Universidade Federal Fluminense

LUCIANO BERTINI

Energy Efficient Web Server Clusters with Stochastic QoS Control

NITERÓI

2009

LUCIANO BERTINI

Energy Efficient Web Server Clusters with Stochastic QoS Control

Doctoral thesis submitted to the Computing Institute Graduate Programme, of the *Universidade Federal Fluminense*, in partial fulfillment of the requirements for the degree of DOUTOR EM COMPUTAÇÃO, main area Parallel and Distributed Computing.

Supervisor:

Julius Leite

Universidade Federal Fluminense

NITERÓI

2009

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

B544 Bertini, Luciano.
Energy efficient web server clusters with stochastic QoS Control / Luciano Bertini. – Niterói, RJ : [s.n.], 2009.
129 f.

Orientador: Julius Leite.
Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2009.

1. Computação paralela e distribuída. 2. Cluster de processador. 3. Sistema em tempo real. 4. Consumo de energia. 5. Ambiente de configuração. 6. Comércio eletrônico. 7. Web - aplicativo. I. Título.

CDD 004.35

Energy Efficient Web Server Clusters with Stochastic QoS Control

Luciano Bertini

Doctoral thesis submitted to the Computing Institute Graduate Programme, of the *Universidade Federal Fluminense*, in partial fulfillment of the requirements for the degree of DOUTOR EM COMPUTAÇÃO.

Approved by:

Prof. Julius Leite, Ph.D. / UFF (Supervisor)

Prof. Daniel Mossé, Ph.D. / U. of Pittsburgh (Co-supervisor)

Prof. Carlos Eduardo Pereira, Dr.Ing. / UFRGS

Prof. Rômulo Silva de Oliveira, D.Sc. / UFSC

Prof. Wagner Meira Jr., Ph.D. / UFMG

Prof. Orlando Loques, Ph.D. / UFF

Prof. Cristina Boeres, Ph.D. / UFF

Niterói, March of 2009.

To my father and mother, who always gave me support, and who pray every day for my success. My sincere apologies to them, for being for such a long time away from home.

To my brother, who I am proud to say that he is, and always were, much more brilliant than me.

To the hero that lies inside my heart. He always comes along when I need, with strength to carry on.

Acknowledgments

Without friends no one would choose to live,
despite having all other goods.
– Aristotle

I first met Dr. Julius Leite in person in *2003* when I went to Niterói for an interview with him. I was not hesitating a bit about the idea of pursuing one of the highest earned academic degrees conferred by a university: the doctor's degree. We talked technically about the need of research in the area of energy-efficient real-time systems, but what impressed me most was the way he presented the *modus operandi* of the process to reach the doctor's degree: "*during the first year you will take courses. In the second year you take the qualify exam, composed of studying and presenting three papers to a board of professors. Then in the third year you go to the United States. This is very important because there you will face a high level competition that will make of you a good researcher. In the fourth year, back to Brazil, you present your proposal, write your thesis, and you are done*". The whole process seemed to be very simple to me, but I knew each phase would be tougher than the other. Then I went back to my home town, Belo Horizonte at that time, happy and motivated to write my doctoral project. From that time on, my electricity bill had a reduction of 10%, because I started to turn everything off at home when not needed. My determination to study energy-efficient systems, and Julius' determinism to state the process of getting a doctor's degree were decisive for having a good start.

Then I started in the Program in March 2004. The writing of the project also yielded a paper for my first WTR workshop, in Gramado-RS. The paper was a review and presented not some thoughts about the most important works to date and future directions. Some of them were from an renowned author, Dr. Daniel Mossé, from the University of Pittsburgh, who was present at that event. My first talk with him was about ideas of what to do in the area of power optimization, and I got impressed with the blowout of ideas that came from that brilliant mind. He gave lots of ideas about problems where it would be interesting to save energy. He talked about some reward-based scheduling and several specific optimization problems where we could apply heuristics in multi-computer systems.

I was amused and amazed. At that occasion, Dr. Julius and Dr. Mossé talked about the sandwich internship and about the possibilities of Dr. Mossé being the host at Pittsburgh.

Then I continued in my first year taking the courses. During that year I realized the first most important characteristic in Dr. Julius that I want to thank him. It is his dedication and idealism for the real meaning of the word Education. He is not interested, at first hand, on producing papers, only for the papers, but he is interested in the level of education that this will bring to his students. He was always proactive about my education needs as if it were for his own benefit. He was constantly worried about providing a high level environment for doing a good work, with the entire infrastructure that I could need. When I had just arrived he gave me a brand new hyperthreding Pentium IV computer that was a top model at that time, and of course I am using it up to now to write these thanks. He never missed a chance of providing everything, and I never had to ask for anything. I owe to him my accomplishments related to the English language, because he helped every time, pushed me up, and stimulated, and this made the difference. I lived one year in the USA, what was an old dream for me, and Julius made it possible. He was completely right about the competitiveness there. Being there pushed me to a higher level of education, professionally, and personally. All happened thanks to Julius' idealism on Education.

Then I went to the so dreamed USA. There I got to know better Dr. Daniel Mossé. I confess that I was afraid of him, because Dr. Mossé is not anyone, he is one of the greatest researchers in the area of real-time systems, in the world. I knew I was going to have a great opportunity working with him. Then I realized that besides being a great researcher, he was accessible, and a "normal" person. Dr. Mossé is fantastic in theory and in practice of Computer Science. When I was in difficulties, he always analyzed theoretically every problem expending all time needed, but he also would comprehend the difficulties. He used to say that the difference between practice and theory is bigger in practice than in theory, and that difficulties are normal. The most invaluable thing that I learned with him is that it is necessary to achieve perfection. For him, only "good enough" is not enough, only "good" is good enough. And I am proud that I had those endless and difficult tasks of rewriting my papers because of his complaints and corrections, what taught me a little of the art of writing. Thank you very much, Dr. Mossé.

I owe also to Dr. Mossé's Power-aware Real-time Systems work group at the University of Pittsburgh a big boost in my work. They work efficiently, and being there made my efficiency increase as well. I need to thank the professors Dr. Rami Melhem, Dr. Bruce

Childers, and of course Dr. Mossé, for all their time devoted to discuss the problems that I was working on, and also for allowing me a time share in their weekly meetings. Also, my colleagues gave very important contributions. I am proud to have worked and shared an office with names like Nevine AbouGhazaleh, Alexandre Ferreira, Cosmin Rusu, Ruibin Xu, and also Mahmoud Elhaddad. Nevine's successful and hard work was motivating. A special thank to Alexandre who helped me a lot inside and outside the lab. Special thanks to Cosmin Rusu for he becoming so quickly my friend. These people took part in the most important year of my life, and every second, from the time I landed in the USA, to the time I left, is unforgettable.

This doctorate made me international. I thank again Julius for all funding provided. The most unthinkable appointment I had with Dr. Julius was to meet him at 7 p.m. in front of the cathedral of Notredame, in Paris. And before that, walking by the streets of Pisa the day before the ECRTS 2007, I met Julius accidentally. Motivated by both Dr. Julius and Dr. Mossé, I gave two lectures, one in Rome, Italy, at the D.I.S.P. – *Dipartimento di Informatica, Sistemi e Produzione Università degli Studi di Roma Tor Vergata*, and one in Kaiserslautern, Germany, at the Real-time and Embedded Systems group of Prof. Dr. Gerhard Fohler, at the *Elektrotechnik und Informationstechnik, Technishe Universität Kaiserslautern*. In July 2009 we will meet again for the ECRTS, but now in Dublin, Ireland. This is going to be another amazing experience abroad. My doctorate and my accomplishments wouldn't be the same without all the international experience I had. Thank you very much, Dr. Julius, and Dr. Mossé.

I need to thank also the committee of my thesis defense, that was formed on March 23rd, 2009. Prof. Dr. Carlos Eduardo Pereira, from UFRGS, Prof. Dr. Rômulo Silva de Oliveira, from UFSC, Prof. Dr. Wagner Meira Jr., from UFMG, Prof. Dr. Cristina Boeres from UFF, Prof. Dr. Orlando Loques, from UFF, and of course, Professors Dr. Julius and Dr. Mossé. Besides the three-hour session of difficult questions, scrutinizing my work, they made invaluable comments and suggestions to improve the final thesis. Thank you all very much. I also would like to thank the whole IC graduate program at UFF and all professors who are building the excellence of the program.

I want to thank all my Brazilian colleagues. A doctorate is an endurance test, and my colleagues were those who helped in times when one beer was necessary (perhaps not sufficient). I want to thank Cristiano Maciel, Janine Kniess, Jacques (the saviour), Sanderson, Luiz Merschmann, Haroldo Santos, Stênio Sã, Tiago Neves, Cristiane, Renatha, Luciana Brugiolo, Adria, Renato Nunes, Jonivan, and Idalmis for their support during most of

the years. Thanks also to Viviane, Rodrigo, Daniela, Kennedy, Johnny, Aleteia, Luciana Pessoa, Rafael, Anand, Mário, Diego Brandão, Leandro, and other colleagues that have been around in many important moments. Special thanks to Raphael Guerra, Carlos Sant'Ana and Vinicius Petrucci for their comments and participation, as we worked close in some projects. And a very special thank goes to Alessandro Copetti, with whom I shared similar difficulties that make of us heroes in accomplishing a doctorate.

Finally, I want to thank who is always beside me. She was with me in the USA giving me emotional support. She would give up on everything just to stay with me, as she did when she decided to go to the USA. With an undisturbing humor, she was always waiting for me, with no complaints when I had to work hard every weekend, depriving us of going out and taking the most of the USA. And her patience and composure continue up to the present. Thank you M', for being with me in the most unforgettable moments, like going to the Phantom of the Opera at Broadway, or drinking Margueritas at the Gullifty's in our neighborhood, in Pittsburgh. If I finished successfully my doctorate, Emilene took a significant role in it.

Thank you very much, reader, for having the patience to read this up to here, and thank you again, in advance, for reading my thesis.

Dr. L. Bertini
April 2009.

Abstract

In this thesis we study the soft real-time web cluster architecture needed to support e-commerce and related applications, with the fundamental goal of optimizing the energy efficiency. The energy consumed by a system and its Quality of Service (QoS) are the two components of the tradeoff that rule the power optimization in such systems. In soft real-time systems like web servers, the QoS is usually defined as the fraction of requests that meet the deadlines. When this QoS is measured directly, regardless of whether the request missed the deadline by an small amount of time or by a large difference, the result is always the same. For this reason, only counting the number of missed requests in a period does not allow an adequate observation of the real state of the system. We give theoretical propositions on how to control the QoS, not measuring the QoS directly, but based on the probability distribution of the tardiness in the completion time of requests. We call this QoS metric *Tardiness Quantile Metric* (TQM). The proposed method provides fine-grained control over the QoS so that we can make a closer examination of the relation between QoS and energy efficiency.

To generalize the TQM idea, we propose the GTQM method that makes it possible to measure the QoS of the system without any assumption on the workload. By using an on line convergent sequential process defined from a Markov chain, we derive quantile estimations that do not depend on the shape of the workload probability distribution. To use GTQM, we need a controller that will keep the system's QoS as defined by the statistical inference. We describe a simplified way to implement performance control in a multi-tier computing system designed for e-commerce applications. We show that the simpler SISO (*Single Input Single Output*) controller, rather than a more complex distributed or centralized MIMO (*Multiple Input Multiple Output*) controller, works well, regardless of the presence of multiple cluster nodes and multiple execution time deadlines. Our feedback control loop acts on the speed of all server nodes capable of Dynamic Voltage Scaling (DVS), with QoS, measured by means of GTQM, being the reference setpoint. We use a SISO PIDF control loop implemented in the multi-tier cluster.

Besides QoS control, we solve the *dynamic configuration* optimization problem in a web server cluster. We model the problem of selecting the servers that will be on and finding their speeds as mixed integer programming. For proof of concept, we implemented this dynamic configuration scheme and the GTQM QoS control in a web server cluster running Linux, and a layer of servers running a MySQL cluster configuration. The system has soft real-time requirements, in order to guarantee both energy-efficiency and good user experience. Our testbed is based on an industry standard, which defines a set of web interactions and database transactions with their deadlines, for generating real workload and benchmarking e-commerce applications.

Keywords

1. Cluster Dynamic Configuration
2. Dynamic Voltage and Frequency Scaling
3. Energy Efficiency
4. Energy Efficient Web Server Clusters
5. Power Aware Real-Time Systems
6. Power Management
7. Quality of Service
8. QoS Control
9. Server Cluster
10. Soft Real-Time Systems
11. Web Cluster

Resumo

Esta tese aborda arquiteturas de tempo real não crítico necessárias para aplicações de comércio eletrônico e outras aplicações *web* em *clusters* de computadores, com o objetivo fundamental de otimização do consumo de energia. O compromisso existente entre a energia consumida por um sistema e sua qualidade de serviço (QoS) rege a otimização de potência em tais sistemas. Em servidores *web*, QoS é normalmente definida como a fração de requisições que atende aos prazos de tempo real definidos. Entretanto, quando essa QoS é medida diretamente, independentemente se o prazo foi descumprido por um pequeno intervalo de tempo, ou por um intervalo muito grande, o resultado é sempre o mesmo. Por isso, somente a contagem do número de requisições com atendimento após o prazo não é suficiente para permitir uma observação adequada do real estado do sistema. Esta tese apresenta proposições teóricas de como controlar QoS sem medi-la diretamente, com base na distribuição de probabilidade da variável *tardiness*, que representa o quão tarde uma requisição termina em relação ao seu prazo. A nova métrica de QoS proposta foi denominada *Tardiness Quantile Metric* (TQM). Esse método provê um controle fino de QoS, que permite avaliar minuciosamente o compromisso entre QoS e a eficiência de energia do sistema.

Com o objetivo de generalizar a idéia do TQM, de modo a tornar o método independente da carga do sistema, esta tese ainda propõe o método GTQM, que permite medir QoS sem qualquer suposição a respeito das características estatísticas do sistema. Utilizando um processo sequencial convergente definido a partir de uma cadeia de *Markov*, o método deriva estimadores de quantil independentes da função de distribuição de probabilidades da carga do sistema. Um método simplificado de controle utilizando GTQM é descrito, para realizar controle de desempenho em um *cluster* de computadores multicamadas projetado para aplicações de comércio eletrônico. Demonstra-se que um controlador simples do tipo SISO (*Single Input Single Output*), ao invés de um controlador mais complexo do tipo MIMO (*Multiple Input Multiple Output*), funciona bem, apesar da presença de múltiplos nós do *cluster* e também múltiplos prazos de execução. O controle realimentado apresentado atua na velocidade dos nós do *cluster* com funcionalidade de DVS (*Dynamic Voltage Scaling*), utilizando-se como referência a QoS medida através do método GTQM.

Além de controle de QoS, esta tese resolve o problema de otimização da configuração dinâmica em um *cluster* de servidores *web*. É modelado o problema da seleção de quais nós do *cluster* devem ficar ligados e quais serão suas respectivas velocidades, através de um problema híbrido de programação linear e inteira. Para provar os conceitos, a configuração dinâmica e o controle GTQM de QoS são implementados em um *cluster* de servidores *web* baseado em *Linux* e *MySQL*. O ambiente de testes construído é baseado em um padrão industrial, que define um conjunto de interações *web* com transações de banco de dados e seus respectivos prazos de tempo real, o que permite a geração de uma carga real de comércio eletrônico.

Palavras-Chave

1. Configuração Dinâmica de Clusters
2. Variação Dinâmica de Tensão e Frequencia
3. Eficiência de Consumo de Energia
4. Cluster de Servidores Web Eficientes em Energia
5. Sistemas de Tempo Real com Otimização de Potência
6. Gerenciamento de Potência
7. Qualidade de Serviço
8. Controle de QoS
9. Cluster de Servidores
10. Sistemas de Tempo Real Não Críticos
11. Cluster Web

Glossary

B2B	:	Business to Business
CVS	:	Coordinated Voltage Scaling
DB	:	Database
DVS	:	Dynamic Voltage Scaling
EB	:	Emulated Browser
EDF	:	Earliest Deadline First
FIFO	:	First In First Out
GLPK	:	Gnu Linear Programming Kit
GMG	:	Motor-Generator Set
GTQM	:	Generalized Tardiness Quantile Metric
HTTP	:	Hypertext Transfer Protocol
HTTPS	:	Hypertext Transfer Protocol over SSL
IT	:	Information Technology
IVS	:	Independent Voltage Scaling
JOP	:	Joules Per Operation
LAN	:	Local Area Network
LAOVS	:	Load-Aware On-off with independent Voltage Scale
MIMO	:	Multiple Input Multiple Output
MIP	:	Mixed Integer Programming
MIPJ	:	Millions of Instructions per Joule
MIPS	:	Millions of Instructions per Second
MISO	:	Multiple Input Single Output
MTTF	:	Mean Time To Failure
NCPI	:	Network-Critical Physical Infrastructure
NDB	:	Network Database
OS	:	Operating System
PARD	:	Power-Aware Request Distribution
PHP	:	PHP: Hypertext Preprocessor
PID	:	Proportional Integral Derivative

PIDF	:	Proportional Integral Derivative with Filter
QOS	:	Quality of Service
Q-Q	:	Quantile-Quantile
RBE	:	Remote Browser Emulator
SIMO	:	Single Input Multiple Output
SISO	:	Single Input Single Output
SLA	:	Service Level Agreement
SP	:	Setpoint
SPSS	:	Statistical Package for the Social Sciences
SQL	:	Structured Query Language
SSL	:	Secure Sockets Layer
SUT	:	System Under Test
SWC	:	Soccer World Cup
TCP	:	Transmission Control Protocol
TPC	:	Transaction Processing Performance Council
TQM	:	Tardiness Quantile Metric
UPS	:	Uninterruptible Power Supply
URI	:	Universal Resource Identifier
VM	:	Virtual Machine
VOVO	:	Vary-On Vary-Off
WIPS	:	Web Interactions Per Second
WIRT	:	Web Interactions Response Time

Contents

List of Figures

List of Tables

Chapter 1

Introduction

To have a great idea, have a lot of them.
– Thomas A. Edison

1.1 Introduction

Recently, it has been a strong concern to system design researchers the development of energy-efficient systems. This concern appeared preeminently for ubiquitous computing systems, because of the need to increase lifetime of the batteries, given the pervasive nature of the applications, where mobile devices depend on the batteries to work. Then it became necessary to reduce energy consumption, as the technology for the batteries doesn't improve as fast as the need for power increases. Furthermore, batteries always have weight and size that sometimes are restricted by the application. Another reason is the cost reduction and improved reliability achieved by the lower heat dissipation.

Ubiquitous computing comprise also all time critical embedded applications. These real-time systems have time restrictions, and need extra reliability. One way to increase dependability of real-time systems is by improving their energy efficiency. However, to show how this concern is new in the research area, by the end of the last decade, the need for energy-efficiency was not predicted. In a survey on the directions of the research on real-time systems [94], in a 10-year prediction about the needs of the these systems, nothing is mentioned about energy consumption.

After a period of development of energy-efficient methods only for the design of mobile systems, the first work to present a case for power management in web servers was published in 2002 [24], from the IBM research group, breaking with the tradition that focused power management research only for portable and handheld devices, to focus on

web servers. The work was motivated by the fact that web servers experience large periods of low utilization, presenting an opportunity for using power management to reduce energy consumption with minimal performance impact.

Energy optimization in server clusters may be done in multiple subsystems, such as the microprocessor, disks, memory, and memory cache. The best result will be obtained when all these subsystems are all deployed with some Power Management techniques. However, each subsystem is a completely different area of research. People working with cache algorithms are proposing energy efficient algorithms, and also for disks or memory. One review that briefly describes some efforts in each subsystem is in [21]. For example, in the memory research area, there are problems such as properly laying out data across the memory banks and chips so that the server can use low-power states more extensively. There are also cache replacement policies that selectively keep blocks from certain disks in the main memory cache to increase their idle times, leaving the disks in low power mode for a longer period. Because of this breadth of research areas, in this thesis we will address local and cluster-wide energy management techniques for heterogeneous systems, attaining only to the processor subsystem. Heterogeneity offers an extra opportunity to energy minimization, because it adds more variables to the problems, and real clusters may become heterogeneous very soon if newer components are added.

This thesis aims at studying energy-efficient techniques for web servers. There are a number of services today that depend on the Internet, such as electronic commerce, business to business (B2B) applications, application and databases servers, etc. In Brazil, a research made by the Science and Technology Ministry, showed that the country is among the ten countries that uses more the Internet [66]. Services like the Federal Revenue, banks, electronic government, and mostly electronic commerce, are growing fast. These services are globally considered essential for the growth of the so called "new economy", that calls for lower costs, lower interest rates, and is very dependent on the information [26]. For all these reasons, power reduction on data-centers represents a new research area that is strategic, as energy-efficient products will dominate the market. And it is even more strategic considering the world wide effort of companies to become more environmentally sustainable, because this reflects in the companies market value.

E-commerce, e-banking, and other related applications involve high complexity, with databases transactions and multi-tiered server clustering. These applications are often modeled with real-time characteristics, because a minimum level of customer satisfaction must be guaranteed for the success and survival of the company. This is so crucial because

today many enterprises rely only on the electronic means to sell their products. Many works have modeled e-applications with real-time characteristics. For example, [31] shows the real-time requirements of e-commerce, addressing mainly the timeliness, among other aspects; [96] presents protocols that can be used to detect when real-time constraints are violated; and [39] describes a real-time middleware to support e-commerce applications.

In addition to real-time characteristics, large systems to host e-commerce applications can show a huge electricity consumption [73], which means high costs of ownership, making power management necessary. In [30], for example, the authors point out that data centers operate at power densities of around 100 Watts per square feet. With all the real-time and energy efficiency issues in mind, architectural challenges arise when we try to deploy architectures to support e-applications which also need to satisfy Quality of Service specifications.

According to [57], the main premise of a power managed architecture is to ensure that the system gets all the power it requires for full performance, and aggressively manages resources that are not in use to consume less power. In real-time systems, however, hitting full performance means that the system is able to meet the predefined real-time specification, including the QoS specifications defined in the Service Level Agreement (SLA). For this reason a real-time system generally offer more room for power management than ordinary systems, because the timing characteristics of the system are well-defined. Thus, one major aspect considered in this thesis is that the system must run with the minimal performance that still can keep the SLA agreement, so that we get the maximum energy savings possible.

1.2 Motivation

The reasons to study energy-efficient web clusters for e-commerce applications are threefold. First, there is a need for a boost in e-commerce web servers efficiency, as the main goals of the e-commerce use in the enterprise are cost reduction and the creation of competitive advantage, and that is what makes energy-efficiency mandatory. Second, the number and variety of e-commerce applications are growing. An example is the integration and customization of such applications, such as the idea of web shopping malls, support for comparative shopping and business-to-business [42]. As the complexity of applications grow, clusters with more servers will be needed, and the power consumption may become prohibitive. The third reason is that the recent work on performance

evaluation for web server clusters, specially for power-efficiency and energy-efficiency, has clamored for more realistic test workloads. In this thesis we will propose new methods that can be independent of the workload, not using assumptions that does not hold in practice, such as M/M/1 queueing models.

Data-centers usually adopt an architecture made of a number of servers, regular computers, or blade servers [107], designed specifically for the composition of clusters for processing or storage. Clusters architecture must be flexible enough to support a range of applications. The question is: how can be attended all the availability and performance requisites face to changes in the demand and resources availability, with minimum energy cost? The available services in a data-center can vary with time, and thus the design must be flexible. Furthermore, it is not convenient to design the system for the peak demand with inadequate resource allocation.

The work presented in [69] shows the electricity demand in data-centers, as part of the debate that took place in the USA about the energy consumption of the computer systems that maintains the Internet. According to that study, data-centers have an energy consumption per area bigger than the consumption presented in some industries, with installed powers of up to 400 W/ft^2. However, although this clearly suggests space for more efficient projects, the work also claim that it is difficult to fill the gap between energy efficiency, and the reliability and fault tolerance required by this industry. Designers still rely on traditional technologies and prefer not to test new energy-efficient options. In [47] it is exemplified that for many IT managers, to improve the energy-efficiency implies in reducing reliability. Today the market understands that ensuring the data-center reliability is associated with high energy consumptions. It is necessary to demystify this fact, by means of good research on energy-efficient and reliable architectures.

A recent report from EPA (U.S. Environmental Protection Agency) [1] shows that data-centers in the United States could save up to $4 billion in annual electricity costs if more energy efficient equipment and operations were applied, along with the use of best management practices. The report pointed that data-centers consumed about 60 billion KWh in 2006, doubling in the past five years, with a tendency to double again in the next five year, costing about $7.4 billion annually. They estimated that existing technologies and strategies could reduce typical server energy use by an estimated 25 percent, with greater energy savings possible with advanced technologies.

The energy consumption of a server determines the operational costs of a data-center, because the high power densities cause cooling and reliability problems. There is the

need to use UPS (*Uninterruptible Power Supply*) units and back-up generators. Besides all problems, at least the generators present environmental issues, considering the main energy comes from clean sources. The cooling systems also represent high costs, because of their complexity. The Figure 1.1 shows the energy configuration adopted in the data-center Cyber, in Brazil [25], owned by Brazil Telecon. The figure shows diesel generators and the back-up unit for uninterruptible energy generation, in a high cost configuration.

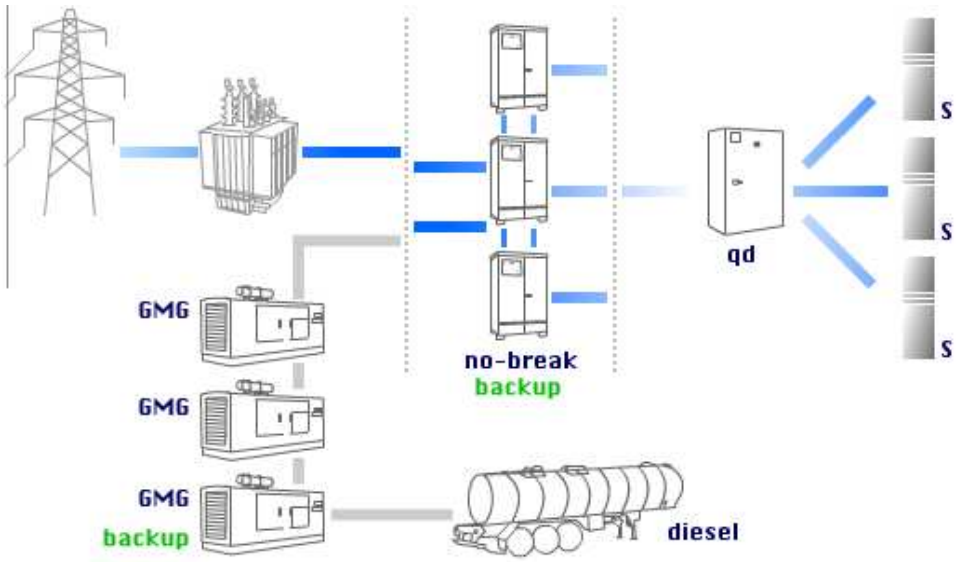

Figure 1.1: Energy delivery system for the Brazil Telecom data-center. Legend: qd = distribution board, GMG = motor/generator set, S = Server

According to [69], the total energy consumption with servers in the USA is close to 22 TWh, what costs about US$2 billion, considering the cost of US$100 per MWh. Talking about the environment, to generate this amount of energy results in 12 M tons of new CO_2 thrown in the atmosphere. Therefore, the main motivation is multiobejective, to minimize both financial costs and environmental impacts. The cost of ownership of a data-center needs to be reduced to maximize the competitiveness of companies that depend on high dependable systems. It is necessary to develop new products that can be energy-efficient, reliable, and also environmentally aware, to demystify that high reliability systems need overdimensioned energy delivery systems.

1.3 Thesis Overview

In this work, we started with the need to specify the system's QoS and then reducing the system's performance to the minimum necessary to keep this QoS. In Chapter 3,

we study the soft real-time web cluster architecture needed to support e-commerce and related applications. To meet this goal, we designed a testbed based on an industry standard, which defines a set of web interactions and database transactions with their deadlines, for generating a real workload and benchmarking e-commerce applications. The QoS is defined as the fraction of requests that meet the deadlines. We found out that, in practice, it is not so simple to measure the QoS. When the QoS is measured directly, regardless of whether the request missed the deadline by a small amount of time or by a large difference, the result is always the same. For this reason, only counting the number of missed requests in a period does not give good observations of the state of the system. Then we made theoretical propositions of how to control the QoS, not measuring the QoS directly, but based on the probability distribution of the tardiness in the completion time of the requests. We called this new QoS metric *Tardiness Quantile Metric* (TQM). The proposed method provides fine-grain control over the QoS so that we can make a closer examination of the relation between QoS and energy efficiency. We validate the theoretical results showing experiments in a multi-tiered e-commerce web cluster implemented using only open-source software solutions.

In Chapter 4, we describe a simplified way to implement performance control in a multi-tier computing system designed for e-commerce applications, so that the GTQM estimation can be put in practice in the system. We show that the simpler SISO (*Single Input Single Output*) controller, rather than a more complex distributed or centralized MIMO (*Multiple Input Multiple Output*) controller, works well, regardless of the presence of multiple cluster nodes and multiple execution time deadlines. Our feedback control loop acts on the speed of all server nodes capable of DVS (*Dynamic Voltage Scaling*), with QoS being the reference setpoint. By changing the speed, we change the position of the p-quantile of the tardiness probability distribution. Then, the control variable will be the average tardiness provided by the GTQM method, and the setpoint the tardiness value that will position this p-quantile at 1.0, value at which a request finishes exactly at the deadline. Doing so will guarantee that the QoS will be statistically p. We test this method in a SISO PIDF control loop implemented in a multi-tier cluster. We use open software, commodity hardware, and a standardized e-commerce application to generate a workload close to the real world. The main contribution of Chapter 4 is to empirically show the robustness of the SISO controller, presenting a sensibility analysis of the four controller parameters: damping factor *zeta*, derivative filter factor *beta*, integral gain *ki*, and zero time constant *tau*.

The TQM method proposed in Chapter 3, although effective, is not perfect, because

it is based on assumptions on the shape of the probability distribution function of the workload. The first question that arise is: how can we generalize? We answer this question in Chapter 5. To generalize this idea we propose the *Generalized Tardiness Quantile Metric* (GTQM). By using an on line convergent sequential process, defined from a Markov chain, we derive quantile estimations that do not depend on the shape of the workload probability distribution, so that the metric can be used with any workload, that is, the QoS measure becomes distribution free. To evaluate the new metric, we also show practical results in a three-tier web cluster with QoS control in an e-commerce environment.

The Chapter 5 is a departure from other works, in that: (a) it presents a method of quantifying the QoS using a metric that is used itself in QoS control (other works use usually the number of deadlines missed [52, 85]); (b) the method works for any kind of probability distribution presented by the workload, and thus we can expect good results for real workloads; and, (c) the results are obtained from a real testbed (not simulations) composed of a three-tiered web server cluster running Linux and TPC-W, a real, industry-standard, e-commerce application.

When dealing with soft real-time web clusters, the bigger the average relative tardiness, the lower is the resultant QoS. Tardiness can then be used as a control variable, because tardiness does not carry only boolean information about QoS (whether the deadline was met or missed), but it is a continuous value possible to be calculated for each web interaction. Tardiness values show how close the execution was to the deadline, which enables fine-grain control over server speeds, and consequently higher energy savings.

After providing a method to carefully estimate and control the QoS, using the DVS as a mechanism to vary the system's performance, a second concern that we study in this thesis is how to turn off servers when they are not needed, and how to optimize the speed selection of servers. Because clusters are often heterogeneous, using different DVS settings in each server produces better energy savings. In a preliminary work, we investigated local nodes adjusting their DVS settings based on a global off-line optimization, and we achieved extra power reduction up to 10%. Then we incorporated on/off mechanisms and an optimization model that results in the optimal combination of servers to handle a specific load.

In Chapter 6 we use a mixed integer programming (MIP) model in the cluster reconfiguration. The QoS is guaranteed by the GTQM statistical quantification of the response time, compared to the deadline of requests. If the system needs to increase its capacity,

to keep the QoS in a predefined level, the cluster may be reconfigured dynamically by selecting the appropriate combination of frequencies, and the best combination of machines. We present two different MIP models, one considering the selection of only existing frequencies, what means the use of the discrete frequency immediately higher to the exact theoretical frequency needed for a given workload (if speeds were continuous), and the use of a combination of two frequencies to achieve the exact theoretical frequency needed. This well known combination scheme was proposed by [53]. It consists on switching periodically between two adjacent frequencies during the execution of a task. As the DVS overhead in modern CPUs is negligible, this second solution presents some key advantages. Besides, recent works [23, 34] showed that switching processor frequency very fast does not cause any reliability problem. In fact, it increases the MTTF of the processor, because it reduces the average device temperature, and consequently also reduces the number of temperature-driven failures.

Still in Chapter 6, we show how to integrate a single QoS controller to the MIP solution. Finally, in order to investigate different alternatives of implementation, we compare the use of one single controller with the MIP optimization, against several independent controllers that simplifies the optimization needed, but looses in power reduction.

1.4 Thesis Contributions

With the proposal of the TQM method in Chapter 3, our objective is to have a means of exploring the trade-off between energy and QoS in complex web systems, and for this we need to have a fine grain control of the QoS. Instead of using a QoS measure based on the counting of missed deadlines, we use the on-line measurement of tardiness in the completion time of the requests, because we verified in practice that counting missed deadlines results in poor accuracy and broad confidence intervals. Our contribution is the statistical guarantee that we can achieve for the QoS based on approximations for the probability density function of the tardiness random variable. We show that the average tardiness is directly related to the QoS. Previous works that are said QoS aware [85, 90] do not allow the maintenance of the QoS at a precise user predefined value. Our work differs from these approaches because we apply a statistical inference solution to guarantee the exact desired QoS level, aside from the fact that our target environment is e-commerce. In addition, most previous work dealt only with requests with a single deadline for all requests, which are not typically representative of e-commerce applications.

Normally the workload of a web system is assumed to have a specific probability distribution, because it simplifies the modeling. For example, when queueing theory is applied, the simplest M/M/1 queueing models are based on Markovian workloads. The more complex G/G/1 queueing models generally do not have closed formulas, and if a G/G/1 model is assumed, bounds based on the tail probabilities are also applied. With the GTQM method presented in Chapter 5, We use no such assumptions, obtaining a three-fold contribution: 1) we present a method of quantifying the QoS so that this metric is used in QoS control; 2) the method works for any kind of probability distribution presented by the workload, and thus we can expect a good result for a real workload; and, 3) the results are not based on simulations. The algorith GTQM is based on stochastic approximations, and was proven to be free of the probability distribution. This is the most derired characteristic in any system modeling, but almost never feasible.

We have shown in a previous work [48] that it is possible to choose the system settings so that the power is minimized and at the same time the average response time, given by queueing theory, is such that a predefined amount of deadlines are met. However, it is difficult to have a good queueing model for a real e-commerce environment that allows simple analytical formulation of the response time, without having many non realistic assumptions about the workload generation and service times. The approach we use in this work is based on a real e-commerce scenario.

