Modeling and control of server systems : application to performance and dependability

Luc Malrait

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Automatique et Productique**

Arrêté ministérial : 7 août 2006

Présentée par

Luc MALRAIT

Thèse dirigée par **Nicolas MARCHAND** et **Sara BOUCHENAK**

préparée au sein du centre de recherche **INRIA Grenoble Rhône-Alpes**,
du laboratoire **GIPSA-lab, département automatique**
et de l'école doctorale **Électronique, Électrotechnique, Automatique et Traitement du Signal**

Modeling and Control of Server Systems

Application to Performance and Dependability

Thèse soutenue publiquement le **3 Juillet 2012**,
devant le jury composé de :

Didier GEORGES, Président
Professeur à Grenoble INP
Gilles GRIMAUD, Rapporteur
Professeur à l'Université de Lille
Fabio GOMEZ-ESTERN AGUILAR, Rapporteur
Professeur à l'Université de Séville
Xiaoyun ZHU, Examinatrice
VMware Inc.
Sara BOUCHENAK, Directrice de thèse
Maître de conférences à l'Université de Grenoble
Nicolas MARCHAND, Co-Directeur de thèse
Chargé de Recherche CNRS, GIPSA-Lab (Grenoble, France)

Acknowledgments

My first thought goes to my parents and my brothers, who continuously supported me throughout the years. This achievement is also the fruit of their labor.

I want to express my deepest gratitude to Sara Bouchenak and Nicolas Marchand, who supervised my thesis work. They provided me with the best working environment, and I was very touched by their kindness.

I would like to thank the jury members, Didier Georges, Gilles Grimaud, Fabio Gomez-Estern Aguilar and Xiaoyun Zhu for agreeing to review my thesis.

I also want to thank all the people who made this thesis possible.

Christian Commault for his help and patience.

Carlos Canudas-de-Wit for his guidance.

The members of the NeCS team and the control engineering department of Gipsa-Lab from whom I have learned so much.

Chloé Codron for her support and all my dear friends.

Abstract

Server technology provides a means to support a wide range of on-line services and applications, such as web services, e-mail services, database services. However, their ad hoc configuration poses significant challenges to the performance, availability and economical costs of applications. In this thesis, we examine the impact of server configuration on the central trade-off between service performance and service availability. First, we model the behavior of single servers using fluid approximations. Second, we develop novel admission control laws of central server systems. We provide several control laws for different combinations of quality-of-service and service level objectives. Among them, $AM\text{-}\mathscr{C}$, the availability-maximizing admission control law, achieves the highest service availability while meeting given performance objective; $PM\text{-}\mathscr{C}$ is a performance-maximizing admission control law that meets a desired availability target with the highest performance. We evaluate our fluid model and control techniques on the TPC-C industry-standard benchmark that implements a warehouse running on the PostgreSQL database server. Our experiments show that the proposed techniques successfully improve performance by up to 30 % while guaranteeing availability constraints. Furthermore, we extend this work to distributed server systems, that are widely used by Internet applications in the farm of server clusters and multi-tier systems. We present a distributed server model as a non-linear continuous-time model using analogies with fluid transfer. We then state an optimization problem for the control of distributed server systems. We provide an admission control that allows to get the highest service availability while a target performance level is guaranteed. Numerical evaluations of the proposed distributed model and control are presented, and show that the optimal configuration of such systems is not intuitive.

Contents

List of Figures

Part I

Context and Motivations

Chapter 1

Introduction

1.1 Challenges and Background

A large variety of Internet services exists, ranging from web servers to e-mail servers [40], streaming media services [18], e-commerce servers [4], and database systems [36]. These services are usually based on the classical client-server architecture, where multiple clients concurrently access an on-line service provided by a server, e.g. reading web pages, sending emails or buying the content of a shopping cart. Such server systems face varying workloads as shown in several studies [6, 13, 5]. For instance, an e-mail server is likely to face a heavier workload in the morning than in the rest of the day, since people usually consult their e-mails when arriving at work. In its extreme form, a heavy workload may induce server thrashing and service unavailability, with underlying economical costs. These costs are estimated at up to US$ 2.0 million/hour for Telecom and Financial companies [19, 35].

A classical technique used to prevent servers from thrashing when the workload increases consists in limiting the total number of concurrent client admitted to those servers. This is also known as the multi-programming level (MPL) configuration of servers.

Existing solutions to server control follow different approaches. Some of them rely on ad hoc techniques and heuristics, but do not provide any guarantee on the optimality of the system configuration [9, 34, 31]. Other approaches apply linear control theory [38, 12]. This does unfortunately not capture the intrinsic nonlinear behavior of server systems. Other solutions follow a queueing theory approach where the system can be accurately modeled. But this is obtained at the expense of a hard model calibration process which makes it unwieldy to use [41, 43, 37].

We believe that modeling server systems is necessary to provide quality-of-service (QoS) guarantees. However, we argue that for the effective adoption of server modeling and control, the models must accurately capture the *dynamics* and the *nonlinear* behavior

of server systems while being *simple* to deploy on existing systems.

In this thesis, we propose novel models and control laws for the on-line configuration and reconfiguration of server systems to provide service performance and dependability guarantees. We apply our solution first to central server systems, then to distributed servers.

We present the design, implementation and evaluation of ConSer, our servers control system. We apply a control engineering methodology in order to model and control the QoS of server systems. We design and validate a non-linear continuous-time model of central server systems from a fluid approximation and the observation of the system dynamics. This model involves very few external parameters, which are easy to identify and have a precise meaning. We design and implement non-linear feedback MPL control laws for central server systems.

First, two variants of control laws are proposed.AM-$\mathscr{C}$ is an availability-maximizing server control that achieves the highest service availability given a target performance level objective. PM-$\mathscr{C}$ is a performance-maximizing server control that meets a desired availability level objective with the highest performance. Furthermore, two additional control laws are proposed for applying performance and availability optimization at multiple levels, these are PA-AM-$\mathscr{C}$ and AA-PM-$\mathscr{C}$ laws.

An evaluation of ConSer is conducted on the TPC-C application, an industry-standard benchmark implementing a warehouse, running on the PostgreSQL database server. A wide range of application workload conditions was considered. The results of the experiments conducted on central servers show that the proposed techniques provide significant benefits on the performance and the availability of the controlled system compared to ad hoc control solutions.

The thesis also proposes MPL control solutions for distributed server systems, i.e. systems consisting of multiple servers such as multi-tier systems and clustered servers. We provide model and control solutions to guarantee QoS levels of distributed server systems. We design a nonlinear continuous-time model of distributed server systems and apply optimal control theory to express the MPL configuration. We propose an instance of MPL control law that that maximizes the availability of the system and guarantees a performance level objective.

Numerical evaluations of the proposed model and control of distributed servers compare the controlled system with ad hoc controlled systems, and shows that the former uses less resources than the latter while providing better performance and availability guarantees.

1.2 Organization of the document

This thesis is organized in four main parts, respectively dedicated to presenting the context and motivations of our work in Part I, the proposed central server modeling and

control in Part II, the proposed variants for modeling and control of distributed servers in Part III, and our conclusions, perspectives and list of publications in Part IV. The different Chapters constituting this document are as follows.

Part I. Context and Motivations

Chapter 2. Background. The background necessary to understand the rest of the document is presented in Chapter 2. The Chapter first recalls the main architecture of client-server computing systems, and describes how these systems organize in central vs. distributed servers. It also discusses the workload of server systems, how it is characterized, and how it dynamically varies over time. It then presents the quality-of-service of such systems considering two aspects, namely performance and availability, before introducing admission control techniques to provide quality-of-service guarantees.

Chapter 3. Related Work. A survey on the different strategies to manage servers configuration in order to improve the quality-of-service (QoS) is presented in Chapter 3. This Chapter first introduces general QoS management techniques, such as service degradation, service differentiation, request scheduling. The Chapter then puts an emphasis on admission control techniques, with a particular interest in MPL control. It thus reviews different approaches in the related work such as heuristic-based approaches, model-based solutions, linear control and queueing theory approaches.

Chapter 4 Motivations and Objectives. This Chapter first illustrates the dynamics of server systems and their nonlinear behavior. It then illustrates the impact of server MPL control on service performance and service availability, two antagonist quality-of-service criteria. The Chapter then defines the objectives of our work.

Part II. Central Server Modeling and Control

Chapter 5. Central Server Modeling. A central server model is developed in Chapter 5. This Chapter details the nature of the model input variables, state variables and output variables. The Chapter then builds step by step the equations that govern the interactions between the variables in the proposed model.

Chapter 6. Central Server Control. Central server MPL control is presented in Chapter 6. This Chapter first states the objectives for the control design. Then four feedback control laws are proposed. They take into account the trade-off between performance and availability of server systems .

Chapter 7. Central Server Experimental Evaluation. The central server model and control introduced in Chapter 5 and 6 are evaluated experimentally in Chapter 7. This Chapter first describes the experimental setup. Then, the model parameters are identified and the model is validated and compared with the real baseline system. The evaluation of the proposed control laws is then described.

Part III. Distributed Server Modeling and Control

Chapter 8. Distributed Server Modeling. A distributed server model is developed in Chapter 8. This Chapter details the nature of the model variables and builds, step by step, the equations that govern their interactions.

Chapter 9. Distributed Server Control. Distributed servers MPL control is studied in Chapter 9. This Chapter first states the objectives for the optimal control design. It then details the resolution steps for this problem. The Karush-Kuhn-Tucker necessary conditions provide an explicit solution.

Chapter 10. Numerical Evaluation of Distributed Servers. The distributed servers model and control introduced in Chapter 8 and 9 are evaluated numerically in Chapter 10. The control evaluation compares the performance and availability of the optimized system with an ad hoc controlled system.

Part IV. Conclusions and Perspectives

Chapter 11. Conclusion. This Chapter draws the conclusions of this thesis. It also discusses mid-term and long-term perspectives of this work.

Chapter 12. List of Publications. This Chapter lists the publications that present the results of this thesis. This includes, among others, publications in IEEEE Transactions on Computers, ACM SIGOPS Operating Systems Review, the European Control Conference (ECC), the IEEE/IFIP Dependable Systems and Networks Conference (DSN).

Chapter 2

Background

The guarantee of the quality-of-service provided by Internet applications is a crucial issue regarding the underlying economical and societal impacts. These applications rely on server systems that host the service. In this Chapter, we present the necessary background to understand the rest of the document. First, we describe server system architectures. We then define server workload and admission control. Finally, we present quality-of-service criteria and the feedback control concept.

2.1 Server systems

2.1.1 Client-server architecture

The client-server architecture is a classical model to build networked computing systems. It is depicted in figure 2.1. Servers can provide some service whereas clients may request a service. A Web server, for instance, is able to send web pages requested by clients over the Internet. Clients and servers are hosted on different computers connected through a communication network. In the following, we make no distinction between the computer machine and the application hosted by that computer. Multiple clients may concurrently access the same server.

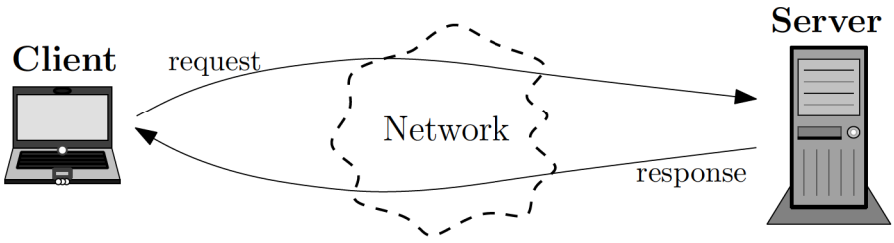


Figure 2.1: Client-server architecture

2.1.2 Session-based versus non session-based systems

We consider two communication models that describe the interactions between clients and servers: session-based systems and non session-based systems.

In session-based systems, a client may send several requests sequentially within the same client session. Typically, when logging in a train ticket booking Web site, a client will send a request and wait for the reply, repeating this process until he has purchased his ticket. More generally, this model reflects that the flow of requests which is submitted to a server depends not only on the number of clients that try to access the server, but also on the server's speed to handle requests.

In non session-based systems, clients who try to access the server are not willing to send multiple requests. In this case, the flow of requests submitted to a server depends only on the number of clients that try to access the server.

2.1.3 Central servers versus distributed servers

A central server is a system made of a unique server upon which the whole service is based. Clients directly interact with the server by sending requests, as shown figure 2.1.

On the other hand, for scalability purposes, a server may be distributed into multiple tiers. A distributed server is made of several servers interconnected by a local area network. Here, the servers follow a multi-tier architecture, where each server is responsible for a specific function. Server 1, server 2, ..., server n interact sequentially to build the response to the client request. For instance, a front-end web server is responsible for serving web pages, a middle-tier application server is responsible for the business logic of the application, and a back-end database server is responsible for storing non ephemeral data. Thus, server k may be seen as a client of server $k+1$. Figure 2.2 illustrates such a system.

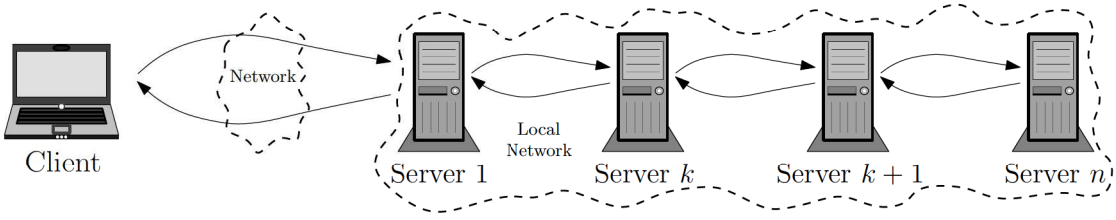


Figure 2.2: Distributed server with n tiers

More precisely, the following properties characterize such distributed server systems:

(P1) Requests arriving at tier 1 come from external clients. Requests arriving at tier k, with $k > 1$, come from tier $k-1$. Responses arriving to external clients come from tier 1. Responses arriving at tier k, with $k < n$, come from tier $k+1$. Figure 2.3 illustrates this property.

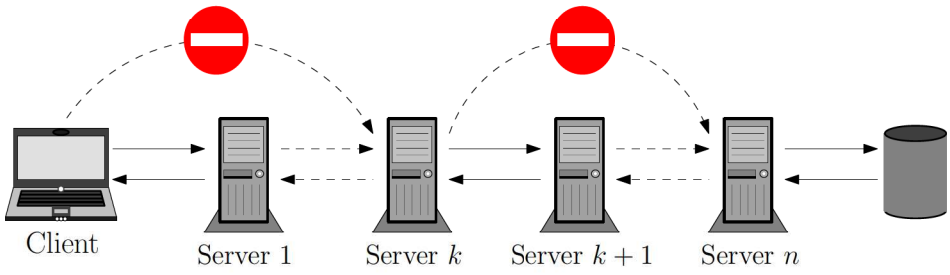


Figure 2.3: Flow of requests in a distributed server with n tiers

(P2) The handling of a request by tier k, with $k > 1$, is terminated when the associated response is sent to tier $k-1$. The handling of a client request by a distributed server is terminated when the associated response is sent to the client by the server at tier 1.

(P3) A request being processed at tier k can generate several requests sequentially at tier $k+1$.

(P4) At any time, a request being processed at tier k can generate at most one request at tier $k+1$.

(P5) As a result of P(4), the number of concurrent requests at tier $k+1$ is less or equal than the number of concurrent requests at tier k. Figure 2.4 illustrates this property. In this example, a request at tier k generates two requests at tier $k+1$. A connection is available when the process of a request, including the process of the sub-requests, is terminated.

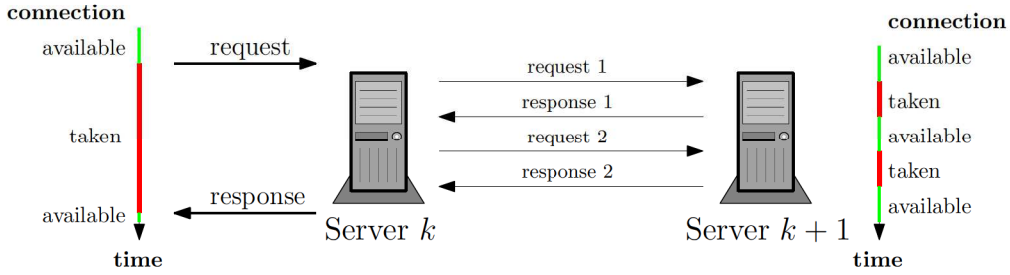


Figure 2.4: Serial access of requests

2.2 Server workload

The server workload is characterized by the workload amount and workload mix that we define in the following.

Workload amount. The server workload amount is the number of clients that try to concurrently access the server. It is denoted as N. The workload amount may vary over time. For instance, an e-mail service usually faces a higher workload amount in the morning than in the rest of the day.

Workload mix. The server workload mix characterizes the nature of requests made by clients. It is denoted as M. As an example, in an e-commerce application, the workload mix can be characterized by read requests, to browse the catalog of products, or by write requests, to perform a purchase. Actually, there is no existing precise way to characterize a workload mix. This can be done, for instance, based on the ratio of each type of request (e.g. TPC-C application [42]). The workload mix may also vary over time. For example, when there is a special offer on a product, the proportion of purchase requests will be higher.

Figure 2.5 illustrates how the server workload amount, i.e. the number of concurrent clients N, and the server workload mix, i.e. the ratio of read, write and read/write requests sent by the clients, can vary over time.

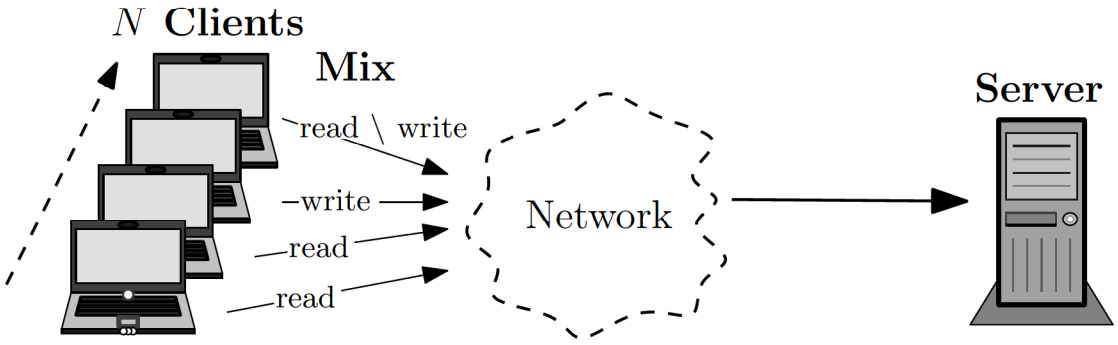


Figure 2.5: Server Workload

2.3 Server admission control

A server is able to concurrently handle multiple clients, this is known as a multiprogrammed server (or multi-threaded server). However, a high number of concurrent clients may induce server trashing. Admission control is a classical technique to prevent a server from thrashing. In practice, servers have a static configuration parameter responsible for admission control. This parameter is the limit for the maximum number of clients allowed to concurrently access a server, the Multi-Programming Level (MPL) configuration parameter of a server. Above this limit, incoming client requests are rejected. Thus, a client request arriving at a server either terminates successfully with a response to the client, or is rejected because of the server's MPL limit. Therefore, due to the MPL limit, among the N clients that try to concurrently access a server, only N_e clients actually access the server, with $N_e \leq MPL$. Figures 2.6 and 2.7 illustrate the case of admission control in a central server. In Figure 2.6, three clients try to concurrently access the server while five concurrent accesses are allowed ($MPL = 5$). The result is that admission control has no effect on the incoming client requests ($N_e = N$). In Figure 2.7, seven clients try to concurrently access the server while five concurrent accesses are allowed ($MPL = 5$). The result is that two request are rejected ($N_r = 2$) because of admission control and five concurrent requests are accepted on the server ($N_e = 5$).

