "Robust identification of linear parameter varying systems," in *Proceedings of American Control Conference*.

[43] Sznaier, M., C. Mazzaro, and T. Inanc (2000) "An LMI approach to control oriented identification of LPV systems," in *Proceedings of American Control Conference*.

[44] Lee, L. and K. Poola (1996) "Identification of linear parameter varying systems via LFTs," in *Proceedings of the IEEE Conference on Decision and Control*.

[45] Apkarian, P. and R. J. Adams (1998) "Advanced gain-scheduling techniques for uncertain systems," *IEEE Transactions on Control System Technology*, **6**(21-32).

[46] Wu, F. and S. Prajna (2004) "A new solution approach to polynomial LPV system analysis and synthesis," in *Proceedings of American Control Conference*, pp. 1362 1367.

[47] Chen, Y., A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam (2005) "Managing server energy and operational costs in hosting centers," *SIGMETRICS*, pp. 303–314.

[48] Qin, W. and Q. Wang (2005) "Feedback performance control for computer systems: an LPV approach," in *Proceedings of the American Control Conference*.

[49] (2007) "An LPV approximation for admission control of an internet web server: Identification and control," *Control Engineering Practice*, in press.

[50] "Web Caching project," Http://www.ircache.net.

[51] Qin, W., Q. Wang, Y. Chen, and N. Gautam (2006) "A First-Principles Based LPV Modeling and Design for Performance Management of Internet Web Servers," in *Proceedings of the American Control Conference*.

[52] Qin, W. and Q. Wang (2007) "Modeling and Control Design for Performance Management of Web Servers via an LPV Approach," *IEEE Transaction on Control System Technology*, **15**(2), pp. 259 275.

[53] Willinger, W., M. Taqqu, and A. Erramilli (1996) *Stochastic Networks: Theory and Applications*, chap. A Bibliographical Guide to Self-Similar Traffic and Performance Modeling for Modern High-Speed Networks, Oxford University Press.

[54] Leland, W., M. Taqqu, W. Willinger, and D. Wilson (1994) "On the self-similar nature of ethernet traffic," *IEEE/ACE transactions on Networking*, **2**, pp. 1–15.

[55] Crovella, M. and A. Bestavros (1997) "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Transactions on Networking*, **5**, pp. 835 846.

[56] Paxson, V. and S. Floyd (1995) "Wide-area traffic: The failure of the Poisson modeling," *IEEE/ACM Transactions on Networking*, **3**, pp. 226 244.

[57] Norros, I. (1994) "A storage model with self-similar input," *Queueing Systems*, **16**, pp. 387 396.

[58] Feldmann, A., A. Gilbert, and W. Willinger (1998) "Data Networks as Cascades: Investigating the Multifractal Nature of Internet WAN Traffic," *ACM Computer Communication Review*, **28**, pp. 42 55.

[59] Karasaridis, A. and D. Hatzinakos (2001) "Network heavy traffic modeling using α-stable self-similar processes," *IEEE Transactions on Communications*, **49**, pp. 1203 1214.

[60] Lopez-Guerrero, M., L. Orozco-Barbosa, and D. Makrakis (2005) "Probabilistic envelope processes for α-stable self-similar traffic models and their application to resource provisioning," *Performance Evaluation*, **61**, pp. 257–279.

[61] Samorodnitsky, G. and M. Taqqu (1994) *Stable Non-Gaussian Random Processes*, Chapman & Hall.

[62] McCulloch, J. (1986) "Simple consistent estimators of stable distribution parameters," *Communications on Statistics-Simulation*, **15**, pp. 1109–1136.

[63] Kogon, S. and D. Williams (1998) "On characteristic function based stable distribution parameter estimation techniques," in *A Practical Guide to Heavy Tails: Statistical Techniques for Analyzing Heavy Tailed Distributions* (R. Adler, R. Feldman, and M. Taqqu, eds.), Birkhauser.

[64] Paxson, V. (1997) "Fast, approximate synthesis of fractional gaussian noise for generating self-similar network traffic," *ACM Computer Communication Review*, **5**, pp. 5–18.

[65] Qin, W., Q. Wang, Y. Chen, and N. Gautam (2007) "An LPV Approach to Performance Control of Web Servers Under Self-Similar Workloads," in *IEEE International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, accepted.

[66] Qin, W., Q. Wang, and A. Sivasubramaniam (2007) "An LPV Based Control Theoretic Approach for Managing Server Performance Under Self-Similar Workloads," *Performance Evaluation*, submitted.

[67] Fujisaki, Y., F. Dabbene, and R. Tempo (2003) "Probabilistic design of LPV control systems," *Automatica*, pp. 1323 1337.

[68] Vidyasagar, M. (2001) "Randomized algorithms for robust controller synthesis using statistical learning theory," *Automatica*, pp. 1515 1528.

[69] Qin, W. and Q. Wang (2007) "Using Stochastic Linear Parameter Varying Control for CPU Management of Internet Servers," in *Proceedings of the Conference on Decision and Control*, submitted.

[70] Buzacott, J. A. and J. G. Shanthikumar (1993) *Stochastic Models of Manufacturing Systems*, Prentice Hall, New Jersey.

Vita

Wubi Qin

3381 Old Darlington Rd
Darlington, PA 16115
United States
Email: qinwubi@gmail.com

Education

The Pennsylvania State University, State College, PA

Ph.D. in Mechanical Engineering (2007)
Dissertation: "Feedback Performance Control for Self-Managing Computer Systems: an LPV Control Theoretic Approach"
Advisor: Dr. Qian Wang

Hong Kong University of Science & Technology
Master of Philosophy in Mechanical Engineering (2001)

Shanghai Jiao Tong University
B.A. in Engineering (1998)

Experience

Hong Kong University of Science and Technology
Teaching Assistant in the Department of Mechanical Engineering (1998-2001).

Advanced Semiconductor Pacific Technology
Research and Development engineer, working on software and controller design for the company's semiconductor packaging machines (2001-2003).

The Pennsylvania State University
Research Assistant working with Dr. Qian Wang on the performance management of computer servers (2003-2007). The work essentially became his Ph.D. thesis in 2007.