In the system reconfiguration presented in Chapter 6 we combine two technologies: QoS control by means of feedback control theory, and operations research. First, the feedback control dynamically adjusts the frequency/voltage of the cluster nodes to control the fraction of deadlines met; frequencies are set proportionally to how late or how early requests finish. To change speeds, we rely on the support of Dynamic Voltage Scaling (DVS), present in most modern CPUs (allowing the dynamic setting of the frequency and voltage of the CPU core), which allows for quadratic reduction in energy, and cubic reduction in the power consumption of the CPU [101].

Second, we show how operations research is introduced in the system to achieve optimal dynamic configuration of the web cluster, that is, which nodes are on and off, and at which frequency. We model the problem of assigning speeds to servers, including zero speed (server off), as a mixed integer programming problem, which is a linear programming problem where some variables are integers and some are real variables, and solve it using traditional linear programming techniques. Then, the contribution is fourfold: (a) a novel way of combining control theory and MIP solutions; (b) modeling the dynamic

configuration problem (i.e., on/off of nodes and speeds of active nodes) through mixed integer programming problems; (c) comparing the efficiency of higher-than-needed discrete speeds (interspersed with idle periods) and pseudo-continuous speeds (based on the two-speed scheme of [53]); and, (d) comparing a centralized SISO (Single Input Single Output) controller with a distributed SIMO (Single Input Multiple Output) controller.

By modeling the optimization problem allowing heterogeneous machines, with different set of frequencies, we can achieve high power reductions compared to other DVS schemes that consider equal frequencies for all machines, and only a predefined sequence of machines to turn on and off. One such solution is adopted in [30], which we simulated to make a comparison. Our optimization solution reduced the power usage up to 40%.

Finaly, one important contribution of this thesis is that it presents solutions and results obtained and tested in a real system. Although we were not able to test for large number of servers (our testbed has up to 10 servers), as far as we know there is no published work for hundreds of servers without being based only in simulations. We based our implementation in a real three-tier e-commerce architecture, using Apache, PHP, MySQL cluster, and a real e-commerce web store implementation, the TPC-W, that was designed for benchmarking e-commerce commercial systems.

Chapter 2

Related Work

A theory has only the alternative of being wrong. A model has a third possibility - it might be right but irrelevant.
– Manfred Eigen

This chapter presents a survey on energy efficient techniques and methodologies. We are going to introduce with general energy efficient research for real-time systems, and then refining to the main goal, which is to pay attention only to the works related to local and cluster-wide energy management techniques for heterogeneous server clusters, preferably with QoS awareness

2.1 Energy-efficient Systems

Energy consumption optimization can take place in any stage of a system's design, from the hardware and microelectronics, the operating system, to the application level. This is a difficult task, because there is always a tradeoff between energy and performance. In real-time systems, the problem is even harder, because there is a timeliness constraint to be met. There are some surveys on the literature on this subject. For example, in [9] power optimization techniques in all system levels are presented, considering that the major energy consumers are the CPU, memory, and communication. In [55], a survey on energy saving techniques for all communication layers is presented. In [49] techniques for all layers are presented, but with focus on multimedia applications and wireless networks. Also in [99], power reduction techniques are presented for the digital logic, the compiler, the operating system, and the network. At the OS level, they take into account the processor speed, input/output devices, measure of energy consumption, quality of service, and *jitter*.

Some energy-efficient related terminologies are important to note. The systems that can reduce their power consumption during runtime are called *power-aware*. There are also the *low-power* systems, which are different. [20] defines power-aware systems as those that can minimize their energy consumption by adapting to the changes in its operating point. On the other hand, low-power systems are those that had taken energy into account in their design, aiming at a better energy consumption without loss of performance.

We presented a brief survey on process scheduling for real-time systems with energy optimization in [10]. Process scheduling is a well known optimization problem, and its associated decision problem, that is, wheather the task set is schedulable or not, even without some power budget restriction to meet, belongs to the $\mathcal{NP}$-complete class. This was proved in [45], by transforming from the 3-PARTITION problem, what makes the problem belong to the $\mathcal{NP}$-hard class. The first paper to address this problem is [106]. They introduced a metric to evaluate the energy consumption called MIPJ, or milions of instructions per joule. They observed that in a ten years of evolution, the processors that improved the performance from 1 MIPS to 200 MIPS, had an increase from 1 MIPJ to only 5 MIPJ. That is, power consumption increased 40 times. They present a technique that consists in reducing the processor frequency and voltage during the busy time, so that the idle time is reduced to a minimum, as idleness means energy loss. Then several other works appeared for process scheduling. Some of them use EDF scheduling, and some use *Slack Reclamation* techniques to try to reduce idleness. A small list of papers in this area is: [6, 67, 68, 72, 77, 83, 110, 112]

Imprecise computation which relates to reward based computing is aimed at delivering a breadth range of QoS for real-time applications. They are usually multiple version software, each with different reward restrictions. Some imprecise computation papers with energy optimization appear in [5, 86, 87]. Techniques for soft-real time systems considering QoS for multimedia applications in mobile systems appear in [111]. In fact, in the beginning, the research on power-aware and low-power systems were mostly directed to mobile systems, which have the limitted battery lifetime problem. One example is the work in [17, 18], where data replication is used in a mobile application, based on ad hoc networks, to avoid that the network become disjoint at some time because of shortage of energy from batteries. A Bayesian-Fuzzy decision model is used as a technique to implement the required decision making process, allowing to deal with the uncertainty present in the ad hoc network.

The mentioned scenario, where power management were only studied for handheld

and portable mobile systems, changed after the paper [24], which showed the importance to reduce power consumption also for web servers, because web servers experience large periods of low utilization, presenting an opportunity for using power management to reduce energy consumption with minimal performance impact, but for a different reason: reduce costs and also environmental impacts associated with energy waste.

2.2 Power Management in Web Servers

The seminal papers addressing energy-efficient web server systems often considered homogeneous systems or single servers to simplify, but they are important as they were the seed to the more complex techniques developed later. They introduced cluster reconfiguration techniques and local or global DVS policies. In this section we will look at the most important seminal work, paying attention on how their energy reduction algorithms work.

The paper [29] is the first to promote a research agenda to improve the energy efficiency of Internet server sites. They use the term JOP (*Joules Per Operation*) to quantify the energy minimization needed to deliver a service at a given level of request thoughput. The proposal is an energy-conscious reconfiguration technique with all servers homogeneous. Machines are turned on and off based on the average utilization. For a threshold T, if the switch detects that the average utilization has fallen below $TN/(N-1)$, where N is the number of servers, then it selects a server to be put in standby. With this simple algorithm, the request traffic is concentrated on the minimal set of servers that can handle the load.

In [28] they propose a resource management architecture for hosting centers called *Muse*. It defines policies for adaptive resource provisioning in hosting centers using an economy approach, considering energy as a resource. The economy framework has a price-setting algorithm that determines efficient resource assignments by "selling" the available resources at a profit to the highest bidders. The system may choose not to sell idle capacity when it is economically preferable to step down that capacity to reduce costs. This work focus on the framework for resource management, not specifically on the algorithm to reduce energy.

In [79] is presented a reconfiguration algorithm for a cluster of servers, with the only purpose to turn on and off servers. They consider multiple resources, and for each resource, they use an independent PID controller which output is the excess demand for

the resource. The controller with largest output is used to determine the ideal cluster configuration at each point in time. An early version of this paper appeared in [78], where the acronym VOVO (Vary-On Vary-Off) was coined to represent the method of turning (varying) on and off the server nodes.

In [40] the IVS (*Independent Voltage Scaling*) and the CVS (*Coordinated Voltage Scaling*) methods were introduced. In the former, each server node decides locally its frequency value, while in the latter scheme, all nodes operate close to the average frequency for the whole cluster. For this, they use a centralized monitor that broadcasts the average frequency to all nodes. They consider that processors may vary its frequencies continuously in a range. They test five combinations: IVS alone, CVS alone, VOVO alone, IVS with VOVO, and CVS with VOVO. They use a simulation model of a web server cluster running workloads constructed from the server access logs constructed from real websites. The process of turning servers on and off with IVS is similar to the one in [29], based in the utilization of each node. For the CVS, they use the model $P(f) = c_0 + c_1 f^3$, representing static and dynamic powers by the constants, and multiplying by the number n of servers to account for the power of the whole cluster. Then they compute this aggregate power for n servers, for $n-1$, and for $n+1$, to obtain the best frequencies to vary on and to vary off a node. Then, given the constants c_0, and c_1, the optimum average frequency range for the clusters with n servers is:

$$f_{varyoff}(n) \leq \text{CPU Frequency} \leq f_{varyon}(n)$$

The IVS method showed a power reduction from 20% to 29%, and CVS achieved a slightly better reduction that may not justify the extra implementation complexity. Up to 50% were achieved by adding VOVO, compared to a not power managed system, that means all machines turned on running at the maximum frequency.

The work in [41] is for single servers. They propose three techniques: the first technique is a task-based DVS policy with a feedback control mechanism. There is a response time goal to be met, and every quantum of time T, the algorithm steps up or down the frequency depending on the measured response time. This DVS policy conserves the most energy for intermediate load intensities. The second technique uses request batching to conserve energy during periods of low load intensity. The network interface processor accumulates incoming requests in memory, while the server processor remains in a low-power state. The server processor awakens when an accumulated request has been pending for longer than a batching timeout. Request batching conserves the most energy for low load

intensities. The third technique uses both DVS and request batching to reduce processor energy usage over a wide range of load intensities.

[80] employ Power-Aware Request Distribution (PARD) at the load-balancing front-end of a cluster serving dynamic web workloads. The load measure is based on the number of connections, and is used to calculate the number of machines to be turned on. This technique, like many others where the on/off algorithm is based only in a number, cannot find an optimal solution, because it cannot specify which particular server will be turned on, therefore is only useful for homogeneous servers. This differs completely from our approach that will be presented in Chapter 6, where we can choose which server will be on, without any predefined sequence.

Our first work to preliminary address the problem of assigning frequencies to the nodes of a cluster was shown in [11]. In that work we used a precomputed table to define each frequency for the servers, normalized to the maximum speed, and compared with the case where all machines receive the same normalized speed. This work was motivated by the work presented in [109], where they adopt the method of using off-line optimization to build a table for on-line look up, but there the intention was to determine the number of active nodes, in a on/off dynamic mechanism.

2.3 DVS and Dynamic Configuration

We now look at the most recent works that used DVS and/or dynamic configuration (VOVO) to reduce power. We must pay attention to what level of QoS awareness each method presents. Some works simply do not take into account QoS, others have some kind of QoS guarantees, reserving resources for the worst case load scenario, but sometimes overprovisioning the system. Our work tries to reduce the QoS at the minimum level specified by the SLA, what will reduce power consumption more than the other methods.

The work in [90] used a feedback loop to regulate the voltage and frequency as a means of providing QoS awareness. Their controller uses utilization as the control variable aiming to keep it around a derived utilization bound that was shown to be a sufficient condition of schedulability. As exceeding this bound does not necessarily imply in missed deadlines, having this utilization bound as a control set-point achieves good results in guaranteeing the QoS close to 1.0. This bound guarantee is based on schedulability tests, making it too conservative, and the system overprovisioned.

One paper that we used extensively as a basis of comparison was [85], which presents

a cluster-wide QoS aware technique based on local DVS and cluster reconfiguration. In that work, they use a local interval-based DVS technique without optimization, based on the calculation per interval of the real-time utilization $U = \sum_i \frac{C_i}{D_i}$. C_i is the average computation time and D_i is the deadline, and they set the frequency to the lowest available discrete frequency bigger than Uf_{max}, where f_{max} is the maximum frequency available in the processor. The dynamic configuration technique is done by defining a sequence of servers to be turned on/off, ordered by the energy-efficiency of each server. The shortcomings are that it must abide by this ordering (to turn them on and off), loosing optimality, and that the DVS scheme is local, making it potentially less power efficient.

The same DVS technique from [85] is used in [109], which presents a technique called LAOVS (Load-Aware On-off with independent Voltage Scale), where the determination of the active node number is made using a table computed off-line. For each load value, the best number of active nodes is obtained considering homogeneous servers. Interval based DVS techniques were studied in [88]. This is an important work because they evaluate DVS policies for power management in systems with unpredictable workloads, the case for web servers. The technique used in [109], and [85], is the application-oblivious prediction, in which the system real-time utilization U is monitored periodically, and if the system is fully utilized, the speed is increased to the next available discrete frequency, otherwise the speed s is updated as the smallest discrete frequency higher than $s.U$. They also show more complex techniques which attempts to predict performance needs by monitoring the arrival rate and CPU requirements of each request, rather than simply observing resource requirements.

Both works [85], and [90], do not allow the maintenance of the QoS at a precise user predefined value. The TQM and GTQM methods presented in Chapter 3 and Chapter 5 differ from these two approaches because we apply a statistical inference solution to guarantee the exact desired QoS level. The goal of TQM and GTQM is to maintain/control QoS at a certain level. This can be done by controlling the QoS directly, as in [64] and [91], but this turned out to be problematic because with the QoS defined as a ratio of deadlines met to the total requests, a large number of requests is necessary to obtain narrowed confidence intervals. Furthermore, the QoS will saturate at 1.0, causing an asymmetry problem and instability, as shown in Chapter 3. In [64] and [91], however, they used a more complicated control, based on a second control loop for the utilization, that can solve the problem of deadline miss ratio saturation at 0, because the saturation condition of both controllers are mutually exclusive. In contrast, in this thesis we propose to control the QoS based on the average tardiness of the web interactions.

In [2], feedback control is used to achieve overload protection, performance guarantee, and service differentiation, based on the same concept of utilization bound presented by [90], thus aiming to meet all deadlines. However, that work applies adaptation of QoS to server load conditions, where the controller actuator can offer degraded service levels accomplished by content adaptation. The web content (e.g., images) is preprocessed and stored in multiple copies that differ in quality and size. Hence, the approach is different, besides the fact that their architecture is primarily aimed for static web content.

Similarly, an autonomic system is described in [102] to allow administrators to set system properties like QoS. For this, they apply control theory with complex feedback optimization techniques where future environment inputs and the future consequences of the control actions are taken into account during optimization, which is multiobjective including power optimization goals. The QoS is defined as response time and is used directly as the controller set-point. However, the focus is more at the control theory rather than the implementation of a real e-commerce environment; the workload is derived from an Internet service provider and they assume continuous DVS settings. For e-commerce environments, an average response time goal alone cannot tell much about the fulfillment of the real-time rules. In this sense, this work is complementary, because it may be applied to our statistical inference to achieve the desired QoS proportion.

Work using queueing theory to model multi-tiered web architectures [61], [62], and [100] is another possibility to compute and control the QoS probabilistically. We have shown in a previous work [48] that it is possible to choose the system settings so that the power is minimized and at the same time the average response time, given by queueing theory, is such that a predefined amount of deadlines are met. However, it is difficult to have a good queueing model for a real e-commerce environment that allows an easy analytical formulation of the response time without having many non realistic assumptions about the workload generation and service times. The approach we use in this thesis is based on a real e-commerce scenario.

A possible approach to dynamic configuration of web servers is to use N $M/M/1$ queues and formulate the optimization problem based on the probability distribution of the waiting time given by queueing theory. The problem is that the assumption of $M/M/1$ queues is far from reality, and the queueing equations make the problem nonlinear, and difficult to solve optimally. A reconfiguration technique for a server cluster based on a $M/M/m$ queueing model was used in [61] to define the optimal number of active servers analytically. The limitation is also that the processors are homogeneous, and the solution

is only the number of servers to be turned on, not an optimal combination of heterogeneous servers like in our work.

Another work related to data center power optimization is [30], which presents a solution to the problem of finding a set of machines to be turned on to run applications in a hosting center. They consider multiple applications sharing the data-center. They apply on/off optimization to determine the number of servers (m_i) allocated to each application i and their frequency (f_i) at any instant, and all m_i servers of application i run at the same frequency f_i. The limitation of that work is the fact of frequencies must be the same for all servers, and all servers have the same power performance. Although we look only at one single application, our optimization consider different machines, with different set of available frequencies, and hence we can look for combinations of servers/frequencies that will reduce power globally inside the cluster. We will show in Section 6.5.1 that for a given combination of servers, we can reduce up to 40% the power consumption of the cluster, compared with [30], and up to 92% compared with a scheme that does only frequency scaling, without turning servers off.

In [50] the authors present the design of a server cluster that can adjust its configuration and the scheduling of requests to optimize the power/throughput ratio. They model analytically the request distribution among servers and from clients, as well as the types of nodes and types of resources. Intra-cluster cooperation is necessary in that case, where a content-oblivious load balancer is used. The optimization problem is to find the request distribution from clients to servers, and among servers, in such a way that the demand for each resource is not higher than its capacity, and to minimize power/throughput. However, the authors do not model DVS nor boot time for changing configurations. Also, their solution is not optimal, as they use simulated annealing in the optimization.

In [52], a multi-tier web system is considered for minimizing the total energy expenditure of the multi-stage pipeline subject to soft end-to-end response-time constraints. They use the average delay of a $M/M/1$ queue for the delay of each stage, and model an optimization problem that can be solved analytically, where the end-to-end delay is computed as $\sum_{i=1}^{N} D_i^{CPU} + D_i^{block}$, where D_i^{CPU} and D_i^{block} are, respectively, the CPU fraction and the I/O fraction of the total delay. The summation is over each tier i, and only one machine is considered for each tier. The solution is applied for multiple homogeneous machines in each layer. Our work differs because we apply intra-layer optimization and consider heterogeneous servers.

In [75] is presented a framework to support dynamic adaptation of applications, such

as a server cluster infrastructure. The proposed method consists of defining a reusable infrastructure to monitor and adapt running applications, and a contract-based adaptation language for the expression of high-level adaptation policies. Adaptation scripts are used to represent the adaptation logic and a contract manager interprets the power management contract. The main advantage is to provide high-level guidance from administrators and developers to control the energy/performance tradeoff, that is, to meet the adaptation requirements of the application.

2.4 QoS Control

Control theory has been used many times, in the last decade, as the solution for performance control in computing systems. A seminal work appears in [95], where the authors change the paradigm of scheduling, applying control theory to maintain the performance of the system stable. Moreover, as pointed out in [56], the computing systems for today's applications will rely on control theory to make systems that can achieve the desired performance objectives.

In [63], different classes of requests are considered. The control actuator does not use DVS, but enforces desired relative delays among classes via dynamic connection scheduling, that is, they use feedback control theory to design an adaptive connection scheduler. They also apply process reallocation. The controller react to load variation by allocating more process to one class and deallocating process to another class, with the goal of providing differentiated services. That work shows clearly the problem of having an unpredictable variable in a control system: the sampling period used was $30s$, and the settling time achieved was $270s$, which is the time for the Web server to enter steady state.

QoS control can also be done by sensing QoS directly [64], [91] rather than by a statistical approach like ours. However, this may be problematic, because the QoS measure will have a saturation point in 1.0 very close to the desired setpoint. This asymmetry can cause instability, as we will show in Chapter 3. In [64] and [91], they solved this problem with a more complicated control, based on a second control loop for the utilization, and the saturation condition of utilization and QoS was proved to be mutually exclusive. These works use actuators that change the scheduling of the system, performing admission control. They also do not apply DVS.

In [90] the authors used a feedback loop to regulate the voltage and frequency as a

means of providing QoS awareness. Their controller uses utilization as the control variable aiming to keep it around a derived utilization bound. However, it differs from our work because their technique is conservative, providing a QoS guarantee always close to 1.0, not controlling QoS at a fine-grain setpoint. Computing systems with utilization control have usually a different goal, which is to enforce a certain utilization by means of admission control, not DVS, to prevent overload conditions. Other recent works in this area are [44, 65, 103, 104].

Control and queueing theories have been proved powerful tools for system modeling, and to be used together. In [65] a queueing model based on Poisson workload is used to compensate, with some predictability, the delayed response of controller metrics caused by averages. The authors claim that the difference between the workload assumption and the real workload can be compensated by the feedback controller. The use of a PIDF controller for stochastic systems is proved to be a valid idea in [84], where the authors introduce the generalized PID for stochastic systems of first and second orders, with filters added to the transfer function of the controller.

In [105] is presented a power-efficient control theoretic architecture for data-centers built up of virtual machines (VMs). Knowing that hardware components subject to power management affect all VMs, they use an upper level MIMO control to find a uniform performance level for all VMs. To control the response time, they use a lower level control with a response time set-point R_s, with the control variable being the average relative response time $r(k)$. As it is based on average response time, it cannot guarantee precisely a specific QoS level. Our work is complimentary, because we offer with the TQM and GTQM workload estimation methods a novel choice of control variable that can be used in any control of computing systems, and that allows the controller to follow with accuracy a predefined QoS setpoint.

Chapter 3

Tardiness Quantile Metric

Measure what is measurable, and
make measurable what is not so.
– Galileo

In this chapter we introduce the Tardiness Quantile Metric (TQM) (published in [13]) to quantify QoS statistically. We make some theoretical propositions of how to control the QoS, not measuring the QoS directly, but based on the probability distribution of the tardiness in the completion time of the requests. The proposed method provides fine-grained control over the QoS so that we can make a closer examination of the relation between QoS and energy efficiency. We study the soft real-time web cluster architecture needed to support e-commerce and related applications. Our testbed is based on an industry standard, which defines a set of web interactions and database transactions with their deadlines, for generating real workload and benchmarking e-commerce applications. In these soft real-time systems, the quality of service (QoS) is usually defined as the fraction of requests that meet the deadlines. When this QoS is measured directly, regardless of whether the request missed the deadline by a small amount of time or by a large difference, the result is always the same. For this reason, only counting the number of missed requests in a period avoids the observation of the real state of the system. We validate the theoretical results showing experiments in a multi-tiered e-commerce web cluster implemented using only open-source software solutions.

3.1 Introduction

Our objective with the TQM is to have a means of exploring the trade-off between energy and QoS in complex web systems, and for this we need to have a fine grain control of

the QoS. Instead of using a QoS measure based on the counting of missed deadlines, we use the on-line measurement of tardiness in the completion time of the requests, because we verified in practice that counting missed deadlines results in poor accuracy and broad confidence intervals. Our contribution is the statistical guarantee that we can achieve for the QoS based on approximations for the probability density function of the tardiness random variable. We show that the average tardiness is directly related to the QoS. To maintain the user specified QoS level, we used feedback control logic, based on a PID controller[1]; the control variable used was the average tardiness instead of number of missed deadlines. Thus, we can show the consequences to the system when the QoS is maintained in a specified level, which is very important for the energy efficiency, because a system maintained in a lower QoS level is generally associated with less resource usage.

We prove the correctness of the proposed theoretical relation between tardiness and QoS. The performance evaluation we present is based on a real implementation of a web store, using commodity hardware and open-source software. The workload is from an industry standard transactional benchmark for e-commerce, the TPC-W (see Appendix A, installed on a heterogeneous cluster running Linux.

3.2 Application and Web Cluster Model

Our cluster model is shown in Figure 3.1, with a front-end server acting as a reverse proxy. The front-end is capable of SSL encryption/decryption, and will distribute the requests to the web server nodes without encryption between front-end and web servers.

Our cluster has two layers after the front-end, with the application server and web server running at the same machine, and a second layer for the databases. As the purpose of this work is to focus on the power management of the web cluster, we replicate the web store in many database servers to avoid bottlenecks at that layer.

3.3 QoS Control

The goal of the system is to maintain/control QoS at a certain level. This can be done by controlling the QoS directly, as in [64] and [91], but it turned out to be problematic because with the QoS defined as a ratio of deadlines met to the total requests, a reasonable

[1] A proportional-integral-derivative controller (PID controller) is a common feedback loop component in industrial control systems.

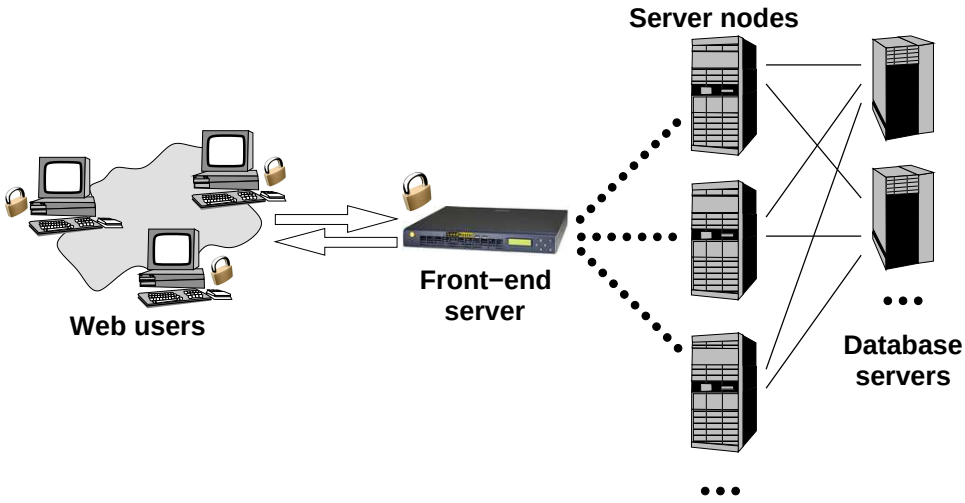

Figure 3.1: Cluster model

number of requests is necessary to obtain narrowed confidence intervals. Furthermore, the QoS will saturate at 1.0, causing an asymmetry problem and instability, as will be shown in Section 3.5. In [64] and [91], however, they used a more complicated control, based on a second control loop for the utilization, that can solve the problem of deadline miss ratio saturation at 0, because the saturation condition of both controllers are mutually exclusive. In contrast, we propose to control the QoS based on the average tardiness of the web interactions. For each web interaction i, we define tardiness by the ratio *web interaction response time* (WIRT) to the respective deadline. That is, $tardiness_i = \frac{wirt_i}{deadline_i}$. A more detailed definition of WIRT will be given in Section 3.4.2. In this section we show the relation between QoS and the average tardiness.

The block diagram for the control logic is shown in Figure 3.2. As will be shown in Section 3.3.1, the user specified level of QoS is applied to a statistical inference method to obtain the necessary average tardiness for that QoS, and if the system is kept with this average tardiness, the QoS is statistically guaranteed to be in the specified value. This average tardiness value is the set-point to the controller.

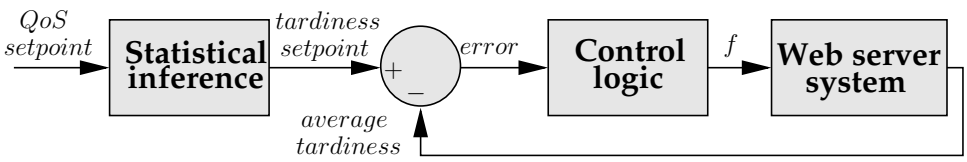

Figure 3.2: QoS control logic block diagram

Described in Section 3.3.2, our PID controller outputs a single frequency scaling factor

u to be used to control the DVS of all the servers. For each server, u specifies the computing capacity. When $u = 0$, the server will run at the minimum frequency, and when $u = 1$, at the maximum frequency. Any value in between will cause the server to cycle periodically between two available discrete frequencies, so that the average frequency is a value proportional to u (see Section 3.3.3 for more details).

3.3.1 Statistical Inference: Tardiness Quantile Metric (TQM)

In this section we show some statistical tests of goodness of fit between the data and the chosen probability distributions. We study more than one distribution in order to choose the best approximation. We also present the theoretical QoS formulation for each distribution.

Using a large dataset, the authors in [35] showed that web traffic, such as response time, can be modeled using heavy-tailed probability density functions, which have self-similarity property, specially the Pareto distribution. We then verified in practice that e-commerce traffic (i.e., WIRT and tardiness) do present a probability distribution close to Pareto. Based on this distribution, we formulated the requirements for the system to meet the specified QoS.

We also propose the use of a second distribution, the Log-normal, which has two parameters that can be easily estimated on-line. The intuition behind using the Log-normal distribution is the fact that the ratio execution time to the deadline has an unreachable lower limit of 0, but has no upper limit, like some variables usually modeled by Log-normal (e.g., personal incomes, tolerance to poison in animals, etc) [51].

The QoS and tardiness value are directly related. The bigger the average relative tardiness, the lower is the resultant QoS. The reason we chose tardiness as a control variable, aside from the problems mentioned earlier, is that tardiness does not carry only a boolean information about QoS: whether the deadline was met or missed, but it is a continuous value possible to be calculated for each web interaction, and its value shows how close the execution was to the deadline. This is a good solution for the problem of choosing the control variable, because the QoS directly does not show as much information as the tardiness measure does. The QoS needs a big amount of web interactions to be calculated with accuracy, making it inappropriate to use in the control, and each interaction carries only a boolean information about QoS: whether its deadline was met or missed. Our experiments show that the confidence interval of the QoS only becomes small after a considerate amount of time. For example, even for a hundred requests, the

confidence interval results in a high value. On the other hand, the tardiness is a continuous value possible to be calculated for each web interaction, and its value shows how close the execution was to the deadline. For this reason, the controller with the tardiness value turns out to be more reactive and without dead-zones, i.e., the variable does not take a long time to react after the actuation of the controller output.

Tho exemplify, supose the two scenarios (a) and (b) of Figure 3.3. In case (a), many requests are finishing execution just a small amount of time before the deadline, and a few are missing the deadline. In case (b), the group of many requests has also missed the deadline. In case (a), the system will react first if using tardiness to quantify QoS, because the tardiness will be with a bad value, while the real QoS will be acceptable, because the requests didn't miss the deadlines yet. On the other hand, using the count of QoS misses, the system will only react in the situation depicted in case (b).

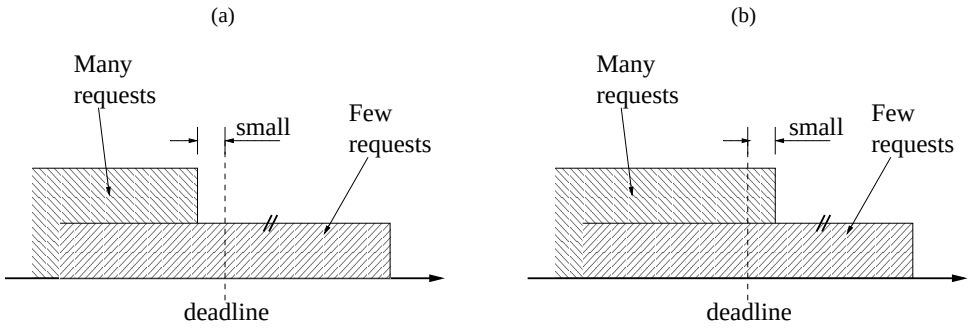

Figure 3.3: Benefit of using tardiness to quantify QoS

The relation between tardiness and QoS is obtained from the probability density function for the tardiness value. We derive this relation from the p-quantile calculation, that is, the tardiness value x such that $P[X \leq x] = p$. Based on the tardiness definition, if the p-quantile is 1.0, then the QoS is p. Hence, we call this method of QoS measuring *Tardiness Quantile Metric* (TQM). In the rest of this section we will show the QoS-tardiness relationship for both Pareto and Log-normal distributions.

TQM with Pareto Distribution

In Figure 3.4, we show the p.d.f. obtained from an experiment run for $2,000$ seconds and $26,255$ web interactions. There is a visual fit between the data and the Pareto distribution, but the Kolmogorov-Smirnov goodness of fit test returns a maximum value between the empirical cumulative distribution and the expected Pareto value of 0.08, while the threshold necessary to accept the data as coming from a Pareto distribution would be 0.01. Figure 3.4 shows that the first bar close to zero is smaller than the second

bar, which does not happen in a Pareto distribution. However, as we will show later, Pareto is still a good approximation to use.

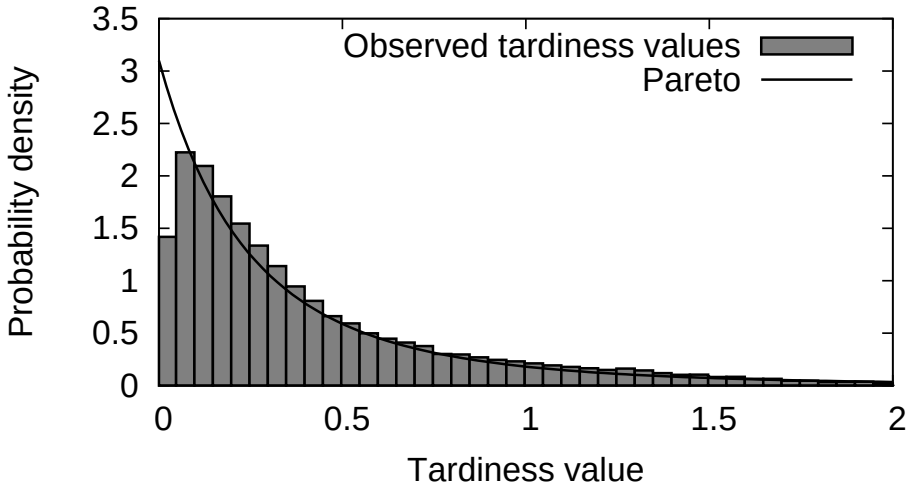

Figure 3.4: Tardiness p.d.f. and Pareto p.d.f.

The representation of a Pareto probability density function is given by $f(x) = k\frac{x_m^k}{x^{k+1}}$, where the parameter k is related to the average μ of the distribution by $\mu = \frac{kx_m}{k-1}$, and x_m is the necessarily positive minimum possible value of X. Note that the tardiness value has a minimum value of 0. For this reason, we use $x_m = 1$ and the transformation $x' = x - 1$. Then we obtain the following equation for the tardiness distribution:

$$f(x) = \frac{k}{(x'+1)^{(k+1)}} \tag{3.1}$$

where $k = \frac{\mu+1}{\mu}$. Let p be the level of QoS desired, that is, $0 \leq p \leq 1$ denotes the fraction of deadlines that must be met. We can formulate the following theorem:

Theorem 3.1 (QoS based on Pareto) *If the tardiness value, defined in Section 3.3, is a random variable with Pareto distribution, a level p of QoS will be achieved, with a confidence level of $1-\frac{c}{2}$, where $1-c$ is the confidence level for the sample mean μ obtained from the system, if the following relation holds:*

$$\mu - z_{\frac{c}{2}}\frac{\sigma}{\sqrt{N}} = \frac{1}{log_2\left(\frac{1}{1-p}\right) - 1} \tag{3.2}$$

where μ is the average value for a set of N samples obtained for the tardiness, σ is the standard deviation for the same set, and $z_{\frac{c}{2}}\frac{\sigma}{\sqrt{N}}$ is the confidence limit for the mean with

the desired significance level c.