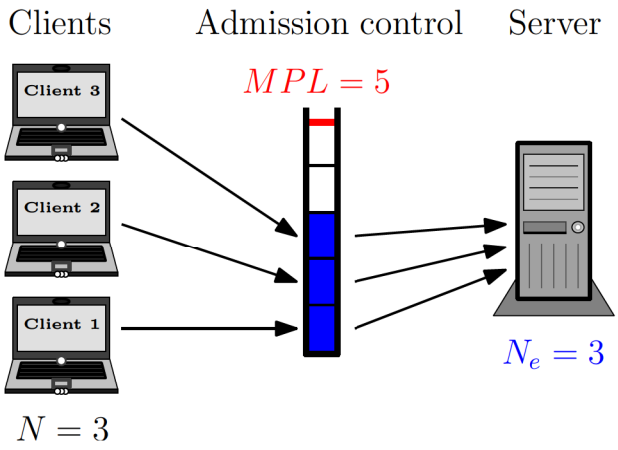


Figure 2.6: Under-loaded server

In the case of a distributed server, admission control can be applied at each tier of the architecture. Thus, similarly, a request arriving at tier k either terminates successfully with a response to tier $k-1$, or is rejected because of the kth server's MPL limit. We consider then the following property:

(P6) A client request rejected at tier k will cause the rejection of the mother request at tier $k-1$, with $k > 1$.

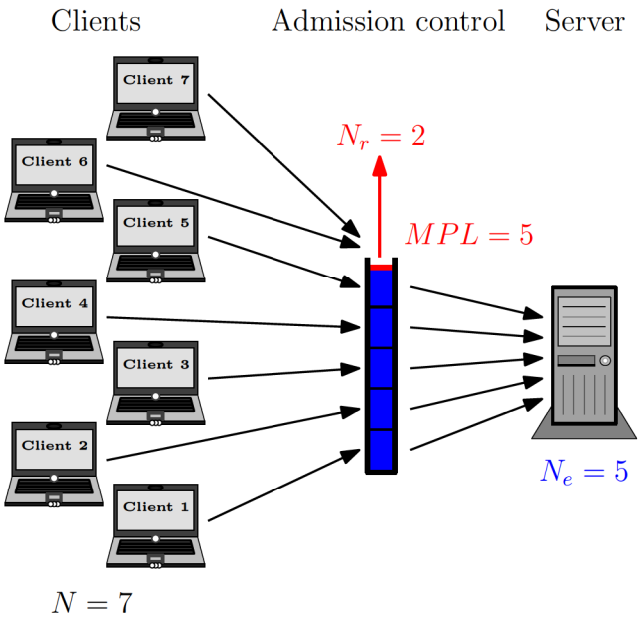


Figure 2.7: Over-loaded server

As shown in figure 2.8, admission control may induce the rejection of requests at any tier of the system. In this example, four clients are interacting with a two tiers distributed server. Admission control limits the number of concurrent clients to four at tier 1 ($MPL_1 = 4$) and three at tier 2 ($MPL_2 = 3$). The result is that one request is rejected at tier 2 ($N_r = 1$), four and three requests are respectively accepted at tiers 1 and 2 ($N_{e_1} = 4$ and $N_{e_2} = 3$).

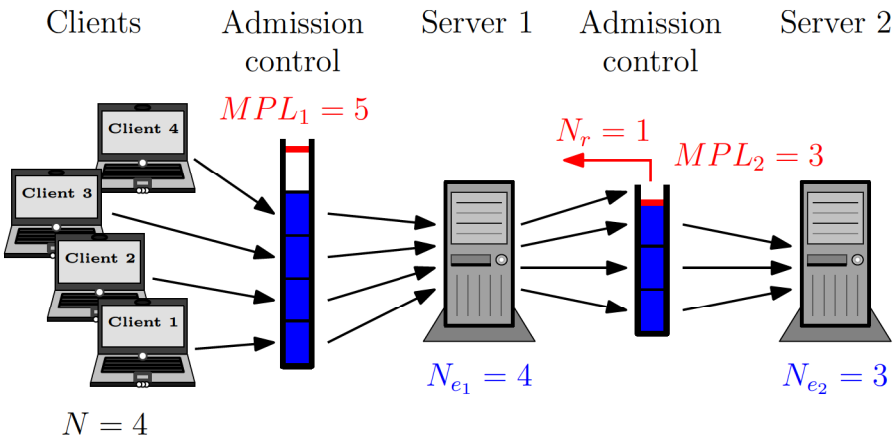


Figure 2.8: Distributed server admission control

2.4 Quality of service

In this section, we focus describe the quality-of-service of server systems.

2.4.1 Performance

Service performance can be characterized by different metrics, which pertinence depends on the application that is considered. The most relevant performance metrics of server systems are usually the server throughput and the client request latency [32].

Server output throughput is the number of handled client requests per unit of time. This metric is used to reflect the performance from the server-side, where a high throughput is desirable.

Client request latency is defined as the time needed by the server to process a request. The latency reflects the performance from the client side, where a low latency renders the responsiveness of the system.

In the following, we will consider, in particular, the average latency as the performance metric. It is denoted as L.

2.4.2 Availability

The availability of a service is characterized by its ability to be operational and to serve clients. In the following, we consider client request abandon rate as a metric of unavailability, the availability being 1 - abandon rate.

Request abandon rate is defined as the ratio between requests rejected due to admission control and the total number of requests that attempt to access a server. Request abandon rate is denoted as α. A low request abandon rate (or abandon rate, for short) is a desirable behavior that reflects service availability.

2.4.3 SLA

Service Level Agreement (SLA) is a contract negotiated between clients and their service provider. Service performance and service availability are parts of the SLA. The SLA specifies the service level objectives (SLOs) such as the maximum latency L_{max} and the maximum abandon rate α_{max} to be guaranteed by the service. For instance, a SLA could state that 95% of client requests must be admitted to the service.

2.5 Open-loop vs. closed-loop control system

An open-loop controller is based only on a mathematical model of the controlled system, whereas a closed-loop controller uses the monitoring of the system outputs to continuously adjust the control inputs (i.e. the tunable variables). This way, the outputs may follow some references. This feedback allows, in particular, to guarantee performance even with model uncertainties, and to reject possible disturbances. Figure 2.9 shows the classical scheme of feedback control.

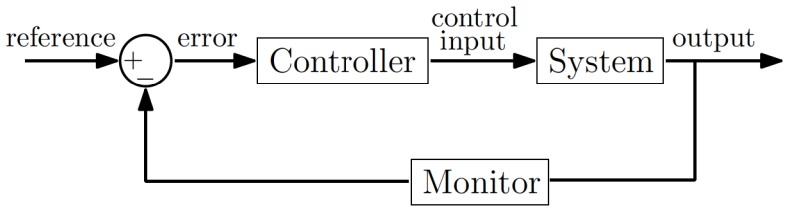


Figure 2.9: Closed-loop control scheme

2.6 Summary

In this chapter, we first presented the general organization of server systems, namely their architecture and their communication model. We then defined the workload of server systems in terms of workload amount and workload mix. We presented server admission control as a means to control high loads of servers, and described quality-of-service metrics to reflect server performance and availability. Finally, we introduced the feedback control concept. This allowed us to describe the necessary background before reviewing the related work in Chapter 3.

Chapter 3

Related Work

3.1 General QoS management

Previous work has noted that system configuration is a crucial issue for the performance and availability of server systems [29, 30]. Much related work has been done in the area of system QoS management [15], investigating techniques such as session-based admission control, service degradation, service differentiation and request scheduling. In the following, we review these techniques.

3.1.1 Session-based admission control

In session-based systems, applications rely on a communication model where a client may send several requests sequentially within the same session. For instance, in an e-commerce service, a client session consists of successive requests from the same client to browse the on-line store and purchase goods. In that case, the client session terminates after the client has confirmed the payment. For these systems, the completed session throughput renders more effectively the quality-of-service than the request throughput. Cherkasova and Phaal describe some characteristics of these systems and propose a session-based admission control mechanism [10]. The proposed technique distinguishes between incoming requests, depending on whether they belong to an existing session or not. New sessions are accepted if the predicted server utilization remains below a given threshold, otherwise, they are rejected. Different control policies derive from the server utilization prediction accuracy. These techniques are evaluated numerically using a simulation model of an e-commerce site.

3.1.2 Service degradation

Service degradation is another technique of QoS management. Abdelzaher and Bhatti describe in [2] three means of degrading a service content: lossy compression of images included in web contents, reduction of the number of objects embedded in a web page and reduction of local links. They renewed and derived a content degradation heuristic. The heuristic gives, for a given ratio of web services, the expected performance improvement in terms of reduction in server utilization. Considering two operating modes for a web service, one serving the full content and one serving a degraded content, they define a bottleneck resource utilization metric and implement a PI controller that maintains this utilization metric at a given level by adjusting the ratio of incoming client requests whose service content is degraded.

3.1.3 Service differentiation

Service differentiation techniques have been studied in order to provide different levels of quality-of-service to different classes of clients. Abdelzaher et al. describe in [3] a client prioritization policy based on web content adaptation. Using the same utilization metric that is used in service degradation [2], they compute different utilization targets for each class of clients, based on the current utilization of the system. They propose then a decoupled control strategy which consists of using as many PI controllers as there are client classes to meet the different utilization targets, by adjusting the web content for each class. This approach is evaluated experimentally on a Apache web servers using the httperf testing tool.

3.1.4 Request scheduling

Another QoS management technique is request scheduling. Server systems such as web servers are applications able to process requests from concurrent clients. The order in which these requests are processed by the server is usually determined by the scheduling policy of the underlying operating system. Crovella et al. study in [11] the impact of a request scheduling mechanism, at the application level, on service performance and availability. They present a server architecture enhanced with request scheduling capabilities and compare the impact of two scheduling policies on the mean response time. Their experiments use the SURGE web workload generator[8] and show that the shorter-job-first scheduling policy improves the average response time by up to 500% compared to the first-in-first-out scheduling policy. As the shorter-job-first policy may lead the server to a starvation state where long jobs (i.e. requests) are never processed, the authors analyze the influence of the incoming request size distribution on the slowdown represented by the ratio between request response time and service demand.

3.2 MPL control

In the following, we briefly overview the work related to admission control, and particularly MPL control for server system management. The Multi-Programming Level (MPL) is the maximum number of clients allowed to concurrently access a server. This parameter is a static configuration parameter of a server and is known as *MaxClients* for the Apache web server, *max_connections* for the MySQL database server. Different works study the impact of this parameter on the performance and availability of server systems and propose to tune it dynamically in order to meet service level objectives.

The improvement of server performance and availability is usually achieved by system administrators using ad hoc tuning [9, 34]. Other approaches were studied such as heuristic-based approaches, model-based approaches, linear control, and queuing theory approaches. We discuss these approaches in the following.

3.2.1 Heuristic-based approaches

Menascé et. al. propose a heuristic for the management of the QoS of servers through the determination of the multi-programming level (MPL) of servers using the hill-climbing optimization technique [31]. They try to maximize a performance function by adjusting the MPL and the maximum queue size of requests at each server. The performance function is a weighted sum of three QoS metrics: the server-side response time, the probability of rejection and the site throughput. They evaluate the proposed strategy with the TPC-W benchmark and compare it against a non-controlled base system. They show that a server system enhanced with the proposed framework is able to adapt its configuration at high loads to meet service level objectives. Although performing well in a variety of applications, the hill-climbing heuristic does not guarantee optimality.

A proxy-based approach for admission control and request scheduling is proposed by Elnikety et al. [14]. A hill-climbing technique is also applied off-line to determine the MPL value that maximizes the throughput of the system. An on-line prediction of the server load is then performed to decide whether the incoming requests should be accepted in the system. This admission control mechanism is enhanced with a shortest-job-first request scheduling policy. A delay bound for longer requests is set to prevent them from starvation. Experiments using the TPC-W benchmark show that this approach prevents the system from thrashing and improves its performance under high load.

Other solutions to MPL identification were proposed specifically to some server technologies. In [39], Shroeder et al. present a heuristic based feedback control strategy to automatically tune the MPL of a transactional database server. This consists in decreasing the MPL value over time until the system throughput is close enough to the maximum throughput. However, this technique does not apply to unknown and varying workloads of database servers.

3.2.2 Model-based approaches

Other approaches aim at modeling the system in order to characterize its capacity. In [16], Heiss et al. conduct a simulation-based study and propose an analytic model to adjust server MPL according to changing workloads. This model has one control input that is the MPL, and one output that is a performance metric of the system. It is assumed that the performance of the system can be modeled as a polynomial of degree 2 according to the MPL. The coefficients of this polynomial are estimated on-line using a recursive least-square estimator and the MPL that maximizes the parabola is computed. This approach is correlated with the extremum seeking control algorithms that are known to be very robust to system uncertainties. A parabola approximation is used to represent the performance function. This restricts the use of this technique to performance functions with a parabola shape and, thus, does not apply to criteria such as request latency and abandon rate that usually underlie service level objectives (SLOs) as perceived by clients.

Robertsson et al. [37] present a simple non-linear model of a server system for control purposes. They design a PI controller that controls the server utilization by adjusting the rate at which requests can be admitted to the system. A numerical evaluation of the model is performed, as well as a numerical evaluation of the controlled system.

3.2.3 Linear control

Other works aiming at applying control theory to server systems appeared in the last decade. A first approach consists in applying well-known linear control theory on servers modeled as SISO (single-input single-output) or MIMO (multiple-inputs multiple-outputs) black-boxes [38, 12]. This approach is often restricted to regulation problem, that is maintaining a desired SLO. It is for instance not possible to model trashing phenomena of saturating queues with linear models. Therefore, linear approaches are often poor to control systems with nonlinear behavior.

In [38], admission control of a Lotus Notes server is derived. Architecturally, Lotus Notes is a client-server system. Client software converts high-level user activity (mouse clicks, etc.) into remote procedure calls (RPCs) that are sent to the server. The server maintains a queue of these in-progress RPCs. Once an RPC is serviced, the appropriate response is sent to the client. Clients operate in a synchronous manner - waiting for the previous request to complete before sending a new request. The service level metric used in this paper - that is the considered output to regulate - is the length of the queue of in-progress RPC requests. The control consists in tuning the parameter that regulates the number of users allowed to access the server at any time. The control is therefore a session-level control as opposed to packet-level RPC objective. If a client is rejected, no RPC from this client will be accepted. An ARMA model is derived [25] based on experiments to derive a SISO transfer function of the system and a saturated integral

control is applied to regulate the queue length of RPC requests according to a reference value given by the administrator of the server.

A quite similar approach was taken in [12]. In this paper, an Apache web server is considered and the SLOs - that is the output -considered here are the CPU and memory utilization of the server. Concurrent access to a limited shared resource is known to produce trashing phenomena similar to the ones considered in this thesis, however this phenomena is not treated in [12]. The tuning parameters used to modify the CPU and memory usage - that is the control of the Apache web server - are the maximum number of clients that can connect to an Apache server, and the *KeepAlive Timeout*, which determines how long an idle connection is maintained. The problem here is MIMO. For this, an ARX model is derived [25] to obtain a state space representation of dimension 2 (actually, the outputs are the states). The SISO with two PI controllers and MIMO with LQR control are compared to show the efficiency of the MIMO approach to regulate CPU and memory usage to desired values.

3.2.4 Queuing theory approaches

Other approaches are based on non-linear models derived from queuing theory. Such models are presented by Tipper et al. [41] and Wang et al. [43], where non-linear dynamic models of queueing systems are derived from a fluid flow equation and steady state results from the queueing theory.

In [23], Kihl et al. design a PI controller that controls the number of requests in the system by adjusting the rate at which requests can be admitted to the system. The model used to tune the controller parameters is a linearized version of the $M/G/1$ queue model presented in [43]. A numerical evaluation is performed.

[21, 27] are other examples of the application of queuing theory models. However, they are restricted to the control of performance and do not consider availability constraints.

In [21] a simple model based on the linearization of a queueing model around an equilibrium point is presented. A self-tuning PI controller which aims at guaranteeing the response time of a server system is then derived. An experimental evaluation is conducted and shows that the closed-loop system behaves well around the equilibrium point. However, when the workload mix varies, the system is not able to reach its objectives.

In [27] and [28], the same queueing model as [21] is used as a feed-forward queueing predictor in an adaptive control scheme. An experimental evaluation to compare the different strategies is presented. It shows that the proposed queueing-model-based adaptive control performs better than the PI control described in [21], particularly under varying workloads. However, the queueing predictor relies on an off-line identification of the mean service time. Thus, in case the actual workload highly differs from the workload used to identify this parameter, one should expect the proposed queueing-model-based adaptive control to behave like an adaptive-only control.

3.3 Discussion

As discussed previously, several issues remain open in MPL control. First, we can notice that model-based approaches, whether they are based or not on queueing theory, provide a necessary framework to analyze and derive control strategies for server systems. However, the proposed models are either too simplistic to render performance and availability metrics, or too complex to be robust enough to the workload variability and its impact on the server behavior. Thus, there is a need for an analytic server model which is generic enough to handle various characteristics of the workload and the server, and which is able to provide an access to performance and availability metrics.

Second, none of the mentioned work provides a framework to guarantee both service performance and availability objectives. Indeed most often, performance functions, that are a mix of different server characteristics, are optimized through admission control but the impact on the service performance and availability is never clearly controlled.

Finally, many strategies suffer from being over-parametrized or at least require that several parameters are computed off-line, such as server capacity or service times. Some attention should be given in that direction as these requirements are highly time consuming and prevent such solutions from being actually and quickly applied.

In this thesis, we precisely address these issues as described in the following Chapters.

Chapter 4

Motivations and Objectives

4.1 Motivations

In this Chapter, we first motivate our work by discussing server workload variation, and showing the impact of server's MPL control on service performance and service availability. We then state the objectives of the thesis.