Proof We will do the proof in two parts. First we show that the right side of the equation represents the value of the real mean of the data that makes the p-quantile equals 1.0. The web interactions with missed deadlines are those for which tardiness resulted bigger than 1. To have p deadlines met, we need the probability of $0 < tardiness < 1$ to be p. Thus we need $\int_0^1 \frac{k}{(x'+1)^{(k+1)}} dx = p$, resulting in $1 - 2^{-k} = p \Rightarrow k = log_2\left(\frac{1}{1-p}\right)$. As the average μ in a Pareto distribution with minimum value positioned at $x = 1$ is given by $\frac{k}{k-1}$, with the transformation $x' = x - 1$ we have $\frac{k}{k-1} = \mu + 1$, giving $k = \frac{\mu+1}{\mu}$. Solving for μ the equation $\frac{\mu+1}{\mu} = log_2\left(\frac{1}{1-p}\right)$, and adding the confidence limit, we obtain equation 3.2.

The second part is to consider the confidence level. The sample mean μ obtained does not represent the real mean of the data, but in half of the cases where the sample mean is obtained, this value will fall below the real mean, and for the other half will fall above. To guarantee the QoS, we need the real mean below or equal to the right side of the equation. Thus, if the sample mean is controlled in the lower limit given by the confidence interval, the unfavorable cases will happen only in $\frac{c}{2}$ of the cases. This limit is represented in the left side of equation 3.2 by the term $z_{\frac{c}{2}} \frac{\sigma}{\sqrt{N}}$. ∎

TQM with Log-normal Distribution

Now we will show the same idea for another distribution, the Log-normal. A data has Log-normal distribution if the natural logarithm of the data has a Normal distribution. Figure 3.5 shows the histogram of the natural logarithm of the tardiness data and the theoretical Normal distribution, and also shows the *Quantile-Quantile* plot (right side) obtained using SPSS [93]. The *Q-Q* plot is used to verify the deviation of a given data to the normality. The normality of the data will cause a straight line in the *Q-Q* plot. The plot is showing that the data is very close to normal, with some variation on both end tails. We also applied the Kolmogorov-Smirnov goodness of fit test in this case and obtained a better fit, with 0.03 maximum difference between the measured and theoretical cumulative distributions, against 0.08 for Pareto (same 0.01 threshold). Thus, we have the following theorem:

Theorem 3.2 (QoS based on Log-normal) *If the tardiness value, defined in Section 3.3, is a random variable with Log-normal distribution, a level p of QoS will be achieved, with a confidence level of* $1 - \frac{c}{2}$*, where* $1 - c$ *is the confidence level for the sample mean* μ *obtained from the system, if:*

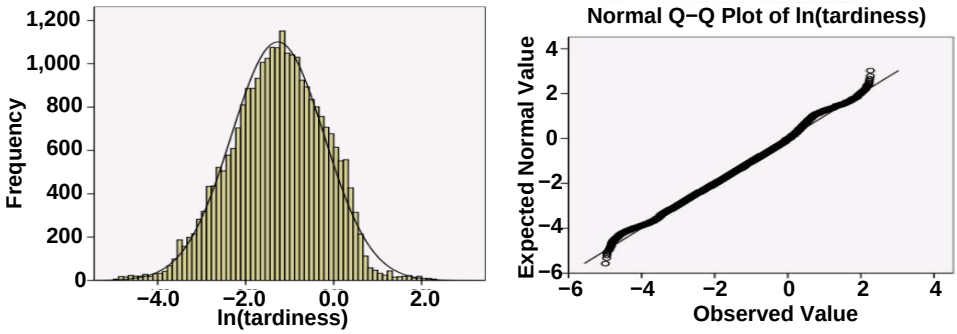

Figure 3.5: P.d.f. of $ln(tardiness)$ with the theoretical normal and the Q-Q plot

$$\frac{\mu - z_{\frac{c}{2}} \frac{\sigma}{\sqrt{N}}}{\sigma} = -\sqrt{ln\left(\frac{1}{1-p^2}\right)} \tag{3.3}$$

where μ and σ are the average value and the standard deviation of the natural logarithm of the tardiness value, considering N samples.

Proof Similarly to the Theorem 3.1, we have the p-quantile calculation and the addition of the same confidence limit. The proof of the right side of the equation follows. Let $f(x)$ be a normal distribution with average 0 and standard deviation σ. Let b be the value of x that results in $\int_{-\infty}^{b} f(x)dx = p$. We have to solve:

$$\frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} dx = p$$

which is solved using the square of this integral equation and the substitution $r^2 = x^2 + y^2$:

$$\begin{aligned}
&\int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} dx \int_{-\infty}^{b} e^{-\frac{y^2}{2\sigma^2}} dy = 2\pi\sigma^2 p^2 \\
&\int_{-\infty}^{b} \int_{-\infty}^{b} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} dxdy = 2\pi\sigma^2 p^2 \\
&\int_{0}^{\sqrt{b^2+b^2}} e^{-\frac{r^2}{2\sigma^2}} 2\pi r dr = 2\pi\sigma^2 p^2
\end{aligned}$$

Using $u = r^2$ and $du = 2rdr$, we obtain $1 - e^{-\frac{b^2}{\sigma^2}} = p^2$, resulting in $b = \sigma\sqrt{ln(\frac{1}{1-p^2})}$. This result is for a normal distribution with $\mu = 0$. In order to have p of the deadlines met, we need a shifted normal distribution so that $b = 0$, because the natural logarithm of the tardiness will be less than 0 whenever the deadline is met. Thus, for this to happen, we need the average of the natural logarithm of the tardiness to be $\mu = -\sigma\sqrt{ln(\frac{1}{1-p^2})}$, which is equation 3.3 without the confidence limit. ∎

Discussion

For the TPC-W specification, where $p = 0.9$, the tardiness average is $\mu = 0.43068$ using the Pareto distribution, and the ratio $ln(tardiness)$ average to the standard deviation of $ln(tardiness)$ is $\frac{\mu}{\sigma} = -1.28869$. In the Pareto distribution, the on-line estimation of the tardiness average has a simpler implementation than in the Log-normal, but both can be done with a low complexity ($O(1)$ for time and $O(N)$ for space). We will show results for many values of specified QoS in Section 3.5, where we used a confidence limit of $\frac{2\sigma}{\sqrt{N}}$ to test both assumptions, yielding a confidence interval of 95.45% for the sample mean, and consequently 97.725% confidence level that the QoS will be equal or higher than the specified value.

3.3.2 Control Logic

We will make use of the classic $z-$transform methodology to derive the equations for the control logic. The $z-$transform is used in signal processing to convert a discrete time domain signal, which is a sequence of numbers, into a frequency domain representation. To make this conversion, the z variable, in the definition of the $z-$transform showed in equation 3.4, must be replaced by $z = e^{sT_s}$, where s is the complex parameter of the Laplace transform and T_s is the sampling interval.

$$X(z) = \sum_{n=0}^{\infty} x_n z^{-n} \tag{3.4}$$

In equation 3.4, where x_n is the n^{th} sample of the signal x, the signal is composed by the most up to date sample, multiplied by z^0, the previous sample, multiplied by z^{-1}, and so on. Thus, this definition can be used to discover the approximate frequency domain representation of a sampled signal. This is used in control theory to build digital filters with the same behavior of the equivalent analog filter.

We applied the $z-$transform to discretize the Laplace equation of a PID controller, given by $G(s) = K_P + \frac{K_I}{s} + K_D s$, where K_P, K_I, and K_D are the proportional, integral, and derivative PID constants, respectively. Using the simplest approximation[2] to find z as a function of s, we obtained the following equation for the controller, which is $O(1)$ in time and space for implementation.

[2]Called the backward difference, which is given by $z = \frac{1}{1-sT_s}$, and is obtained from a first order series approximation to the $z-$transform

$$out_k = out_{k-1} + \left(\frac{K_D}{T_s} + K_P + T_s K_I\right) error_k - \left(\frac{2K_D}{T_s} + K_P\right) error_{k-1} + \frac{K_D}{T_s} error_{k-2} \quad (3.5)$$

where out_k is the k^{th} sample for the output (i.e., the frequency factor u) of the controller, and $error_k$ is the k^{th} sample for the error, which is the difference between the set-point and the actual value of the output (see Figure 3.2). For implementing equation 3.5, it is necessary only to keep in memory the two latest error values, $error_{k-1}$, and $error_{k-2}$.

The average and standard deviation were obtained by using a sliding window of size N. The implementation is $O(1)$ in time for both the average and the standard deviation. At each sample, the average value is updated by the sum of the new value and the subtraction of the oldest value. The space complexity is $O(N)$ for both.

As the focus of this chapter is not the controller itself, we will not address it here. In Chapter 4 we address the controller showing an analysis of sensitivity to the parameters, and with improved control dynamics applying filters in the derivative part. Here, for the proof of concept, we use values $K_P = 0.02$, $K_I = 0.05$, and $K_D = 0.02$, and also the number of samples $N = 200$ that resulted in good responsiveness and stability.

3.3.3 Speed Setting

We use a simple DVS scheme that consists in switching between the two discrete values adjacent to the desired frequency [53]. This scheme is a good solution to the case of a controller actuator, because it offers a continuous, rather than discrete, operating point, so that the controller can have a continuous output. In this scheme, a high priority daemon executes periodically with a duty cycle α with the exact width to stay in the higher frequency, and the remaining of the period in the lower frequency.

As we mentioned earlier, the DVS overhead in modern CPUs is negligible, and switching frequency does not cause any reliability problem, it in fact increases the MTTF of the processor, because it reduces the average device temperature, and consequently also reduces the number of temperature-driven failures [34, 23].

The frequency scaling factor u output by the QoS controller is broadcast to each server node and each server node i calculates the desired frequency f_i given by $f_i = u(F_{max} - F_{min}) + F_{min}$. The duty cycle of the DVS mechanism is α, so that $\alpha||f_i||^- + (1-\alpha)||f_i||^+ =$

f_i, where $||f_i||^-$ is the highest available discrete frequency smaller than f_i, and $||f_i||^+$ is the lowest available discrete frequency bigger than f_i.

We will show in Appendix B that the efficacy of this switched DVS scheme is better than using the lowest available discrete value higher than the necessary frequency.

3.4 Implementation Issues

We describe the system components used in the implementation of our web store on the cluster, and show some implementation issues not directly related to the QoS control, such as the request distribution mechanism, important time measurements, and servers turn-on/turn-off policy.

3.4.1 Hardware and Software

The hardware used in the testbed, summarized in Table 3.1, is composed of the front-end, four machines for the web server tier, and three machines for the database tier, besides one machine to execute the emulated browsers, in the same configuration as Figure 3.1. We chose this configuration so that we were able to focus on the web/application server layer. This configuration puts a load, including SSL processing, of 64% on the front-end and about 80% on the database servers, avoiding bottlenecks.

Table 3.1: Hardware used

Node	Function	Freq. available (MHz)	Specifications
yellow	front-end	Not applicable	AMD Athlon 64 X2 Dual Core 4200+ 2GB RAM
pm1	web server	600, 800, 1000, 1200, 1400, 1600, 1800	Pentium M 1GB RAM
black	web server	1000, 1800, 2000	AMD Athlon 64 3000+ 1GB RAM
silver	web server	1000, 1800, 2000, 2200, 2400	AMD Athlon 64 3400+ 1GB RAM
green	web server	1000, 1800, 2000	AMD Athlon 64 3000+ 1GB RAM
antimony	database	Not applicable	1 CPU Intel Xeon 3.80GHz 8GB RAM
oxygen	database	Not applicable	4 CPUs Intel Xeon 3.60GHz 4GB RAM
hydrogen	database	Not applicable	4 CPUs Intel Xeon 3.60GHz 4GB RAM

The software used was the Apache web server, the PHP scripting language, and the database PostgresSQL. For the TPC-W we used the specification compliant implementation available at the PgFoundry PostgreSQL development group [76]. The front-end

works as a reverse proxy, with the load-balancing Apache module mod_backhand [7], which allows easy addition of new request distribution policies. For the database, it is mandatory to have a distributed database solution in this architecture. In spite of that, as our focus was to study the power management in the web server layer, we used multiple databases without replication. In each database, we deployed an independent web store with 10,000 items and 1,000 customers each. For example, for a load of 600 EBs, we start 200 EBs accessing each independent web store. For the web servers it makes no difference. That is, any request is treated equally, and any server is able to process any request, regardless of what database server will respond to the queries.

3.4.2 Time Measurements

The main problem that makes the implementation in [85] inappropriate to the TPC-W application is that we need to have a way to measure the web interaction response time (WIRT) as a whole, and it is impossible to be made locally in one web server node. The WIRT is defined by the TPC-W specification as the time from the sending of the PHP request by the EB until the receiving of the last byte of the last image embedded in that PHP request. The problem is that the requests to the embedded objects may be sent to different web server nodes in the cluster.

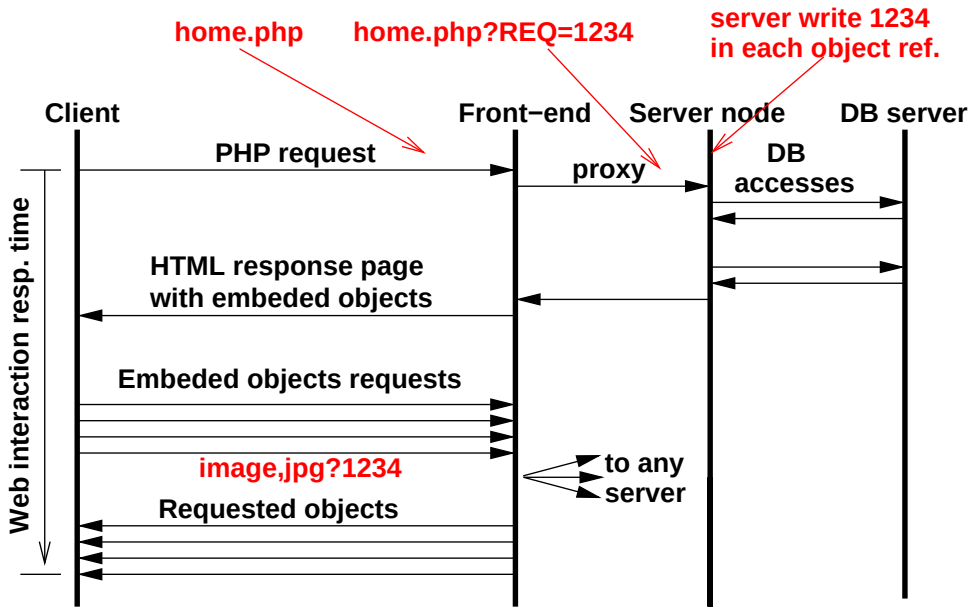

Figure 3.6: WIRT time components

We measure the approximate WIRT at the front-end, excluding only the local network time between the EBs and the front-end. For this, we implemented a new Apache module

that labels the requests before sending them to the server nodes, as shown in Figure 3.6. When the PHP request arrives at the front-end (e.g., home.php), the module creates a unique number and attaches it as a new parameter in the URI of the PHP request. When the web server node receives the request, it gets the label and puts it, also as a parameter, in every embedded object reference. Each subsequent request for every embedded object will come with the label to which PHP request it belongs to. When the request for the last object finishes, the front-end knows the time for the whole web interaction and can compute the QoS and tardiness. We note that this solution does not modify the client at all, and therefore is backward compatible with existing systems. In the case that the application has a very dynamic behaviour, and the number of embedded objects varies on time, the algorithm can still be used, by having the front-end count the number of objects for every request.

Another implementation issue was that we needed to know the average CPU time spent, in user space and kernel space, by each PHP request, for the load estimation in the front-end. We attempted measuring them with direct measurements, but the precision is very poor, because the minimum CPU time, given by a system call called from the PHP script, had resolution of the same order than the execution time itself. Our solution was to design microbenchmarks using functionality from the EBs implementation [76], namely to have them generate specific interactions, in order to exercise each of the interactions separately.

The methodology for the microbenchmarks works as follows. During a period of T seconds, N_r requests type r are issued and the CPU achieves an utilization U. This way, the average CPU time t_r for request r is $\frac{UT}{N_r}$. However, there is a restriction. The TPC-W benchmark specifies a transition diagram with the possible set of transitions allowed after one specific web interaction, and thus, it is not possible to generate all kinds of interactions in isolation. For example, the request to display an order the client has made cannot be issued before the customer actually asks for that order. Similarly, the *Buy Confirm* interaction cannot happen before the *Buy Request* interaction. For the cases with this type of precedence restriction, we used the average value of the precedent interaction to calculate the average CPU time of the next interaction. In a sequence of n interactions, the CPU time of interaction r_i, say t_i, is given by $\sum_{i=1}^{n} N_i t_i = UT$.

The average value measured by this methodology, with $T = 20$ minutes is shown in Table 3.2. This resulted in about $10,000$ interactions in each measurement. The scripts *admin_ confirm* and *admin_ request* could not be determined with precision because they

are not requested very often. In a 20-minute experiment, only 250 such interactions occurred, along with 40,000 other precedent interactions. In fact these interactions are not important, because typical customers do not change or administer the database. Their CPU time, though, is approximately 4 ms, measured directly inside the script for one execution. Again, this measurement is not precise because the granularity of the time function used is 4 ms.

Table 3.2: Average CPU time (system + user) for each PHP script

PHP script	avg. time (ms)	PHP script	avg. time (ms)
admin_confirm	–	new_products	5.417
admin_request	–	order_display	5.456
best_sellers	5.578	order_inquiry	4.126
buy_confirm	6.929	product_detail	4.643
buy_request	6.039	search_request	4.576
customer_reg	4.242	search_result	5.406
home	5.012	shopping_cart	5.336

3.4.3 Request Distribution

As the PHP application depends on the session ID that the server generates and writes in the browser cookies, requests with the same session ID must go to the same server. This is implemented by the mod_backhand software, and is commonly called as a distribution with *sticky sessions*. The web request distribution adopted is based on current load, that is, the amount of work outstanding at the server. The web request is sent to the web server with lowest load, providing that the sticky session rule is not violated. The front-end estimates the load of each server as follows: for each web request, the average CPU time is added to the load estimator when the request arrives at the front-end, and the same value is subtracted after sending the response to the client.

3.4.4 On/Off Policy

The policy used to turn servers on and off affects the QoS control limiting the maximum load of the system and determining the moment to turn a node on, as in [85]. The difference is that we use suspend to RAM, and Wake on LAN and therefore we needed to adopt new values of overhead of time and energy when turning a machine on and off. In Figure 3.7 the activity line is the output of one parallel port pin measured by the same

data acquisition system used to measure the power (in other words, clock skew is zero). A process is started at the same time of the command to shut down the machine ($t = 4$), switching this output. After that, any state different than switching (black part) means that the machine is not operational. It can be seen in the plot that the time overhead to turn off is the period between 7 and 10 seconds. Similarly, the time to turn on goes from 18 to 24 seconds.

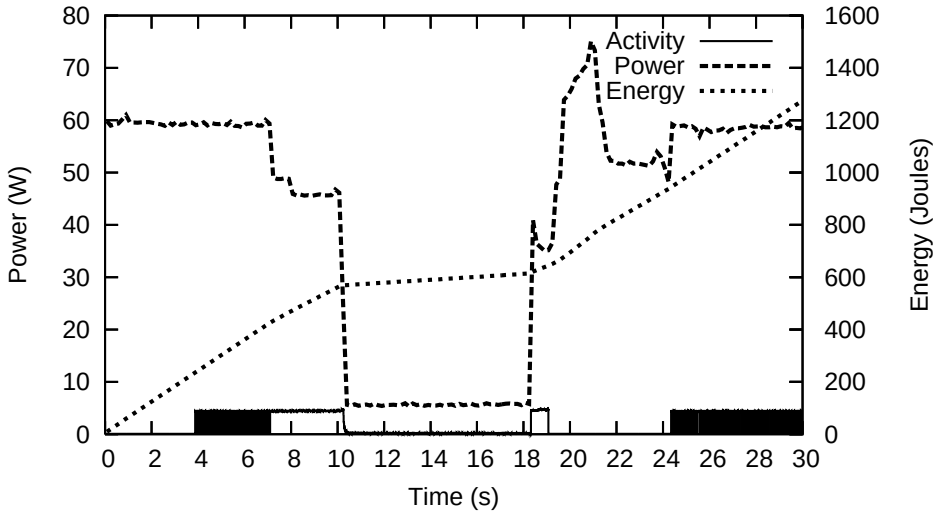

Figure 3.7: Overhead of time and energy for turning on/off the Pentium M server

3.5 Performance Evaluation

Before evaluating the proposed method, we will show empirical proofs for the impracticality of controlling the QoS using a direct measure of the QoS. The plot on Figure 3.8 shows the QoS being measured in a sliding window of size sufficient to store 10 seconds of web interactions information about whether it met or missed the deadline. This size was the biggest size that showed not to compromise the responsiveness of the control. The control set-point was set to 0.98, shown in the plot as a reference line. The plot also shows the control output (u in Figure 3.2) for the two cases: based on the direct QoS measure (sliding window), and based on the tardiness measure (with Pareto distribution).

The first of two problems of measuring the QoS is the broad confidence interval. The confidence interval in this experiment, not shown in the plot, resulted in values up to 0.06. For 0.98, for example, the confidence interval is 0.04, meaning that the real mean will lay between 0.96 and 1.0. For this reason, as can be observed in the plot, more often than not the QoS measure assumes the value 1.0 (for example, between $t = 370$ and $t = 470$), even

though the real mean value (not the sample average) is something different, resulting in instability.

The second problem is that the maximum value of QoS is 1.0. The plot shows several intervals (e.g., $370 < t < 470$ and $t > 550$) where the measured QoS is bigger than the set-point 0.98, giving an error limited to 0.02, resulting in a long decreasing output, because 0.02 is too small. After this period, in most cases the output reached a position that caused an error much larger than 0.02 (e.g., $t = 250$, $t = 350$, and $t = 530$), resulting in a fast increasing of the output. On the other hand, the curve for the output based on tardiness shows a more constant behavior, and there is no asymmetry related to the set-point. Furthermore, the QoS measured in a sliding window during the control with tardiness is more constant, although higher than 0.98, because of the broad confidence interval. As a result, the control based on the direct QoS measure gives periods of high probability of meeting the deadline, followed by periods where it is more likely of missing the deadline than the previous period. Even though the final accumulated QoS for the whole experiment were correct for the two cases (close to 0.98), what is expected is that every web interaction have the same probability of meeting its deadline, uniformly, and the use of tardiness achieves this goal. The energy consumption is higher in the case of controlling the QoS without tardiness, because of the higher variability of the output u.

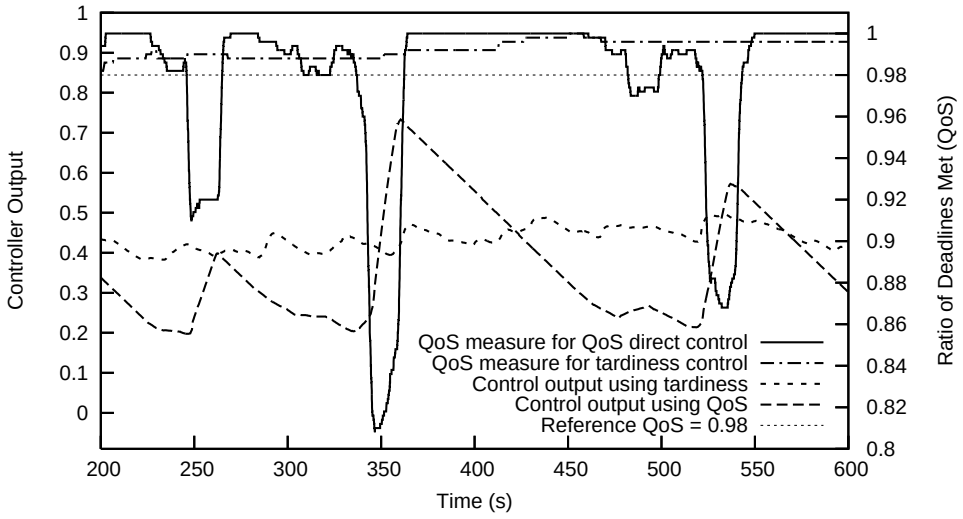

Figure 3.8: Control using direct QoS measure

The most important evaluation we made is to prove the correctness of Theorems 3.1 and 3.2, for the Pareto and Log-normal distributions. We executed the tests with 360 EBs, a number that represents half of full load and requires 4 servers turned on, divided equally into the three database servers and monitored the QoS obtained for each value of

specified QoS. The obtained QoS (accumulated) was measured by the ratio $\frac{missed\ deadlines}{total\ requests}$ for each class of web interaction, and the tardiness values were from the web interaction class with the minimum QoS. In other words, the controller is directed to control the worst QoS among all classes of web interactions. Although conservative, this is to guarantee that all web interactions will stay with a QoS above the specified limit, as it is stated in the TPC-W specification.

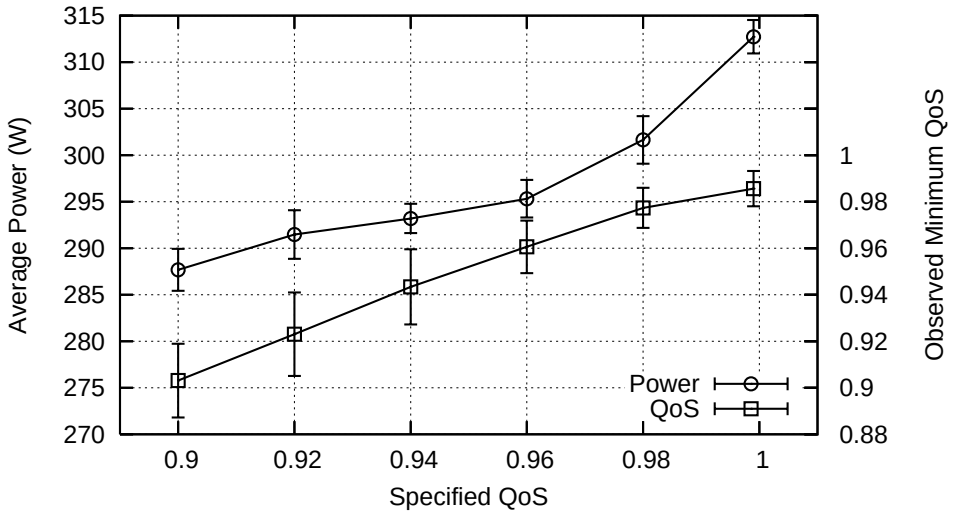

Figure 3.9: Evaluation of the Pareto distribution

The plots in Figure 3.9 and Figure 3.10 show average power and the minimum QoS obtained by our scheme as a function of the specified QoS when using Pareto distribution and Log-normal distribution, respectively. The confidence interval plotted is obtained in each measure by the confidence interval for a proportion, given by $\pm 1.96\sqrt{\frac{p(1-p)}{N}}$, for a 95% interval, where p is the proportion, or the QoS measured.

The Pareto distribution showed very accurate results for QoS values not close to 1.0. The Log-normal showed an error approximately constant of 0.02, and was consistently worse for all values. This is because the Pareto distribution has a better goodness of fit for the tail, which contains most of the requests with missed deadlines. On the other hand, the log-normal distribution had the worse fit exactly in both tails.

Both models, based on Pareto and on Log-normal, have some difficulty to be correct for QoS close to 1.0, as it can be expected examining the theorems. The points in both plots (Figures 3.9 and 3.10) close to 1.0 were actually user specified QoS of 0.999. This happens because in the case of QoS 1.0 the distributions will have no tail at all.

The TPC-W specifies 0.90 for QoS. Normally, when using the TPC-W to measure an

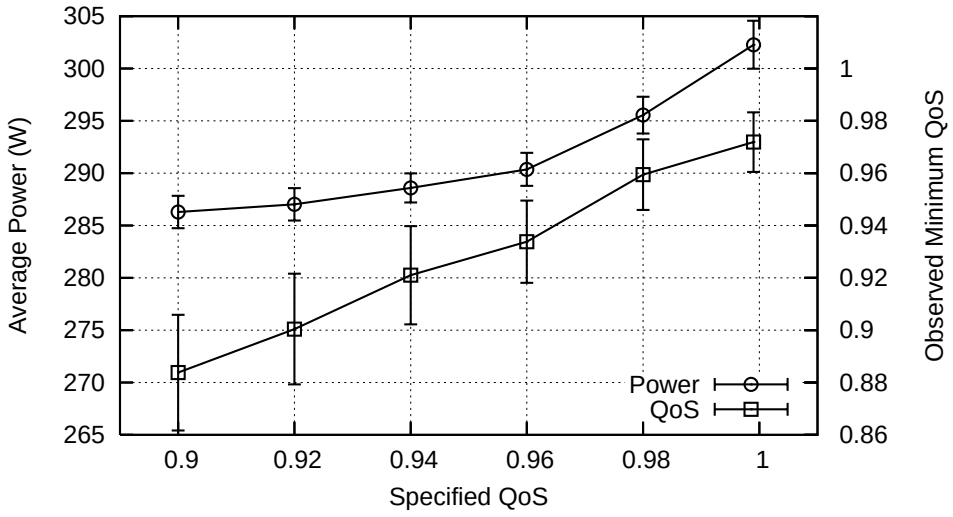

Figure 3.10: Evaluation of the Log-normal distribution

e-commerce system performance, the number of items in the database must be scaled up until the server has minimum QoS of 0.90, and it is found the maximum scaling factor that the system under test can sustain. Thus, to get TPC-W results the system must be in full load. Our system, when not at full load, will slow down to stay in a similar condition of load with the accurate QoS of 0.90, and consequently will reduce the energy cost.

We compared the results, with QoS control based on Pareto, with the implementation in [85]. We made some few modifications in that implementation to accommodate the new real-time model and to support the bigger number of request types. The first result is shown in Figure 3.11, where the TPC-W test was executed for 30 minutes, with a load of 400 EBs. The QoS in the proposed scheme was set to 0.95 ([85] also had a target QoS of 95%). The QoS for [85] is not plotted, because, for this load, the QoS remained very close to 1.0 for all requests. The average power for the scheme proposed in [85] was 320.9 W, while it was 303.2 W for our scheme. This shows that our scheme can accurately specify QoS in a fine-grain manner.

It is important to note that the QoS does not have a value far from the specification at the beginning of the experiment, as it may be wrongly concluded from the plot in Figure 3.11. In the beginning, the measured value does not correspond to the real value, because there is not enough information for a precise measurement. Note that for the first 100 seconds the QoS measured is 1.0, and right after that, the value is not close to the real 0.95 value to which there is a convergence at the end of the experiment, and also, there is a large confidence interval at this point. These are the evidences we mentioned

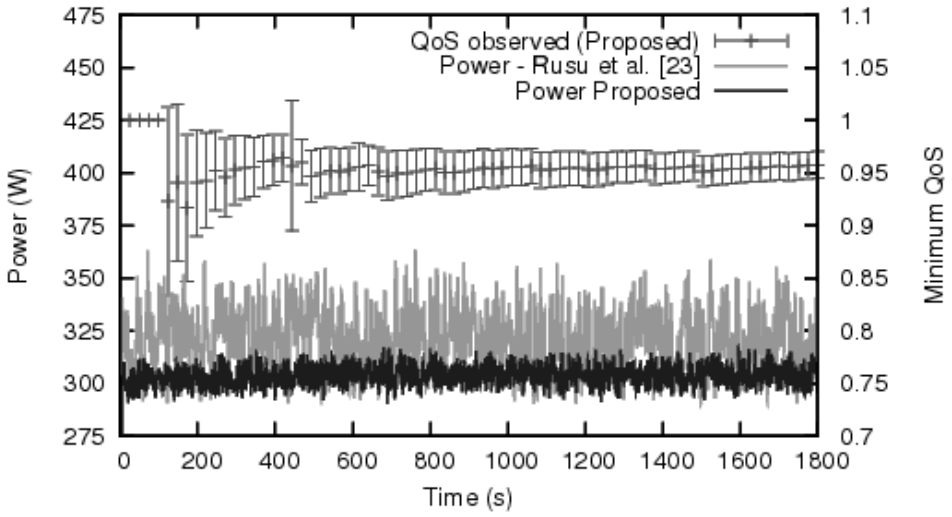

Figure 3.11: QoS and power in the TPC-W test

earlier that the QoS cannot be used as a control variable directly. Because the value has a large error, the control simply could not work. To make the the direct control by the QoS value work, one could argue that the resulting deadzone caused could be made negligible by lowering the frequency to which the CPU frequency and voltage are adjusted. This solution would not help, because the time necessary to wait for a good measure is too high, as the plot shows. Thus, the QoS has to be measured in a long experiment, with the number of requests necessary for achieving a narrowed confidence interval. We also note that the QoS changes in some points. This is because we are plotting the minimum QoS, and sometimes the class of web interaction responsible for the minimum QoS gets a bigger value and the minimum comes from another web interaction.

We also compared the new scheme with [85] for several different loads (see Figure 3.12), using the specified QoS of 0.95, the same as in [85]. The experiments show that we can save power by having an accurate control of QoS.

In Figure 3.13 we show that, even though we are focusing on the web server layer, the energy consumption of the web servers depends on the load of the database layer. We executed the same load in two different scenarios. In the first, all clients were directed to only one database, and in the second the clients were distributed to the three available databases. In the first scenario the database showed almost full utilization, against about 30 percent in the other option. When the load at the databases is higher, the web server layer has to speed up to compensate the response time increase at the database layer. Thus, the question arises on how to cleverly integrate the power management among the different tiers in a multi-tiered architecture. Figure 3.13 also shows the QoS obtained for

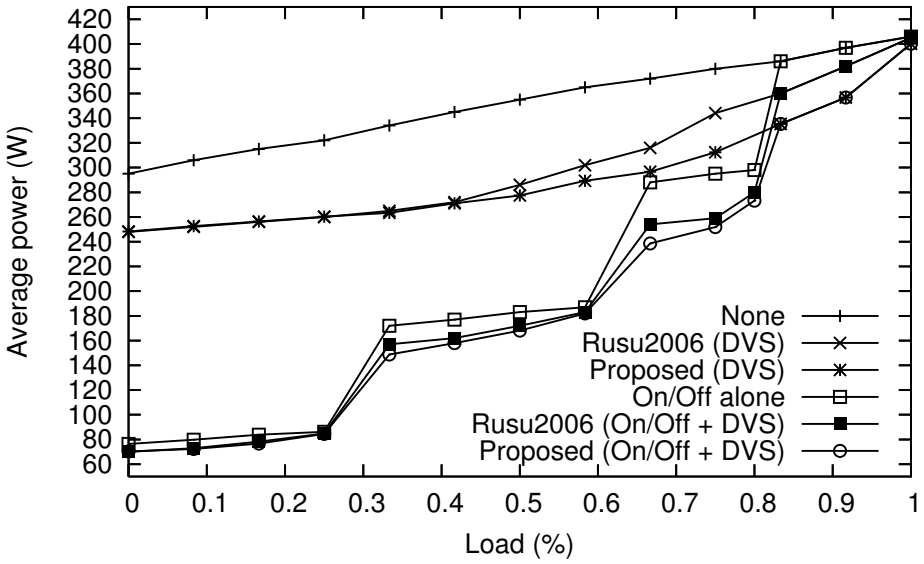

Figure 3.12: Evaluation of the proposed scheme

both scenarios. We omitted the confidence intervals for better clarity, because they were superposed. It is important to note that both stayed close and above the QoS specified at 0.95.

We noticed also in the experiment of Figure 3.11 that the QoS always stays above and very close to the specified. This happens because we used the confidence limit as stated in theorems 3.1 and 3.2.

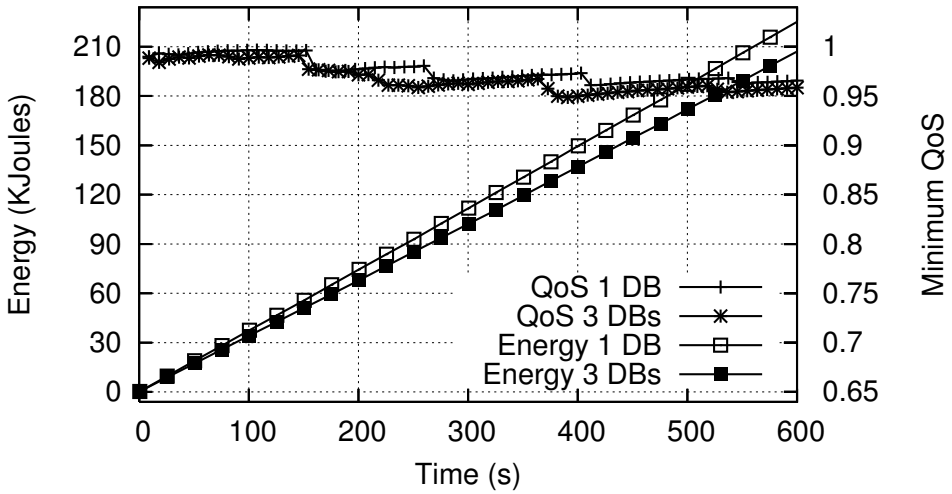

Figure 3.13: QoS and energy consumption for two scenarios with different database load

The recomended performance metric by TPC-W is WIPS, which we measured for our proposed scheme and for the scheme proposed in [85]. For the experiment shown

in Figure 3.11, the averages were 49.79 and 54.99 WIPS for our proposed and for [85], respectively. We had 10.4% less performance, but with a controlled quality of service in a value that attends the minimum level specified by TPC-W for achieving customer satisfaction. We used 95% in this experiment, but we can achieve even more power savings with the TPC-W requirement of 90%. The scheme proposed in [85] achieved better performance because of overprovisioning the system with respect to the real-time specifications.

3.6 Conclusions

In this chapter we presented a scheme to relate QoS to tardiness in a multi-tiered environment designed for e-commerce, based on the statistical distribution of the tardiness of web interactions. This QoS metric was shown to be very useful because some practical difficulties arose when we tried to use the measured QoS in the control. On the other hand, tardiness is a continuous value that can be calculated for each web interaction, and its value depicts how close the execution was to the deadline.

We proposed two approaches, based on the probability density function adopted to represent the tardiness data: using the Pareto distribution and using the Log-normal distribution. We showed that the Pareto distribution achieves better results in the accuracy of the resultant system QoS, for values of user defined QoS not close to 1.0, and Log-normal showed to have a constant error due to differences in the fit of the data to the distribution. Our proposed scheme using Pareto was shown to be better than existing schemes like [85] and [90], because it meets with precision the real-time specification, not overprovisioning the system, and thus saving energy. Although easy to implement, because there is a closed formula to express the QoS, a shortcoming of our approach is when the goal is to meet all deadlines, the tardiness would have an upper bound of 1, and thus the assumption on the tail distribution does not hold. In this case, which is not the goal of our method, the cited existing schemes would be more precise.

The major drawback of the TQM presented is that it is based on predetermined probability distributions: Pareto and Log-normal. The approximations showed to be good, but the first questions that arise are: what if the workload is not Pareto and not Log-normal? Is it possible to generalize? Fortunately the answer is yes, we can generalize, by using stochastic approximation algorithms that can measure some characteristics of a random variable regardless of its probability distribution function. Then we came up with

the Generalized Tardiness Quantile Metric (GTQM), which we present in the Chapter 5.

Before presenting the GTQM, in the next chapter we show more details of the control logic used by the QoS controller. We show the implementation of a feedback control loop that acts on the speed of all server nodes capable of dynamic voltage scaling, with QoS measured using TQM being compared to the QoS setpoint.

Chapter 4

QoS Control

A penny saved is a penny earned.
– Benjamin Franklin

In this chapter we describe a simplified way to implement performance control in a multi-tier computing system designed for e-commerce applications. A paper was published in [12]. We show that the simpler SISO (*Single Input Single Output*) controller, rather than a more complex distributed or centralized MIMO (*Multiple Input Multiple Output*) controller, works well, regardless of the presence of multiple cluster nodes and multiple execution time deadlines. By changing the speed, we change the position of the p-quantile of the tardiness probability distribution, a variable that enables to measure QoS indirectly. Then, the control variable will be the average tardiness, and the setpoint the tardiness value that will position this p-quantile at 1.0, value at which a request finishes exactly at the deadline. Doing so will guarantee that the QoS will be statistically p. We test this new Tardiness Quantile Metric (TQM) in a SISO PIDF control loop implemented in a multi-tier cluster. We use open software, commodity hardware, and a standardized e-commerce application to generate a workload close to the real world. The main contribution is to empirically show the robustness of the SISO controller, presenting a sensibility analysis of the four controller parameters: damping factor *zeta*, derivative filter factor *beta*, integral gain *ki*, and zero time constant *tau*.

4.1 Introduction

As people increase their trust on Internet means for services like banking and commerce, electronic applications become everyday more popular and widespread. The complexity

of the computing systems for these applications are increasing fast, both for well established popular kind of applications such as e-banking and e-commerce, and also for less known business-to-business applications, such as e-sourcing, where businesses auction the willingness to purchase from the seller who can offer lowest prices and best contracts. Due to the needed complexity and size, computing systems are becoming complicated, dense, and of high cost of ownership. As pointed out in [56], because of this growing complexity, the computing systems for today's applications need to be able to do self-configuration and self-optimization, and act in an autonomic way, such that it can optimize itself seamlessly to the desired performance objectives. With the motivation that control theory will play a crucial role in the development of complex and large scale computing systems, we present in this chapter a practical use of control theory for multi-tier clusters to host e-commerce and related applications.

Following the work in [38], where the authors discussed the scaling aspects of control problems that arise in large computer systems, our control borrows some characteristics from the centralized MIMO (*Multiple Input Multiple Output*) models. They used as a target architecture a multi-tier e-commerce system composed of multiple layers of web clusters, each layer used to process a different part of the web request, namely, request distribution (layer 1), static and dynamic requests (layer 2), and database access (layer 3). In their classification, for any performance control, an e-commerce system has to be either MIMO centralized, where there is a centralized controller with multiple actuators and multiple sensors, or MIMO distributed, with several distributed independent controllers. The authors claim that the controller for an e-commerce system has to be MIMO by necessity, for example, because of the existence of multiple web request types with different response time objectives. However, in our practical implementation of a multi-tier e-commerce web cluster, the industry standard e-commerce application used presented some restrictions that make it impracticable to read the control metric from the multiple servers. The reason is that the information, or control metric, is distributed across the cluster, and the only way to measure it is at the front-end server where the controller runs. This prompted us to build a SISO *Single Input Single Output* controller, using a normalized response time among classes of requests to obtain a single control metric that normalizes the several different time constraints.

We based our implementation on open source software and industry standard workloads. Open-source software offers a huge advantage for controlled computing systems, because virtually any metric or measurement can be derived from the system, as we have total access to the source code, from the core kernel level to the application user level.

Our objective is to accomplish energy consumption minimization and QoS (*Quality of Service*) guarantee. We build a feedback control loop that regulates the performance of all dynamic voltage scaling (DVS) capable server nodes (i.e., layers 2 and 3), with QoS being the reference control objective. But rather than sensing the QoS directly, which is measured as a ratio of number of requests that executed within their deadlines to the total number of requests, we use a new metric of QoS based on the tardiness of the completion of web requests proposed in Chapter 3, where tardiness, the control variable, is defined as the ratio of web request response time to the deadline. This metric is based on the probability distribution of tardiness, and because it presents more information about the completion of tasks than the QoS, it offers a better metric for using in a feedback control loop.

We will apply the theory of a PIDF controller, which is basically a proportional-integral-derivative (PID) controller augmented with a low pass filter (F) in the derivative part. The workload of a web system is a composition of random variables, and consequently, present the random fluctuations that is characteristic of any stochastic process. We consider the unpredictability of the workload as being similar to sensor noise. With the low pass filter, the process disturbance caused by random oscillation will be rejected by the controller. In such a web system, it is desirable to have the derivative component, because as the plant dynamic presents a dead time delay, it is important to have the predictive characteristic given by the derivative part. Besides, we need also to include averages in the control variable to handle the intrinsic randomness. We will measure the plant dynamics after the inclusion of the averages and apply some tuning rules for the controller.

The contribution of this chapter is the practical implementation and robustness evaluation of the control loop for a real e-commerce web server cluster, with sensitivity analysis to the parameters of the PIDF controller. The goal was to have a means of proving the concept of TQM and GTQM, by having a control loop implemented in the cluster computing system.

4.2 Background

In this section we describe briefly the cluster model used and give some basic concepts of control theory.

4.2.1 Cluster Model

The cluster architecture is composed of a central web server that serves as a front-end to the whole system (layer L1), a layer L2 of servers to process dynamic and static requests, and the L3 layer to execute a distributed database that will store all the information related to the application. The front-end node implements a request distribution policy based on the amount of work that each second-tier server has. The front-end server acts as a reverse proxy, that is, it redirects requests to other servers and also returns the server's response to the client. The front-end is capable of SSL encryption/decryption as required for the e-commerce application. The load distribution among the database servers is done statically. We replicate the web store in many independent database servers to avoid bottlenecks, and the total load is divided equally to each database. To implement this architecture we used in layer L1 the Apache web server with the module *backhand* [7] for load balancing and a new module to implement the controller, in layer L2 we have Apache with PHP scripting language support for the dynamic pages, and in L3, PostgreSQL for the databases. The workload generation used is based on TPC-W.

4.2.2 PID Control

The PID control is the most used control technique in industry. The first PID controller appeared in the 1930s [58], and is still used and researched because of its operational simplicity, and because it provides generic and efficient solutions to control problems. A PID controller computes the error between the controlled variable measured in the plant, and the desired setpoint value, and according to this error signal, it generated a signal sent to the actuator that will eliminate this error. The actuator is part of the system that can produce a change in the controlled variable. For example, in a computing system where we want to control the system's performance, the actuator may be the DVS speed or frequency. If we change the DVS frequency of the processor, it will show a faster or slower average response time. The response time in this case would be the control variable. The setpoint would be a predefined response time that we desire the system achieves in the average. Thus, the PID controller will compare the measured response time with the setpoint and adjust the DVS frequency in such a way that the measured response time reaches the setpoint in steady-state. The settling time is the time it takes to reach this steady-state.

The PID controller has three components: proportional (P), integral (I), and derivative (I). The computed signal u sent to the system's actuator is a combination of these

factors. The proportional term provides a control action proportional to the error signal, and the goal is to be able to instantly calculate an output that is independent on the initial conditions of the system. The proportional action alone is unable to eliminate the error in steady-state. The integral term reduces steady-state errors, because the output increases as a calculation of the integral of the error signal. The derivative term improves transient response with some predictive capability. The classic PID controller is then given by:

$$u(t) = k_P e(t) + \frac{k_P}{k_I} \int e(t)dt + k_P k_D \frac{d}{dt} e(t) + u_0 \tag{4.1}$$

This is called the parallel PID controller, because all components are computed in parallel and then summed. The proportional factor k_P is multiplied in all components. T_I and T_D are the integral and derivative factors respectively. T_I and T_D have time units, and normally the integral and derivative constants used are k_I and k_D, where $k_I = \frac{1}{T_I}$ and $k_D = T_D$, what we adopt in this work.

4.2.3 Implementation in a Discrete System

In a computing system, time is discretized. Any dynamic system can be represented in a discrete system by using the $z-$transform. After representing the system in the frequency domain, that is, applying the Laplace transform, the discrete equivalent system is derived by substituting the Laplace variable s by the variable z using a function $s = f(z)$. The variable s can be related to the z variable by the expression:

$$z = e^{sT} \tag{4.2}$$

where T is the discretization time, or sampling period, that is, the interval between values of a discretized signal. The Figure 4.1 shows a signal which value is known only every T seconds. We have a sequence of samples, and the last sample is the k-th sample.

If we take z^{-1} from Equation 4.2, we will get $z^{-1} = e^{-sT}$, what is the known Laplace formula for a delay of T seconds. That is, if a signal is multiplied by e^{-sT} in the frequency domain, the signal will be delayed by T seconds in the time domain. Then, the discretized signal X of Figure 4.1 can be represented by:

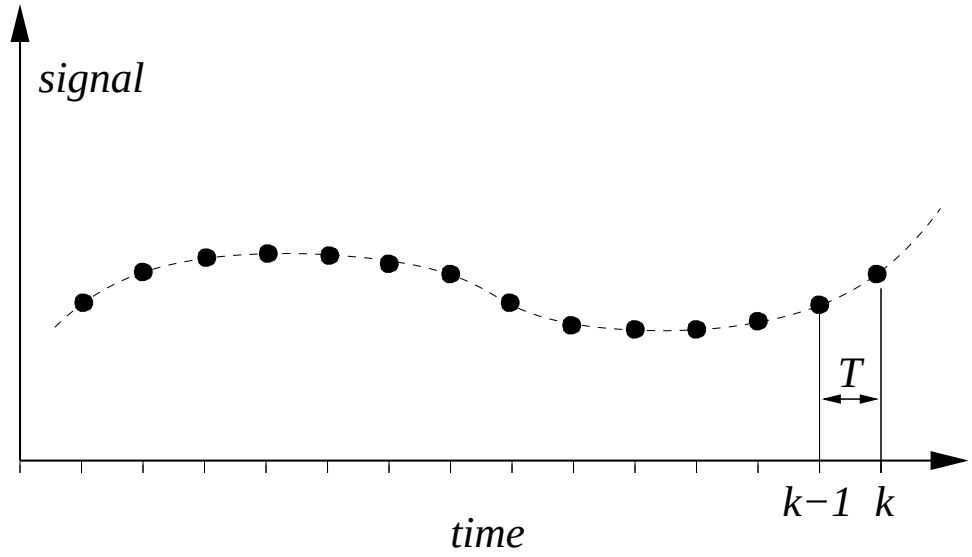

Figure 4.1: Discretized Signal

$$X(z) = \sum_{n=0}^{\infty} x(n) z^{-n} \tag{4.3}$$

where n is an integer, and the signal is supposed to be defined only for $t \geq 0$.

The $z-$transform can be used to represent a signal in the discrete domain, or even be used to represent a formula of a controller or a filter in order to be implemented in a discrete system. The backward difference, given by $1 - sT = z^{-1}$, that is obtained from a first order series approximation to the $z-$transform, is the most used approximation for the Equation 4.2. One way to see this relation from Figure 4.1 is by the approximated calculation of the derivative of the signal between points $k-1$ and k. The derivative $\frac{d}{dt}x$ is $\frac{x(k)-x(k-1)}{T}$. In the frequency domain, the derivative is represented by the Laplace variable s. Then, $sx = \frac{z^0 x - z^{-1}x}{T}$, or $s = \frac{1-z^{-1}}{T}$. What gives $1 - sT = z^{-1}$.

When the laplace formula of the controller is transformed by $1 - sT = z^{-1}$, the result is a recurrence formula that implements the dynamic behavior of the original formula. This will be shown in the next section when applying this idea to our PID controller.

4.3 Control Logic

The control logic uses the TQM control input metric as described in Chapter 3, that is, based on the probability distribution of $tardiness_i = \frac{wirt_i}{deadline_i}$, for all web interaction i. The DVS Actuator mechanism is the same as defined in Section 3.3.3.

Figure 4.2 shows the control logic block diagram adopted. As suggested in [58], we model the noise as the input signal $w(t)$; in our model, noise is present in the measure because of the stochastic nature of the workload $v(t)$ (the process disturbance), which will cause the randomness present in the tardiness measure. The controller output is $u(t)$, and the transfer function $K(s)$ of the controller has a minus because it has to invert the output related to the input error. When the error is negative, the p-quantile for the QoS p is bigger than 1.0, and the deadline miss ratio is bigger than $1-p$, and therefore the server must increase the speed. $G(s)$ is the unknown plant transfer function; we will measure its dynamics in Section 4.4.2. $A(s)$ represents the averaging included in the control variable.

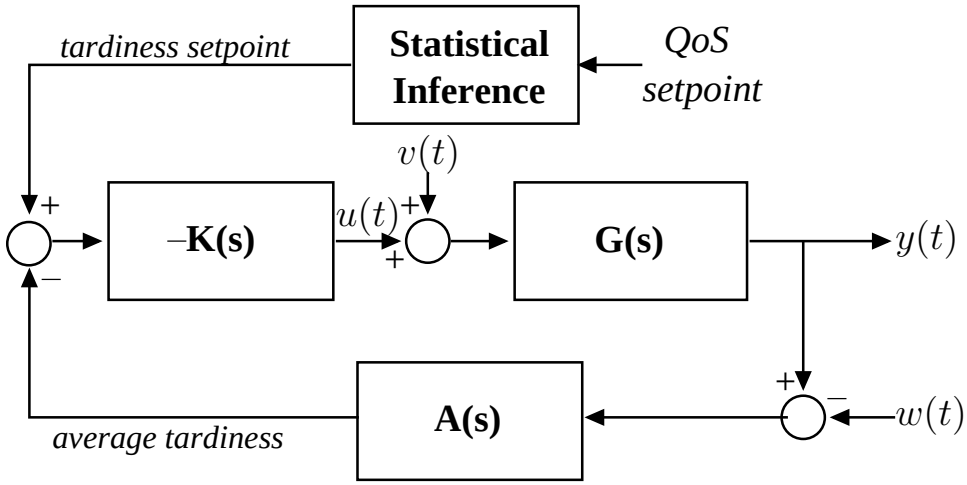

Figure 4.2: Control logic block diagram

We have used in Chapter 3 a simple PID controller given by $K(s) = k_P + \frac{k_i}{s} + k_D s$. To improve it, as suggested in [43], we insert a lowpass filter in the derivative part to make it reduce the noise, and we change the parameterization of the controller as proposed in [43]. With only the lowpass filter, the controller becomes: $K(s) = k_P + \frac{k_i}{s} + \frac{k_D s}{1+sT_f}$. The new parameterization will use the four parameters: dumping factor (ζ), derivative filter factor (β), integral gain (k_i), and zero time constant (τ). The advantage of using these parameters is better stability, because it reduces the freedom of the traditional parameters in a way that the controller is easily kept in a stable region. This parameterization also makes the controller tuning procedure easier. The resultant controller is:

$$K(s) = k_i \frac{1 + 2\zeta\tau s + \tau^2 s^2}{s\left(1 + s\frac{\tau}{\beta}\right)} \tag{4.4}$$

where $\beta = \frac{k_\infty}{\tau k_i}$, and $k_\infty = \lim_{s\to\infty} K(s)$.

The damping factor ζ dictates the responsiveness of the controller. With a increased

ζ, the system becomes slower to achieve steady-state, and with a small ζ, the overshoot increases. The zero time constant τ is dependent on the plant dynamics. In [58] a very simple method of tuning the controller is to make $\tau = \frac{T}{3}$, where T is the time constant of the plant (for a first order plant, the time the output takes to achieve 63.2% of the input in the step response). The filter factor β is related to the high-frequency gain, or control activity, $k_\infty = \beta\tau k_i$. If β is small, the system may lose control activity and perform as if in a positive retrofit (see Section 4.4). Increasing k_i will increase the performance of the controller.

In the discrete domain, the controller equation relating the discrete output u_k to the discrete error e_k becomes:

$$K(z) = k_i \frac{T_s + 2\zeta\tau + \frac{\tau^2}{T_s} - \left(\frac{2\tau^2}{T_s} + 2\zeta\tau\right) z^{-1} + \left(\frac{\tau^2}{T_s}\right) z^{-2}}{T_s + \frac{\tau}{\beta} - \left(T_s + 2\frac{\tau}{\beta}\right) z^{-1} + \left(\frac{\tau^2}{T_s}\right) z^{-2}} \tag{4.5}$$

The discrete equation obtained by straightforward manipulation of Equation 4.5 is in the recurrence formula in Equation 4.6.

$$\begin{aligned} u_k = \frac{(\beta T_s + 2\tau)\, u_{k-1}}{\beta T_s + \tau} - \frac{\tau^2 k_i \left(u_{k-2} - e_{k-2}\right)}{T_s \left(T_s + \frac{\tau}{\beta}\right)} + \\ \frac{\left(T_s + 2\zeta\tau + \frac{\tau^2}{T_s}\right) k_i e_k}{T_s + \frac{\tau}{\beta}} - \frac{\left(\frac{2\tau^2}{T_s} + 2\zeta\tau\right) k_i e_{k-1}}{T_s + \frac{\tau}{\beta}} \end{aligned} \tag{4.6}$$

4.4 Evaluation and Sensitivity Analysis

In this section we present a set of experiments with the controller. The first step is to measure the process dynamics in open loop and then tune the controller accordingly. We adopted the tuning procedure given by Equation 11 of [58]. For the closed loop, all tests use a QoS setpoint of 0.95.

4.4.1 Process Dynamics

We adopt the first order plant with delay of Equation 4.7, where L_d is the lag delay, or the time it takes for the output to change after a step response, and T is the time constant.

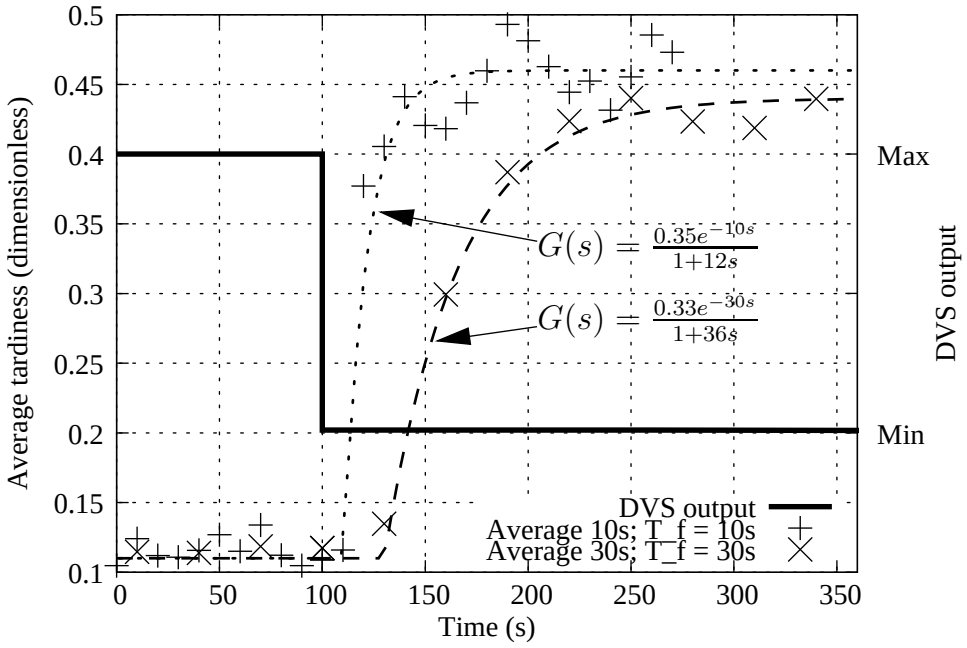

Figure 4.3: Step response in open loop, for $10s$ average with $T_f = 10$ s and $30s$ average with $T_f = 30$

$$G(s) = k\frac{e^{-sL_d}}{1 + sT} \tag{4.7}$$

We will show results (how the process dynamics change) for two different time windows for computing average tardiness (which are also the sampling period T_s). We will use an average of 10 seconds plus an additional filter with constant $T_f = 10s$, and to test bigger averages, we use window average of 30 seconds plus an additional filter with constant $T_f = 30s$ (a sampling period and average of 30 seconds was also used in [63]). This lowpass filter in the measurement is required for smoothing and improving the measurement of the control variable. With this averaging scheme implemented, we measured the step response for both cases, and the result is in Figure 4.3. We will use this figure in the next section for fitting with the plant model adopted.

4.4.2 Tuning

We did curve fitting from the results in Figure 4.3 to extract the parameters of the plant model. We obtained $L_d = 10s$, $T = 12s$, and $k = 0.35$ for the $10s$ case and $L_d = 30s$, $T = 36s$, and $k = 0.33$ for the $30s$ case. Applying these values to the tuning rule described by Equation 11 of [58], we obtain $\zeta = 0.83$, $\tau = 6.52$, $k_i = 0.29$, and $\beta = 3.91$, for $10s$ case, and $\zeta = 0.83$, $\tau = 19.56$, $k_i = 0.10$, and $\beta = 3.68$, for the $30s$ case. The study in [58]

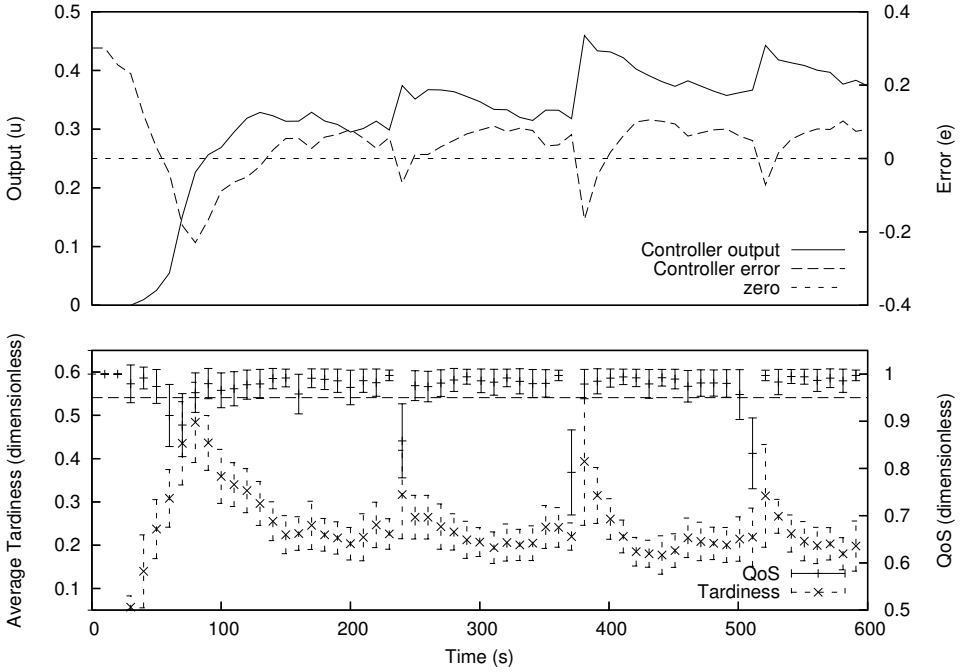

Figure 4.4: Control performance with $10s$ average

showed that these values yield closed-loop behavior close to optimal, for first order plants with moderate time delay. In our case, with $10s$ delay resulted in good stability, but a $30s$ delay was too large and did not yield good results (see Section 4.4.3).

4.4.3 Results

In all experiments, the control variable used is not only the average tardiness, but the average tardiness added to the confidence limit calculated every sampling interval. For example, if in one given sampling interval the average tardiness measured with its confidence interval is 0.30 ± 0.05, the control variable will be 0.35 rather than 0.30. This is to guarantee, with the confidence level adopted (95%), that the QoS will lay above the specified value.

In Figure 4.4, the tuning rules resulted in stable operation of the controller with $10s$ average. The QoS measured every interval remained above, in most cases, the specified value of 0.95, as expected, because we controlled by the confidence limit. The points close to $t = 240s$, $t = 380s$, and $t = 510s$ with low QoS were caused by load imbalancing that is difficult to avoid when all servers run almost with full utilization.

Figure 4.5 shows the $30s$ case. As the lag delay was too big, the tuning rules failed. With a too small β, the integral part is not sufficient to recover from a negative error. The effect is of a positive retrofitted system. We solved this by increasing β and increasing

k_i, for better performance and better control activity. The result is in Figure 4.6, which also shows the increase in control activity with higher β. For the remaining experiments, one parameter will be changed, while the others will remain the same given by the tuning rules.

In Figure 4.7, we show that increasing the integral gain k_i, the performance increases. The curve with $k_i = 0.1$ is much slower than with $k_i = 0.3$. However, $k_i = 1.0$ is too big, and resulted in instability.

Figure 4.8 shows the effect of varying the damping factor ζ. As was expected, an increase in ζ lowers the overshoot of the system, but increases the time to reach the setpoint.

In Figure 4.9 we show the effect of the parameter τ. The zero constant must be tuned with the plant dynamics. The value $\tau = 6.5$ was the value returned by the tuning rule. We also experimented with $\tau = 3$, which was too small and did not allow the system to correct the positive error, and $\tau = 12$, which caused difficulty in correcting a negative error.

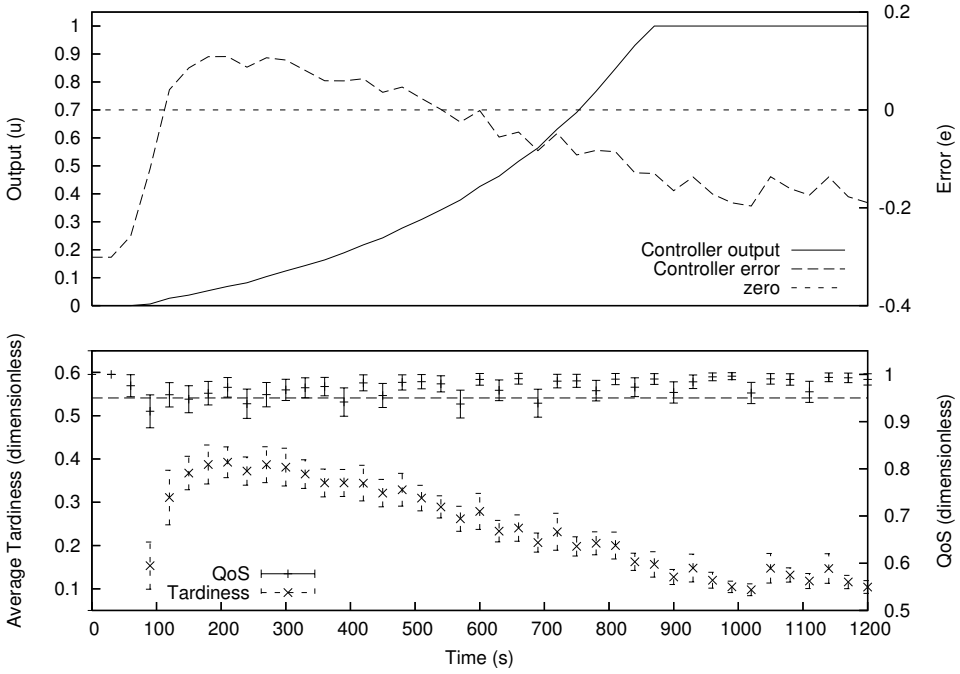

Figure 4.5: Control performance with 30s average

In this chapter we have not shown any energy measurement because we focused more in the stability analysis and sensitivity to parameters, issues that we could not assess in Chapter 3, where we compared the energy consumption with other interval based DVS mechanisms and we showed that extra energy savings can be achieved with the fine-grain

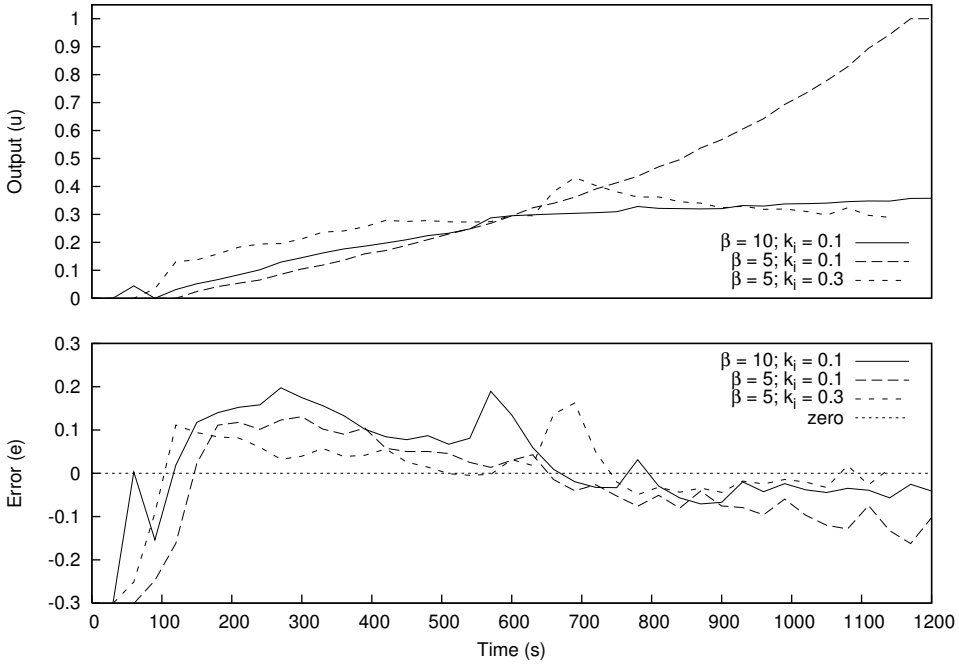

Figure 4.6: Control performance with 30s average: changing tuning parameters

QoS control proposed. We did not evaluate, however, the energy-efficiency of the system during the settling time, which will depend on the tuning rules. This is not an important issue because the settling time of 150 seconds, observed in Fig 4.4, about half the settling time obtained in [63], is sufficiently small to accommodate the workload variation.

4.5 Discussion

In this work we followed the general framework for describing control problems presented in [38]. They use a multi-tier e-commerce system as illustration and classify the possible control architectures, including *SISO, MISO*, and *MIMO*, which refer to the number of inputs and outputs of the controller (S = single, M = multiple). MIMO, in particular, can be further divided in centralized and distributed. The authors argue that e-commerce systems are MIMO by necessity, because the target system must have multiple inputs in order to achieve multiple objectives, and must have multiple outputs in order to measure the multiple objectives (see Fig. 4.10a).