4.1.1 Server workload variation

As it was defined in Section 2.2, server workload is characterized by the workload amount, i.e. the number of clients that try to concurrently access the server, and the workload mix, i.e. the nature of the requests the clients send. The formal description of such workloads is not straightforward [7], [33] since it deals with the human behavior. As we can see in Figure 4.1, which represents the daily traffic volume to the 1998 football world cup web site, the server workload amount is strongly varying over time. In this example, we see that the number of requests received per day by the world cup web server can vary up to 400% in one day. The highest peak, which occurs at 30th June, corresponds to the game Argentina - England. We made an experiment to observe the impact of a server's workload on the throughput and the latency. The test bed comprises two dedicated computers. One computer hosts the PostgreSQL 8.2.6 [36] database server and another one hosts the client emulator. The TPC-C benchmark [42] is used to emulate a warehouse system where a set of concurrent clients perform transactions on warehouses stored on a database [1]. Figure 4.2 shows the impact of server's workload amount on its performance in terms of throughput and latency[1]. One can notice that when the workload amount becomes too high, from 65 clients in this example, the latency is dramatically increasing while the throughput is decreasing. This phenomena is known as trashing, and induces service unavailability.

[1] Details on the underlying experimental test bed are given in Section 7.1.

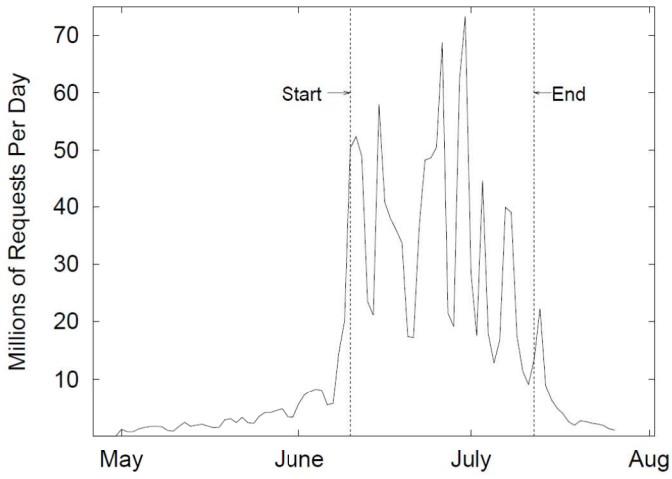


Figure 4.1: Daily traffic volume to the 1998 world cup web site taken from[5]

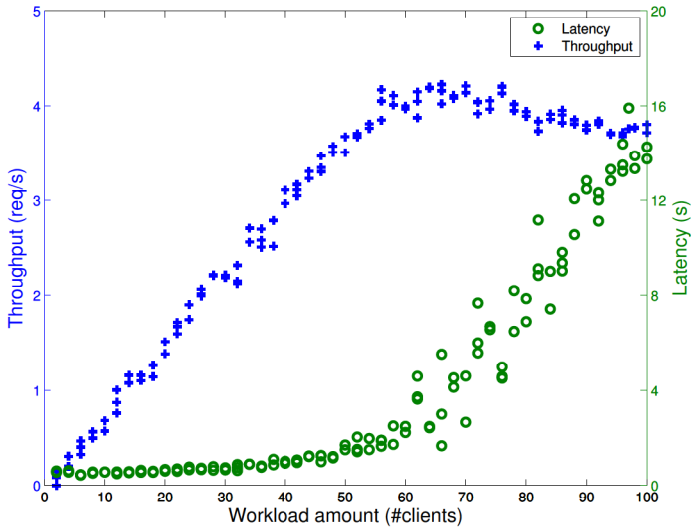


Figure 4.2: Impact of server's workload on throughput and latency

4.1.2 Impact of server control on performance

A proxy-based approach was followed to be able to tune the concurrency on server side. This proxy allows no more than MPL concurrent connections to the database server. Figure 4.3 describes the impact of server's MPL value on client request latency, when the workload amount varies [1]. Here, three values of MPL are considered, a low value (1), a medium value (25) and a high value (75). The low MPL is very restrictive regarding client concurrency on the server and thus, keeps the server under-loaded and implies a low client request latency. In contrast, with a high MPL, when the server workload amount increases client request latency increases too. Thus, a low MPL is desirable for a low client request latency, i.e. a good service performance.

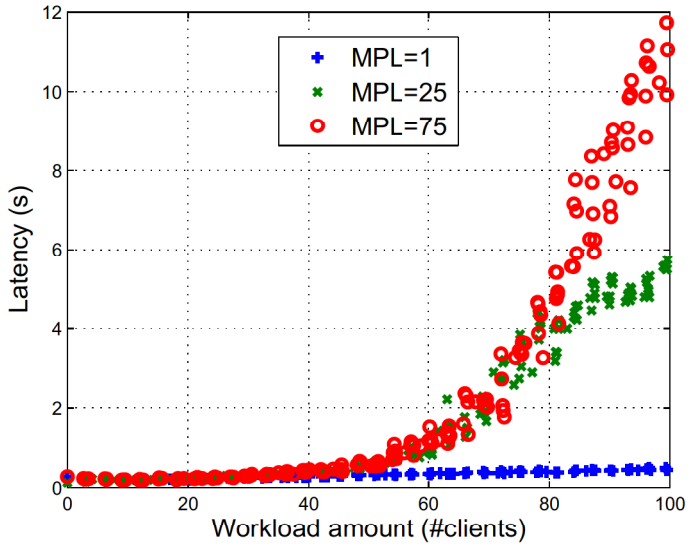


Figure 4.3: Impact of MPL on performance

4.1.3 Impact of server control on availability

Figure 4.4 describes the impact of MPL on client request abandon rate when the workload amount varies [1] . A low MPL is very restrictive regarding client concurrency on the server, and obviously implies a higher abandon rate compared to a high MPL which accepts more clients. Thus, a high MPL is desirable for a low client request abandon rate, i.e. a high service availability.

[1] Details on the underlying experimental tested are given in Section 7.1.

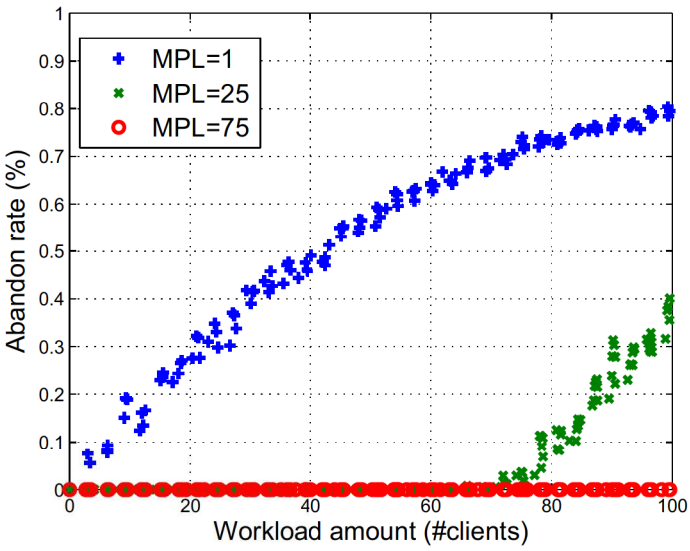


Figure 4.4: Impact of MPL on availability

4.2 Objectives

The previous observations showed that the server's MPL and workload have a strong influence on its quality-of-service. Whereas the server's workload is difficult to characterize, the server's MPL is a tunable parameter. Thus it seems relevant to choose the server's MPL as a control parameter in order to fulfill QoS objectives. The observations also emphasize the nonlinearities that exist between the characteristic variables of a server. Figures 4.3 and 4.4 show that performance and availability are antagonist. A trade-off between performance and availability must be considered when acting on server's MPL. The control objectives have to take this behavior into account.

The objectives of our thesis are as follows:

- Design a model for central servers that is simple to use and which renders the behaviors of server systems
- Design control laws that aim at guaranteeing QoS objectives in terms of performance and availability
- Evaluate the proposed model and control laws experimentally on real settings
- Propose an extended model to handle more complex systems, such as distributed servers
- Design control strategies for distributed servers
- Evaluate the proposed model and control for distributed servers

4.3 Summary

In this chapter, we first presented our motivations to use the servers MPL as a dynamic control variable to manage servers quality-of-service. We showed that server MPL has a reciprocal impact on servers performance and availability. Then, we detailed our objectives. In the following Chapters 5 and 6, we detail the design a central server model and feedback control laws based on server MPL control.

Part II

Central Server Modeling and Control

Chapter 5

Central Server Modeling

Mathematical modeling of server systems is not straightforward. A compromise between empirical and first principles approaches has to be made. In this chapter, we present our contributions in terms of a central server modeling. We describe first our methodology and then develop the design of a general model.

5.1 Methodology

Numerous server models exist in the literature, but they are often too complex to be used for control purpose, or too simple to render the behavior of such systems. We propose to design a model from a first principle approach, enhanced with heuristic elements deduced from observations. The goal is to get a description model that is useful for the design of feedback control laws for server performance and availability. In that sense, the model doesn't need to render the microscopic phenomena involved in such systems. That is the reason why we choose to build a continuous time model which captures the main characteristics and dynamics of servers that reflect the state of the server in terms of performance and availability.

Thus, we will consider all the variables of the system - that are most integers - as real variables in $\mathbb{R}$. This approximation usually has a low impact if the considered variables take large values. The model variables can be divided into four classes.

Control inputs are the tunable parameters of a system.

Exogenous inputs represent the surrounding conditions that have an impact on the system but which are not under control.

State variables are interdependent variables that are impacted by inputs.

Output variables are the measured or the controlled variables of a system, and depend on the state of the system.

The approximation described above enables to write the infinitesimal variation of characteristic state variables of the system with respect to time. Those variations can be seen as fluid flows, e.g. client request flows in the present case; and a request queue on the server is similar to a fluid tank [1]. The model is therefore built as a set of differential equations - as for most physical systems in mechanics, physics, electricity, etc. - that describes the time evolution of state variables. This approach is known to reproduce the mean behavior of the system, in addition to some filtering of the measurements as shown in section 7.1.

5.2 Central server model

5.2.1 Model structure

In the present case, we identify three state variables that describe and have an impact on server performance and availability, namely the current number of concurrent client requests in the server N_e, the server throughput T_o and the client request abandon rate α. State variables are usually influenced by themselves and by input variables.

The inputs of the proposed model can be divided in two classes: controlled (also denoted by control inputs) and uncontrolled inputs (denoted by exogenous inputs). The first class includes the server MPL tunable parameter that can be used to control the admission to the server. In the second class of input variables, we find the server workload amount N and the server workload mix M.

In addition to input and state variables, the model has output variables such as the average latency L to process a client request on the server. In the following, we describe the proposed fluid model through the formulas of its state and output variables.

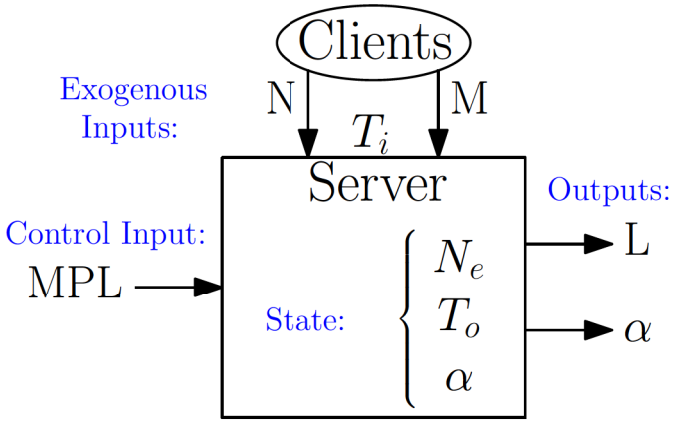


Figure 5.1: Model inputs/outputs

5.2.2 Model state variables

Among the N concurrent clients that try to connect to a server, MPL control authorizes N_e concurrent clients to actually enter the server, with $0 \leq N_e \leq N$ and $0 \leq N_e \leq MPL$. It clearly means that the MPL is not a control in the classical sense. If the server is not saturated, that is $N_e < MPL$, then the control MPL has no effect on the system. The MPL acts as a saturation on the exogenous input N_e.

Let $cr(t, t+dt)$ be the number of client connections created on the server between t and $t+dt$, and $cl(t, t+dt)$ be the number of client connections closed on the server between t and $t+dt$.

Thus, a balance on N_e between t and $t+dt$ gives

$$N_e(t+dt) = N_e(t) + cr(t, t+dt) - cl(t, t+dt) \tag{5.1}$$

Let T_i be the incoming throughput of the server, measured as the number of client connection demands per second. It comes that the number of connections created between t and $t+dt$ is

$$cr(t, t+dt) = (1 - \alpha(t)) \cdot T_i(t) \cdot dt \tag{5.2}$$

where α is the abandon rate of the server.

Similarly, let T_o be the outgoing throughput of the server, measured as the number of client requests a server is able to handle per second. Thus, the number of connections closed between t and $t+dt$ is

$$cl(t, t+dt) = T_o(t) \cdot dt \tag{5.3}$$

Deriving from (5.1), (5.2) and (5.3), we have $\dot{N}_e$, the derivative of N_e

$$\dot{N}_e(t) = (1 - \alpha(t)) \cdot T_i(t) - T_o(t) \tag{5.4}$$

Moreover, we assume that the system reaches a steady state in a reasonably short period of time Δ; this is particularly reflected in state variables outgoing throughput T_o and abandon rate α. During this short period of time, the workload is relatively stable, which is consistent with studies such as [5]. Thus, the dynamics of T_o and α can be approximated by first order systems through their derivatives as follows

$$\dot{T}_o(t) = -\frac{1}{\Delta}\left(T_o(t) - \bar{T}_o\right)$$
$$\dot{\alpha}(t) = -\frac{1}{\Delta}\left(\alpha(t) - \bar{\alpha}\right)$$

where $\bar{T}_o$ and $\bar{\alpha}$ are the steady state values of respectively the outgoing throughput and the abandon rate of the server.

The next step naturally consists in finding the expression of $\bar{T_o}$ and $\bar{\alpha}$. A balance on the number of served client requests (or outgoing requests) N_o gives

$$N_o(t+dt) = N_o(t) + sr(t, t+dt)$$

where $sr(t, t+dt)$ is the number of served request between t and $t+dt$. Since there are N_e concurrent clients on the server and the average client request latency is L, the number of served requests during dt will be $sr(t, t+dt) = \frac{dt}{L} N_e$. Thus, we get $\dot{N_o} = \frac{N_e}{L}$, that is

$$\bar{T_o} = \frac{N_e}{L}$$

which is an expression of Little's law [26].

By definition, $\bar{\alpha}$ is equal to zero if N_e is smaller than MPL, and $\bar{\alpha}$ is equal to $1 - \frac{T_o}{T_i}$ if $N_e = MPL$. However, the stochastic nature of the client request arrival may lead to situations where the measured average N_e is smaller than MPL but where punctually, the number of clients that try to access the server is actually higher than MPL, and thus, some clients are rejected. We conducted an experiment on our test bed to observe the evolution of the client request abandon rate when the server workload amount increases and the MPL is equal to 25 [1]. The results are shown in Figure 5.2, which compares the actual measured abandon rate with the naive model prediction of the abandon rate, showing a mismatch between the two. In order to take this behavior into account, we choose to write

$$\bar{\alpha} = \frac{N_e}{MPL} \cdot \left(1 - \frac{T_o}{T_i}\right)$$

This renders that the probability to reject a client connection is higher when the average N_e is close to MPL. Figure 5.2 shows that this improved method provides a more accurate estimation of the abandon rate. Finally, it follows that

$$\dot{T_o}(t) = -\frac{1}{\Delta}\left(T_o(t) - \frac{N_e(t)}{L(t)}\right) \tag{5.5}$$

$$\dot{\alpha}(t) = -\frac{1}{\Delta}\left(\alpha(t) - \frac{N_e(t)}{MPL(t)} \cdot \left(1 - \frac{T_o(t)}{T_i(t)}\right)\right) \tag{5.6}$$

5.2.3 Distinction between session-based and non session-based systems

In the case of a non-session based system, the incoming request throughput T_i is considered as an exogenous input.

In the case of a session based system, T_i must be considered as an internal variable since it depends on the state of the system. Thus, one has to give the expression of T_i according to the inputs and the state of the system.

[1] Details on the underlying experimental test bed are given in Section 7.1.

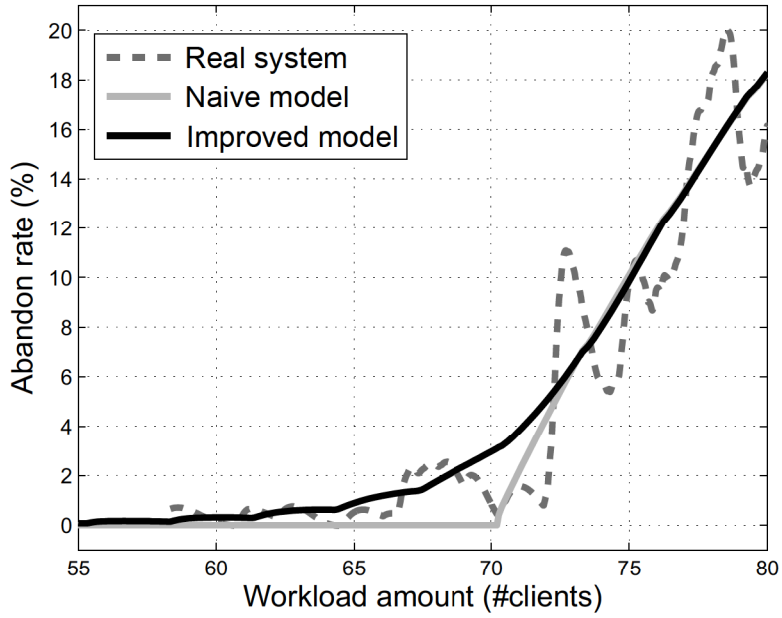


Figure 5.2: Accuracy of modeled abandon rate

We define the L_{tt} parameter as the "think time" of the clients. This parameter represents the time elapsed between the receipt by the client of a server response and the sending of another request. The average time between arrivals of a client requests is then the sum of the think time and the average latency of the accepted requests on the server, that is $(1-\alpha)L + L_{tt}$. Thus the incoming request throughput can be written as

$$T_i = \frac{N}{(1-\alpha)L + L_{tt}} \tag{5.7}$$

5.2.4 Model output variables

Now that we have defined the model state variables, the last step consists in expressing the model output variable latency L. Latency obviously depends on the global load of the server, i.e. the workload mix M and the number of concurrent clients on the server N_e. We conducted an experiment on our test bed to observe the evolution of the client request latency when the server workload amount increases [1] . The server's MPL is set to 100, so that no client request can be rejected. Figure 5.3 describes the evolution of latency L as a function of N_e, for a given workload mix. One can see that a second degree polynomial in N_e is a good approximation of the latency L. Thus:

$$L(N_e, M, t) = a(M,t)N_e^2 + b(M,t)N_e + c(M,t) \tag{5.8}$$

[1] Details on the underlying experimental test bed are given in Section 7.1.