However, although this classification is very reasonable, there are practical issues to implement the e-commerce web system, and it turns out that it is possible to use a simpler SISO architecture, as shown in Figure 4.10b. As the chosen metric to be used in the controller was the tardiness of web interactions, and because of the definition of

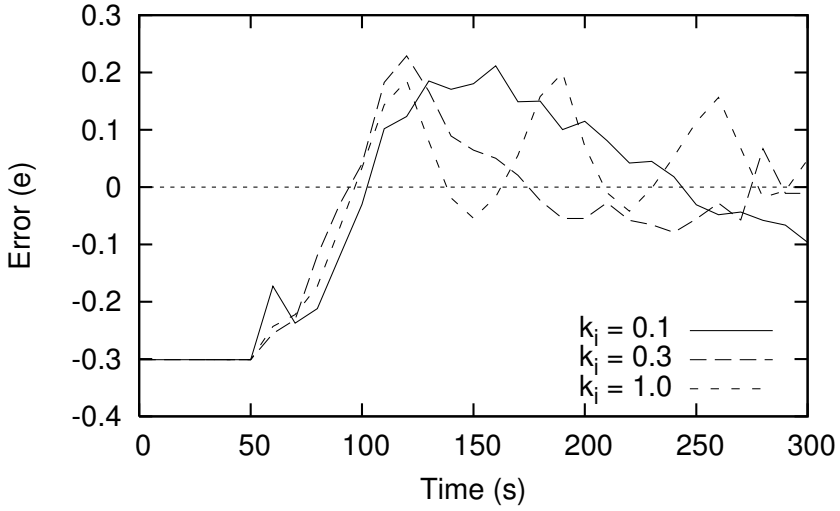

Figure 4.7: Experimentation with parameter k_i

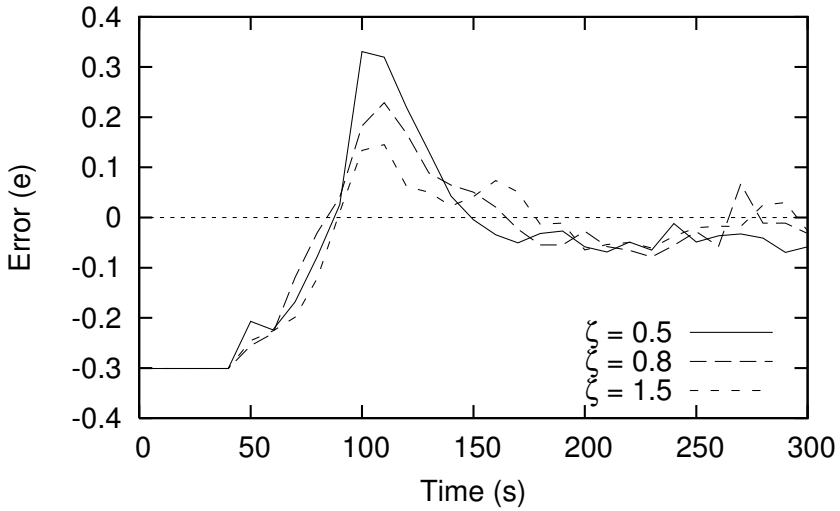

Figure 4.8: Experimentation with parameter ζ

web interaction given by the TPC-W standard, the MIMO model is not convenient. The reason is that the TPC-W standard defines a web interaction as a sequence of several HTTP requests, and the real-time requirements in this standard determine that a certain level of QoS must be achieved for the end-to-end service time of each web interaction. Since the metric must account for the whole web interaction, and since each of the HTTP subrequests may be serviced by different L2 server nodes with a certain level of parallelism, it is impossible to obtain the response time at the server nodes. In our implementation, the centralized controller runs in the front-end server, where all requests and responses go through and the end-to-end time is measured.

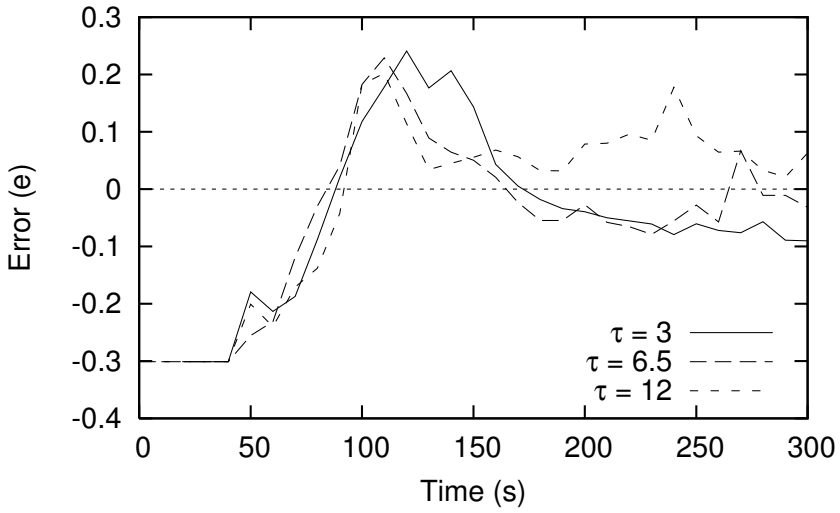

Figure 4.9: Experimentation with parameter τ

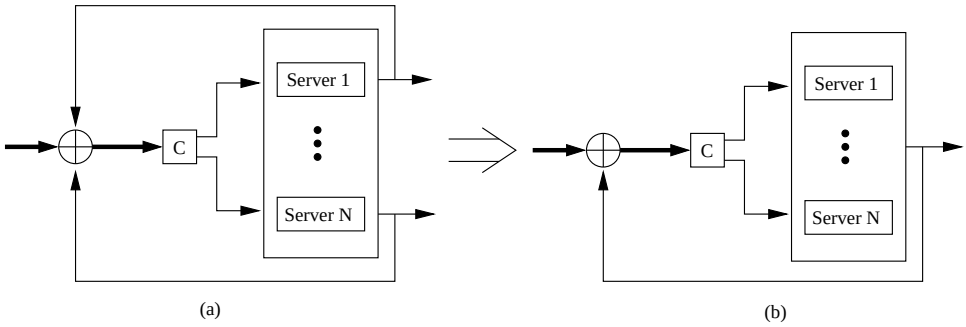

Figure 4.10: Comparison with the classification in [38]. (a) The expected MIMO-C controller for QoS control. (b) The simplified SISO controller implemented

4.6 Conclusion

In this chapter we showed a practical implementation of a feedback control loop in a multi-tier web server system for e-commerce. We used DVS to adjust the system performance to save energy, but with the QoS specification being guaranteed by the control loop. We showed practical issues that arise in the implementation of a controller in a real web cluster application. The experiments showed that the parameterized controller is easy to tune, because tuning has a limited degree of freedom, which helps stability. Our experiments showed an analysis of sensitivity to the controller parameters that can help in achieving the best performance for the controlled system. The fine-grain QoS control showed in this work is useful in achieving extra energy savings for interval based DVS schemes where the goal is to meet all deadlines, avoiding overprovisioning the system according to the real-time specifications.

The next chapter introduces the Generalized Tardiness Quantile Metric (GTQM)

to be used instead of assuming a specific probability distribution. We will show what can be the loss of energy by using the closed formulas derived for the QoS, which are approximations, instead of the GTQM that is distribution independent.

Chapter 5

Generalized Tardiness Quantile Metric

Insanity: doing the same thing over and over
again and expecting different results.
– Albert Einstein

Performing QoS control in large computing systems requires an on line metric that is representative of the real state of the system. The Tardiness Quantile Metric (TQM) presented in the Chaper 3 allows control of QoS by measuring efficiently how close to the specified QoS the system is, assuming specific distributions. In this chapter we generalize this idea and propose the Generalized Tardiness Quantile Metric (GTQM). By using an online convergent sequential process, defined from a Markov chain, we derive quantile estimations that do not depend on the shape of the workload probability distribution. We then use GTQM to keep QoS controlled in a fine grain manner, saving energy in soft real-time web clusters. To evaluate the new metric, we show practical results in a real web cluster running Linux, Apache, and MySQL, with QoS control, and for both a deterministic workload and an e-commerce workload. The results show that the GTQM method has excellent workload probability estimation capabilities, which immediately translates in more accurate QoS control, allowing for slower speeds and larger energy savings than the state-of-the-art in soft real-time web cluster systems. This generalized method will be published as a regular paper in [16].

5.1 Introduction

Large soft real-time computing systems, like a web server cluster, service thousands of web interactions per second and requires that a certain percentage of requests be serviced within their specified deadlines, the workload being built based on several aggregated

random variables. Probabilistic measures of success are the norm in such systems and thus deterministic tools, such as simple real-time scheduling are not appropriate to efficiently deal with such workloads. On the other hand, approximate methods and stochastic modeling techniques such as Markov chains, queueing theory, or stochastic estimation algorithms are powerful tools for system modeling. These tools offer a way to find some regularity, such as convergent estimations and steady state analysis, in the nondeterminism of the random phenomena. Newer versions of these tools, such as real-time queueing theory [60], make strides towards transforming such theories deadline aware.

We show in this chapter an application of a stochastic approximation to solving the classic problem of finding θ in the equation $Pr[X < \theta] = p$. If some performance metric in a computing system is a random variable, this can be used as a means of statistically guarantee the system QoS. The difficulty, however, is that generally one does not know the distribution of the random variable, and it cannot usually fit to any known probability function. One solution is to use generic bounds that depend only on the first k moments of the data, so that the moments can be observed and the limits applied regardless of the probability distribution shape. Finding bounds for this problem was first introduced by Chebyshev in 1874 and proved later by Markov [19].

Principles like Markov's Inequality, Chebyshev's Inequality, and Chernoff's Inequality give upper bounds of the type $Pr[X > \theta] < p$. For example, they can be used to estimate system resources without breaking a certain level of agreed-upon QoS. One example of an application of these statistical inequalities for tail distributions appears in [22], for the performance measure of telephone traffic. In a web server, we are also interested in the tail distribution of a random variable, namely the workload represented by the tardiness. However, bounds are generally not tight enough, usually because web traffic can be difficult to predict [8]. In particular, bounds are not tight when it is necessary to have not only a conservative policy, but a precise measure of the tail probability.

While bounds are not exact, we notice that there are stochastic estimators that can be used to give a value of a specific probability characteristic (such as the quantiles or even the mean). We will use the 1951 Robbins-Monro algorithm for stochastic approximations [82]. We chose this method because it is based on a simple Markov chain and appropriate for online computations. The method allows for transition probabilities that change during the estimation process, while allowing for the p-quantile of an unknown distribution to be estimated with reasonable accuracy. The Robbins-Monro algorithm is a recursive algorithm that can be used to find the root of an unknown function $g(\theta)$, from noisy

observations of θ [59]. The interesting aspect of doing this is that the result is independent of the distribution shape, and hence, the system will reach the specified QoS for any kind of workload.

The difference with the TQM method presented in Chapter 3 is that we apply Robbins-Monro to find a specific quantile of an indirect representation of the workload (tardiness), so that it can be used to control the QoS. This p-quantile estimation will serve as the guide to the frequency settings of a complex multi-tier web server cluster, such that the probability of servicing web requests within their deadlines (i.e., the QoS) is maintained at a specified value. Another important goal is to make the system QoS adaptive under workload variations.

The application we give to the quantile estimation method is to serve as the guide to the performance operating point of a complex multi-tier web server cluster, actuating in the speed of the processors, such that the probability of servicing web requests on time is maintained at a specified value and the QoS of the system is some desired value, and also to make the system QoS adaptive under workload variations. Changing speed is possible when the processors have support to dynamic voltage and frequency scaling. The voltage scaling is tied with the frequency, and the voltage setting is done automatically by the operating system when the user sets a different operating frequency. Our motivation is to allow an energy-efficient architecture to work in the best configuration possible without overprovisioning the system, and therefore spending not more energy than the necessary to attend the QoS level agreed. In systems without a fine grain QoS control, if one contracted a QoS level of 90%, and the system provides 98%, the service provider is giving a product that was not paid for, and thus money is being wasted. This is what is called *quality give away*.

Normally the workload of a web system is assumed to have a specific probability distribution, because it simplifies the modeling. For example, when queueing theory is applied, the simplest queueing models M/M/1 are based on Markovian workloads. The more complex queueing models for G/G/1 queues generally do not have closed formulas, and if G/G/1 model is assumed, bounds based on the tail probabilities are also applied. We use no such assumptions, obtaining a three-fold contribution: 1) we present a method of quantifying the QoS so that this metric is used in QoS control; 2) the method works for any kind of probability distribution presented by the workload, and thus we can expect a good result for the real workload; and, 3) the results are not based on simulations.

5.2 Cluster Model

Our cluster model is shown in Figure 5.1, with a front-end server acting as a reverse proxy. The front-end is capable of SSL encryption/decryption, but we force it to distribute the requests to the web server nodes without encryption between front-end and web servers, in order to decrease the load on all servers. We also use the MySQL cluster architecture combined with Apache reverse proxying to put in the same application layer all the CPU intensive tasks present in a multi-tier architecture. As shown in Figure 5.1, the PHP programs used for building the dynamic pages are executed at the same server that runs the MySQL server, responsible for processing the database queries. Our intention is to do Power Management only at the application level, and this cluster architecture is the best choice because the more CPU intensive are the tasks, the better is the energy saving obtained with DVS.

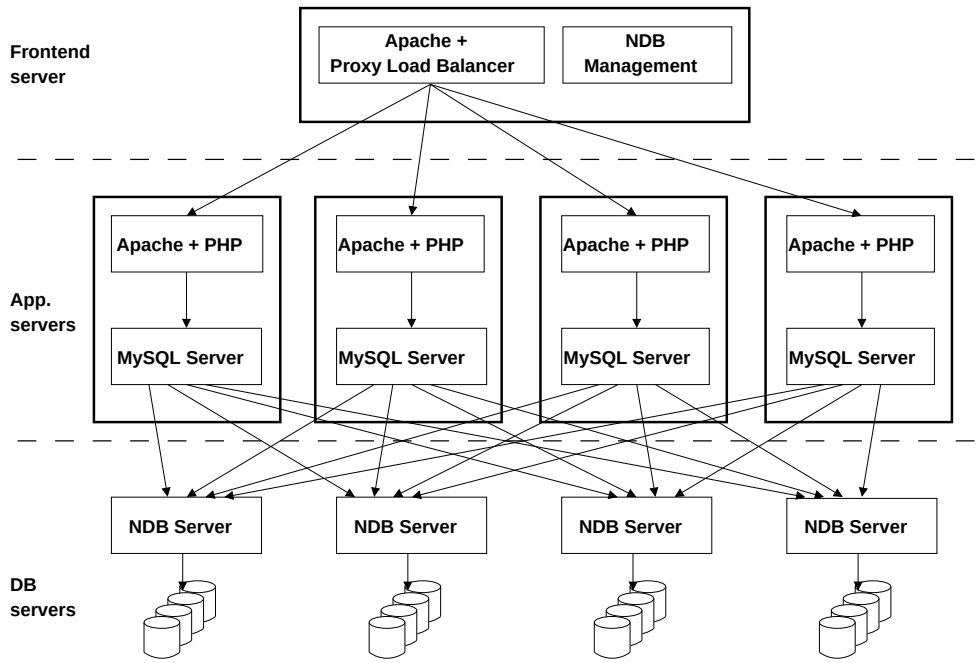

Figure 5.1: Cluster model

Figure 5.1 shows elements of a typical MySQL cluster setup, composed of the mentioned MySQL servers and the Network Database (NDB) nodes, which are the storage engines. When data is stored in the NDB nodes, this data is directly accessible from all other MySQL servers in the cluster. Thus, if one application updates some data, all other MySQL servers that query this data can see this change immediately. There is also a process for the cluster management running in the front-end, the NDB management. The role of this process is to manage the nodes within the MySQL cluster, providing

configuration data, starting and stopping nodes, running backup, etc. Our focus is on controlling the application layer DVS.

5.2.1 Adaptive QoS Control

The cluster front-end has an internal control loop that was built using control theory. It is a PIDF controller, a very common controller found on many industrial plants. The PIDF is a Proportional-Integral-Derivative (PID) controller augmented with a low-pass filter in the derivative part. In industrial plants, this filter is needed to eliminate the noise present in the sensing process. In the e-commerce web cluster, the variable we will sense is the tardiness, a random variable, and because it is stochastic, presents variations that are very similar to noise.

For the QoS control with the assumption of a specific distribution (e.g., TQM), the QoS controller maintains the average tardiness around the values expressed by Equations 3.2 or 3.3. When using our proposed quantile estimator (GTQM), that is independent of the workload distribution, the reference of the controller will be to maintain the p-quantile controlled in 1.0. In other words, the tardiness will have an average of zero, guaranteeing statistically a QoS of p.

The actuator of the control system is based on dynamic voltage scaling (DVS). Changing the voltage and frequency of all servers we can slow down the system, resulting in a greater average tardiness and greater energy savings, or it can speed up the servers, pushing the average tardiness to values closer to zero at the expense of higher energy consumption.

5.3 Tardiness and QoS

Tardiness is defined as the ratio of web interaction end-to-end response time to the specified deadline for that web interaction request. The advantage of using this definition is that tardiness gives more detailed information about the completion of web interactions, rather than simply counting how many interactions finished by the deadline and dividing by the total. In other words, this is the same as answering yes or no whether the interaction finished on time, or answering how close to the deadline the interaction completed.

The tardiness in a web server will depend on several system factors, such as the

workload, the speed of the processors, the time to access resources, etc[1]. The tardiness ratio can, thus, represent the workload of the system and is, therefore, a random variable. The QoS metric can be obtained as a function of statistical characteristics of the random variable tardiness, such as the average, the p-quantile, or expressions relating both.

A p-quantile, or $Q(p)$, is the value θ of a random variable for which $Pr[X \leq \theta] = p$. It may also be defined in terms of the inverse of the cumulative distribution function of the random variable:

$$Q(p) = F^{-1}(p) = \inf\{x : F(x) \geq p\}, \quad 0 < p < 1 \tag{5.1}$$

where $F(x)$ in the cumulative distribution function of the random variable X.

For specific probability distributions, the relation between the tardiness and the QoS can be obtained analytically, such as the expressions for Pareto and Lognormal distributions (Equations 3.2 and 3.3).

5.4 Robbins-Monro Algorithm

The Robbins-Monro stochastic estimation method was originally presented in [82], and more recent works present applications to signal processing, communications, control systems, and analysis of convergence [59]. The simpler application is to find $\bar{\theta}$, which is the root of the equation $m(\theta) = \bar{m}$, where $m(\theta) = \int yG(dy, \theta)$, or the mean of a random variable. For each parameter θ, $G(dy, \theta)$ is the unknown probability distribution function of the random variable. In our instantiation of a real system, the random variable can be the response time of the web requests given an internal parameter θ (in our case, the unknown load arriving at the front-end). The experimenter can get noise-corrupted observations for specific values of θ. The algorithm consists of estimating θ by the recursive formula $\theta_{n+1} = \theta_n + \epsilon_n(\bar{m} - Y_n)$, where Y_n is the observation taken at time n. Because θ is a parameter of the system, as the estimation θ_n changes, the observed value Y_n changes altogether, and θ_n will converge to the desired value, that is, the θ which makes the system to give an average response time $\bar{m}$. The ϵ_n can be set in two ways, as a decreasing value, with some restrictions to guarantee the convergence, or a fixed value ϵ. The original work of Robbins-Monro chooses an appropriate sequence satisfying: $\epsilon_n > 0$, $\epsilon_n \to 0$, $\sum_n \epsilon_n = \infty$. The ϵ_n value is in fact a step value, and the choice of sequence $\{\epsilon_n\}$

[1]The bottleneck is the CPU, as usual when there are many third tier machines, large memories and fast networks.

is central to the effectiveness of the algorithm. A small and constant ϵ_n will be our choice, because it allows tracking of the system if its probabilistic properties change. More on this will be explained in Section 5.5.

The method is straightforward to be applicable to a computing system for which we want to set its DVS clock frequency, so that it yields the desired average response time. To illustrate this, we made the following experiment. We want to sort a fixed size set of random numbers using the quicksort algorithm. As expected, the execution time varies, depending on the input. We used a CPU that can set any frequency between 1000MHz and 2600MHz. Then we wanted to use the recursive stochastic algorithm to find the best frequency (parameter θ) so that the average execution time to sort $10,000$ random 10-digit numbers was 300ms. At the minimum frequency, the measured average execution time was 578.7ms, and at the maximum frequency, the measured average execution time was 221.4ms. The result is shown in Figure 5.2.

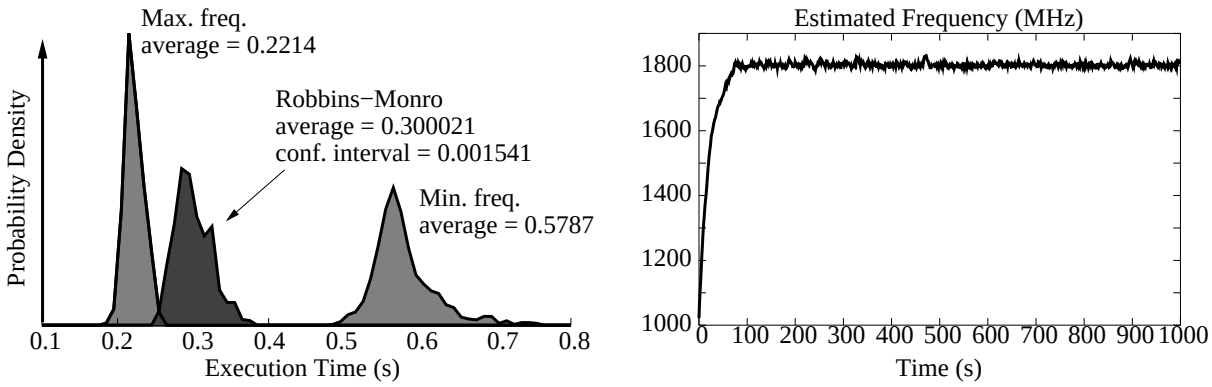

Figure 5.2: Quicksort: Frequency estimation for a given response time

At every 1s the sorting algorithm is executed and the execution time is stored. The top part of Figure 5.2 (Probability Density) shows the computed histogram for the execution times at the maximum frequency, at the minimum frequency, and at the estimated frequency to achieve the desired average. The bottom part of Figure 5.2 (Estimated Frequency) shows the evolution of the estimation. The result was that the frequency achieved a value around 1800MHz, which made the average execution time to be 300ms. The confidence interval with 95% of confidence was 1.5ms. In this experiment, the recursive algorithm was $freq_{n+1} = freq_n - 200 * (0.300 - time_n)$. The value $time_n$ is the measured execution time at time n, and $\epsilon = -200$. The negative value is because we need to increase the frequency, if we want to reduce the execution time, and vice-versa.

5.5 Generalized Tardiness Quantile Method

To quantify the p-quantile and use it in the system to statistically guarantee a QoS with probability p, we propose the use of a quantile estimator based on the Robbins-Monro method, explained in Section 5.4, The estimation method consists of a sequence of real numbers $\{x_n\}$, proved to converge to θ, as the solution to the problem $Pr[X \leq \theta] = \alpha$. The convergent process is obtained from a non-homogeneous (transition probabilities depend on time) Markov chain $\{x_n\}$ defined by:

$$x_{n+1} = x_n + \epsilon_n (\alpha - y_n) \tag{5.2}$$

where $\{\epsilon_n\}$ is a fixed sequence of positive constants, and $\{y_n\}$ are values from a random variable $Y = Y(x)$ that depends on the random variable X. This dependence is expressed as follows:

$$y_n = \begin{cases} 1 & \text{if} z_n \leq x_n \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

where z_n is an outcome of the experiment, or, in the case of the web server system, z_n is one single measure of time delay or tardiness. From Equation 5.3 we can derive the transitions probabilities as $Pr[y_n = 1|x_n] = F(x_n)$, and $Pr[y_n = 0|x_n] = 1 - F(x_n)$ (see Figure 5.3).

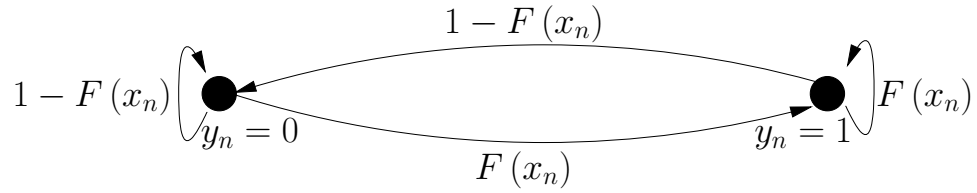

Figure 5.3: Markov transition diagram

For each outcome z_n of the experiment with unknown probability distribution, a new x_n value is determined by Equation 5.2, making the transition probability vary on time, until the convergence to θ occurs. The rationale behind this algorithm is that if x_n is already at the p-quantile of the distribution, the probability that x_n will increase by $\epsilon_n(\alpha - 0)$ is $\alpha/(1-\alpha)$ times the probability that x_n will be decrease by $\epsilon_n(1-\alpha)$. For example, if we are estimating the 0.9-quantile, the probability that $y_n = 0$ is 10%, and 90% for $y_n = 1$. But when $y_n = 1$, the estimator is decreased by a size that is 9 times smaller (e.g., $0.9 - 1$) than when $y_n = 0$ (e.g., $0.9 - 0$). This makes the estimator

asymptotically converge to the p-quantile. Additional analysis of the convergence is given in Section 5.5.1.

The proof described in [82] is made by showing that, independently of the initial value of x, namely x_1, the limit of $E\left[x_n - \theta\right]^2$ will tend to zero as n tend to infinity. This happens for an appropriate sequence of positive constants $\{\epsilon_n\}$, such that $\frac{c'}{n} \leq \epsilon_n \leq \frac{c''}{n}$, for two positive constants c' and c''. The common choice is the sequence $\{\epsilon_n\} = \frac{1}{n}$. However, if this parameter is fixed in a relatively small value, the estimation can be run indefinitely, and we obtain a tracking system in which the probability distributions of the system can change during time, and the estimator will still work. This is obviously important for a web system, because the QoS measure can be done for different load scenarios. In the tracking estimation, the strictly nonzero step size ϵ generates a cyclically updated value with an added noise that does not go to zero as $n \to \infty$, as in the other case, with a decreasing step ϵ_n. This tracking algorithm makes it possible to follow changes in the probabilistic properties of the system, and has been used in other applications for tracking time varying parameters in radar systems and networks, such as wireless communications with varying channels [59].

5.5.1 Test of Convergence

The estimation method just described is very simple to implement and can be run in constant time (that is, with insignificant overhead in practice). Before applying to the real web server system, we wanted to test the independence of the method to the shape of the probability density function of the workload. We tested the estimation capability and convergence response with several known distributions, so that we could validate the method by comparing the results obtained with the theoretical given by the distribution equations.

To evaluate the p-quantile estimator in terms of accuracy and convergence time, we used a random number generator to generate 6 different distributions: Exponential, Pareto, Lognormal, 1-Erlang, 2-Erlang, and 3-Erlang. Figure 5.4 shows the inverse of the cumulative distribution function, both the theoretical (T) and the obtained by the estimator (E). The plots show excellent accuracy for all probability range. For each distribution, we plot 40 points, equally distributed between 0.10 and 0.98. Each point consists of an average of 20 runs, with a confidence interval of 95% (given by $\pm 2.086 \frac{\sigma}{\sqrt{N}}$). Although the confidence interval is larger for very high probabilities (i.e., close to the tail), the method is clearly capable of providing an unbiased quantile estimation.

In these tests, we used $\{\epsilon_n\} = \frac{1}{n}$. The inconvenience of choosing the $\frac{1}{n}$ value is that we need to wait until a convergence is achieved. That is, we cannot have an estimation every time, like a moving average, as we can when we use the tracking algorithm. We can only have a single estimation after a defined time window. As mentioned, an alternative way is to fix the step size $\epsilon_n = \epsilon$ with a relative small size so that we can track the estimated value. We show in Figure 5.5 the convergence of the estimator for the 0.90-quantile, using $\epsilon_n = 0.02$ and $\epsilon_n = 0.002$, to give a sense of how this parameter affects the method. In Section 5.6 we will show a better sensitivity analysis for this parameter. In Figure 5.5, before iteration 25,000, the input data has 0.98-quantile = 1.0, and 0.90-quantile = 0.504. After iteration 25,000, the distribution changes to 0.90-quantile = 1.0. We note that there is a tradeoff between time to converge and the error in steady state.

5.6 Experiments

In this section we show experiments performed on a real web cluster implemented as described in Section 5.2, where we measured QoS and compared it to the user-specified value. The experiments used both TPC-W and a deterministic workload, as described next.

Table 5.1 shows the frequencies available for DVS for the processors in the cluster, and the respective power consumptions for the four machines in the application layer (see model in Section 5.2), where we apply DVS: coulomb, hertz, ohm, and joule.

Table 5.1: Frequencies, power busy and idle, and performance for the application servers

Node	**Frequencies**	**frequency (MHz)**, idle power (W), busy power (W), performance (req/s)
coulomb	5	**1000** 67.40 75.20 53.80; **1800** 70.90 89.00 95.40; **2000** 72.40 94.50 104.80; **2200** 73.80 100.90 113.60; **2400** 75.20 107.70 122.30
hertz	5	**1000** 63.90 71.60 53.60; **1800** 67.20 85.50 92.90; **2000** 68.70 90.70 103.40; **2200** 69.90 96.50 112.40; **2400** 71.60 103.20 122.80
ohm	6	**1000** 65.80 82.50 99.40; **1800** 68.50 99.20 177.40; **2000** 70.60 107.30 197.20; **2200** 72.30 116.60 218.00; **2400** 74.30 127.20 234.60; **2600** 76.90 140.10 255.20
joule	4	**1000** 66.60 74.70 51.20; **1800** 73.80 95.70 91.20; **2000** 76.90 103.10 101.40; **2200** 80.00 110.60 111.40

5.6.1 Deterministic Workload

The deterministic workload is a constant stream of web requests to a web server cluster, for which the average execution time is 20ms, and with a fixed deadline of 200ms. This

type of workload is characteristic of servers that service mostly static webpages, or servers with large internal memory that can cache (in memory) most of their information. The intent is to show that the correlation of the expected QoS with the observed QoS does not depend on the workload. The observed QoS was measured counting deadline misses in 40-second intervals. We changed the QoS setpoint every 10 minutes to ensure our GTQM would adapt to new values.

5.6.2 Results for the Deterministic Workload

The experiment shown in Figure 5.6 compares the GTQM against the TQM with Pareto distribution. We can clearly see the effect of approximating the real tardiness distribution by a known distribution. The Pareto-TQM line is the $p-$quantile equation derived in Chapter 3 (Equation 3.2). For example, say that we need a QoS of 0.95. Then, using Pareto-TQM, the system will control the tardiness around 0.3. Following the horizontal line of 0.3, in the GTQM line, this corresponds to a QoS value between 0.97 and 0.98. This means that the Pareto-TQM is more conservative and that the controlled web server will thus consume more energy. For the QoS of 0.95, the tardiness could be controlled close to 0.45 with GTQM, making the system slower while still observing the contracted QoS, but saving more energy.

The plot in Figure 5.7 shows the obtained QoS for each specified QoS value, using the deterministic workload. The correspondence is an almost perfect match even for high QoS values (e.g., 0.99). As shown in Chapter 3, although QoS control based on TQM method (assuming Pareto distributions) is the best among the tested methods, it does not perform so well for QoS levels close to 1.0. We see from Figure 5.7 that with GTQM this problem is practically eliminated.

Figure 5.8 shows a more detailed experiment, again for the deterministic workload. As the overhead to obtain a $p-$quantile is minimal, we estimated a range of $p-$quantile values, with a 0.1 step, that allowed us to see online the shape of the execution time distribution (cdf). We plotted the random variable Execution Time for every $p-$quantile, and for all desired QoS values from 0.90 to 0.99, with a 0.01 step. The point between 0.9 and 1.0, for $Pr[X < x]$, is the set QoS value for each curve. Note that all are aligned at an execution time of 200ms, which is the deadline for the requests. The alignment of the last point with the 200ms value shows the precision of the estimator for any QoS value.

We carried out sensitivity analysis on the a_n factor in the Robbins-Monro equation and on the parameter k_i of the controller. We experimented with values 0.01, 0.05, 0.1,

0.3, and 0.5 for a_n, and values 0.01, 0.05, 0.1, and 0.3 for k_i. As shown in Figure 5.9, the small values of a_n showed better performance, attaining an excellent match between the 0.95 QoS setpoint and the observed value. A small value of a_n makes the system slower, because the estimation needs more time to reach the steady state value. As for the value of k_i, greater values of this parameter provided better approximations, specially when a_n is small. However, the sensitivity to the k_i parameter is smaller.

The last experiment using the deterministic workload is a comparison of the power and QoS of GTQM with the method presented in [85]. In that work, the QoS control is not done in a fine-grain manner, keeping the QoS close to 1.0. Figure 5.10 shows that GTQM maintains the system precisely at the specified QoS level. In this experiment, the specified QoS was 0.95, and only one machine was turned on.

Figure 5.11 shows the power consumed by the system during a period of 1.8h. The GTQM method reduced the average power as shown in the plot, and reduced the energy consumed from 173Wh to 164Wh, an energy saving of 5.2%. Considering that this savings is relative to an already power managed system, this amounts to significant savings. We note that such saving is bigger than that achieved by the daylight *savings* time in Brazil, reportedly around 4%.

5.6.3 Results for TPC-W

In Figure 5.12 we again show the relation between measured QoS and the setpoint, but now for the TPC-W workload. In this case the obtained QoS is slightly higher than the expected for the lower range. This happens because although the system is set as slow as possible, these minimum resources in our test cluster are still too high, and, thus, it is not possible to further reduce the QoS. The results prove again the ability of GTQM to estimate the system state and to adjust the controller to the exact expected QoS.

To be able to examine the long term behavior of GTQM, we carried out a continuous time experiment, assuming the TPC-W workload, where we leave the system running for over 1600 seconds (see Figure 5.13). We measured QoS every 40s, for three setpoints: QoS = 0.91, QoS = 0.95, and QoS = 0.99. As can be seen, the variability is smaller for QoS values close to 1.0. This happens because the confidence interval for a proportion is given by $\pm 1.96\sqrt{\frac{p(1-p)}{N}}$ [54], and if the $deadlines_{met}/num_{reqs}$ proportion is 1.0 the confidence interval is zero. The observed variability of the QoS value was within 1% of the requested QoS, which is a very good result for the random process we want to control.

Finally, in Figure 5.14 we show the trade-off of power and QoS, assuming the TPC-W generated workload. As the need for a better QoS increases, the power needed increases in a exponential way. This clearly shows the benefit of our mechanism to keep the system with the QoS specified in the SLA, avoiding overprovisioning the system (due to the achievable fine-grain control) and the consequent energy/power waste.

5.7 Conclusion

In this chapter we presented GTQM, a generalized method for the Tardiness Quality Metric presented in Chapter 3. By using the online convergent sequential process proposed in [82], defined from a Markov chain, we derived quantile estimations that do not depend on the shape of the workload probability distribution, so that the metric can be used in any workload. To evaluate this new approach, we showed practical results in a real web cluster with QoS control in an e-commerce environment. We used the tardiness to control the speed of the servers, and showed that GTQM performs better than TQM: it allows for finer-grain control of the requests, making it possible to further reduce the speeds when comparing to an already optimized technique.