The parameter c is positive as it represents the zero-load latency. a and b are also positive since they model the processing time of requests.

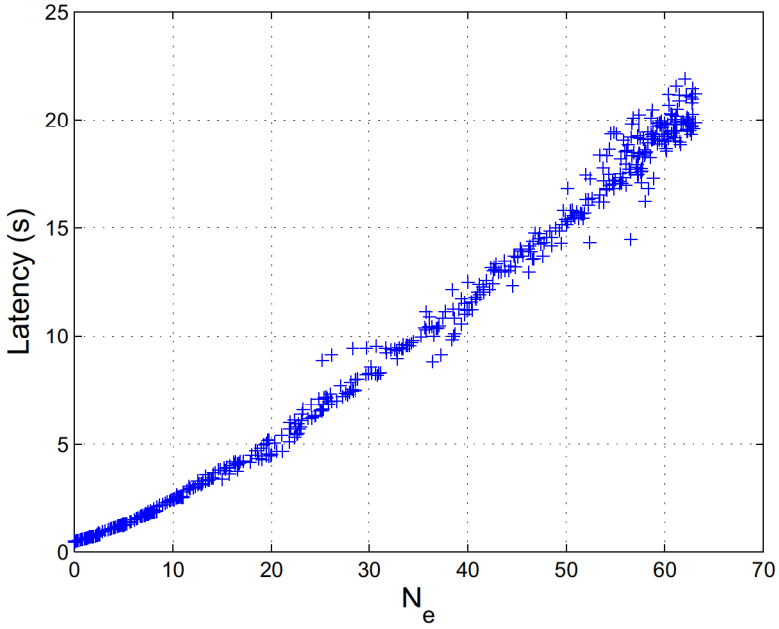


Figure 5.3: Latency as a function of N_e

To sum up, the proposed fluid model is given by equations (5.4) to (5.8) that reflect the dynamics of the state and outputs of server systems in terms of performance and availability.

5.3 Summary

In this chapter, we first presented the methodology we used to handle the problem of modeling server systems. We defined the model variables as input, output and state variables. From a fluid flow equation and the Little's law we derived a set of first order differential equations that link these variables together. A distinction between session-based and non session-based systems has been made as this characteristic has an impact on the exogenous nature of the inputs. In the following Chapter 6, we use this model to design and analyze feedback control laws for servers performance and availability. The experimental evaluation of the proposed model is presented in Chapter 7.

Chapter 6

Central Server Control

6.1 Problem statement: Trade-off between performance and availability

In the following, we study the trade-off between the performance and the availability of server systems, and derive the optimal MPL control of server systems based on the proposed fluid model, that is the optimal number of concurrent clients admitted to the server with respect to a between the two QoS metrics.

We provide four variants of control laws. AM-$\mathscr{C}$ is an availability-maximizing server MPL control that achieves the highest service availability given a fixed performance constraint. Symmetrically, PM-$\mathscr{C}$ is a performance-maximizing server MPL control that meets a desired availability target with the highest performance. In the present case, service availability is measured as the client request acceptance rate (i.e. 1 - α), and service performance is measured as the average client request latency (i.e. L).

AA-PM-$\mathscr{C}$ and PA-AM-$\mathscr{C}$ are two other control laws that extend respectively PM-$\mathscr{C}$ and AM-$\mathscr{C}$as discussed in Sections 6.4 and 6.5 .

6.2 AM-$\mathscr{C}$: availability-maximizing control

AM-$\mathscr{C}$ aims at guaranteeing a trade-off between service performance and service availability with the following properties:

(P1) the average client request latency does not exceed a maximum latency $L_{\max}$, and

(P2) the abandon rate α is made as small as possible.

To that end, a feedback control law is proposed to automatically adjust the MPL server control parameter in order to satisfy this trade-off. The basic idea behind this law is

to admit clients in such a way that the average client request latency L is close (equal) to $L_{\max}$. By construction, this maximizes the number of admitted clients N_e, which induces a minimized abandon rate α.

A first approach could consist in solving Eq. (5.8) in such a way that $L = L_{\max}$. Although accurately reflecting the system, such an approach is unwieldy since it requires the knowledge of accurate values of parameter a, b and c in equation 5.8, through an online identification of these parameters since the workload may change over time.

We propose another approach which avoids this online identification of model's parameters. It is a Lyapunov approach. First, let us assume that the dynamic of the load's variations is much smaller than the dynamic of T_o and α. It follows:

$$\begin{aligned} L(N_e) &= aN_e^2 + bN_e + c \\ \alpha(t) &= \bar{\alpha} \\ T_o(t) &= \bar{T}_o \end{aligned}$$

We choose $V(N_e) = \frac{1}{2}(L - L_{max})^2$ as a Lyapunov function candidate.

Then with (5.4) we get

$$\dot{V}(N_e) = (L - L_{max})\,(2aN_e + b)\left(1 - \frac{N_e}{MPL}\right)(T_i - \bar{T}_o)$$

Taking

$$MPL = \frac{N_e}{1 + \gamma_L(L - L_{\max})} \tag{6.1}$$

will give:

$$\dot{V}(N_e) = -\gamma_L(L - L_{max})^2(T_i - \bar{T}_o)(2an + b)$$

In case of overload ($T_i - \bar{T}_o > 0$), since $2aN_e + b > 0$, we have $\dot{V}(N_e) \leq 0$ for every $\gamma_L > 0$. Invoking Lasalle's Invariance principle, L globally asymptotically converges to L_{max}.

We also have to ensure that MPL takes positive finite values, that is $1+\gamma_L(L-L_{\max}) > 0$. This is guaranteed for $\gamma_L < \frac{1}{L_{\max}}$.

The proposed admission control technique requires a unique external parameter, that is γ_L. This parameter has an impact on both convergence time of the control and stability of the system.

In summary, it is interesting to notice that the feedback control law given in (6.1) will reflect one of the following situations. If the current latency L is higher than L_{max}, property (P1) is not guaranteed and the control law will produce an MPL as a decreased value of the current number of admitted concurrent clients N_e (since $(1+\gamma_L(L-L_{\max})) > 1$), which aims at meeting (P1). Symmetrically, if L is lower than L_{max}, property (P1) holds but property (P2) may not hold, and the control law will produce an MPL as an increased value of N_e (since $(1 + \gamma_L(L - L_{\max})) < 1$), which aims at meeting (P2). Finally, if L is equal to L_{max}, both properties (P1) and (P2) hold.

6.3 PM-$\mathscr{C}$: performance-maximizing control

Similarly, PM-$\mathscr{C}$ aims at guaranteeing the following trade-off between service performance and service availability where:

(P3) the client request abandon rate does not exceed a given maximum abandon rate $\alpha_{\max}$,

(P4) with the lowest average client request latency.

In this context, (P4) will be ensured given (P3) if and only if the MPL converges to the smallest value that guarantees $\alpha \leq \alpha_{max}$. It is obtained via a simple input-output linearization approach [22], taking α as the output, to solve the problem

$$\dot{\alpha} = -\gamma_{\alpha}(\alpha - \alpha_{\max}) \tag{6.2}$$

with $\gamma_{\alpha} > 0$. Furthermore, since the workload remains relatively stable during a short period of time, as stated previously, $\dot{N}_e = 0$. Then, from equations (5.4) and (5.6), we get

$$\alpha = 1 - \frac{T_o}{T_i}$$

$$\dot{\alpha}(t) = -\frac{1}{\Delta}\alpha(t)\left(1 - \frac{N_e(t)}{MPL(t)}\right) \tag{6.3}$$

Thus, from Eq. (6.2) and (6.3) and with the following control applied to MPL, α will converge to α_{max}

$$MPL = \frac{\alpha N_e}{\alpha - \gamma'_{\alpha}(\alpha - \alpha_{\max})} \tag{6.4}$$

where $\gamma'_{\alpha} = \gamma_{\alpha}\Delta$.

We also have to ensure that MPL takes positive finite values, that is $\alpha - \gamma'_{\alpha}(\alpha - \alpha_{\max}) > 0$. This is guaranteed for $\gamma'_{\alpha} < \frac{1}{1-\alpha_{\max}}$.

The proposed admission control technique requires a unique external parameter, that is γ_{α}. This parameter has an impact on both convergence time of the control and stability of the system.

In summary, it is interesting to notice that the feedback control law given in (6.4) will reflect one of the following situations.

6.4 *AA-PM-𝒞*: availability-aware performance-maximizing control

The *AA-PM-𝒞* availability-aware performance-maximizing control law is an *MPL* control law that extends the previously presented *PM-𝒞* law. Indeed, as we will see in Chapter 7, Figure 7.7 illustrates the behavior of *PM-𝒞* where the client request abandon rate is kept below a service level limit while the request latency is minimized. However, this may result in a situation where client request latency has a reasonable value (i.e. a low value) whereas client requests are rejected. For instance, Figures 7.7(b) and 7.7(c) respectively show that between the 14^{th} and 37^{th} minutes of the experiment, 10% of client requests are rejected while request latency is below 8 seconds. During that period of time, and if availability is prioritized over performance, availability could be maximized (i.e. rejection rate minimized) as long as performance meets a given service level objective (i.e. request latency is below a limit). Then, as long as availability objective is guaranteed (i.e. abandon rate is below a limit), performance is maximized.

Thus, *AA-PM-𝒞* aims at guaranteeing the following trade-off between server performance and availability, with a priority to availability as follows:

(P5) the client request abandon rate does not exceed a given maximum abandon rate $\alpha_{\max}$,

(P6) furthermore, the client request abandon rate is minimized as long as request latency does not exceed a given maximum latency $L_{\max}$, and

(P7) the request latency is minimized as long as abandon rate reaches its limit $\alpha_{\max}$.

Therefore, the *AA-PM-𝒞* control law takes into account two limits, a request abandon rate limit and a request latency limit. This law consists in applying the *AM-𝒞*-based control when the latency is below its limit. Then if the load is too heavy to guarantee both performance and availability constraints, *AA-PM-𝒞* switches to the *PM-𝒞*-based control.

6.5 *PA-AM-𝒞*: performance-aware availability-maximizing control

Similarly, *PA-AM-𝒞* extends the previously proposed *AM-𝒞* law with service level limits for both performance and availability, and a priority of performance over availability. Thus, *PA-AM-𝒞* aims at guaranteeing the following trade-off between server performance and availability:

(P8) the client request latency does not exceed a given maximum latency $L_{\max}$,

(P9) moreover, the client request latency is minimized as long as request abandon rate does not exceed a given maximum abandon rate $\alpha_{\max}$, and

(P10) the request abandon rate is minimized when the latency reaches its maximum authorized limit.

Thus, *PA-AM-*$\mathscr{C}$ consists in first applying the *PM-*$\mathscr{C}$-based control when the abandon rate is below its limit. Then if the load is too heavy to guarantee both availability and performance constraints, *PA-AM-*$\mathscr{C}$ switches to the *AM-*$\mathscr{C}$-based control.

6.6 Summary

In this Chapter we first drew up the objectives of the proposed control strategies. Based on these objectives, we built the *AM-*$\mathscr{C}$control, that achieves the highest service availability given a fixed performance constraint, and the *PM-*$\mathscr{C}$control, that achieves the highest performance given a fixed service availability constraint. The design methods we used in Sections 6.2 and 6.3 ensure the stability of the closed loop system for both of these strategies. Two other control laws, namely *AA-PM-*$\mathscr{C}$and *PA-AM-*$\mathscr{C}$, that provide another compromise between performance and availability have been introduced. The experimental evaluation of these control laws is presented in Chapter 7.

Chapter 7

Experimental Evaluation: Central Server Modeling and Control

The proposed central server fluid model and control laws introduced in Chapter 5 and 6 are evaluated in the present Chapter. The Chapter first describes the underlying experimental setup, before presenting experimental evaluation results of modeling and control.

7.1 Experimental setup

The evaluation of the proposed fluid model has been conducted using the TPC-C benchmark [42]. TPC-C is an industry standard benchmark from the Transaction Processing Council that models a realistic database server application as a warehouse system where clients request transactions on warehouses stored on a database server. TPC-C comes with a client emulator which emulates a set of concurrent clients that remotely send requests to the database server. The TPC-C client emulator allows to specify the number of concurrent clients to launch (i.e. the workload amount N). It also specifies the client think time, that is the inter-arrival time between two consecutive client requests. We extended the client emulator in order to be able, on the one hand, to vary the workload amount N over time, and on the other hand, to vary the workload mix M over time. For the latter extension, we considered two mixes of workload, one consisting of read-only requests, and another consisting of a mix of read-write requests.

Our experiments have been conducted on a set of two computers connected via a 100 Mb/s Ethernet LAN, one computer dedicated to the database server and another to the client emulator. The database server is PostgreSQL 8.2.6 [36]. The proposed model was implemented using an online monitoring of the system which allows to maintain the state of the model. Well-known Kalman filtering techniques were therefore applied [20]. Both client and server machines run Linux Fedora 7. The server machine

is a 3 GHz processor with 2GB RAM, while the clients' computer is a 2 GHz processor with 512MB RAM.

7.1.1 Estimation of throughputs

Some variables like throughputs are not directly accessible on the system. Indeed, throughputs corresponds to a fluid-based approximation of discrete phenomena. Throughputs are therefore estimated using the measurement of the amount of the corresponding discrete variable that are available. For instance, the output throughput T_o is estimated using the number of connections cl that are release after client request termination. Similarly, the input throughput is deduced using the number of new connections cr for new client requests. Denoting generically T_c a throughput and c the corresponding discrete variable, we have the following equation:

$$\dot{T}_c = c$$

If T_c can not be measured, c is usually simply accessible. Therefore, we build the following classical observer:

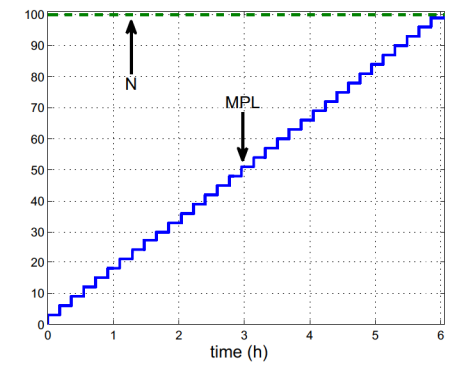
$$\begin{aligned} \dot{\hat{T}}_c &= \hat{c} + L_1(c - \hat{c}) \\ \dot{\hat{c}} &= L_2(c - \hat{c}) \end{aligned}$$

where $\hat{T}_c$ and $\hat{c}$ represent the filtered estimated values of T_c and c. The gain $L = (L_1\ L_2)$ can then be tuned using the optimal framework as in [20]. A discretized version of the observer can be used for implementation purpose. Usually, tuning the cut frequency of the filter is not difficult since c corresponds to variables with fast changes (e.g. the instantaneous number of connections is rapidly varying) while the whole dynamics of the server remains quite slow. It is therefore easy to smooth c without any influence on the closed loop server control.

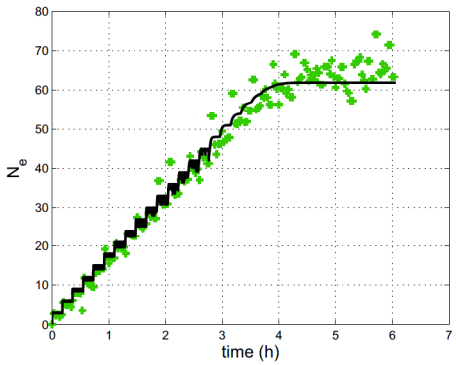
7.2 Model Evaluation

7.2.1 Model identification

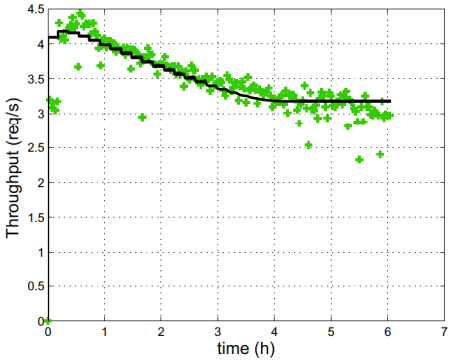
Figure 7.1 describes the case of an open loop system where the workload amount N trying to access the database server is fixed (to 100 clients) and where the MPL value of the server varies (see Figure 7.1(a)). Figures 7.1(b), 7.1(c) and 7.1(d) show the evolution over time of respectively the number N_e of concurrent clients admitted in the server, the outgoing throughput T_o and the abandon rate α, for both the real system (+) and the model (solid line).



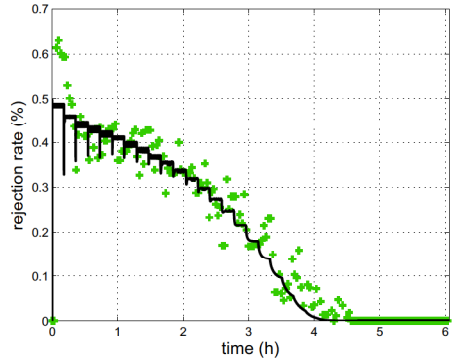
(a) Varying MPL with a fixed workload amount



(b) Admitted concurrent clients



(c) Throughput



(d) Abandon rate

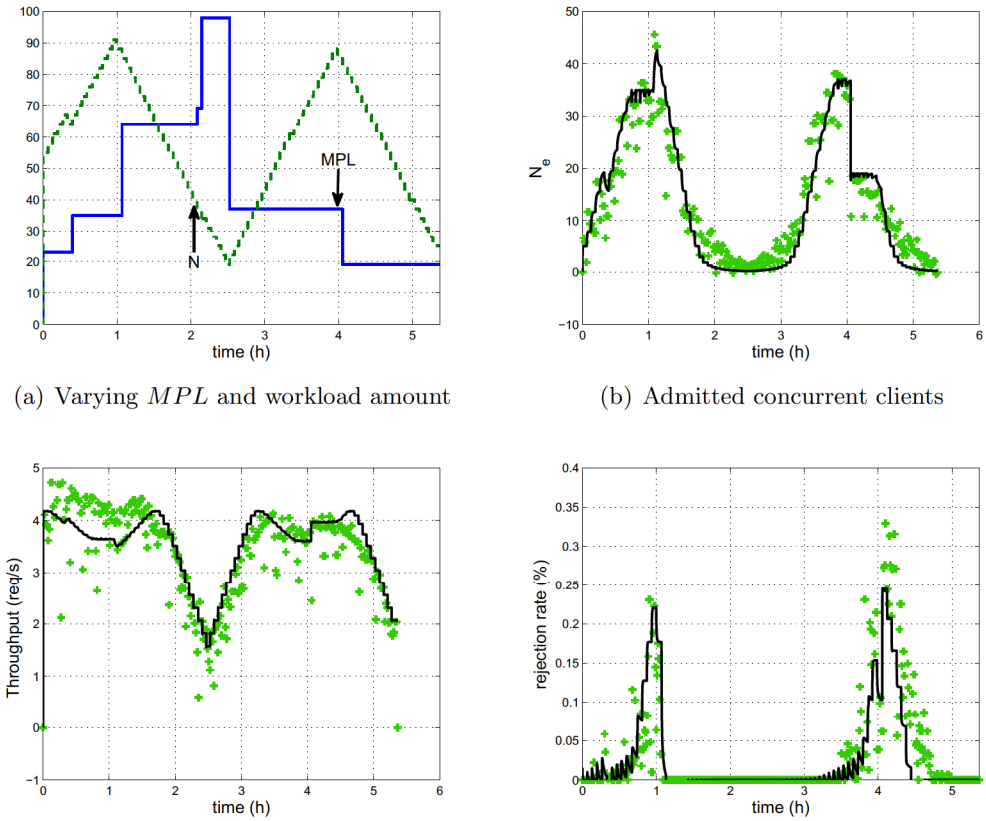
Figure 7.1: System behavior with a varying MPL and a fixed workload amount – Real system (+) versus modeled system (solid line)

Results show that the model accurately reflects the behavior of the real system. For instance, we can observe a thrashing phenomenon of the server when T_o decreases whereas N_e increases. And the model is able to render that behavior, which would not be possible without an over-linear term with respect to N_e in Equation (5.8).

7.2.2 Model validation

Figure 7.2 illustrates the case of a dynamic open loop system where both the workload amount N and the server MPL vary over time (see Figure 7.2(a)). Figures 7.2(b), 7.2(c) and 7.2(d) present the evolution over time of respectively the number N_e of concurrent clients admitted in the server, the outgoing throughput T_o and the abandon rate α, for both the real system (+) and the modeled system (solid line).

Results show that the model is able to render the dynamic behavior of the real system.



(a) Varying MPL and workload amount

(b) Admitted concurrent clients

(c) Throughput

(d) Abandon rate

Figure 7.2: System behavior with varying MPL and workload amount – Real system (+) versus modeled system (solid line)

7.3 Control Evaluation

This section presents the results of the evaluation of the implemented feedback controllers presented in Chapter 6 when applied to the PostgreSQL database server that runs the TPC-C benchmark. The results of the experiments conducted with the AM-$\mathscr{C}$ availability-maximizing controller are presented in Section 7.3.1, and then the results of the PM-$\mathscr{C}$ performance-maximizing controller are described in Section 7.3.2.

We used the same experimental environment as the one described in Section 7.1. The proposed controllers were deployed as follows. A proxy-based approach was followed to implement the AM-$\mathscr{C}$ and PM-$\mathscr{C}$ controllers where a proxy stands in front of the database server to implement real-time feedback server control.

7.3.1 *AM-𝒞* evaluation

In this section, we evaluate the proposed *AM-𝒞* availability-maximizing feedback controller presented in Section 6.2. Here, we consider a performance constraint limiting the maximum average client request latency to $L_{max} = 8s$. To ensure the stability of the controller, we initialize *AM-𝒞* with $\gamma_L = 0.1$. This value satisfies the constraint $\gamma_L < \frac{1}{L_{\max}}$ obtained in Section 6.2, and is a compromise between responsiveness and stability.

The role of *AM-𝒞* is thus to guarantee that performance constraint while maximizing service availability, through on-line feedback control of the server MPL. We use two scenarios to evaluate this controller, each one illustrating a variation of one of the two exogenous input variables of the system, i.e. the first scenario considers a changing workload mix M, and the second scenario handles a varying workload amount N.

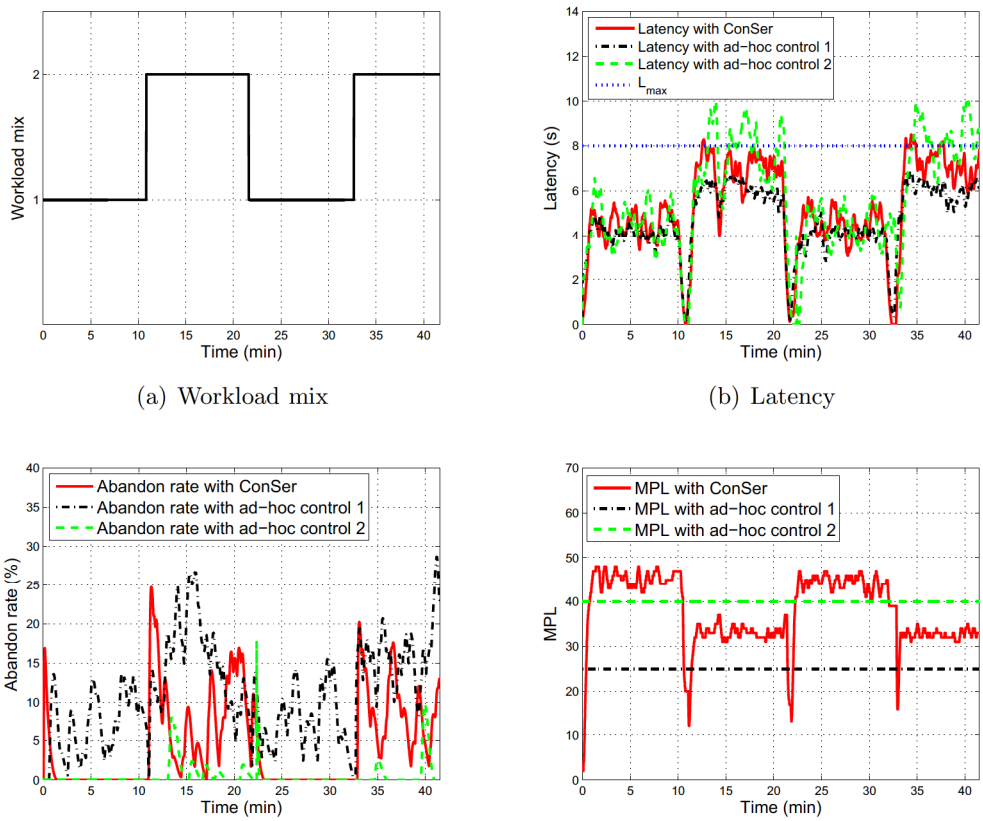
AM-𝒞 with workload mix variation

Figure 7.3 describes the first scenario where the workload mix varies from $M1$ to $M2$ twice (c.f. Figure 7.3(a)), while the workload amount N is of 80 clients. The workload mix $M1$ consists of read-only requests while the workload mix $M2$ generates read-write requests. The average request processing time differs from one mix to another. As an example, when 10 clients enter the system, the average processing time with mix $M1$ is $0.23s$ and the one with mix $M2$ is $0.55s$. Figures 7.3(d), 7.3(b) and 7.3(c) present the variation over time of respectively the server MPL, the average client request latency and the client request abandon rate, comparing two ad hoc controlled base systems, the first one with $MPL = 25$ and the second one with $MPL = 40$, with a closed loop-based controlled system. Notice that the sudden change of MPL after the 10th, 20th and 30th minutes correspond to workload mix changes, which also has an impact on the latency and abandon rate.

Results demonstrate that the *AM-𝒞* controller is able to dynamically adjust MPL in order to guarantee the latency performance constraint while keeping the service availability to its maximum, with an abandon rate minimized to 0% with $M1$ and to 10% in average with $M2$. Whereas none of the two ad hoc controlled base systems is able to guaranty the QoS, with a latency overhead of up to 25% with ad hoc control 2 and an abandon rate overhead of up to 28% with ad hoc control 1.

AM-𝒞 with workload amount variation

Figure 7.4 presents another dynamics of the system, that is the variation of the server workload amount N over time (c.f. Figure 7.4(a)) when the workload mix remains at $M2$. Figures 7.4(d), 7.4(b) and 7.4(c) present the variation over time of respectively the server MPL, the average client request latency and the client request abandon rate,



(a) Workload mix

(b) Latency

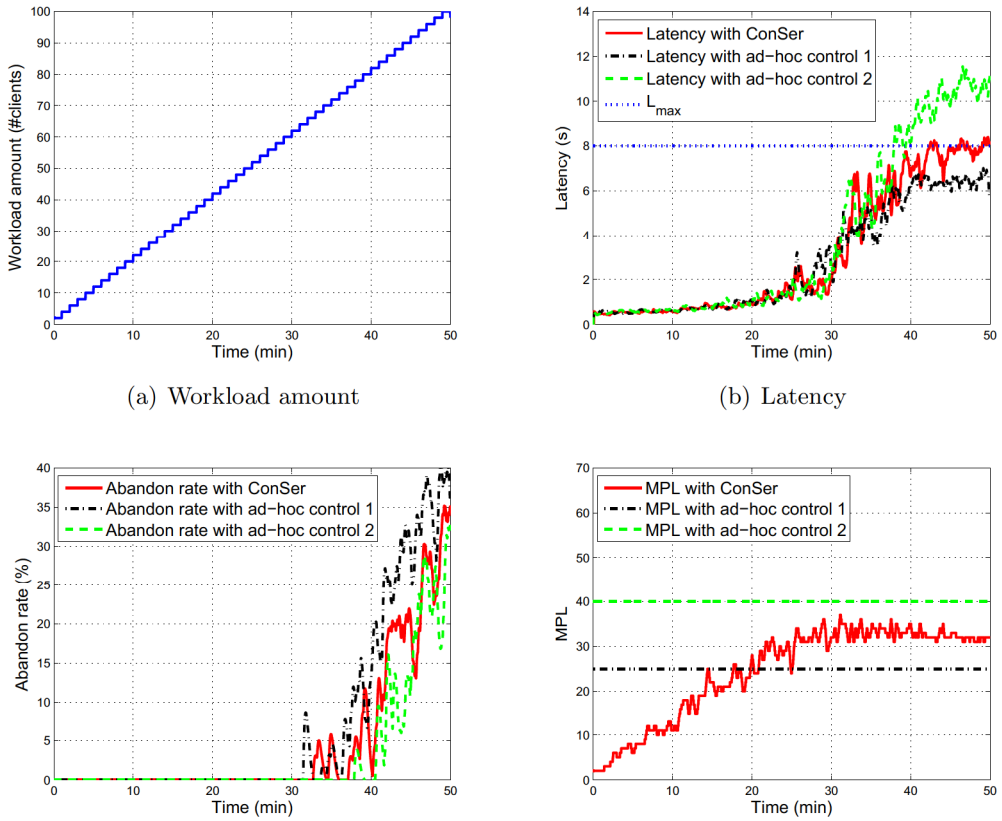
(c) Abandon rate

(d) MPL of controlled system

Figure 7.3: System behavior upon workload mix variation – *AM-𝒞*-based controlled system versus non-controlled system

comparing two ad hoc controlled base systems with the controlled system. Notice that, due to TPC-C client think time, the number of active clients at any given time may be different from (lower than) the actual load generated by TPC-C client emulator at that time. Results show that the controlled MPL is able to adjust its value to the optimal value so that the performance constraint is guaranteed. Whereas in the case of the ad hoc controlled system 1, the latency grows up to 11.5 s, with an overhead of up to 44 % compared to the controlled system. The ad hoc controlled system 2 allows to guaranty the performance constraint but the abandon rate grows up to 40 %, with an overhead of up to 14 %.

In the controlled system, the abandon rate is maintained at 0% with up to 70 clients. Then, the abandon rate increases with the increase of concurrent clients in the system, to attain its highest value when the number of clients is maximum, in order to keep latency below the target maximum latency. Notice that at the end of the experiment (between the 40th and 50th minutes), it seems justifiable to have a high abandon rate



(a) Workload amount

(b) Latency

(c) Abandon rate

(d) MPL of controlled system

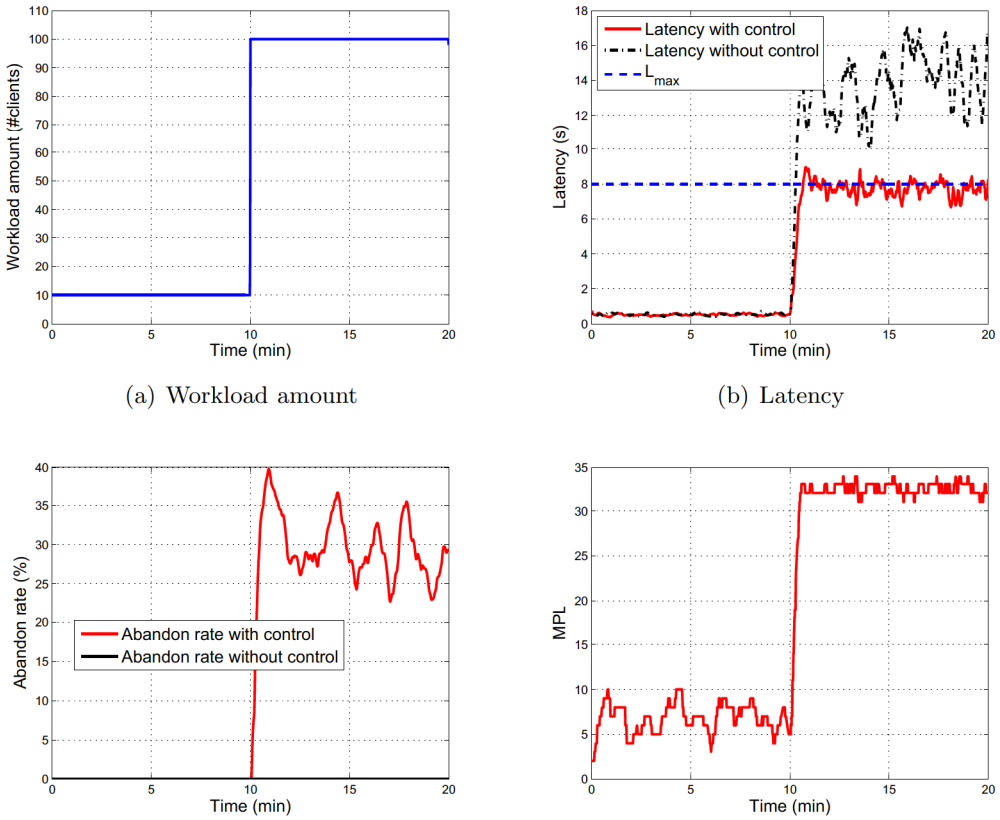
Figure 7.4: System behavior upon workload amount variation – *AM-𝒞*-based controlled system versus non-controlled system

since latency attains its maximum authorized value (c.f. Figure 7.4(d)) and client request rejection is necessary at that time to guaranty the latency constraint.

AM-𝒞 with sudden workload amount variation

Figure 7.5 presents the behavior of the system for a quick variation of workload amount (c.f. Figure 7.5(a)) when the workload mix remains at $M1$. Figures 7.5(d), 7.5(b) and 7.5(c) present the variation over time of respectively the server MPL, the average client request latency and the client request abandon rate, comparing the non-controlled base system with the controlled system. Results show that the controlled MPL is able to adjust its value to the optimal value so that the performance constraint is guaranteed. Whereas in the case of the non-controlled system, the latency grows up to 17 s, with an overhead of up to 110 % compared to the controlled system. In the controlled system,

the abandon rate is mainained at 0% until the workload amount changes (i.e. during the first half of the experiment).



(a) Workload amount

(b) Latency

(c) Abandon rate

(d) MPL of controlled system

Figure 7.5: System behavior upon quick workload amount variation – *AM-𝒞*-based controlled system versus non-controlled system

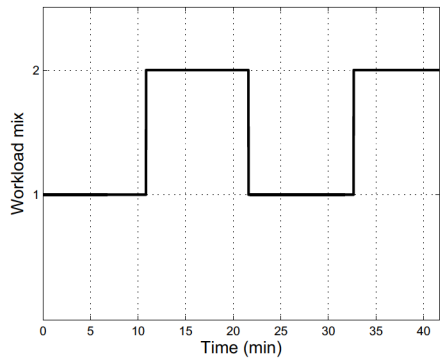
7.3.2 *PM-𝒞* evaluation

In this section, we evaluate the proposed *PM-𝒞* performance-maximizing feedback controller presented in Section 6.3. Here, we consider an availability constraint limiting the maximum client request abandon rate to $\alpha_{max} = 10\%$. To ensure the stability of the controller, we initialize *PM-𝒞* with $\gamma'_\alpha = 0.3$. This value satisfies the constraint $\gamma'_\alpha < \frac{1}{1-\alpha_{\max}}$ obtained in Section 6.3, and is a compromise between responsiveness and stability.

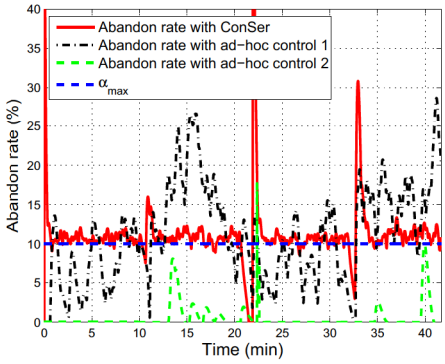
The role of *PM-𝒞* is thus to guarantee this availability constraint while maximizing service performance, through on-line feedback control of the server *MPL*.

PM-$\mathscr{C}$ with workload mix variation

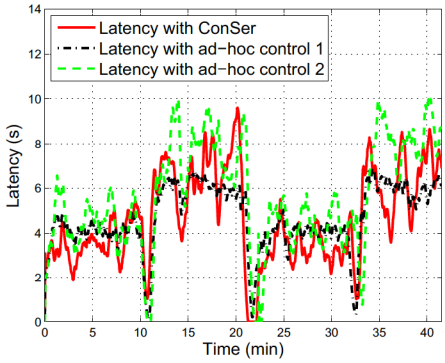
Figure 7.6 presents the variation of system behavior and dynamic control when the exogenous input variable of workload mix M changes. In Figure 7.6(a), the workload mix varies from $M1$ to $M2$ two times when the workload amount N is of 80 clients. The workload mix $M1$ consists of read-only requests while the workload mix $M2$ generates read-write requests. Figures 7.6(d), 7.6(b) and 7.6(c) present the variation over time of respectively the server MPL, the client request abandon rate and the average client request latency, comparing two ad hoc controlled base systems with a controlled system. Here again, we notice a sudden change in the MPL when the workload mix suddenly changes, with an impact on the latency and abandon rate. Results demonstrate that the PM-$\mathscr{C}$ controller is able to dynamically adjust MPL in order to guaranty the abandon rate constraint while keeping service performance to its maximum, with an average latency minimized to 4 s with $M1$ and to 6 s with $M2$. Whereas none of the two ad hoc controlled base systems is able to guaranty the QoS, with a latency overhead of up to 66% with ad-hoc control 2 at 14th and 35th minute, and an abandon rate overhead of up to 250% with ad hoc control 1 at 15th and 41th minute.