In the next chapter, we show how operations research is introduced in the system to achieve optimal dynamic configuration of the web cluster, that is, which nodes need to be on and off, and at which frequency. We model the problem of assigning speeds to servers combining linear programming and integer programming, and solve it using traditional linear programming techniques. This will complete the goal of the thesis, addressing the cluster-wide energy management technique for heterogeneous systems based on cluster reconfiguration, again based on a real implementation.

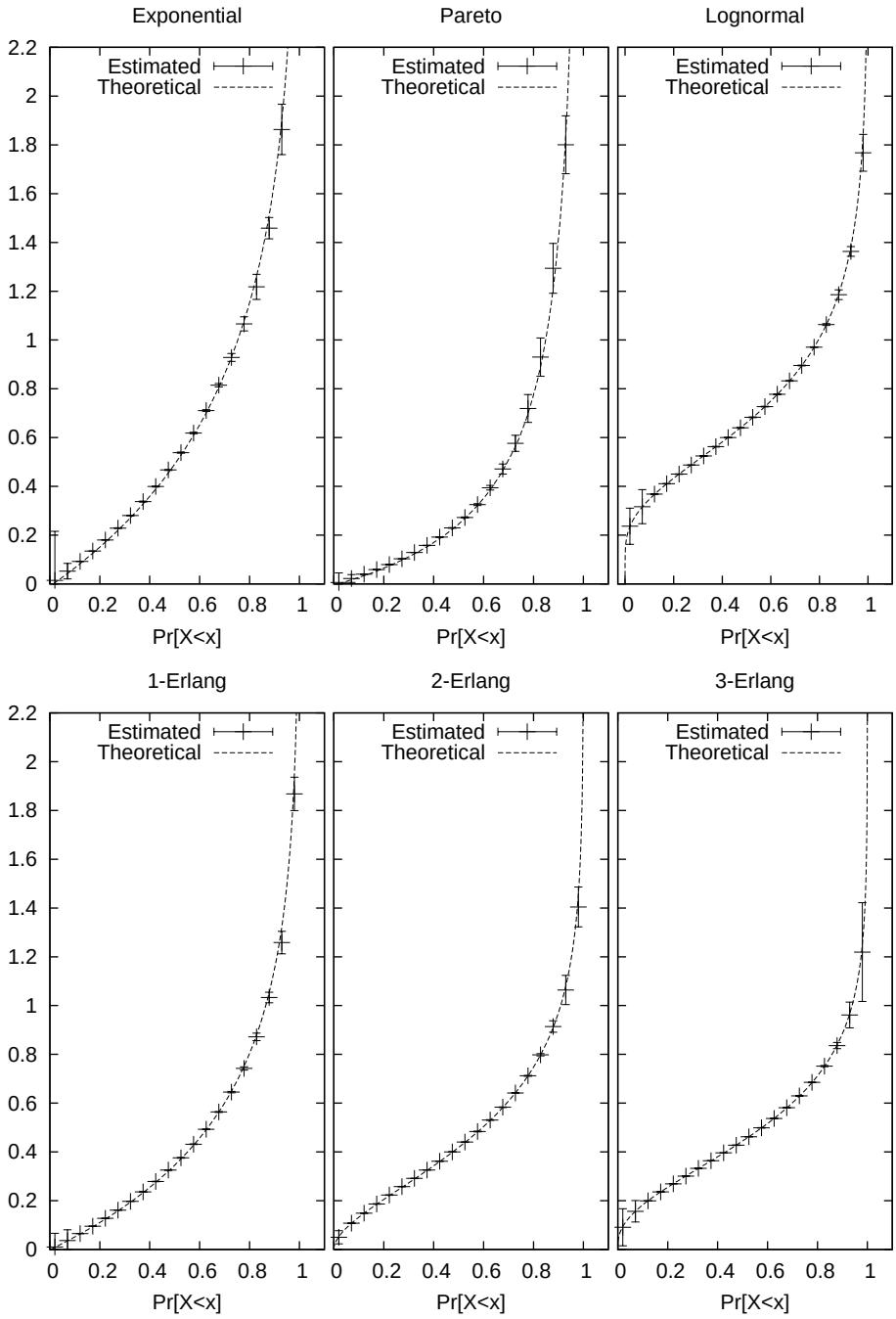

Figure 5.4: Tests with 6 different distributions: Exponential, Pareto, Lognormal, 1-Erlang, 2-Erlang, and 3-Erlang

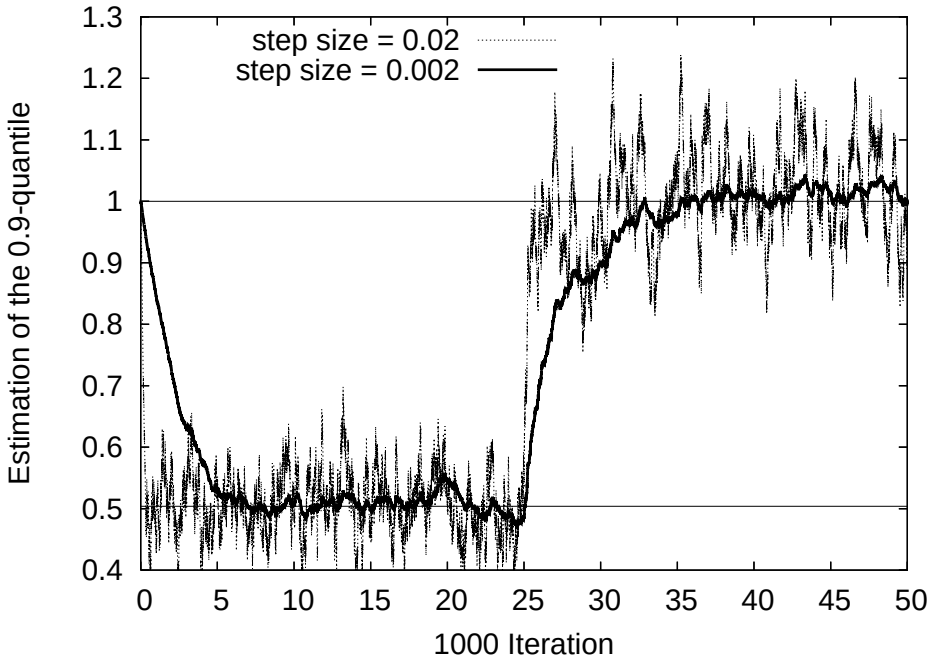

Figure 5.5: Experimentation with the tracking step size in the recursive algorithm

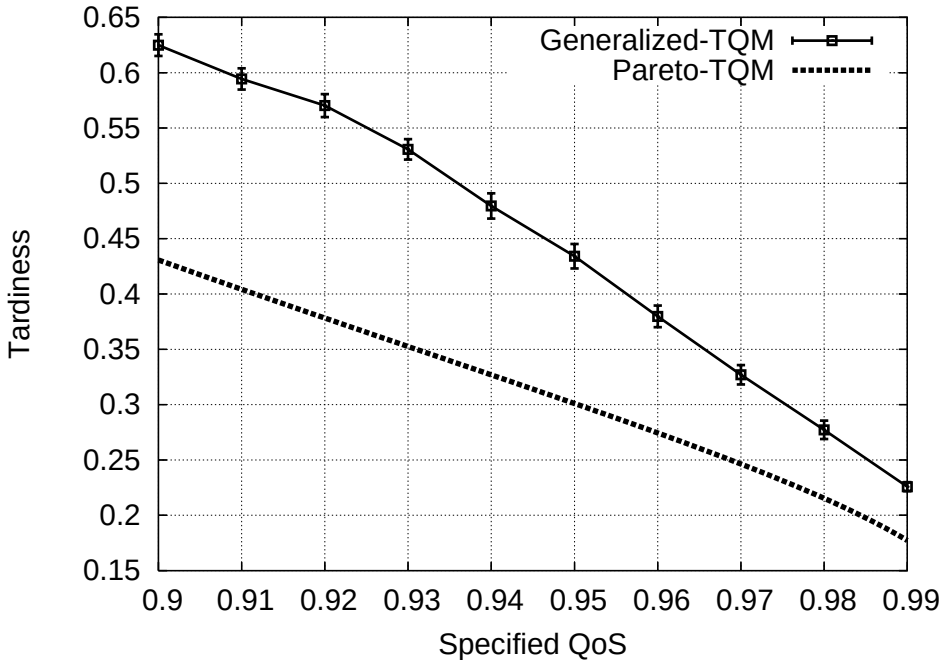

Figure 5.6: Comparing GTQM against TQM with Pareto distribution

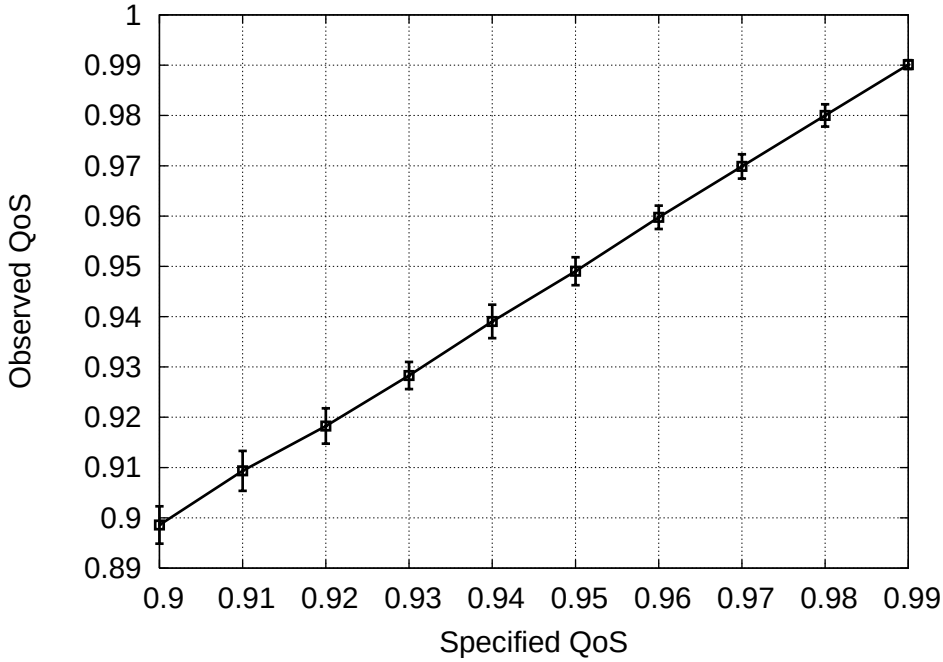

Figure 5.7: Observed QoS and setpoint for the deterministic workload

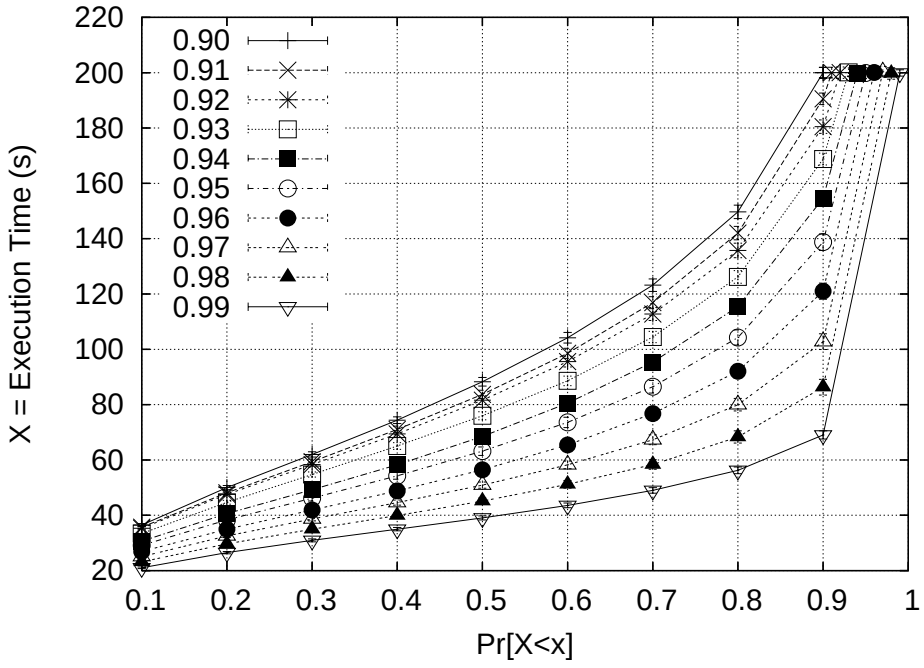

Figure 5.8: Execution time distribution for QoS values between 0.90 and 0.99

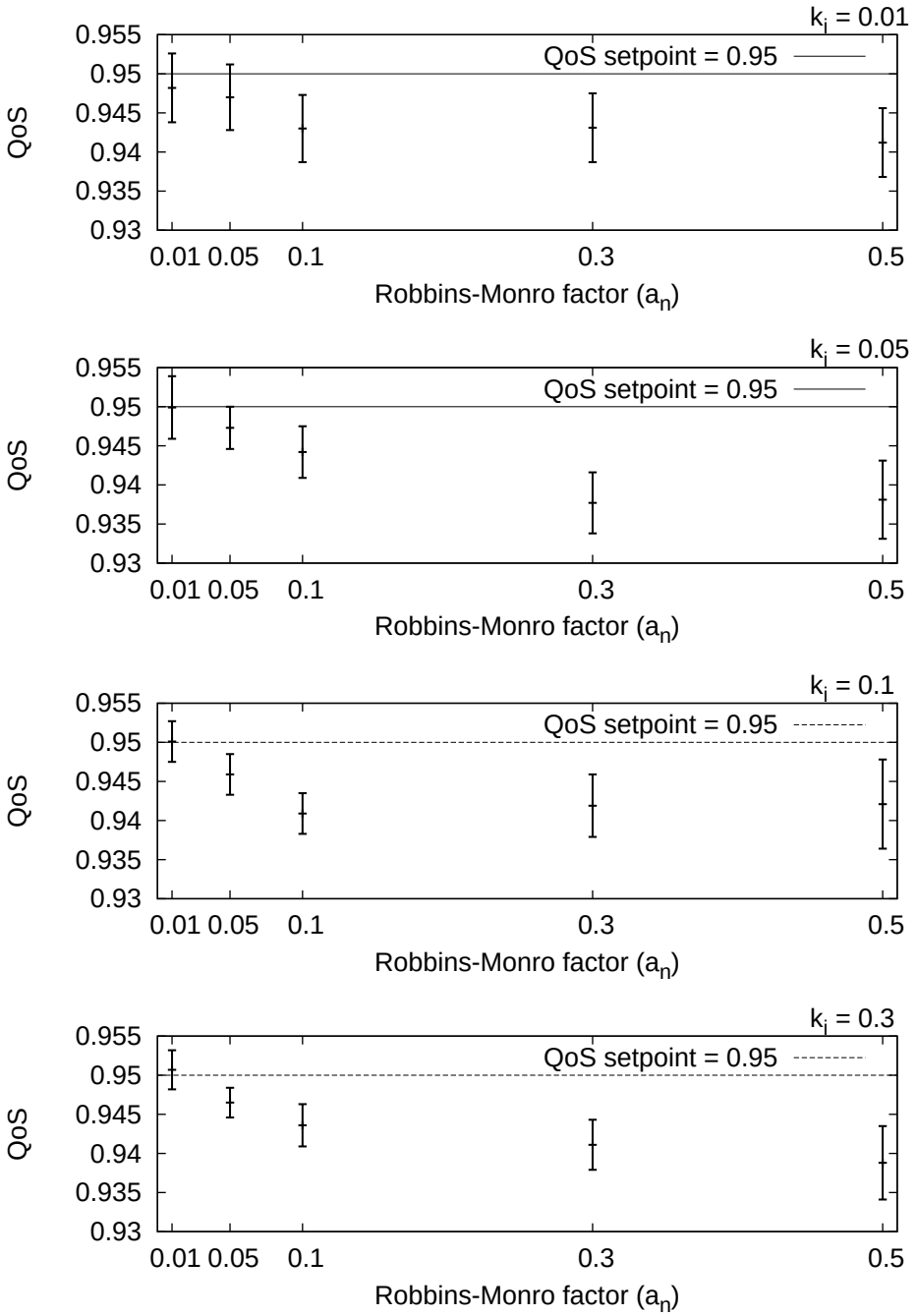

Figure 5.9: Sensitivity analysis for a_n and k_i

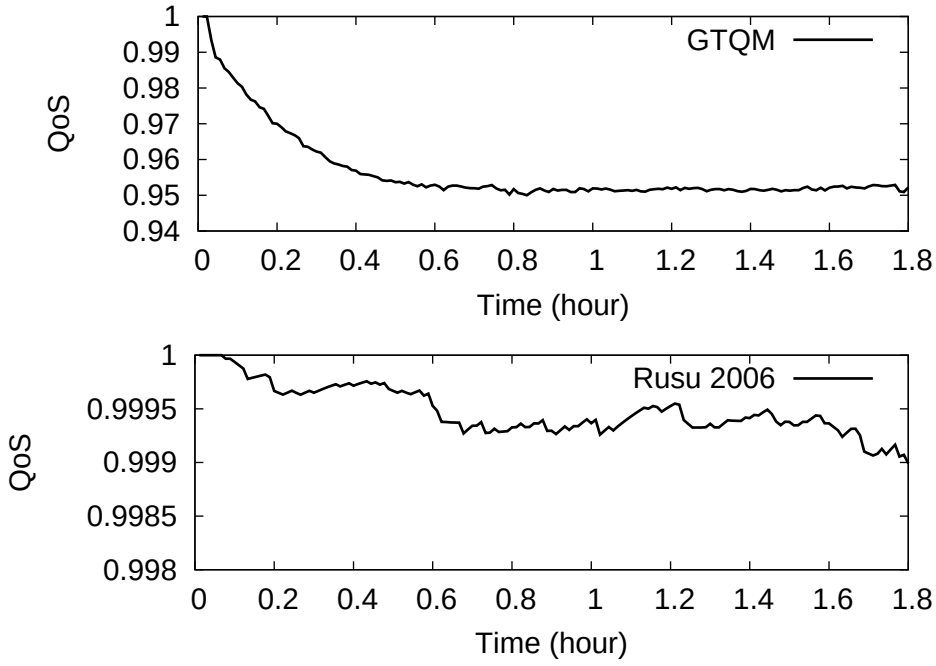

Figure 5.10: QoS comparison between GTQM and Rusu 2006

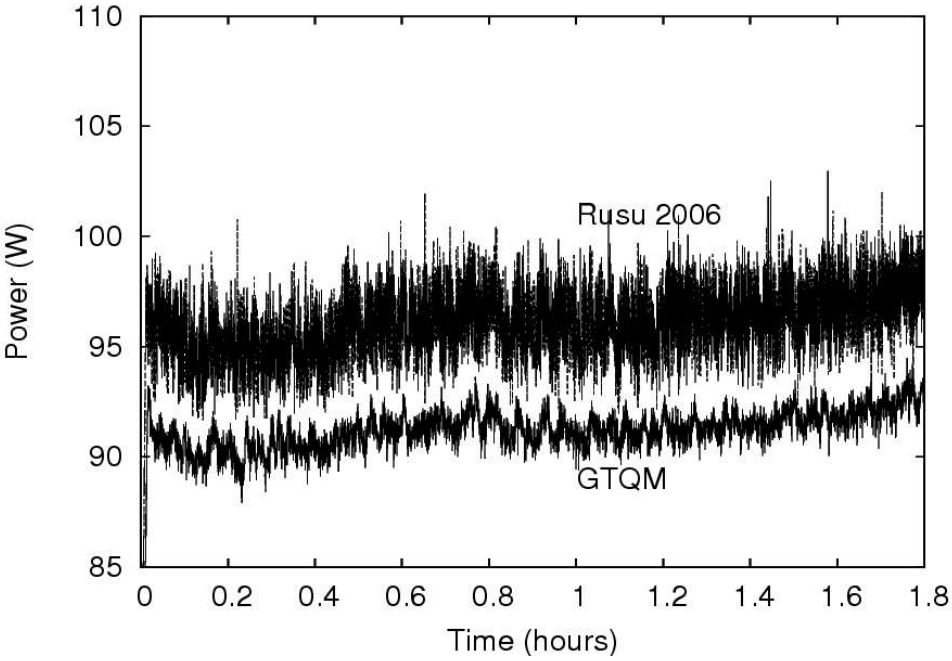

Figure 5.11: Power comparison between GTQM and Rusu 2006

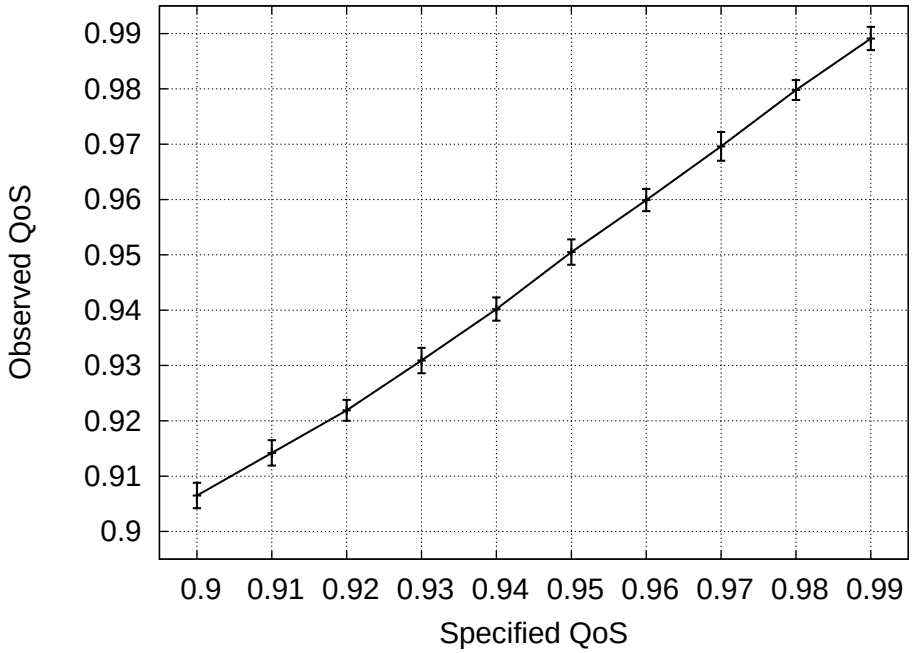

Figure 5.12: Correspondence between observed QoS and setpoint for the TPC-W workload

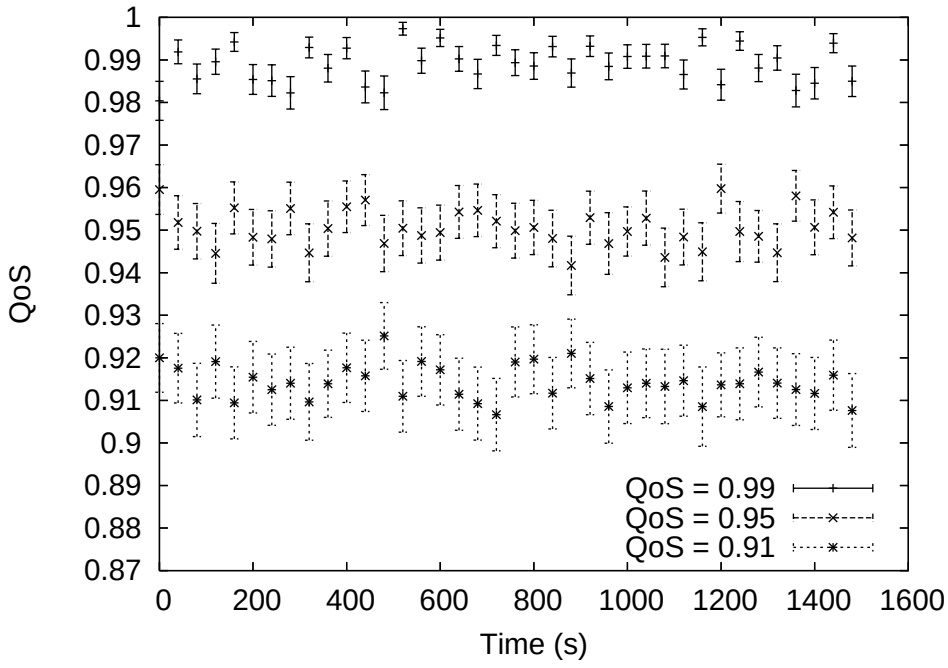

Figure 5.13: Evolution of the QoS in time for three setpoints when running TPC-W

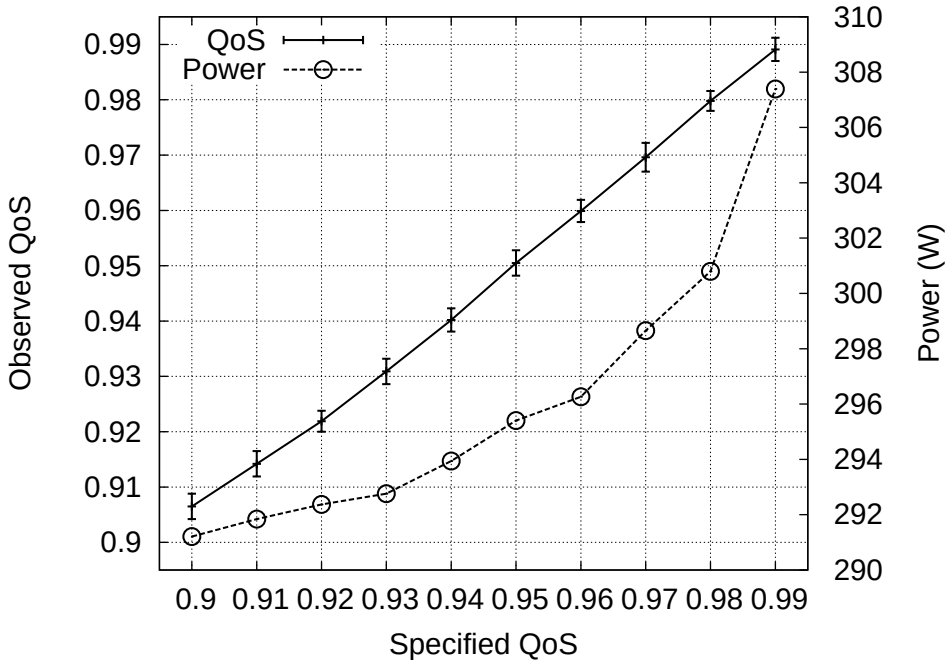

Figure 5.14: QoS vs. power trade-off for GTQM when running TPC-W

Chapter 6

Dynamic Configuration

There can be no economy where there is no efficiency.
– Benjamin Disraeli

To reduce the environmental impact, it is essential to make data centers green, by turning off servers and tuning their speeds for the instantaneous load offered, that is, determining the *dynamic configuration* in web server clusters. We model the problem of selecting the servers that will be on and finding their speeds through mixed integer programming; we also show how to combine such solutions with control theory. For proof of concept, we implemented this dynamic configuration scheme in a web server cluster running Linux, with soft real-time requirements and QoS control, in order to guarantee both energy-efficiency and good user experience. In this chapter, we show the performance of our scheme compared to other schemes, a comparison of a centralized and a distributed approach for QoS control, and a comparison of schemes for choosing speeds of servers. Some partial results of this chapter was published as a WiP paper [14].

6.1 Introduction

Energy consumption is a real concern in these times of global warming and related environmental threats. In many parts of the world, initiatives for deploying green data centers have already appeared. One initiative in development in Germany [32, 33] will save 25% of energy, what will correspond to 16,000MWh per year, putting annually in the atmosphere up to 11,000 tons less carbon dioxide than conventional data centers of the same size. Another example is the 247,000-square-foot Urbana Technology Center that was the first data center in the USA certified by standards for environmentally sustainable construction [97]. In that project, as an example on how every saving opportunity counts,

even the lighting is reduced by 50%. In light of this, why should a web server node be turned on when it is not needed?

The energy cost of a web server cluster can be reduced by sizing the system appropriately. As an example, in a 5-machine cluster with a very low workload, close to 77% of power reduction is possible if the correct configuration of only one server is used. From [81] we can have an idea on what is the energy expenditure of a real data center: a typical cost of $US\$0.12$ per KWh, according to that paper, during the 10-year life of a data center, translates to approximately $US\$10,000$ per KW. In a data center, half of the energy cost is due to the IT equipment itself, and the other half comes from the Network-Critical Physical Infrastructure (NCPI), which is the underpinning for the IT equipment to function, such as backup power, cooling, physical housing, security, fire protection, etc. This makes a 200KW data center have a 10-year electricity bill of 4 million dollars.

This chapter aims at studying the dynamic resizing of web server clusters as well as setting the speeds of the servers that will be turned on. Servers that compose the cluster may be heterogeneous, that is, they may have different set of frequencies and may perform differently in each frequency. The goal is to enable the server cluster to configure itself according to the load, while providing the same user experience of a high performance oversized server. To achieve this goal we consider the cluster as a soft real-time system. This is a convenient approach, because with the specification of deadlines (i.e., the time limit for executing each web interaction) we impose limits for the minimization of energy consumption, without compromising the expected quality of the user experience. In fact, standards for benchmarking e-commerce web servers such as the TPC-W characterize the application as soft real-time, because in addition to defining the deadlines, they specify also a fraction of the web interactions that statistically have to complete before the deadline. We expect stochastic systems to perform well on average and save energy, rather than model it for the worst case execution, causing overprovisioning and consuming more energy than needed. For such systems, our metric of interest will be Quality of Service (QoS), defined as the percentage of web interactions that can execute before its deadline.

We combine two technologies: QoS control by means of feedback control theory, and operations research. First, the feedback control dynamically adjusts the frequency/voltage of the cluster nodes to control the fraction of deadlines met; frequencies are set proportionally to how late or how early requests finish. To change speeds, we rely on the support of Dynamic Voltage Scaling, present in most modern CPUs (allowing the dynamic setting of the frequency and voltage of the CPU core), which allows for quadratic reduction in

energy, and cubic reduction in the power consumption of the CPU [101].

Second, we show how operations research is introduced in the system to achieve optimal dynamic configuration of the web cluster, that is, which nodes are on and off, and at which frequency. We model the problem of assigning speeds, including zero speed (server off), to servers as a mixed integer programming (MIP) problem, which is a linear programing problem where some variables are integers and some are real variables, and solve it using traditional linear programming techniques.

The contribution is fourfold: (a) a novel way of combining control theory and MIP solutions; (b) modeling the dynamic configuration problem (i.e., on/off of nodes and speeds of active nodes) through mixed integer programming problems; (c) comparing the efficiency of higher-than-needed discrete speeds (interspersed with idle periods) and pseudo-continuous speeds (based on the two-speed scheme of [53]); and (d) comparing a centralized SISO (Single Input Single Output) controller with a distributed SIMO (Single Input Multiple Output) controller.

By modeling the optimization problem allowing heterogeneous machines, with different set of frequencies, we can achieve high power reductions compared to other DVS schemes that consider equal frequencies for all machines, and only a predefined sequence of machines to turn on and off. One such solution is adopted in [30], and our solution reduced the power usage up to 40%.

6.2 Optimization Problems

We present a scheme to find the best configuration for the cluster of N nodes and one front-end. This is a cluster-wide optimization for both the on/off and the DVS, unlike what is done in [85], where the DVS is local to each node, and the on/off is cluster wide. We will compare two possible models for the problem, the *traditional DVS* scheme and the *switched DVS* scheme. The former allows tasks to use only one of the available discrete processor speeds at a time, and thus the solution will choose the discrete frequency immediately higher than the exact theoretical frequency needed for a given workload (if speeds were continuous). In the switched DVS, based on [53], the solution allows the CPU to switch between the two discrete values adjacent to the exact theoretical frequency. The latter is convenient for building a feedback control with the CPU frequencies being the final control element, because it simulates a CPU with continuous frequencies (the output of the controller can be immediately fed to the DVS module). Although both use discrete

frequencies, from now on we will call, interchangeably, the first "discrete" or "traditional" DVS, and the second "continuous" or "switched" DVS.

6.2.1 Switched DVS

We adopted a combination model for piecewise problems similar to what is done in [37]. A server i with S_i frequencies has $S_i - 1$ frequency ranges. For any range s, we have two endpoint frequencies F_i^s and F_i^{s+1}. We need to find the range s that combining linearly the frequencies F_i^s and F_i^{s+1} will result in the optimal frequency f_i for each server i, allowing this combination be zero, that will represent the server turned off. Let us denote the power when busy at frequencies F_i^s and F_i^{s+1} as $P_{busy}^{i,s}$ and $P_{busy}^{i,s+1}$, respectively. Similarly, for the power when idle: $P_{idle}^{i,s}$ and $P_{idle}^{i,s+1}$. In the same way, H_i^s and H_i^{s+1} are the maximum load attained in each server for each frequency endpoint. The amount of requests per second the cluster has to process is represented by H_{base}. The problem is modeled as follows:

$$\textit{Minimize:} \quad \sum_{i=1}^{N} \sum_{s=1}^{S_i-1} \left\{ \alpha_i^s P_{busy}^{i,s} + \beta_i^s P_{busy}^{i,s+1} \right\} \tag{6.1}$$

$$\textit{subject to:} \quad \sum_{i=1}^{N} \sum_{s=1}^{S_i-1} \left\{ \alpha_i^s H_i^s + \beta_i^s H_i^{s+1} \right\} = H_{base} \tag{6.2}$$

$$\alpha_i^s + \beta_i^s - y_i^s = 0, \quad \forall i \in \{1 \dots N\}, \forall s \in \{1 \dots S_i - 1\} \tag{6.3}$$

$$\sum_{s=1}^{S_i-1} y_i^s \leq 1, \quad \forall i \in \{1 \dots N\} \tag{6.4}$$

$$y_i^s \in \{0, 1\}, \quad \forall i \in \{1 \dots N\}, \forall s \in \{1 \dots S_i - 1\} \tag{6.5}$$

This is a piecewise optimization problem because the objective function is a sum of several discontinuous line segments. The main variables, α_i^s and β_i^s, mean how much of the frequency end points F_i^s and F_i^{s+1}, in a given range s at the server i, we will combine to obtain the desired frequency f_i for that server. After solving the problem and obtaining α_i^s and β_i^s, the necessary frequency value f_i for each server i is given by:

$$f_i = \sum_{s=1}^{S_i-1} \alpha_i^s F_i^s + \beta_i^s F_i^{s+1} \tag{6.6}$$

With this convenient modeling, we can solve the problem using traditional linear

programming techniques, and the solution also includes the reconfiguration of servers (on/off). The problem is classified as MIP because y (see Equation 6.5) is an integer variable. This variable allows the identification of the elected frequency range, and allows the solver to search for any combination, including turning a server off, what is represented by $y = 0$. When $y_i^s = 0$, the frequency for that segment on a given server is set to zero. Restriction (6.4) ensures that at most one y is 1 for each server i. If $y_i^s = 0, \forall s$, then server i is turned off. In this solution, the utilization is ideally always 1.0, because we combine two frequencies in such a way that it is just enough to handle the load. An exception is when there is only one server working at its lowest frequency; in this case there is no optimization to do.

We solved this problem using the free software Gnu Linear Programming Kit (GLPK) [46]. For small clusters it can be executed online, with execution times of tens of milliseconds. However, we run it offline building a table that can be looked up online. We will use this formulation also for optimizing the control output, with a simplified version of the problem to allow only a subset of nodes always turned on. We run this modified version online in the controller, because the offline version would need a large number of tables. For up to 30 nodes we measured the execution time, and in the worst case it fell below $100ms$. This change is made on restriction (6.4) and will be described in more details in Section 6.2.4.

6.2.2 Traditional DVS

The problem with discrete frequencies is slightly different. Let us consider s not as a frequency range, but one of the available discrete frequencies between $1, \ldots, S_i$. The power consumption will be the combination of the idle power $P_{idle}^{s,i}$ and the busy power $P_{busy}^{s,i}$, for the selected frequency. Here we also have the main variables α_i^s and β_i^s, and β_i^s, incidentally, is the resulting utilization, because it is the part representing the busy power. Differently from the previous formulation, here the cluster may be configured with a capacity bigger than the load demand H_{base}. The optimal frequency f_i is the frequency s for which the binary variable y_i^s is non zero. Mathematically, $f_i = \sum_{s=1}^{S_i} y_i^s F_i^s$. The problem for discrete frequencies is:

$$Minimize: \quad \sum_{i=1}^{N} \sum_{s=1}^{S_i} \left\{ \alpha_i^s P_{idle}^{i,s} + \beta_i^s P_{busy}^{i,s} \right\} \tag{6.7}$$

$$subject\ to: \quad \sum_{i=1}^{N} \sum_{s=1}^{S_i} \beta_i^s H_i^s \geq H_{base} \tag{6.8}$$

$$\alpha_i^s + \beta_i^s - y_i^s = 0, \quad \forall i \in \{1 \dots N\}, \forall s \in \{1 \dots S_i\} \tag{6.9}$$

$$\sum_{s=1}^{S_i} y_i^s \leq 1, \quad \forall i \in \{1 \dots N\} \tag{6.10}$$

$$\sum_{s=1}^{S_i} \beta_i^s y_i^s = \sum_{s=1}^{S_j} \beta_j^s y_j^s, \quad \forall i \in \{1 \dots N-1\}, j = i+1 \tag{6.11}$$

$$y_i^s \in \{0, 1\}, \quad \forall i \in \{1 \dots N\}, \forall s \in \{1 \dots S_i\} \tag{6.12}$$

The additional restriction (6.11) has to do with load distribution. It is present to enforce equal utilization values β_i^s for all servers i that are turned on[1]. In our implementation, we dynamically assign weights proportional to the measured performance that is reported back to the front-end, by each server, periodically. In this way, for a given load, and at any combination of frequencies for the servers, the utilization of every server tend to have all the same value.

Although restriction 6.11 makes the problem nonlinear, we were still able to devise an elegant solution to solve it as a MIP using GLPK. This solution requires an approximation, and uses GLPK twice, once to determine the set L of active servers, and the second time using only the restriction without the product with y_i. The restriction 6.11 then becomes:

$$\sum_{s=1}^{S_i} \beta_i^s = \sum_{s=1}^{S_j} \beta_j^s, \quad \forall i, j \in L, j = i+1 \tag{6.13}$$

This slightly modified optimization problem produces different results. For example, with 10 servers, say that there is a load that would fill roughly 80% of each server in the load balanced version. In the optimal version without load balancing, the optimizer will fill up the most power efficient servers first and then the last server would be with a lower load. This fact says that an unbalanced load distribution method could be more power efficient than using load balancing. This difference would be more evident if, for

[1] Apache and other web servers implement this type of restriction, by distributing the fraction of the total work proportional to the server performance, measured in terms of their current frequency settings, needed due to cluster heterogeneity. This is guaranteed by our modification in the Apache load balancer module making it dynamic.

example, we did not allow for turning on/off nodes, that is, in the case that all servers must be turned on, and for low loads, because in a low utilization scenario, it results that one server receives 0% utilization and hence it is turned off.

It turned out that in practice there is a very small difference in power for the unbalanced versus the balanced load optimization, even when all machines must be turned on. The plot in Figure 6.1 shows the evaluation of the objective function for the optimization with 10 servers. The result is the expected average power consumption that comes from the combination of average powers for idle and busy at each frequency. As expected, the bigger power difference is for small loads, with a peak of 0.4%, which corresponded to $3W$. This happens because the solution is a combination of frequencies that is minimal to handle each load. A considerable difference would appear if the combination of frequencies were much larger than the necessary for the actual load. This does not happen because the optimizer finds frequencies very close to the total amount of cycles necessary to handle the load.

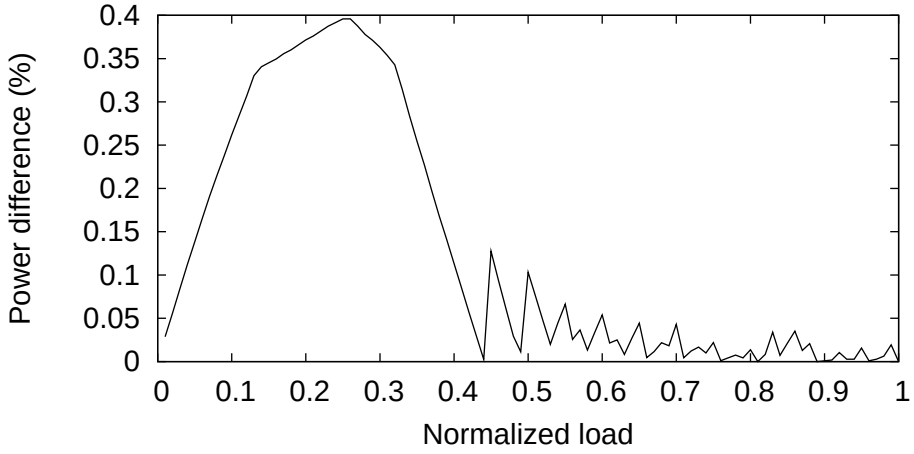

Figure 6.1: Difference in power consumption between load balanced and load unbalanced optimizations; note that load balancing is not significant

6.2.3 Considering Boot Time

We designed an algorithm based on the MIP problem defined in Section 6.2.1 where for each load demand (H_{base}), discretized in a small bin, we know which machines have to be turned on and at which frequency. However, practical issues arise. One is concerning how to handle the boot time of the machines. In [85], in the worst case, if the load is increasing at a defined maximum rate *max_load_increase*, a new machine is turned on earlier, so that when the load reaches the specific point where it is optimal to have one more machine, the new machine is already booted and active.

Because we consider an optimization problem that may turn machines off even if the load increases, the above solution will not work. We improved on this idea based on the following observation: it is possible that the point at which it is optimal to have one more machine does not coincide with the point at which the currently-on machines are at full speed. In the case where the machines that are already on have more room to increase their frequencies, we can make the system more deterministic[2] by turning on the new machine closer to the optimal point, or exactly at the optimal point. In the latter case, the already turned on machines have to have room to increase their frequencies up to the point the load can reach at the worst case rate from the optimal point.

6.2.4 Hysteresis Algorithm

Another practical issue is the way servers oscillate between the on/off states caused by small fluctuations of the workload. We implemented a hysteresis to define the turn-on and turn-off points. Hysteresis consists of building a state machine that defines two different points for a state transition, with a lag between turning on and off. We want to turn a server on at a specific load, say A, and turn off at the lower load $A - h$, where h is the hysteresis lag. In the range between $A-h$ and A, the system may be found with k or $k-1$ active servers. Thus, we need to optimize in two situations, for k and $k-1$ servers. As the optimization is done offline, we need to have two tables that will show the optimum frequencies for each server in the two situations.

To obtain these two tables we need a slightly different MIP problem, which takes as input which machines will be on and which will be off. This is done by changing the restriction (6.4). Letting S_{on} be the set of machines that are to stay on,

$$\sum_{s=1}^{S_i-1} y_i^s = 1, \quad \forall i \in S_{on} \tag{6.14}$$

Letting S_{off} be the set of machines that are to stay off,

$$\sum_{s=1}^{S_i-1} y_i^s = 0, \quad \forall i \in S_{off} \tag{6.15}$$

We use the above modified version of the minimization problem of Equation 6.2 to optimize the output of the controller for the machines that are on. The algorithm will

[2] *More deterministic* because we reduce one random variable, which is whether the actual load rate is at the *max_load_increase* or not.

find the best instant to turn a node on, considering how far the set of turned on machines can sustain the load. Recall that a node may be turned on when the currently-on still can sustain the load in cases that a new combination of servers is more power-efficient. If the currently-on can sustain the load including the *max_load_increase*, a new machine is turned on only at the optimal point given by the solution of the MIP problem defined in Equations (6.1) to (6.5). Otherwise, if the turned on machines cannot sustain the load, the new machine is turned on earlier considering a worst case assumption on the load increase, as in [85]. The state machine that builds the hysteresis and the algorithm are detailed in [15].

6.3 Testbed

Based on open source software and commodity hardware, we implemented a web server cluster with the network topology shown in Figure 6.2.

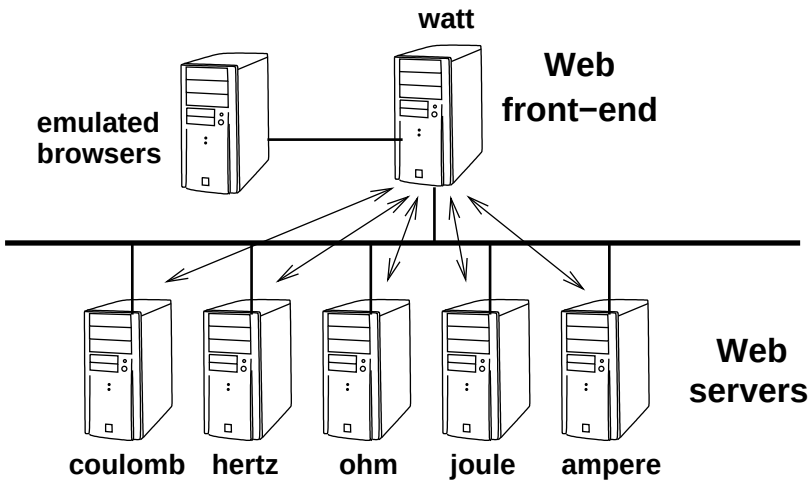

Figure 6.2: Network topology

Table 6.1 shows the hardware used to build the cluster. The machines have the same architecture but are heterogeneous in terms of maximum performance and number of frequencies.

We use machines that can suspend the execution to RAM. The power consumption when suspended to RAM is about 5.5 Watts, only 1 Watt bigger than a machine that is off. Even when turned off there is power consumption because the system maintains the motherboard powered on, with some parts activated (e.g., for the wake on LAN feature). This 1 Watt difference is only to maintain the memory refreshed, and it is worth spending it, because the boot time decreases from 30s (from when the server is off) to 3s (from

Table 6.1: Specification of web cluster nodes used in testbed

Node	**Freq. (GHz)**	**Specifications (AMD Athlon 64)**
watt	2.0	3200+, 2GB RAM
coulomb	1.0, 1.8, 2.0, 2.2, 2.4	3800+, 2GB RAM
hertz	1.0, 1.8, 2.0, 2.2, 2.4	3800+, 2GB RAM
ohm	1.0, 1.8, 2.0, 2.2, 2.4, 2.6	5000+ (dual core), 2GB RAM
joule	1.0, 1.8, 2.0, 2.2	3500+, 2GB RAM
ampere	1.0, 1.8, 2.0	3800+ (dual core), 2GB RAM

suspend to RAM).

6.4 Web Cluster Model

Our web server model is a cluster of N heterogeneous servers capable of Dynamic Voltage Scaling (DVS) and a front-end. The front-end is a server acting as a reverse proxy and serving as a gateway to the actual web servers that process the requests. This is a common architecture used in data centers, although to achieve bigger systems, data centers may use several clusters. A review of web server clusters architectures is very well presented in [27]. As in [85], we consider CPU-bound dynamic requests. The front-end assigns a frequency f_i to each server i, based on performance information that is reported back by the servers. We allow for turning off a server by making f_i to be zero. The power consumption of a server, for each available discrete frequency, is linear in relation to the CPU utilization, and we consider different idle and busy powers for each frequency. The linearity comes from the weighted averaging of idle and busy powers resulted when a whole duty cycle is observed, for a given utilization.

With the web server Apache, requests arrive at the front-end and are redirected in FIFO order to the web servers using a load distribution mechanism, which is similar to weighted round robin. The weights are set dynamically according to the frequency and performance of each web server, and disabled (i.e., weight = 0) if the server is turned off. We modified Apache to include the QoS control module and to make the weights of the load distribution dynamic.

6.4.1 Power Measurement

We have built a data acquisition system for power measurement and data logging. Using a data acquisition board (USB6009 from National Instruments [74]) capable of 48K samples per second, we configured 2 channels to measure voltage and current for the composite cluster, using voltage and current transformers. The power measures are taken from the AC power, and are given by $P = \frac{1}{T}\int_0^T v(t)i(t)dt$, where T, v and i are time of experiment, voltage, and current, respectively. This sampling rate is more than enough to measure power at a one 60Hz cycle granularity.

We used the LabVIEW graphical environment to automate the logging of data and to build online a power versus load plot. We used a TCP connection from the LabVIEW to the front-end server, and the load information is sent to LabVIEW every 500ms. Power is acquired during this period and enqueued. For each load information that arrives, the average power for the last 500ms is obtained. Besides the load data, the front-end also sends other data such as utilization, frequency, QoS, and the fraction of time remaining to the deadline, what we call tardiness. A high level view of this implementation is shown in Figure 6.3.

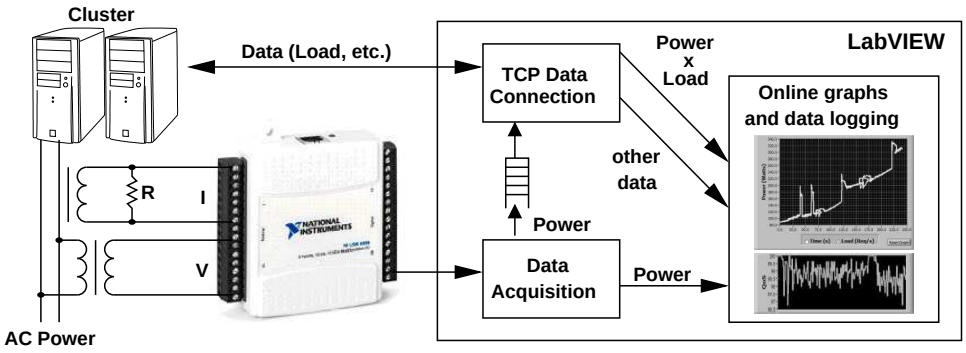

Figure 6.3: Power acquisition system using Labview

6.4.2 Load Balancing Algorithm

The application layer is built based on Apache 2.2 and its proxy load balancer module. It has a request counting algorithm that can distribute evenly the requests based on weights (*lbfactors*). Our early experience in this project showed that the load distribution algorithm that sends requests to the servers with lowest load or servers with lowest utilization does not work well, because it results in bursts that can overload the servers. Instead, a better algorithm is one that can control the flow of requests based on some parameter that can be proportional to the server's performance. It works as if the load balancer had

a "knob" through which it could reduce or increase the flow of requests, rather than only turning the flow on and off. The algorithm present in the Apache 2 distribution does this. It has a *lbfactor* that defines the work quota given to a specific server, and a *lbstatus*, that shows how fast a server has to work to fulfill its quota. Details on how this algorithm works can be found in [3].

Although *lbfactors* are statically defined in Apache 2, in our implementation servers change their frequencies dynamically, and we need a mechanism to alter the *lbfactor* dynamically. We modified the Apache load balance module to implement this idea. Each server periodically sends information to the front-end, including the average frequency that the server ran during the last measurement window. A time window of $4s$ is big enough to make the overhead negligible, and small enough for the granularity needed, since the load in a web server vary slowly. For example, in [63], a feedback controlled web server shows good performance even with a settling time of 270s, and the settling time must be small if the system is to react fast to the load.

Our method is independent of the DVS algorithm being used, because we read the frequency from the statistics file provided by the Linux kernel, describing the amount of time each frequency was used ($time\_in\_state$). The *lbfactor* of a server i is then calculated by:

$$lbfactor_i = 100 \frac{K_i freq_i}{max_j \{K_j freq_j\}}$$

We used the frequency as a measure of performance, multiplied by a factor K_i to consider machines with heterogeneous architectures. K_i can be determined by benchmarking the server. We used $K_i = 2$ for the dual core processors, and $K_i = 1$ for single core CPUs, because our servers have the same performance for a given frequency, although they are heterogeneous with different number of frequencies and maximum frequency. The above normalization limits the maximum *lbfactor* in 100, as required by Apache.

This setting of the *lbfactors*, proportional to the server's performance, showed good results, guaranteeing full utilization of all servers without overload. However, it also showed some problems in practice. When a new machine enters the cluster, and receives a *lbfactor* value, as soon as this server is enabled in the load balancer, a massive load is directed to it at once. Apache creates children processes as the load increases, and the overhead to do this caused what we called a *balloon effect*, showed in the upper plot of Figure 6.4. This plot shows an experiment where traffic is an increasing workload

generated by the *httperf* tool [71]. Load refers to the number of requests actually processed per second. If the server has enough performance, the ramp is determined by *httperf*, but the load can decrease if the server is not fast enough. When a new server is just added to the cluster (see every discontinuity at $Load = 75, 220, 270$, and 315), the figure shows the load decreasing because Apache is creating threads to attend the demand. The new server cannot handle the instantaneous workload that appeared. As soon as new threads are created, they start processing requests and this makes the load and power consumption increase. The balloons happen in the clock-wise direction, because while new threads are being created, the server cannot service requests, and the load goes down. When all new threads are ready to service requests, power increases as long as the load is normalized. As the plot in Figure 6.4 is power $\times$ load, this appears as a circle or balloon. When the balloon is over, the load and power continue to increase normally.

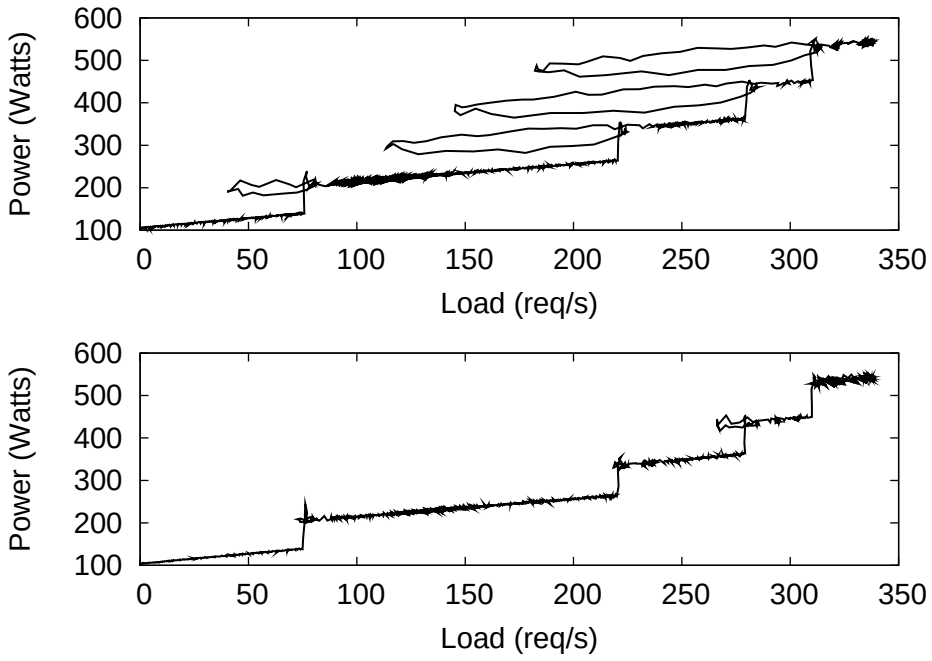

Figure 6.4: Transient effect caused in the addition of a new server. Balloon effect at top and no balloon effect at the bottom, showing the slowly increasing load at new server

We found two solutions for the balloon effect problem. The first is to increase the parameter *StartServers* in the Apache configuration, so that when a server is turned on, before receiving requests, a larger number of threads is already created. The problem with this solution is an increase of the power consumption because the number of context switches increase. This is significant, and we measured a loss of about 0.5 Watt per server. We show an experiment relating power consumption and context switch activities in [15].

The second solution is more power-efficient and consists in enforcing that the load

will be transferred to the new server slowly. When we determine the *lbfactor*, we apply an exponential filter. The result is showed in the bottom plot of Figure 6.4, applying only the exponential filter. As can be seen, a small occurrence of the problem persisted in one case, around $Load = 270$. We solved it completely by combining both solutions.

6.5 Evaluation

In this section we will show some optimization results and compare our cluster implementation with a baseline implementation and show some comparative results of the two approaches for QoS control.

6.5.1 Simulation Results

We first show by simulation an evaluation of the optimization technique itself, without the real cluster implementation, which will be shown in Section 6.5.2. We cited in Section 2.3 the work by Chen et al. [30], which presents a solution to the problem of finding a set of machines to be turned on to run applications in a cluster of servers of a hosting center. For a given set of machines, we will show that our approach can achieve almost 40% of power reduction compared to their work. This happens because our approach has a broader degree of freedom for assigning frequencies to servers. In [30] all servers are considered identical, and what they determine is the number of servers m_i allocated to each application i, and their frequency f_i at any instant. Thus, all m_i servers of application i run at the same frequency f_i. We can do better because we can choose, for any load situation, the best combination of servers that must be turned on, and each server j running at a different frequency f_j, considering the result for one application. Because servers are considered heterogeneous, the possibility to specify which server must be on, and with which frequency, makes a sizeable difference. In the sense of choosing the best combination of frequencies and servers, the method in [30] is unoptimized.

Figures 6.5.a and 6.5.b show the gain we obtain compared to the method in [30]. We used the real data from 10 servers, which are shown in Table 6.2. Figure 6.5.a show the power consumption for each number of servers turned on, and all using the same normalized frequency. That is, $f = 1$ represents the maximum speed for all servers. We compare the two optimizations by finding a different combination of servers and frequency that will achieve the same cluster capacity given by the method in [30]. The dotted lines in Figure 6.5.a represent the power of servers if they were not turned off. As load increases,

Table 6.2: Frequencies, power busy and idle, and performance for 10 servers

Node	**Number of frequencies**	**frequency (MHz)**, idle power (W), busy power (W), performance (req/s)
coulomb	5	**1000** 67.40 75.20 53.80; **1800** 70.90 89.00 95.40; **2000** 72.40 94.50 104.80; **2200** 73.80 100.90 113.60; **2400** 75.20 107.70 122.30
hertz	5	**1000** 63.90 71.60 53.60; **1800** 67.20 85.50 92.90; **2000** 68.70 90.70 103.40; **2200** 69.90 96.50 112.40; **2400** 71.60 103.20 122.80
ohm	6	**1000** 65.80 82.50 99.40; **1800** 68.50 99.20 177.40; **2000** 70.60 107.30 197.20; **2200** 72.30 116.60 218.00; **2400** 74.30 127.20 234.60; **2600** 76.90 140.10 255.20
joule	4	**1000** 66.60 74.70 51.20; **1800** 73.80 95.70 91.20; **2000** 76.90 103.10 101.40; **2200** 80.00 110.60 111.40
ampere	3	**1000** 66.30 81.50 99.80; **1800** 70.50 101.80 179.60; **2000** 72.70 109.80 199.60
pentium-M	7	**600** 42.00 44.00 37.30; **800** 43.00 45.00 50.00; **1000** 43.00 47.00 62.40; **1200** 44.00 49.00 74.40; **1400** 45.00 51.00 88.40; **1600** 47.00 55.00 97.60; **1800** 49.00 60.00 111.40
silver-athlon	5	**1000** 68.00 77.00 55.30; **1800** 70.00 89.00 96.90; **2000** 74.00 100.00 107.00; **2200** 79.00 115.00 115.80; **2400** 85.00 136.00 124.60
black-athlon	3	**1000** 69.00 78.00 56.90; **1800** 73.00 101.00 98.90; **2000** 76.00 112.00 108.90
green-athlon	3	**1000** 65.00 72.00 55.50; **1800** 75.00 105.00 96.30; **2000** 84.00 124.00 108.60
blue-athlon	3	**1000** 64.00 73.00 54.30; **1800** 74.00 108.00 96.40; **2000** 81.00 124.00 107.90

nodes are turned on, and our method is better because the best combination of machines and frequencies are chosen.

6.5.2 Baseline Comparison

Now we show results of our real cluster implementation. From now on, in all experiments, for the sake of comparison, the deadline and average execution time of a request are the same as in [85], 200ms and 24.5ms respectively. The workload is a ramp of dynamic requests, starting from zero load until it reaches the full load of the system. This is necessary because we need to see the power reduction that will be obtained for every load level. In [85] this is done executing different experiments with a constant load at some load points, and using the ramp is equivalent of testing for the whole continuous range. We adopted a deterministic workload, using *httperf*, in order to make controlled comparisons with other dynamic configuration schemes. To generate the ramp, we set *httperf* to access the 24.5ms script and set the session parameter (*wsess=N1,N2,X*), and rate of calls (*rate=r*). These parameters have the following meaning: $N1$ sessions will be created, each consisting of $N2$ calls with intervals of X. The creation of sessions is not done at once, but following the rate r. For our experiments, $X = 0.1$s, and $N1$

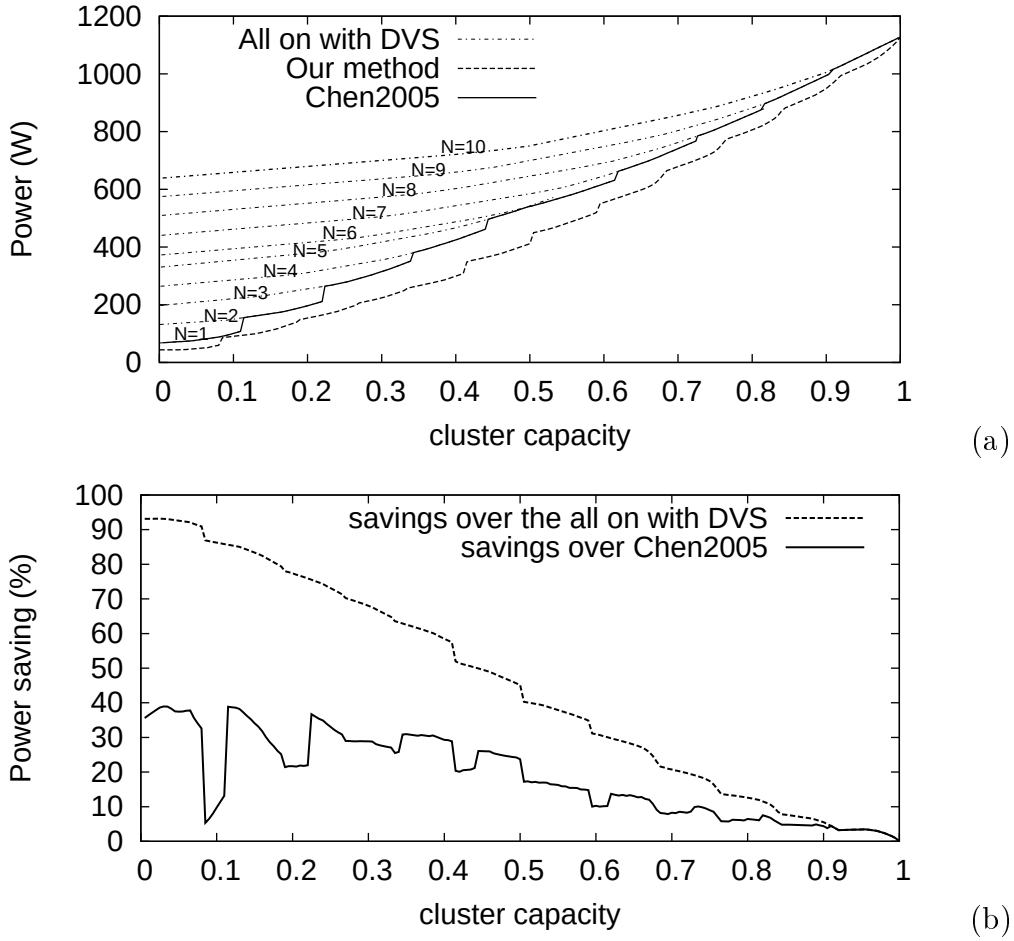

Figure 6.5: Comparison of the switched DVS scheme with an unoptimized method. (a) absolute values, and (b) relative comparison for 10 machines

and $N2$ are large enough, so that we reach the maximum load, and the workload does not finish before the end of the time of the experiment (400s in our experiments). The rate parameter will define the ramp length, and we used a 3s interval. To reproduce the experiments, these parameters do not need to be exactly the same, they must be adjusted to achieve the maximum load on the desired time.

We also compared our DVS policy given by the QoS control with the *ondemand* Linux DVS governor, but in a single server only. This is done to evaluate how our real-time-aware DVS method differs in performance to the Linux built-in DVS governor. Our experiments showed that for some load values, the Linux governor cannot keep the QoS within the specified, and for other load values it overprovisions the server achieving a QoS close to 1.0, but with higher energy consumption. For lack of space, the details of this last experiment are described in a technical report [15].

We compared our on/off MIP optimization with the work presented in [85] which

uses a real-time utilization to determine the DVS policy, and on/off is done based on a predefined sequence of machines. That work compared with, and improved on, the work published in [90]. The work in [85] defined a real-time utilization $U = \sum_i \frac{C_i}{D_i}$ computed based on the deadline D_i, and recent utilization U_{recent}, which is given by the number of requests times the average execution time C_i, divided by a recent time period. Then the frequency used is $\max\left(U, \frac{U_{recent}}{0.8}\right) \times f_{max}$ [85], where f_{max} is the maximum CPU frequency, and the factor 0.8 is a target maximum CPU utilization.

Figures 6.6 and 6.7 show a comparison of *no power management* (i.e., a regular cluster of machines, without DVS and no on/off), *only on/off*, and *QoS control* (that is, our approach that combines on/off and DVS with the QoS control). This experiment, and the next, uses the ramp workload to exercise the whole load range. For 5 machines, and very low load, the power decreases from 390 to 90 Watts, reducing up to 77% the consumption, and up to 15% of energy saving by using DVS with QOS control compared with using just on/off.

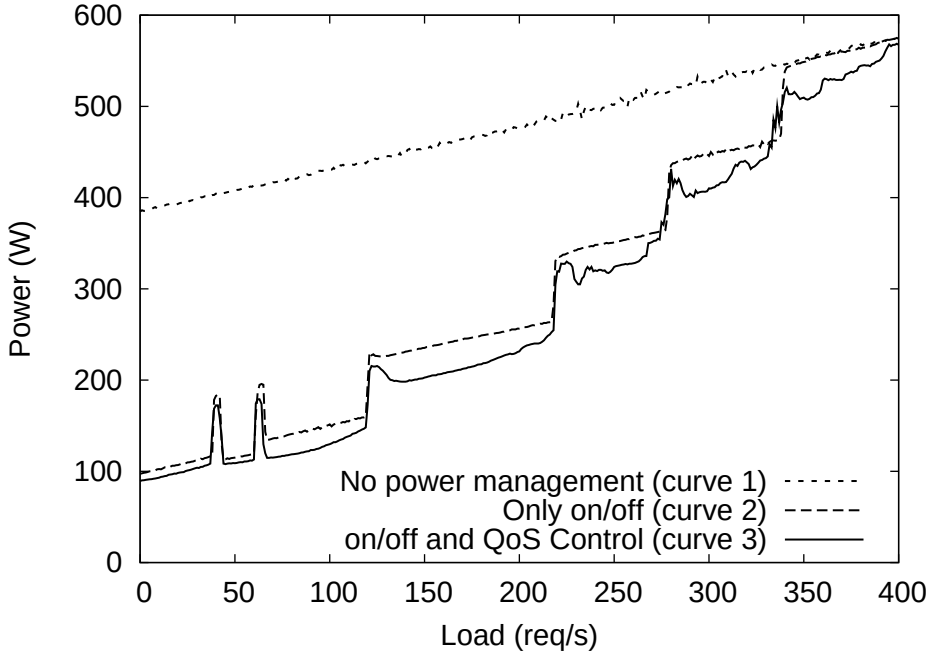

Figure 6.6: Comparing power management with no power management

Figure 6.8 compares our method with the method of on/off and DVS presented in [85], where the machines are turned on in order, from the more power efficient to the less power efficient, and the DVS is done locally at each server, using the real-time utilization method mentioned, and thus without global optimization. Furthermore, this DVS scheme is not able to control the QoS in a fine grain manner, and most of the time results in 100% QoS (see Figure 6.9). The reduction in peak power that can be observed is about

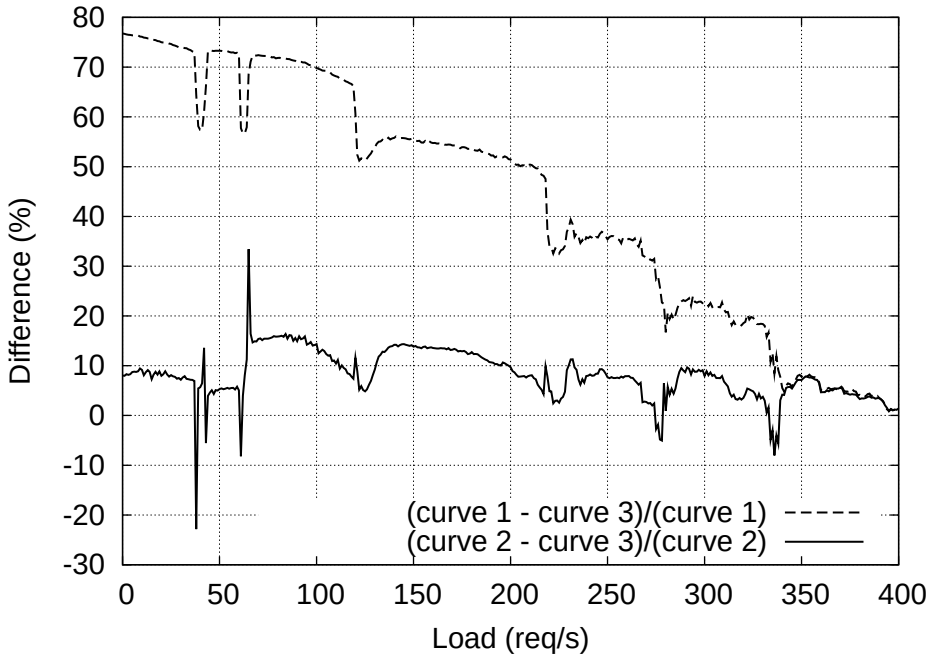

Figure 6.7: Comparing power management with no power management: percentage of power savings

15%. This represents a good improvement, considering that the baseline [85] is already a power managed system.