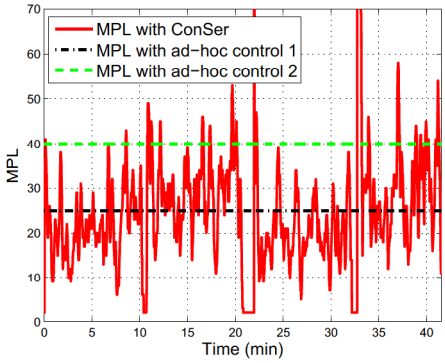
(a) Workload mix



(b) Abandon rate



(c) Latency

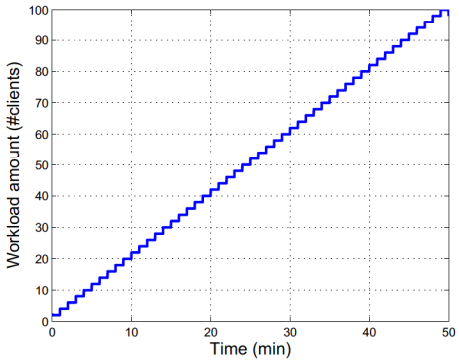


(d) MPL of controlled system

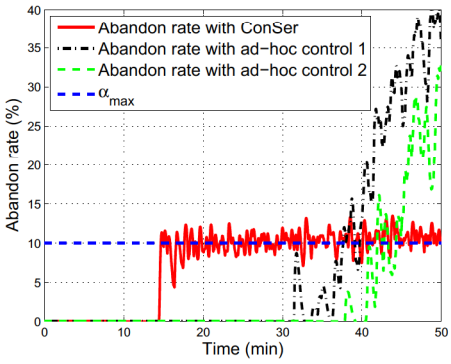
Figure 7.6: System behavior upon workload mix variation – PM-$\mathscr{C}$-based controlled system versus non-controlled system

PM-$\mathscr{C}$ with workload amount variation

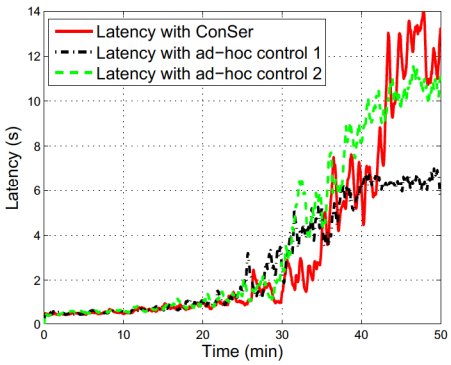
Figure 7.7 shows the variation of the system behavior with an increasing workload amount (c.f. Figure 7.7(a)) and a constant workload mix of read-write requests. Figures 7.7(d), 7.7(c) and 7.7(b) present the variation over time of respectively the server MPL, the average client request latency and the client request abandon rate, comparing the non-controlled base system with the controlled system. Results show that the controlled MPL is able to adjust its value to the optimal value so that the availability constraint is guaranteed. We can notice that before the 15th minutes of the experiment, the controlled system does not reject any request since too few clients are trying to connect to the server. In the case of the ad hoc controlled system 1 (respectively the ad hoc controlled system 2), the abandon rate grows up to 40 % (respectively 32%), with an overhead of up to 400 % (respectively 320%) compared to the controlled system.



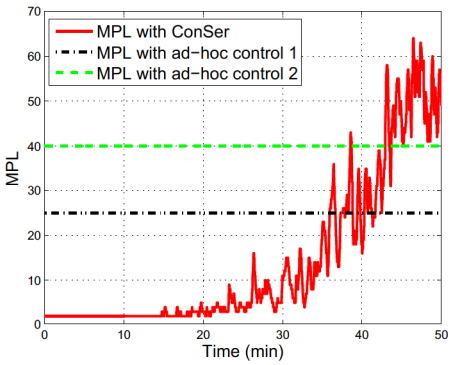
(a) Workload amount



(b) Abandon rate



(c) Latency

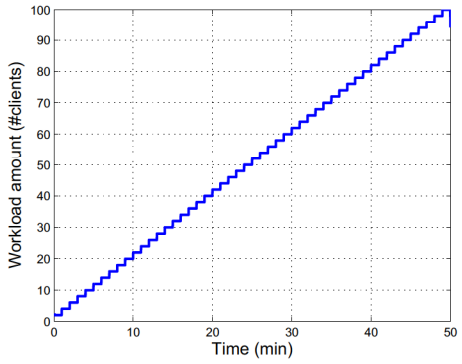


(d) MPL of controlled system

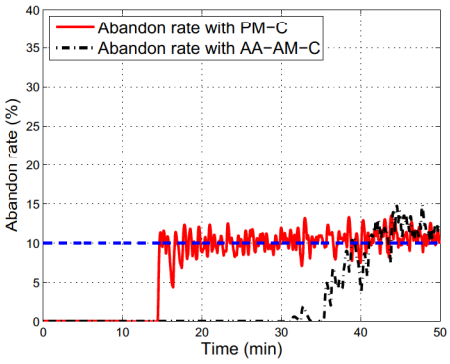
Figure 7.7: System behavior upon workload amount variation – PM-$\mathscr{C}$-based controlled system versus non-controlled system

7.3.3 *AA-PM-𝒞* evaluation

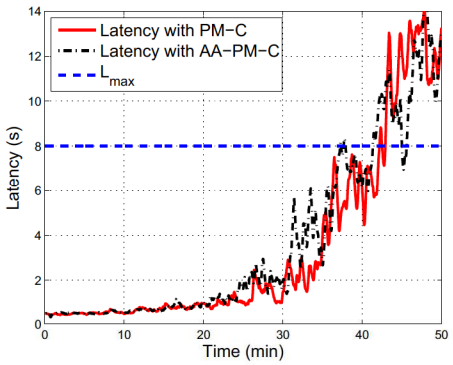
We conducted experiments with *AA-PM-𝒞* that are presented in Figure 7.8. This figure shows the variation of the system behavior according to an increasing workload amount (cf. Figure 7.8(a)) and a constant workload mix of read-write requests. The results show that while *AA-PM-𝒞* provides similar latency guaranties as *PM-𝒞* (cf. Figure 7.8(c)), the former is also able to improve service availability compared to the latter (cf. Figure 7.8(b)). Indeed, between 15th and 30th minute, *PM-𝒞* throws away 10 % of client requests against 0% for *AA-PM-𝒞*. The performance degradation during this period remains low with a 25 % overhead on the latency in average with *AA-PM-𝒞*.



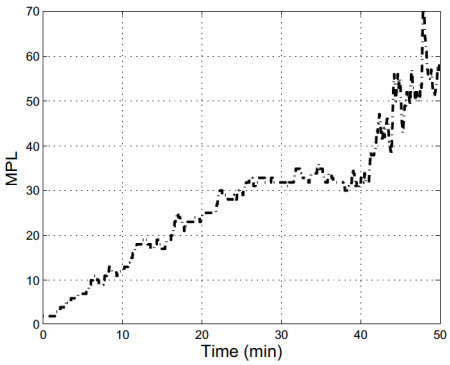
(a) Workload amount



(b) Abandon rate



(c) Latency



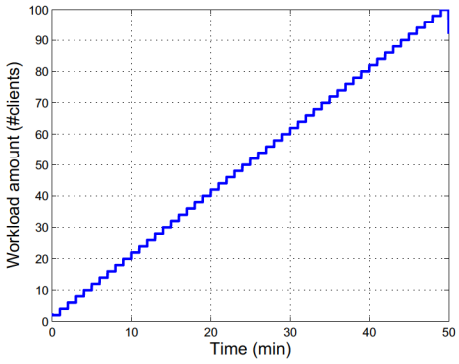
(d) MPL of controlled system with *AA-PM-𝒞*

Figure 7.8: System behavior upon workload amount variation – *AA-PM-𝒞*-based controlled system versus *PM-𝒞*-based controlled system

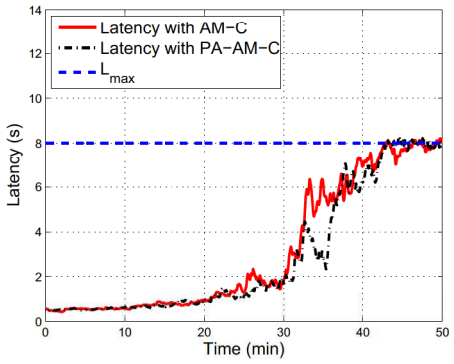
7.3.4 *PA-AM-𝒞* evaluation

We conducted experiments with the *PA-AM-𝒞* controller and we present their results in Figure 7.9. This figure shows the variation of the system behavior according to an increasing workload amount (c.f. Figure 7.9(a)) and a constant workload mix of read-write requests.

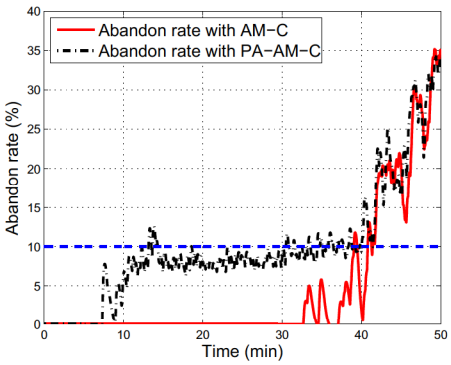
Here, *PA-AM-𝒞* specifies that latency should not exceed $L_{\max} = 8s$ and is reduced as long as abandon rate remains below $\alpha_{\max} = 10\%$. Figure 7.9(a) shows that the server workload amount is increasing over time while the workload mix remains at $M2$. Figures 7.9(b) and 7.9(c) show that during the first 40 minutes of the experiment the abandon rate with *PA-AM-𝒞* remains below 10% while the latency is slightly improved compared to *AM-𝒞*. Indeed, latency may be reduced by up to 54%.



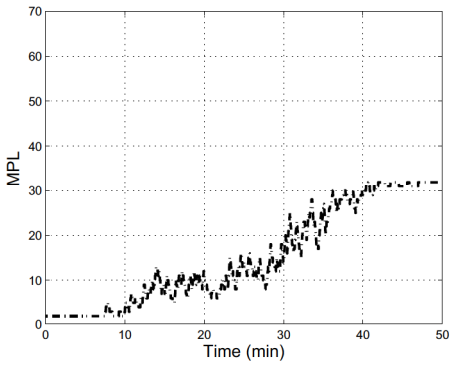
(a) Workload amount



(b) Latency



(c) Abandon rate



(d) MPL of controlled system with *PA-AM-𝒞*

Figure 7.9: System behavior upon workload amount variation – *PA-AM-𝒞*-based controlled system versus *AM-𝒞*-based controlled system

7.4 Summary

In this Chapter we have experimentally evaluated the model developed in Chapter 5 and the control laws designed in Chapter 6. We first identified the model parameters using an input-output data set. We successfully validated the proposed fluid model and control laws to dynamically control an on-line database server with various scenarios.

Part III

Distributed Server Modeling and Control

Chapter 8

Distributed Server Modeling

8.1 Methodology

In this Chapter, we extend results of the previous Chapters to systems with multiple servers, such as multi-tier distributed servers (c.f. Section 2.1.3). Whereas using fluid approximation to model a server is rare in the literature and quasi restricted to linear approximations, as far as we know, this thesis is the first study to derive a fluid model for multi-tier servers. This chapter proposes a dynamic model of multi-tier server systems based on the model proposed in the previous Chapter. Using fluid approximations, we derived differential equations that allow to render the of such systems.

8.2 Distributed server model

8.2.1 Model structure

A multi-tier server system consists in a series of n servers serially connected through a communication network. Here, external clients send requests to the front-end server, which itself acts as a client of a back-end server (see Figure 8.1). As a reminder of Sections 2.1.3 and 2.3, the following properties characterize such systems:

(P1) Requests arriving at tier 1 come from the clients. Requests arriving at tier k, with $k > 1$, come from tier $k - 1$. Responses arriving at the clients come from tier 1. Responses arriving at tier k, with $k < n$, come from tier $k + 1$ (see Figure 2.3).

(P2) The handling of a request by tier k, with $k > 1$, is terminated when the associated response is sent to tier $k - 1$. The handling of a client request by a distributed server is terminated when the associated response is sent to the client by the server of the first tier.

(P3) A request being processed at tier k can generate several requests sequentially at tier $k+1$.

(P4) At any time, a request being processed at tier k can generate at most one request at tier $k+1$.

(P5) As a result of P(4), the number of current requests at tier $k+1$ is less or equal than the number of current requests at tier k (see figure 2.4): $\forall k \in [1; n-1], N_k \geq N_{k+1}$.

(P6) A client request rejected (because of the MPL limit) at tier k will cause the rejection of the mother request at tier $k-1$, with $k > 1$.

For each tier k, we define:

- T_k^i, the incoming throughput at tier k.
- T_k^o, the outgoing throughput at tier k.
- T_k^r, the rejection throughput caused by MPL at tier k.
- N_k, the number of concurrent requests admitted at tier k.
- MPL_k, the server's MPL at tier k.
- λ_k, the probability that a request being processed at tier $k-1$ generates requests at tier k.
- V_k, if a request on tier $k-1$ generates subsequent requests on tier k, the average number of subsequent requests is V_k.
- S_k, the processing time of a request on tier k.

Basically, the incoming, outgoing and rejection throughputs, respectively T_k^i, T_k^o and T_k^r are expressed in number of requests per second. N_k and MPL_k are positive integers. $\lambda_k \in \mathbb{R}$ belongs to the interval $[0, 1]$ and S_k is a time in seconds. From property P(6) we deduce that the rejection throughput T^r of the whole system can be expressed as

$$T^r = \sum_{k=1}^{n} T_k^r$$

8.2.2 Model state variables

In Figure 8.2, we propose a methodology to model the dynamic behavior of the system. Each server is considered as a group of tanks. The numbered arrows of figure 8.2 represent the flows of requests in a three-tier system as follows:

(1) Incoming client requests arrive at tier 1.

(1') Requests may be rejected at tier 1 because of MPL_1.

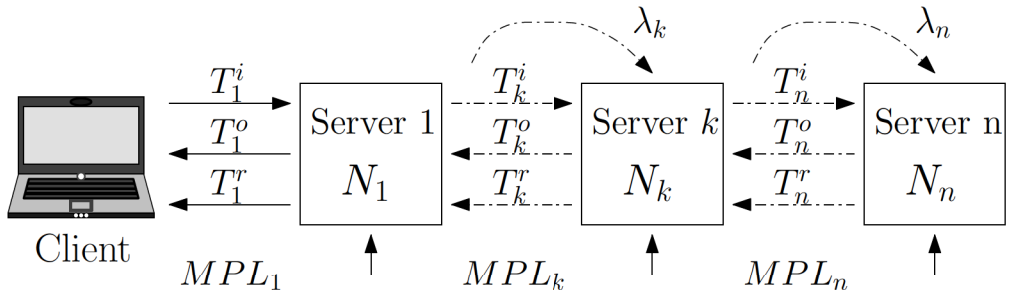


Figure 8.1: Model inputs/outputs and internal states

2 Some requests accepted at tier 1 generate subsequent requests at tier 2.

2' Other requests accepted at tier 1 do not generate subsequent requests at tier 2.

3 Requests sent to tier 2.

3' Requests rejected at tier 2 because of MPL_2.

4 Requests accepted at tier 2.

5 Requests processed by tier 2 that are sent back to tier 1 for a final processing.

5' Requests processed by tier 2 that are sent back to tier 1 for further processing.

6 Requests processed by tier 1 that are sent back to the client.

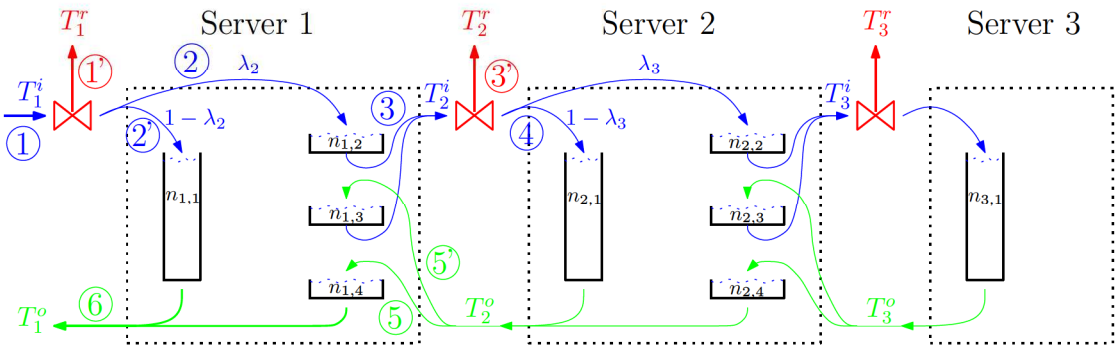


Figure 8.2: Request flow in a three-tier distributed system

For $k < n$ we have:

$$\dot{n}_{k,1} = (1 - \lambda_{k+1}) \cdot (T_k^i - T_k^r) - \frac{n_{k,1}}{S_k} \tag{8.1}$$

$$\dot{n}_{k,2} = \lambda_{k+1} \cdot (T_k^i - T_k^r) - \frac{n_{k,2}}{S_k} \tag{8.2}$$

$$\dot{n}_{k,3} = \frac{V_{k+1} - 1}{V_{k+1}} \cdot T_{k+1}^o - \frac{n_{k,3}}{S_k} \tag{8.3}$$

$$\dot{n}_{k,4} = \frac{1}{V_{k+1}} \cdot T_{k+1}^o - \frac{n_{k,4}}{S_k} \tag{8.4}$$

And

$$\dot{n}_{n,1} = T_n^i - T_n^r - T_n^o \tag{8.5}$$

With

$$T_n^o = \frac{n_{n,1}}{S_n} \tag{8.6}$$

$$T_{k+1}^i = \frac{n_{k,2} + n_{k,3}}{S_k} for 1 \leq k < n \tag{8.7}$$

$$T_k^o = \frac{n_{k,1} + n_{k,4}}{S_k} for 1 \leq k < n \tag{8.8}$$

And with the processing time of requests on server k

$$S_k = \begin{cases} a_k (N_k - N_{k+1}) + b_k & \text{for } k < n \\ a_n (N_n) + b_n \end{cases} \tag{8.9}$$

Where

$$N_k = \sum_{i=k}^{n} \sum_{j=1}^{4} n_{i,j} \tag{8.10}$$

8.2.3 Model output variables

We are interested in two QoS metrics that reflect service performance and service availability, respectively, client request latency and client request abandon rate. According to the transition probability described before, the average client request latency is

$$L = \sum_{k=1}^{n} S_k \cdot \prod_{j=1}^{k} \lambda_j V_j$$

By definition, the abandon rate is the ratio between requests rejected due to admission control and the total number of requests received by the system. It may be written as

$$\alpha = \frac{T^r}{T_1^i}$$

8.3 Summary

Based on the distributed server model introduced in this Chapter, we will discuss how to control the MPL of the servers at the different tiers of such systems in Chapter 9.

Chapter 9

Distributed Server Control

9.1 Motivation

In Chapter 6, we presented different MPL control strategies for central servers. As we have seen in Chapter 8, distributed server systems are more complex to model, and they also exhibit more control inputs that can be used to achieve given objectives. These control inputs are the different $MPLs$ at each tier of the system. Our approach consists in applying a centralized control policy, where the knowledge of the global system's state is used to compute each MPL. In the following, we study the trade-off between performance and availability of distributed server systems for this approach, and derive the optimal MPL control based on the proposed fluid model. This is the optimal number of concurrent clients admitted to the server at each tier of the system with respect to a given trade-off.

9.2 dAM-$\mathscr{C}$: availability-maximizing control of distributed servers

We propose the dAM-$\mathscr{C}$ distributed availability-maximizing control that aims to provide the following objectives:

(O1) the average client request latency does not exceed a maximum latency L_{max}, and

(O2) the rejection throughput T^r is made as small as possible

We state and solve an optimization problem to meet the previous objectives. We assume that all the parameters of the model are known. Practically, an on-line identification of these parameters would be necessary if the nature of the workload may change over time. Otherwise, an off-line identification is sufficient. The optimization problem considered here is to find $MPL^\star = (MPL_1^\star, MPL_2^\star, ..., MPL_n^\star)$ that minimizes the rejection

throughput at tier 1 and maintains the client requests average latency under L_{max} when the system faces a high load. For that we consider the system is saturated, which means

$$\forall k \in [1; n], \quad N_k = MPL_k$$

We express the steady-state of the system as follows:

$$\forall (i, j) \in [1; n] \times [1; 4], \quad \dot{n}_{i,j} = 0$$

That is

$$0 = T_1^i - T_1^o - T^r$$

From 8.10 and 8.1 we get

$$T_1^o = \sum_{k=1}^{n} (1 - \lambda_{k+1}) \left[\prod_{j=1}^{k} \lambda_j - \frac{V_j - 1}{V_j} \right] \frac{MPL_k - MPL_{k+1}}{S_k} + \left[\prod_{k=1}^{n} \lambda_k - \frac{V_k - 1}{V_k} \right] \frac{MPL_n}{S_n} \tag{9.1}$$

Then the cost function to minimize, that is the global rejection throughput T^r, may be written as:

$$T^r = T_1^i - T_1^o \tag{9.2}$$

where

$$S_k(t) = \begin{cases} a_k \left[MPL_k(t) - MPL_{k+1}(t) \right] + b_k & \text{for } k = 1, \ldots, n-1 \\ a_n MPL_n(t) + b_n & \text{for } k = n \end{cases}$$

In order to force the system not to exceed a given maximal latency L_{max} specified in the SLO, the constraint to consider is

$$L \leq L_{max} \tag{9.3}$$

where L is defined by:

$$L = \sum_{k=1}^{n} S_k \cdot \prod_{j=1}^{k} \lambda_j V_j \tag{9.4}$$

Taking $MPL_{n+1} := 0$, the definition of S_k becomes:

$$S_k(t) = a_k \left[MPL_k(t) - MPL_{k+1}(t) \right] + b_k \quad \text{for } k = 1, \ldots, n$$

For $N_k = MPL_k$, it gives in equation (9.4):

$$\begin{aligned} L &= \sum_{k=1}^{n} \left[a_k \left[MPL_k - MPL_{k+1} \right] + b_k \right] \cdot \prod_{j=1}^{k} \lambda_j V_j \\ &= a_1 \lambda_1 V_1 MPL_1 + \sum_{k=2}^{n} a_k MPL_k \cdot \prod_{j=1}^{k} \lambda_j V_j - \sum_{k=1}^{n} a_k MPL_{k+1} \cdot \prod_{j=1}^{k} \lambda_j V_j + \sum_{k=1}^{n} \left(\prod_{j=1}^{k} \lambda_j V_j \right) b_k \\ &= a_1 \lambda_1 V_1 MPL_1 + \sum_{k=1}^{n-1} \left[\prod_{j=1}^{k} \lambda_j V_j \right] \cdot (a_{k+1} \lambda_{k+1} V_{k+1} - a_k) \cdot MPL_{k+1} + \sum_{k=1}^{n} \left(\prod_{j=1}^{k} \lambda_j V_j \right) b_k \end{aligned} \tag{9.5}$$

To solve this problem we write the first order Karush-Kuhn-Tucker necessary conditions [24]. Assuming $MPL^\star$ is a local minimum for the optimization problem, then there exists a unique $\mu^\star$ such that

$$\begin{aligned} \nabla_{MPL}\mathcal{L}(MPL^\star, \mu^\star) &= 0 && (9.6)\\ \mu^\star \cdot (L(MPL^\star) - L_{max}) &= 0 && (9.7)\\ \mu^\star &\geq 0 && (9.8) \end{aligned}$$

where the Lagrangian is given by $\mathcal{L} = T^r + \mu\,(L - L_{max})$.

By differentiating the Lagrangian with respect to $MPL = (MPL_1\, MPL_2\, \ldots\, MPL_n)$ we get, using the fact that the input throughput T_1^i is not dependent of the MPL's choices:

$$\nabla_{MPL}\mathcal{L} = \begin{pmatrix} -\frac{(\lambda_1 - \frac{V_1-1}{V_1})(1-\lambda_2)b_1}{(a_1\Delta_1+b_1)^2} + \mu a_1\lambda_1 V_1 \\ \frac{(\lambda_1 - \frac{V_1-1}{V_1})(1-\lambda_2)b_1}{(a_1\Delta_1+b_1)^2} - \frac{(\lambda_1 - \frac{V_1-1}{V_1})(\lambda_2 - \frac{V_2-1}{V_2})(1-\lambda_3)b_2}{(a_2\Delta_2+b_2)^2} + \mu\lambda_1 V_1(a_2\lambda_2 V_2 - a_1) \\ \frac{\left[\prod_{k=1}^{2}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_3)b_2}{(a_2\Delta_2+b_2)^2} - \frac{\left[\prod_{k=1}^{3}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_4)b_3}{(a_3\Delta_3+b_3)^2} + \mu\prod_{j=1}^{2}\lambda_j V_j(a_3\lambda_3 V_3 - a_2) \\ \vdots \\ \frac{\left[\prod_{k=1}^{n-2}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_{n-1})b_{n-2}}{(a_{n-2}\Delta_{n-2}+b_{n-2})^2} - \frac{\left[\prod_{k=1}^{n-1}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_n)b_{n-1}}{(a_{n-1}\Delta_{n-1}+b_{n-1})^2} + \mu\prod_{j=1}^{n-2}\lambda_j V_j(a_{n-1}\lambda_{n-1}V_{n-1} - a_{n-2}) \\ \frac{\left[\prod_{k=1}^{n-1}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_n)b_{n-1}}{(a_{n-1}\Delta_{n-1}+b_{n-1})^2} - \frac{\left[\prod_{k=1}^{n}(\lambda_k - \frac{V_k-1}{V_k})\right]b_n}{(a_n MPL_n+b_n)^2} + \mu\prod_{j=1}^{n-1}\lambda_j V_j(a_n\lambda_n V_n - a_{n-1}) \end{pmatrix} \quad (9.9)$$

where $\Delta_k = MPL_k - MPL_{k+1}, k < n$.

Equation (9.6) gives for $\mu^\star \neq 0$ at line k of (9.9):

$$MPL_k^\star = MPL_{k+1}^\star + \frac{1}{a_k}\cdot\left(-b_k + \sqrt{\frac{b_k(1-\lambda_{k+1})\prod_{j=1}^{k}(\lambda_j - \frac{V_j-1}{V_j})}{\mu^\star a_k\prod_{j=1}^{k}\lambda_j V_j}}\right) \quad (9.10)$$

and at line n:

$$MPL_n^\star = \frac{1}{a_n}\cdot\left(-b_n + \sqrt{\frac{b_n\prod_{k=1}^{n}(\lambda_k - \frac{V_k-1}{V_k})}{\mu^\star a_n\prod_{k=1}^{n}\lambda_k V_k}}\right) \quad (9.11)$$

Then equation (9.7) gives for $\mu^\star > 0$

$$L(MPL^\star) = L_{max} \quad (9.12)$$

Putting (9.10) and (9.11) in (9.12) gives after some fastidious calculations:

$$\mu^\star = \frac{1}{L_{max}^2}\left[\sum_{k=1}^{n-1}\sqrt{(1-\lambda_{k+1})\prod_{j=1}^{k}\left(\lambda_j - \frac{V_j-1}{V_j}\right)\prod_{j=1}^{k}\lambda_j V_j \frac{b_k}{a_k}} + \sqrt{\prod_{k=1}^{n}\left(\lambda_k - \frac{V_k-1}{V_k}\right)\prod_{k=1}^{n}\lambda_k V_k \frac{b_n}{a_n}}\right]^2 \quad (9.13)$$

We finally obtain an explicit expression of $MPL^\star$ with (9.11), (9.10) and (9.13). This minimizes the rejection throughput T^r ensuring the latency is lower than the SLO's requirements assuming all tiers at each level are saturated.

9.3 Discussion

9.3.1 Some hints to use the $MPL^\star$

There is probably many ways to use the optimal configuration $MPL^\star$. The aim of this subsection is to give the two more intuitive ones.

A first solution consist in directly setting the $MPL^\star = (MPL_1^\star\, MPL_2^\star \ldots MPL_n^\star)$ on the n tiers. This is probably the simplest one but probably also the less robust since there is no closed-loop control on each tier. Results using this approach are given in the next chapter.

A second approach consists in finding a fictive maximal latency $L_{k_{max}}$ for each tier that achieves a global latency of L_{max} latency. Using the expression of the latency (9.4), it follows that taking

$$L_{k_{max}} := S_k \prod_{j=1}^{k}\lambda_j V_j \quad (9.14)$$

and when all tiers are saturated, the global latency is such that:

$$L = L_{max} = \sum_{i=1}^{n} L_{k_{max}} = \sum_{k=1}^{n} S_k \prod_{j=1}^{k}\lambda_j V_j$$

Clearly, using *AM-𝒞* availability-maximizing control presented in section 6.2 page 47 with maximal latency set-point $L_{k_{max}}$ will ensure that the average latency L_k of tier k will remain below $L_{k_{max}}$. The overall latency will, that way, remain below the latency L_{max}.

9.3.2 On the saturated tiers assumption

Clearly, the optimal control result relies on the assumption that all tiers are saturated, that is $N_k = MPL_k$ for all k. Applying this control approach when the tiers are not saturated may, however, lead to over-conservative strategy. Indeed, assuming the tier i is not saturated:

$$N_i < MPL_i$$

The latency introduced by the data processing on the tier will lower that expected. It is given by:

$$S_k = \begin{cases} a_k \left(N_k - N_{k+1}\right) + b_k & \text{for } k < n \\ a_n \left(N_n\right) + b_n \end{cases}$$

As previously, defining $N_{k+1} := 0$, it unifies the expression for all $k \in \{0, \ldots, n\}$:

$$S_k = a_k \left(N_k - N_{k+1}\right) + b_k$$

If all the tiers are saturated except the ith tier, it gives:

$$S_k = \begin{cases} a_k \left(MPL_k - MPL_{k+1}\right) + b_k & \text{for } k \neq \{i, i-1\} \\ a_i \left(N_i - MPL_{i+1}\right) + b_i \\ a_{i-1} \left(MPL_{i-1} - N_i\right) + b_{i-1} \end{cases}$$

In the throughput expression (9.1) it gives:

$$\begin{aligned} T_1^o = \sum_{\substack{k=1 \\ k \neq i, i+1}}^{n} (1 - \lambda_{k+1}) \left[\prod_{j=1}^{k} \lambda_j - \frac{V_j - 1}{V_j} \right] \frac{MPL_k - MPL_{k+1}}{S_k} \\ + (1 - \lambda_{i+1}) \left[\prod_{j=1}^{i} \lambda_j - \frac{V_j - 1}{V_j} \right] \frac{N_i - MPL_{i+1}}{S_i} \\ + (1 - \lambda_i) \left[\prod_{j=1}^{i-1} \lambda_j - \frac{V_j - 1}{V_j} \right] \frac{MPL_{i-1} - N_i}{S_{i-1}} \\ + \left[\prod_{k=1}^{n} \lambda_k - \frac{V_k - 1}{V_k} \right] \frac{MPL_n}{S_n} \end{aligned} \quad (9.15)$$

and in the latency expression (9.5):

$$\begin{aligned} L = a_1 \lambda_1 V_1 MPL_1 + \sum_{\substack{k=1 \\ k \neq i-1}}^{n-1} \left[\prod_{j=1}^{k} \lambda_j V_j \right] \cdot \left(a_{k+1} \lambda_{k+1} V_{k+1} - a_k\right) \cdot MPL_{k+1} \\ + \sum_{k=1}^{n} \left(\prod_{j=1}^{k} \lambda_j V_j \right) b_k + \left[\prod_{j=1}^{i-1} \lambda_j V_j \right] \cdot \left(a_i \lambda_i V_i - a_{i-1}\right) \cdot N_i \end{aligned} \quad (9.16)$$

Recomputing the partial derivative of the Lagragrian $\mathcal{L}$ with respect to the tiers MPL's gives:

$$\nabla_{MPL}\mathcal{L} = \begin{pmatrix} -\frac{(\lambda_1-\frac{V_1-1}{V_1})(1-\lambda_2)b_1}{(a_1\Delta_1+b_1)^2} + \mu a_1\lambda_1 V_1 \\ \vdots \\ \frac{\left[\prod_{k=1}^{i-3}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{i-2})b_{i-3}}{(a_{i-3}\Delta_{i-3}+b_{i-3})^2} - \frac{\left[\prod_{k=1}^{i-2}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{i-1})b_{i-2}}{(a_{i-2}\Delta_{i-2}+b_{i-2})^2} + \mu\prod_{j=1}^{i-3}\lambda_j V_j(a_{i-2}\lambda_{i-2}V_{i-2} - a_{i-3}) \\ \frac{\left[\prod_{k=1}^{i-2}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{i-1})b_{i-2}}{(a_{i-2}\Delta_{i-2}+b_{i-2})^2} - \frac{\left[\prod_{k=1}^{i-1}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_i)b_{i-1}}{(a_{i-1}[MPL_{i-1}-N_i]+b_{i-1})^2} + \mu\prod_{j=1}^{i-2}\lambda_j V_j(a_{i-1}\lambda_{i-1}V_{i-1} - a_{i-2}) \\ 0 \\ \frac{\left[\prod_{k=1}^{i}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{i+1})b_i}{(a_i[N_i-MPL_{i+1}]+b_i)^2} - \frac{\left[\prod_{k=1}^{i+1}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{i+2})b_{i+1}}{(a_{i+1}\Delta_{i+1}+b_{i+1})^2} + \mu\prod_{j=1}^{i}\lambda_j V_j(a_{i+1}\lambda_{i+1}V_{i+1} - a_i) \\ \vdots \\ \frac{\left[\prod_{k=1}^{n-2}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_{n-1})b_{n-2}}{(a_{n-2}\Delta_{n-2}+b_{n-2})^2} - \frac{\left[\prod_{k=1}^{n-1}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_n)b_{n-1}}{(a_{n-1}\Delta_{n-1}+b_{n-1})^2} + \mu\prod_{j=1}^{n-2}\lambda_j V_j(a_{n-1}\lambda_{n-1}V_{n-1} - a_{n-2}) \\ \frac{\left[\prod_{k=1}^{n-1}(\lambda_k-\frac{V_k-1}{V_k})\right](1-\lambda_n)b_{n-1}}{(a_{n-1}\Delta_{n-1}\,|\,b_{n-1})^2} - \frac{\left[\prod_{k=1}^{n}(\lambda_k-\frac{V_k-1}{V_k})\right]b_n}{(a_n MPL_n\,|\,b_n)^2} + \mu\prod_{j=1}^{n-1}\lambda_j V_j(a_n\lambda_n V_n - a_{n-1}) \end{pmatrix} \tag{9.17}$$

Since at the optimum, this partial derivative of the Lagrangian vanishes, (9.10) keeps unchanged for $k < i-2$ while $MPL^\star_{i-1}$ is given by:

$$MPL^\star_{i-1} = N_i + \frac{1}{a_{i-1}} \cdot \left(-b_{i-1} + \sqrt{\frac{b_{i-1}(1-\lambda_i)\prod_{j=1}^{i-1}(\lambda_j - \frac{V_j-1}{V_j})}{\mu^\star a_{i-1}\prod_{j=1}^{i-1}\lambda_j V_j}} \right) \tag{9.18}$$

where $\mu^\star$ still denotes the optimal value of μ. For $k > i+1$, $\nabla_{MPL}\mathcal{L}(MPL^\star, \mu^\star) = 0$ gives summing the $n-i$ last row of (9.17):

$$\frac{\left[\prod_{k=1}^{i}(\lambda_k - \frac{V_k-1}{V_k})\right](1-\lambda_{i+1})b_i}{(a_i[N_i - MPL^\star_{i+1}] + b_i)^2} - \frac{\left[\prod_{k=1}^{n}(\lambda_k - \frac{V_k-1}{V_k})\right]b_n}{(a_n MPL^\star_n + b_n)^2} + \mu^\star\left[a_n\prod_{j=1}^{n}(\lambda_j - \frac{V_j-1}{V_j}) - a_i\prod_{j=1}^{i}(\lambda_j - \frac{V_j-1}{V_j})\right] = 0 \tag{9.19}$$

Chapter 10

Numerical Evaluation: Distributed Server Modeling and Control

10.1 Numerical setup

In this Chapter, we simulate a three-tier server system with the model described in Chapter 8. We chose the following model parameters for this numerical evaluation:

$a_1 = 0.04$
$b_1 = 0.0125$
$a_2 = 0.08$
$b_2 = 0.025$
$a_3 = 0.08$
$b_3 = 0.05$

10.2 dAM-$\mathscr{C}$ evaluation

In this section, we evaluate the proposed dAM-$\mathscr{C}$ distributed availability-maximizing optimal controller presented in Chapter 9. Here, we consider a performance constraint limiting the maximum average client request latency to $L_{max} = 8s$. The role of dAM-$\mathscr{C}$ is thus to guarantee that performance constraint while maximizing service availability, through server MPL control. We use two scenarios to evaluate this controller, each one illustrating a variation of one of the two exogenous input variables of the system. The first scenario considers a changing workload mix M, and the second scenario handles a varying workload amount N.

dAM-𝒞 with workload mix variation

Figure 10.1 describes the first scenario where the workload mix varies from $M1$ to $M2$ twice (c.f. Figure 10.1(a)) while the incoming throughput of client requests is 30 requests per second. Workload mix $M1$ represents a mix in which 85% of service requests at tier 1 generate subsequent requests at tier 2, that is $\lambda_2 = 0.85$. Workload mix $M2$ represent a mix in which 95% of service requests at tier 1 generate subsequent requests at tier 2, that is $\lambda_2 = 0.95$. For both mixes, V_2 and V_3 are equal to 1.5. The average processing time differs from one mix to another since the load distribution across the system tiers differs.

Figures 10.1 present the variation over time of the average client request latency and the client request abandon rate, comparing three ad hoc controlled base systems against the proposed optimized system. Table 10.2 gives the different MPL for each configuration:

	$MPL1$	$MPL2$	$MPL3$
ad-hoc control 1	96	96	96
ad-hoc control 2	40	40	40
ad-hoc control 3	50	30	10
optimal control for Mix 1	96	40	24
optimal control for Mix 2	74	41	24

Table 10.1: MPL Configurations

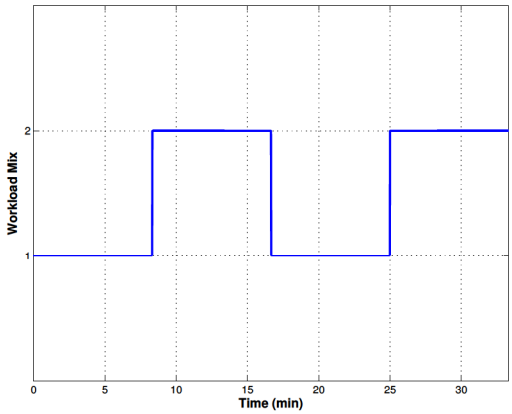
Ad hoc control 1 is a front-tier admission control policy that allows 96 concurrent client requests in the system. Figure 10.1(b) this configuration does not meet the objectives since the average client requests latency is far above 8 seconds for both mixes (18.5s for mix M1 and 20.6s for mix M2).

Ad hoc control 2 is a more restrictive front-tier admission control that keeps the latency below the 8s limit for mix M1 (7.6s) but not for mix M2 (8.5s).

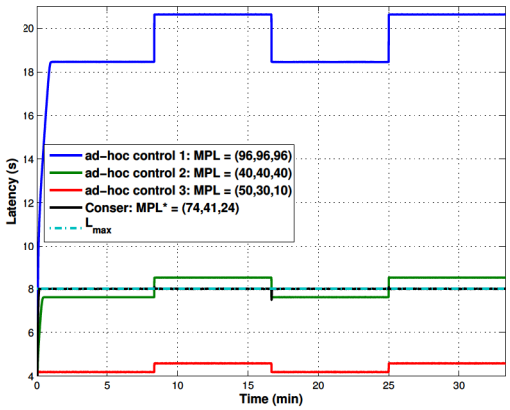
Ad hoc control 3 is a distributed control policy which is more efficient than the two previous strategies since it keeps the client request latency below the limit and allows to reject less requests (see Figure 10.1(c)). Compared to these ad hoc controls, ConSer optimal control provides the best compromise since it meets the performance objective, keeping the latency at the 8s limit, while maximizing the system availability, with an abandon rate between 73 and 76% for both mixes.

dAM-𝒞 with workload amount variation

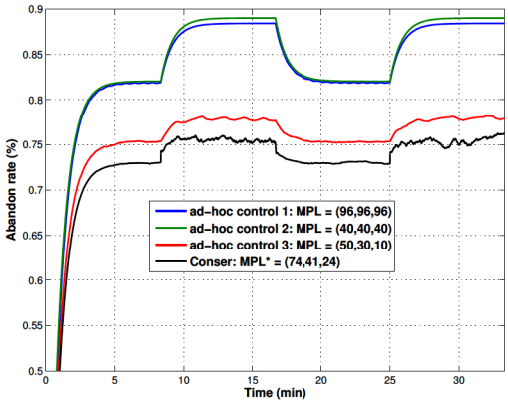
Figure 10.2 presents the evolution of the system when the server workload amount N is varying over time (c.f. Figure 10.2(a)) and the workload mix remains at $M1$. Figures 10.2(b) and 10.2(c) present the variation over time of the average client request latency and the client request abandon rate, comparing three ad hoc controlled base



(a) Workload Mix



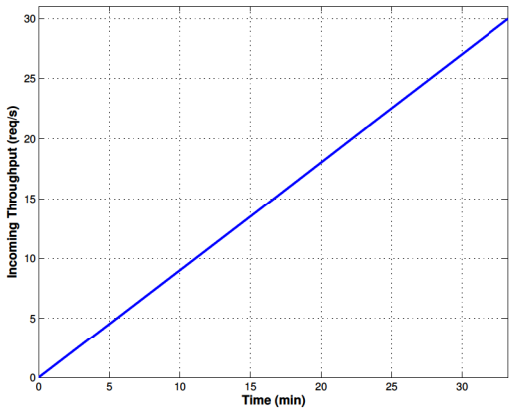
(b) Latency



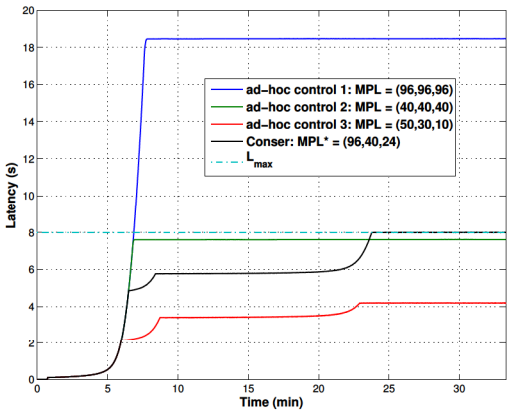
(c) Abandon rate

Figure 10.1: System behavior upon workload mix variation – *dAM-$\mathscr{C}$*-based controlled system vs. ad hoc controlled system

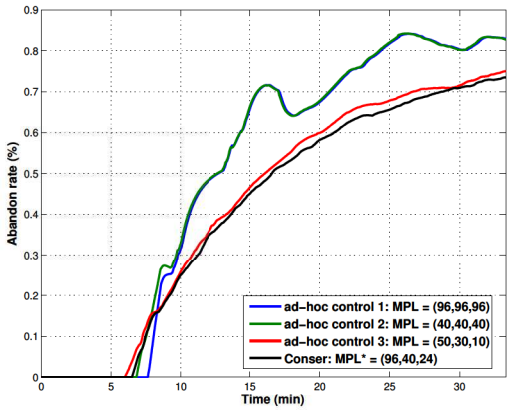
systems with ConSer base system. Results show that ConSer, ad hoc control 2 or ad hoc control 3 base systems meet the performance objective while ad hoc control 1 system does not achieve this goal, with a client request latency up to 18.5s. Figure 10.2(c) show that from the 8^{th} minute of the simulation, the client request abandon rate obtained with ConSeris lower than the one obtained with any of the proposed ad hoc control. The abandon rate obtained with ConSer is in average 13% lower than the one with ad hoc control 1 and ad hoc control 2, 1.5% lower than the one with ad hoc control 3.



(a) Workload Amount



(b) Latency



(c) Abandon rate

Figure 10.2: System behavior upon workload amount variation – *dAM-𝒞*-based controlled system vs. ad hoc controlled system

Part IV

Conclusions and Perspectives

Chapter 11

Conclusion

11.1 Summary

This thesis is a contribution to the management, quality-of-service and optimization of computing server systems, such as database servers, web servers, etc.

Server systems can host many different applications that can be accessed remotely through a network. The Internet is an environment that makes an intensive usage of such systems, and this usage has grown dramatically in recent years. The development of new technologies, such as the mobile Internet, and also the expansion of the range of cloud services that are available might explain that. Server systems have a dynamic behavior and face varying workloads. In its extreme form, a heavy workload may induce server thrashing and service unavailability, with underlying economical costs. Admission control is a classical technique used to prevent servers from thrashing when the workload increases consists in limiting the total number of concurrent client admitted to those servers. This is also known as the multi-programming level (MPL) configuration of servers. We believe that modeling server systems is necessary to provide quality-of-service (QoS) guarantees. However, we argue that for the effective adoption of server modeling and control, the models must accurately capture the *dynamics* and the *nonlinear* behavior of server systems while being *simple* to deploy on existing systems.

In this thesis, we first presented the general organization of server systems, namely their architecture and their communication model. We define their workload in terms of workload amount and workload mix. We describe quality-of-service metrics to reflect server performance and availability, and introduce the feedback control concept.

We then provided an overview of existing approaches that aim at managing server systems quality-of-service.

In this thesis, we proposed novel models and control laws for the on-line configuration and reconfiguration of server systems to provide service performance and dependability

guarantees. We apply our solution first to central server systems, then to distributed servers. We present the methodology we used to handle the problem of modeling such systems. We first define the model variables as input, output and state variables. We then derive a set of first order differential equations that link together dynamically these variables from a fluid flow equation and the Little's law. We build the *AM-𝒞* and *PM-𝒞* controls laws. The stability of the closed loop system for both of these strategies is proven. Finally two other control laws, namely *AA-PM-𝒞* and *PA-AM-𝒞*, are introduced.

We also propose a distributed server model. We build step by step the equations that govern the interactions between the model variables.The Karush-Kuhn-Tucker necessary conditions provide an explicit solution for the optimal control of distributed servers.

We present the design, implementation and evaluation of ConSer, our servers control system. We apply a control engineering methodology in order to model and control the QoS of database server systems. We present the experimental and numerical evaluation of the server model and control laws. The results obtained on an industry standard benchmark are conclusive. The model parameters have been identified easily and the validation results show that the model is able to render the main dynamics of the system. The controlled system behaves as expected with a quick convergence time.

The results of the experiments conducted on central servers show that the proposed techniques provide significant benefits on the performance and the availability of the controlled system compared to ad hoc control solutions. Furthermore, numerical evaluations of distributed servers compare the controlled system with ad hoc controlled systems, and show that the former uses less resources than the latter while providing better performance and availability guarantees.

11.2 Perspectives

The results of this thesis open the door to prospective directions. In the following, we discuss a number of interesting research directions that can complement or extend our work.

The proposed central server model and control have been applied successfully to a database server. We believe that they could be easily applied to other sever systems with multi-programming such as web servers, application servers, etc. Furthermore, real experiments to evaluate the proposed distributed server model would allow a better validation of this model. Target experimental environments could consider multi-tier server systems consisting of a web tier, a business tier and a database tier.

In this thesis, we used the client request response time as a performance metric, and the client request abandon rate as the dependability metric. Although these metrics are widely used, we believe that our work could be extended to other quality-of-service

metrics.

Finally, we focused in this thesis on admission control techniques. However, we believe that a combination of admission control with other techniques could further improve the QoS. Other complementary techniques include service differentiation, degradation and service provisioning to enable fully elastic cloud.

Chapter 12

List of Publications

[01] Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Experience with ConSer: A System for Server Control Through Fluid Modeling. *IEEE Transactions on Computers*, 60:951-963, 2011.

[02] Luc Malrait. Qos-oriented control of server systems. *ACM SIGOPS Operating Systems Review*, 44(3):59-64, August 2010.

[03] Luc Malrait. Qos-oriented control of server systems. *In Proceedings of the Fifth Inter- national Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, FeBiD'10, pages 16-21, 2010. ACM.

[04] Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Modelisation et controle d'un serveur. *In Actes de Toulouse'2009 (RenPar'19 / SympA'13 / CFSE'7)*, Toulouse, France, 2009.

[05] Luc Malrait, Nicolas Marchand, and Sara Bouchenak. Average Delay Guarantee in Server Systems Using Admission Control. *In 8th Workshop on time delay systems*, Sinaia, Romania, 2009.

[06] Luc Malrait, Nicolas Marchand, and Sara Bouchenak. Modeling and Control of Server Systems: Application to Database Systems. *In Proceedings of the European Control Conference, ECC'09*, Budapest, Hungary, August 2009.

[07] Luc Malrait, Sara Bouchenak, and Nicolas Marchand. Fluid modeling and control for server system performance and availability. *In IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* , pages 389-398. IEEE , 2009.

Bibliography

[1] T. Abdelzaher, Ying Lu, Ronghua Zhang, and D. Henriksson. Practical application of control theory to Web services. *American Control Conference*, June 2004.

[2] Tarek F. Abdelzaher and Nina Bhatti. Web content adaptation to improve server overload behavior. *Comput. Netw.*, 31(11-16):1563–1577, 1999.

[3] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002.

[4] Amazon.com Inc, 2007. http://www.amazon.com/.

[5] Martin Arlitt and Tai Jin. Workload Characterization of the 1998 World Cup Web Site. Technical Report HPL-1999-35(R.1), HP Laboratories Palo Alto, September 1999.

[6] Martin Arlitt and Carey L. Williamson. Web server workload characterization: the search for invariants. *SIGMETRICS Perform. Eval. Rev.*, 24(1):126–137, 1996.

[7] Martin F. Arlitt and Carey L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5):631–645, 1997.

[8] Paul Barford and Mark E. Crovella. Generating representative web workloads for network and server performance evaluation. In *Proceedings of Performance '98/SIGMETRICS '98*, pages 151–160, July 1998. Software for Surge is available from Mark Crovella's home page.

[9] Martin Brown. Optimizing Apache Server Performance, February 2008. http://www.serverwatch.com/tutorials/article.php/3436911.

[10] Ludmila Cherkasova and Peter Phaal. Session-based admission control: A mechanism for peak load management of commercial web sites. *IEEE Trans. Comput.*, 51(6):669–685, 2002.

[11] Mark E. Crovella, Robert Frangioso, and Mor Harchol-Balter. Connection scheduling in web servers. In *Proceedings of the 2nd conference on USENIX Symposium on*

Internet Technologies and Systems - Volume 2, pages 22–22, Berkeley, CA, USA, 1999. USENIX Association.

[12] Yixin Diao, N. Gandhi, J.L. Hellerstein, S. Parekh, and D.M. Tilbury. Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the ApacheWeb Server. *Network Operations and Management Symposium*, 2002.

[13] John A. Dilley. Web Server Workload Characterization . Technical Report HPL-96-160, HP Laboratories, December 1996.

[14] Sameh Elnikety, Erich Nahum, John Tracey, and Willy Zwaenepoel. A Method for Transparent Admission Control and Request Scheduling in E-Commerce Web Sites. In *13th international conference on World Wide Web*, New York, NY, May 2004.

[15] Jordi Guitart, Jordi Torres, and Eduard Ayguadé. A survey on performance management for internet applications. *Concurr. Comput. : Pract. Exper.*, 22(1):68–106, 2010.

[16] Hans-Ulrich Heiss and Roger Wagner. Adaptive Load Control in Transaction Processing Systems. In *17th International Conference on Very Large Data Bases*, San Francisco, CA, 1991.

[17] J. Hyman, A. A. Lazar, and G. Pacifici. Joint Scheduling and Admission Control for ATS-based Switching Nodes. In *ACM SIGCOMM)*, Baltimore, MA, August 1992.

[18] Apple Inc. Quicktime streaming server, 2007, http://www.apple.com/quicktime/streamingserver/.

[19] Iron Mountain. The Business Case for Disaster Recovery Planning: Calculating the Cost of Downtime, 2001. http://www.ironmountain.com/dataprotection/resources/CostOfDowntimeIrnMtn.pdf.

[20] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(1):35–45, 1960.

[21] Abhinav Kamra. Yaksha: A self-tuning controller for managing the performance of 3-tiered web sites. In *In International Workshop on Quality of Service (IWQoS*, pages 47–56, 2004.

[22] Hassan Khalil. *Nonlinear Systems*. Prentice Hall, 2002.

[23] M. Kihl, A. Robertsson, and B. Wittenmark. Analysis of admission control mechanisms using non-linear control theory. *8th IEEE International Symposium on Computers and Communication*, pages 1306–1311 vol.2, July 2003.

[24] H.W. Kuhn and A.W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, 1951.

[25] I.D. Landau. *Identification Et Commande Des Systèmes*. HERMES, 1988.

[26] J. D. C. Little. A proof for the queueing formula L = λ W. *Operation Research*, 9:383–387, 1961.

[27] Xue Liu, Jin Heo, Lui Sha, and Xiaoyun Zhu. Adaptive control of multi-tiered web applications using queueing predictor. In *10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 106–114, 2006.

[28] Xue Liu, Jin Heo, Lui Sha, and Xiaoyun Zhu. Queueing-model-based adaptive control of multi-tiered web applications. *IEEE Transactions on Network and Service Management*, 5(3):157–167, 2008.

[29] Chris Loosley, Frank Douglas, and Alex Mimo. *High-Performance Client/Server*. John Wiley & Sons, November 1997.

[30] Evan Marcus and Hal Stern. *Blueprints for High Availability*. Wiley, September 2003.

[31] D. A. Menascé, D. Barbara, and R. Dodge. Preserving QoS of E-Commerce Sites Through Self-Tuning: A Performance Model Approach. In *ACM Conference on Electronic Commerce*, Tampa, FL, October 2001.

[32] Daniel A. Menascé and Virgilio Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[33] Daniel A. Menascé, Virgilio A. F. Almeida, Rodrigo Fonseca, and Marco A. Mendes. A methodology for workload characterization of e-commerce sites. In *EC '99: Proceedings of the 1st ACM conference on Electronic commerce*, pages 119–128, New York, NY, USA, 1999. ACM.

[34] Microsoft. Optimizing Database Performance. http://msdn.microsoft.com/en-us/library/aa273605(SQL.80).aspx.

[35] North American Systems International Inc. The True Cost of Downtime, 2008. http://www.nasi.com/downtime_cost.php.

[36] PostgreSQL, 2008. http://www.postgresql.org/.

[37] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson. Admission control for web server systems - design and experimental evaluation. *43rd IEEE Conference on Decision and Control*, December 2004.

[38] S. Parekh and N. Gandhi and J. Hellerstein and D. Tilbury and T. Jayram and J. Bigus. Using Control Theory to Achieve Service Level Objectives In Performance Management. *Real-Time Syst.*, 23(1-2):127–141, 2002.

[39] Bianca Schroeder, Mor Harchol-Balter, Arun Iyengar, Erich Nahum, and Adam Wierman. How to determine a good multi-programming level for external scheduling. In *22nd International Conference on Data Engineering*, Atlanta, GA, April 2006.

[40] Sendmail.org, 2007. http://www.sendmail.org/.

[41] D. Tipper and M.K. Sundareshan. Numerical methods for modeling computer networks under nonstationary conditions. *IEEE Journal on Selected Areas in Communications*, 8(9):1682–1695, December 1990.

[42] TPC-C. Tpc transaction processing performance council, 1992. http://www.tpc.org/tpcc/.

[43] Wei-Ping Wang, D. Tipper, and S. Banerjee. A simple approximation for modeling nonstationary queues. *IEEE INFOCOM*, March 1996.