The two up-and-down steps of power around 50 req/s in Figures 6.6 and 6.8 are due to the change of configuration where one server has to be turned off for another server to be turned on in our scheme, as determined by the solution to the MIP optimization. It is worth doing this switch depending on the trend analysis of the workload.

Figure 6.9 shows the QoS and tardiness for both schemes, but now we use a fixed load workload, not the ramp, because we want to show how the QoS is maintained during a larger period of time. There is also a QoS reference line of 95% plotted that is the target QoS for both cases. It is interesting to note that when a machine is turned on (e.g., at time $t = 1000$ in this figure), the two schemes have opposite QoS behavior. Our QoS controller goes to 100% QoS because the controller output is too high for the new configuration. This is a transient effect that disappears as soon as the controller finds a new control output to satisfy the QoS in the new configuration. In the baseline case the QoS decreases, because the QoS awareness of the method is based on turning a node on just before the old configuration cannot handle a QoS of 95%, accounting for the prediction on the load given by the $max\_load\_increase$ parameter. In this experiment, the scheme Rusu 2006 shows a tardiness curve that stays usually below that in our method because it runs more overprovisioned most of the time.

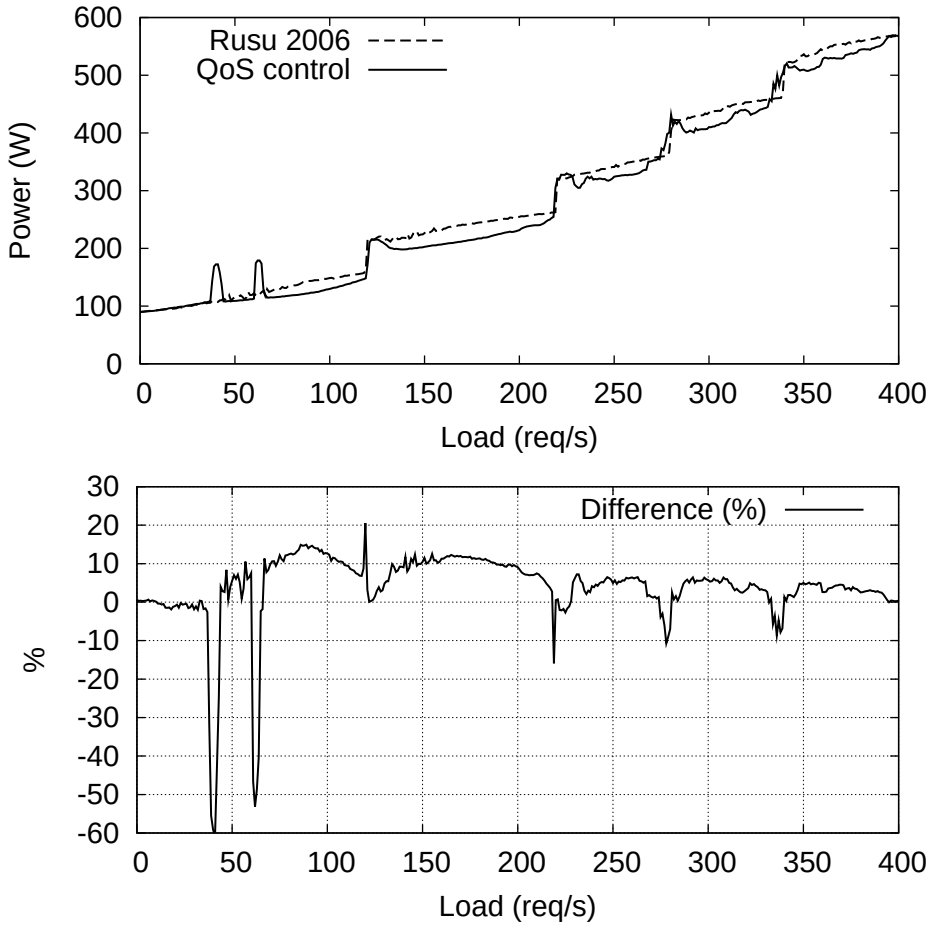

Figure 6.8: Comparing with [85]: power

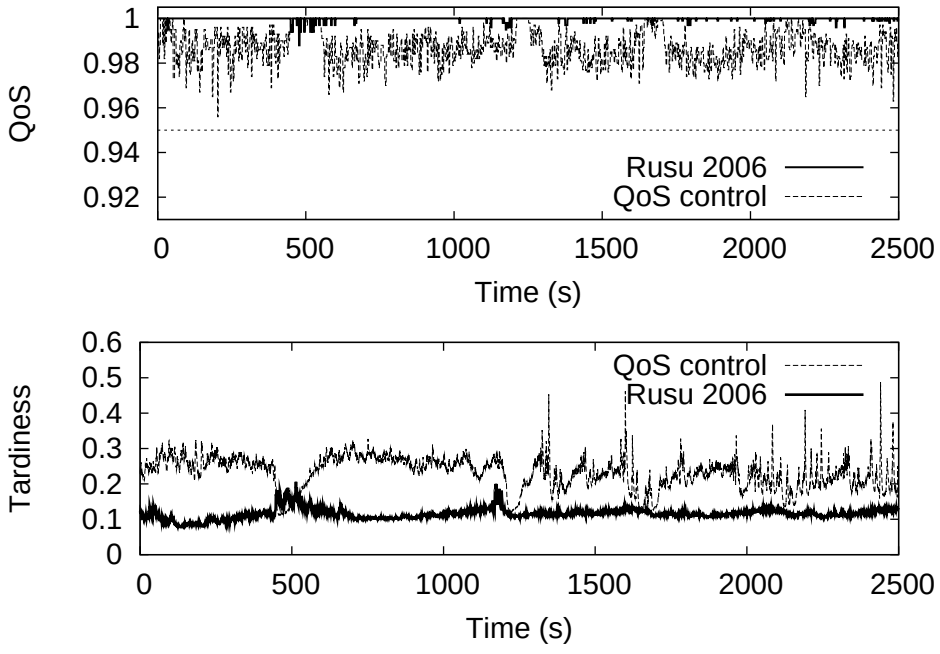

Figure 6.9: Comparing with [85]: QoS and tardiness

Another observation we made from our experiments is that the DVS overhead is not due to changing the frequency, but due to the scheduling of the DVS task. If a period of 10ms is used (like in the original work [85]), it results in 300 context switches per second (cs/s), and 70W power when idle. When the period is increased to 50ms, we get 100cs/s and 68W. Again, for lack of space, we focus only on showing comparison results of our method with [85]; the details of the effect on the period can be seen in [15].

6.5.3 Discrete Versus Continuous Frequency

To see the advantages of the two DVS assignment policies described by the MIP problems in Sections 6.2.1 and 6.2.2, namely the switched and the traditional DVS policies, we generated a ramp workload for both cases. The first advantage of the continuous case over the discrete case is because it is more appropriate for a feedback controller that relies on the continuity of the actuator to compute the output. The experiment shown in Figure 6.10 consists of only doing DVS for the 5 servers (without the on/off capability). The problem of not having a continuous actuator in the discrete case appears as instabilities (i.e., power goes up and down). This happens because as the frequencies are discrete, when the tardiness (see also Figure 6.11) reaches a value that causes the output to increase, the frequency will in some cases increase one step higher than the needed frequency, making the system act too fast. The tardiness value then drops and this will be the beginning of the instability.

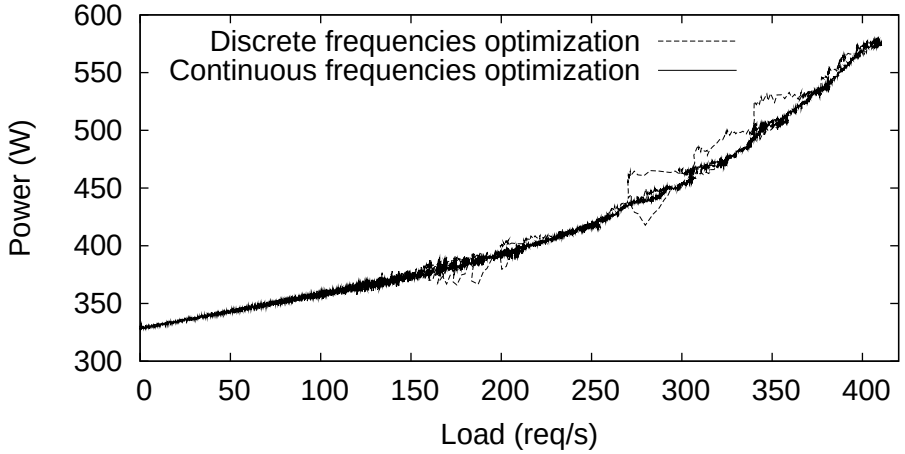

Figure 6.10: Power comparison for the continuous and discrete frequencies

The top part of Figure 6.11 shows also how the aggregate frequency vary in both cases. The frequency indicated is the sum of all frequencies of the servers, showing the capacity of the cluster in cycles per second.

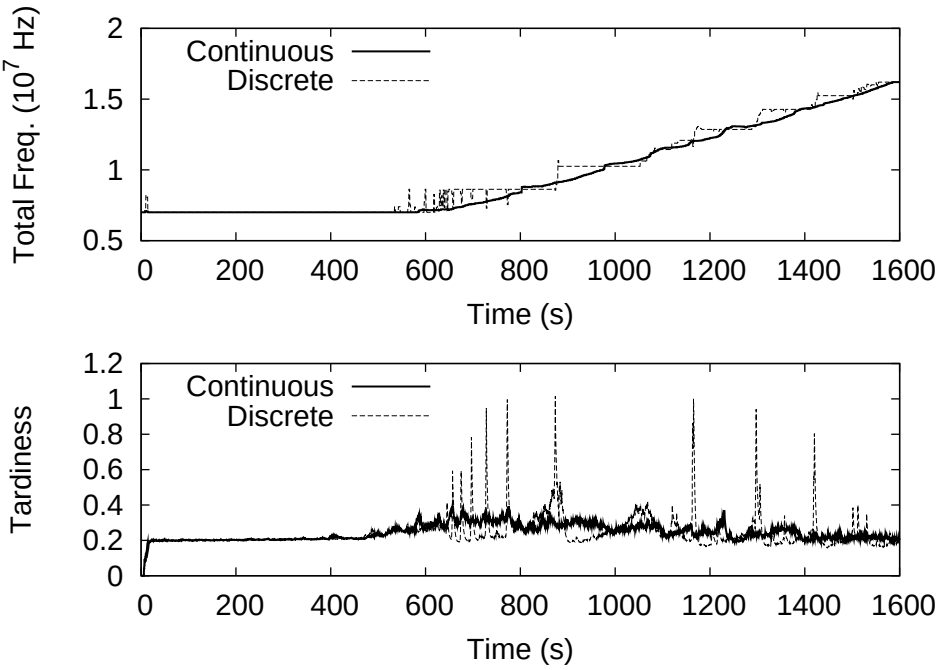

Figure 6.11: Frequency and tardiness comparison for continuous and discrete frequencies

A disadvantage of the discrete frequencies was observed during the on/off optimization. The continuous case will make fewer changes in the set of active servers, minimizing the occurrence of turning on/off servers. This happens because the continuous case offers more flexibility to the MIP solver. In a situation that, say, only one cycle per second is needed in addition to a given configuration, the continuous case will be able to attend using the same configuration, by just increasing this one cycle. On the other hand, the discrete case will need to choose a different frequency, and it is more likely that a configuration with a different set of servers will outperform the original set of servers in terms of power consumption. For this reason, the discrete case will change configuration more often.

We show this difference between the two cases in Figure 6.12. The bars show the frequencies assigned to the servers. It is sufficient to look at the first 3 servers in a 10-node system to see the difference. If a region has no bar, it means that another server (from 4 to 10) is running at a nonzero frequency for that load value. Consequently, there will be more online swapping of servers (i.e., a server turning on and another server turning off at the same load value), resulting in smaller power efficiency. Note that server 3 is always on in the continuous case, while the same server had to be turned off 4 times after it had been turned on for the first time in the discrete frequency case.

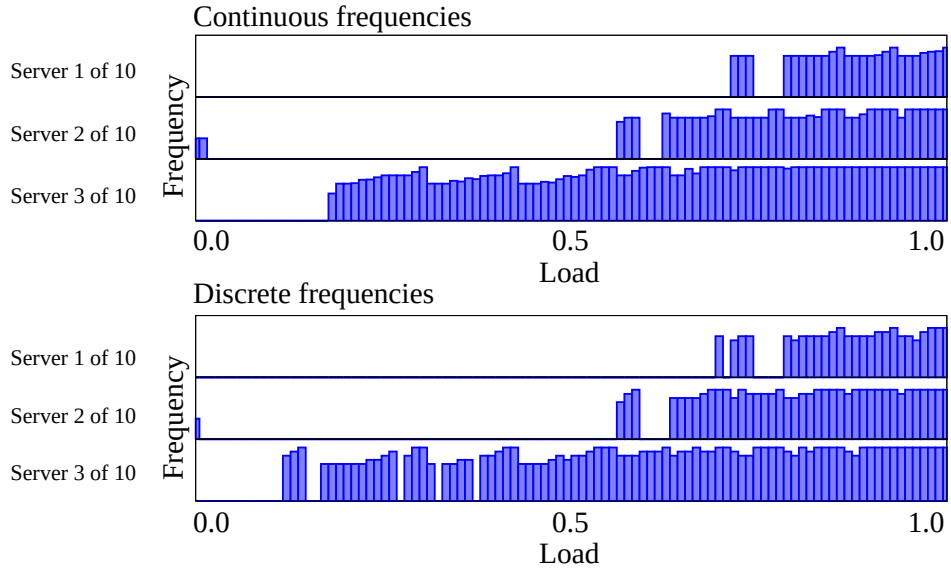

Figure 6.12: Disadvantage of discrete frequencies with discontinuities: processors switch on and off more often

6.6 QoS Control Evaluation

QoS is controlled by a PIDF controller with a parameterization that eases the tuning process. It is very simple and shows a good response for a stochastic system such as a computing system, due to the filter component added in the derivative part of the PID controller. The implementation of it inside the Apache server is done in the same Apache module that controls the DVS. Recursive equations are straightforward to obtain, by using the *Z-transform* to convert from the Laplace domain to the discrete domain (more details in [12]).

The QoS control is done indirectly by controlling the tardiness of completion of web requests, defined as the ratio of response time to the deadline. The rationale of measuring and controlling QoS by measuring tardiness is described in [13]. here it is only important to know that the controlled variable is the ratio of execution time to the deadlines, and this variable is controlled by the manipulated variable frequency of the servers. The idea is, to achieve a QoS of, say, 95%, that the probability distribution of the execution time has its 0.95-quantile exactly at the deadline. In other words this means that the probability of meeting the deadline is 0.95. A statistical inference, based on calculations of the quantile, relates the average tardiness to the desired QoS value, which is defined as a fraction of deadlines met. The controller output will be a normalized performance factor that will be used as an input to the MIP solver, to define the H_{base} load demand.

The use of the QoS controller can be done through different topologies. We will analyze

and compare two here: a SISO topology (Single Input Single Output) where there is only one centralized controller, with a single output, and a SIMO controller (Single Input Multiple Output), where the same input is sent to N independent controllers, therefore generating multiple independent outputs.

6.6.1 Centralized SISO Controller

The SISO controller takes a single measure of tardiness, the ratio execution time by the deadline, at the front-end and computes one single output that is used as an input to the MIP optimization process. Figure 6.13a gives an illustration, where $K(s)$ is the transfer function of the PIDF controller.

6.6.2 Distributed SIMO Controller

One disadvantage of the SISO controller presented in Section 6.6.1 is the added work of precomputing the offline tables, for optimizing the controller output, for a large number of servers (e.g., bigger than 30). Instead of just using the SISO architecture for the controller, we are going to also compare it to a distributed SIMO control architecture that wins in simplicity at a cost of loosing optimality.

Figure 6.13b shows an alternative scheme that runs without optimization, with N independent controllers. This scheme reaches the same QoS and does not need tables for the DVS. It simplifies the implementation because there is no need to run the MIP optimization for the controller, only for defining the points to turn servers on/off. When the system stabilizes, however, it will operate in a suboptimal point (i.e., higher power consumption), as shown in Figure 6.14. Note that this figure is divided in two plots for better clarity of the results.

6.6.3 Distributed Versus Centralized Control

Each method has advantages and disadvantages, but in terms of energy consumption the SISO controller is better. The SIMO controller showed to be more stable (using the same parameters) for increasing number of servers. Because the controllers are independent, each one shows a performance similar to the SISO controller for one server. The gain of the SISO over the SIMO was up to 10%, as can be seen in Figure 6.14. We observed some load fluctuations for the SISO model, but above the QoS setpoint, that is, the system load fluctuates in a permitted zone where the QoS is still satisfied. Thus, both

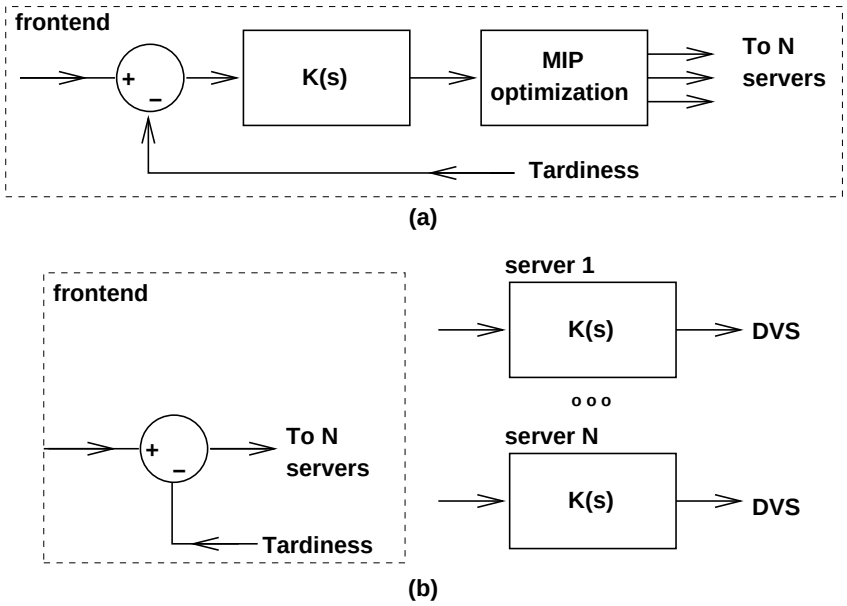

Figure 6.13: (a) one centralized SISO controller. (b) N distributed SISO controllers building one SIMO controller.

configurations achieve a QoS within the specification (in this experiment, 95%). The QoS for this experiment can be seen in the right y-axis of Figure 6.15 and Figure 6.16. Besides being more power efficient, the centralized scheme with optimization achieved a slightly better QoS.

Plots in Figure 6.15 and Figure 6.16 also show the utilization and frequency resultant of the two cases: SISO and SIMO. Note that in both cases there is a tendency of the average utilization to reach 100%. The closer to this maximum, the more harmonious is the synergy between the load balancer and the DVS algorithms. These plots also show an important difference in both implementations. The variations in frequency and utilization such as the one that happens at time $t = 1000$s are smaller in the SIMO than in the SISO scheme. This is due to the initialization of the controller. In the SIMO case, when a new server is turned on, its controller output is initialized to zero, and then increases until control is achieved. In the SISO case, when a new server is turned on, because there is a single output the initial value is the value with which the controller was operating before the new server comes in. It results in an overprovision of the system. The QoS goes to 100% until the system stabilizes again. The SISO case is more conservative, and the SIMO case can result in a underprovisioned system, as happened just after $t = 500s$ in Figure 6.15, where a short QoS drop to 80% can be seen. The more conservative case is preferred, and the SIMO case can be modified to copy this behavior. It is only necessary to define what are the contour conditions of the controller equation. The most

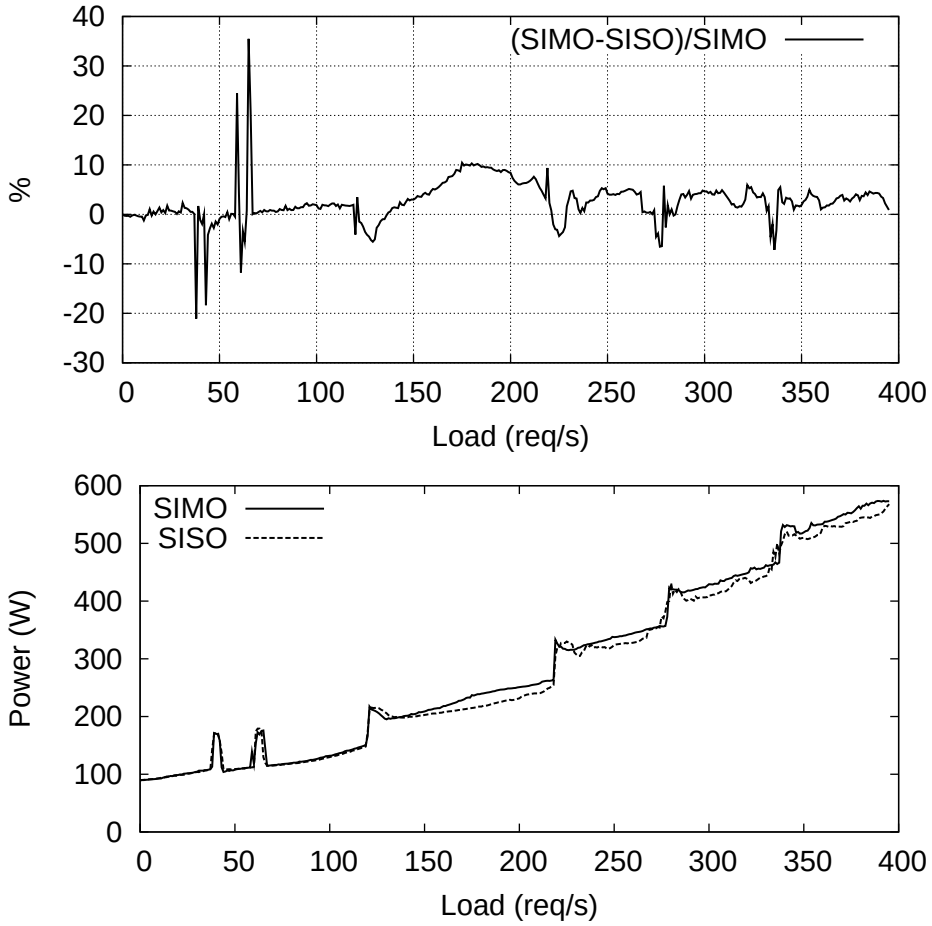

Figure 6.14: Comparison of the power consumption for the SISO and SIMO models

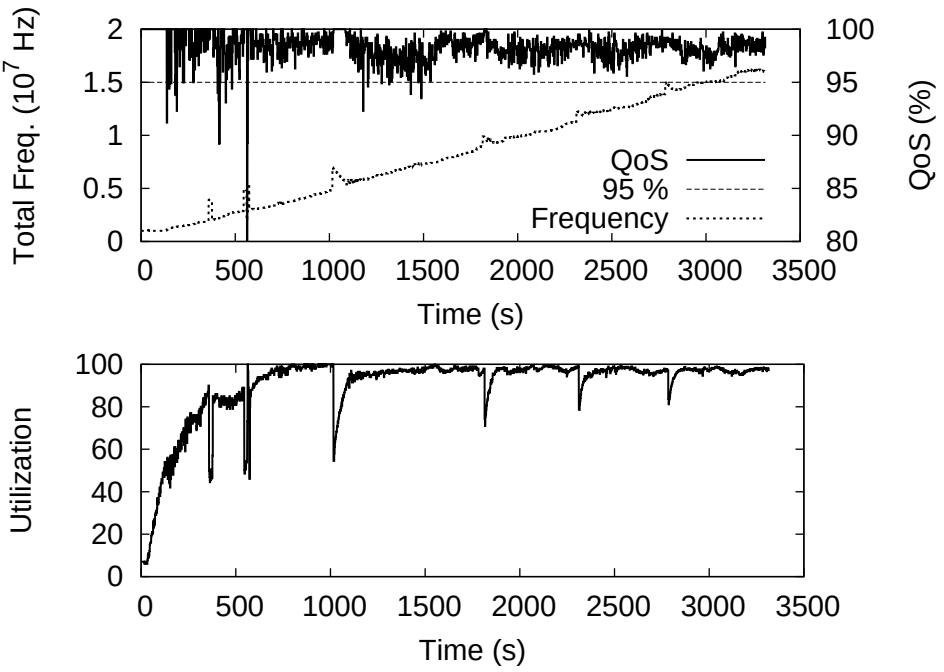

Figure 6.15: Utilization and frequency x QoS for the SIMO controller

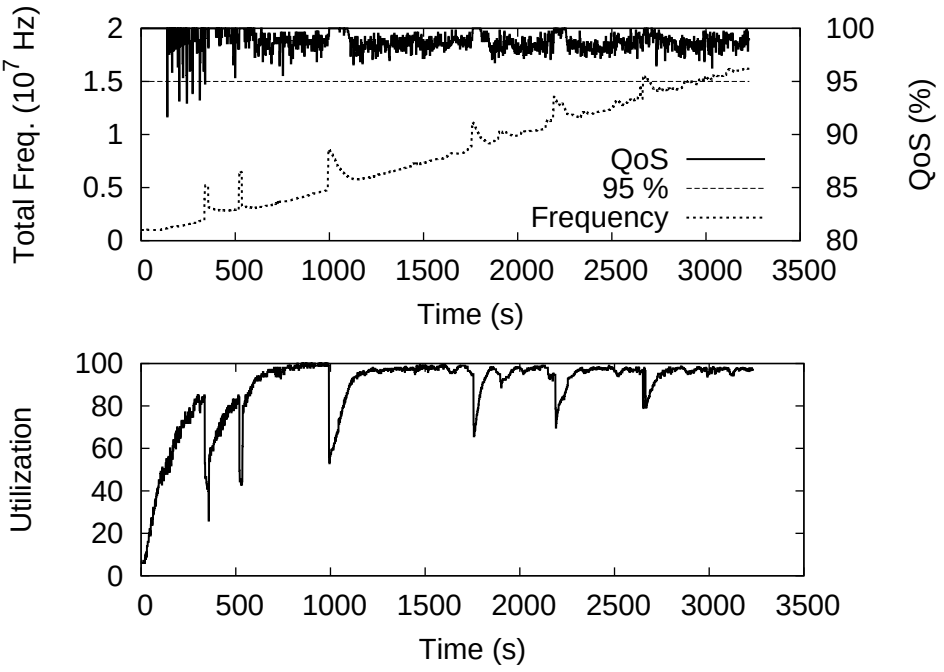

Figure 6.16: Utilization and frequency x QoS for the SISO controller

conservative way is to set it as 1.0, which means that a new server starts at the maximum frequency, and decreases to some steady state point.

6.6.4 A Real Workload Scenario

The experiments shown so far were all based on a well-behaved ramp workload generated by the *httperf* tool. We also experimented with traffic that mimics the shape of a real workload. We used the well-known and available workload for the 1998 Soccer World Cup (*SWC98*) web site [4]. We only mimic the workload shape because the replay of the real workload would not fit to our application. Too many static requests would make the system I/O bound, and thus the DVS would not make a big effect. Furthermore, our server is much smaller than the *SWC98* server, in addition to our request being CPU bound. We then map the maximum load of the *SWC98* to our maximum sustainable load, and adjust dynamically the interval between calls of *httperf* (the X parameter of *wsess*). As the load is generated, we look up the *SWC98* load and then generate the proportional load to our server.

The plot in Figure 6.17 shows the rate of requests for the original *SWC98* workload between the days 64 and 66 of the event. The plot in Figure 6.18 shows the load generated by *httperf*. We translated the maximum load from the original workload to the limits of our cluster, and modify the *httperf* user think time to shape the rate of requests. We also

compressed the time tenfold to result in a smaller time window of about 6 hours.

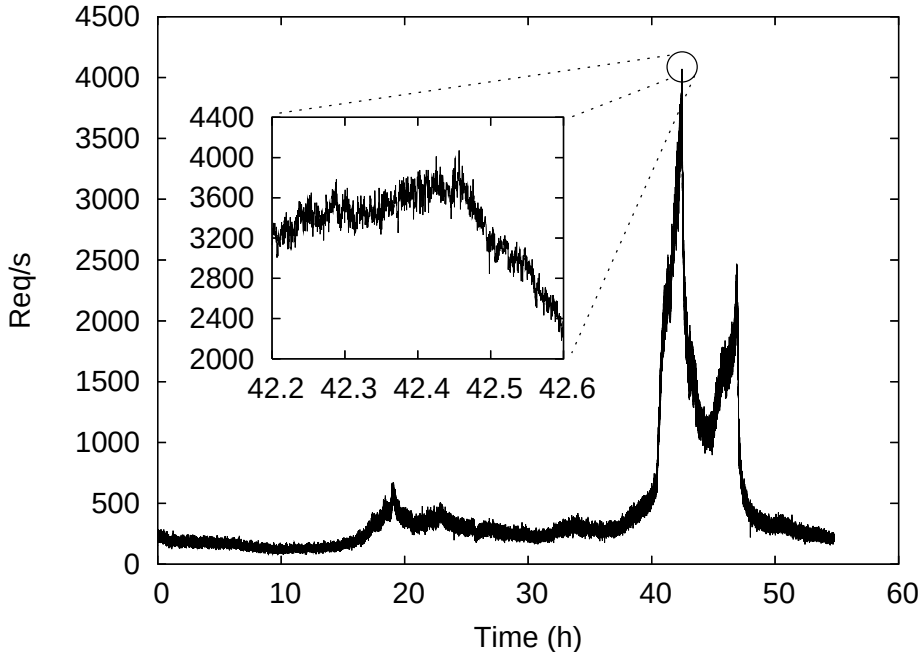

Figure 6.17: 1998 world cup web site workload

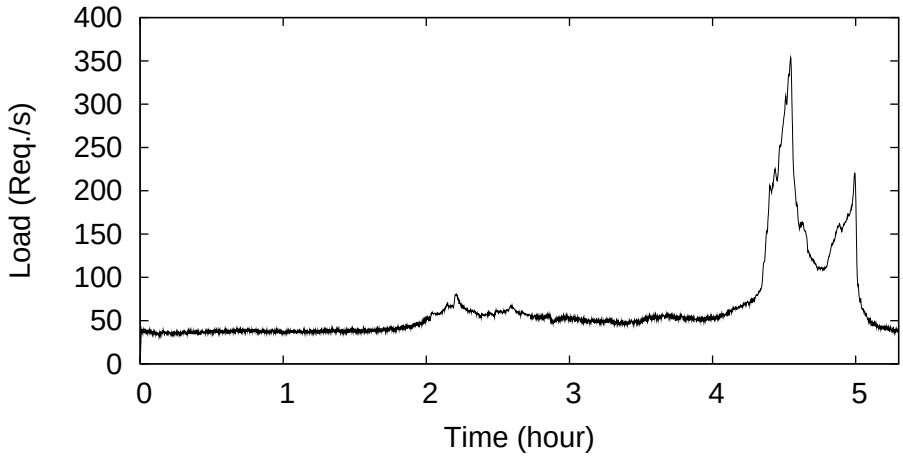

Figure 6.18: Reproduction of the SWC98 workload with *httperf*

The next plots show the power and QoS results of the proposed method, namely QoS Control, compared to the sequential on/off method proposed in [85]. Figure 6.19 shows the power expenditure for the whole experiment in each method, and the value of the integral of the curves for the interval, that is, the energy consumed. At $t = 2h$, our optimization decides that a different machine is better for handling the load, that increases a little after $t = 2h$. As Rusu's method cannot do this, it starts with the machine named *ohm* and goes with it until after $t = 4h$. In our method, the system starts using the server *hertz* and then, at $t = 2h$, changes to *ohm*. The spike of power at $t = 2h$ corresponds to the power

of both machines turned on. The whole experiment resulted in an energy consumption of about $587Wh$ for our QoS Control method, against $624Wh$ for Rusu's method (6.3% higher).

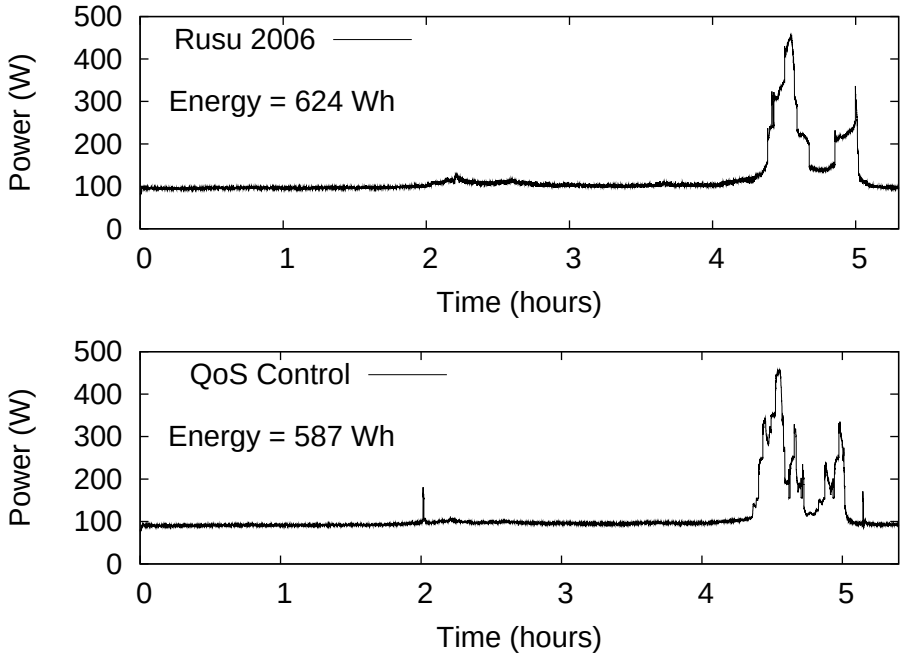

Figure 6.19: Power comparison: full experiment

The plot in Figure 6.20 shows the same data for the interval before $t = 2h$. The energy is reduced from about $192Wh$ to $182Wh$, an expenditure of 5.5% higher for Rusu's method, against our QoS Control method. We observed that the overhead of power for switching machines, if not done too frequently as in this case, is negligible.

We also took a closer look at the experiment where 4 machines of the cluster are turned on to handle the peak load. This is shown in Figure 6.21. The energy reduction was from about $170Wh$ to $163Wh$, a reduction of 4.1% by our QoS Control method. Note that for high loads the energy savings is smaller because there is less room for optimization.

We are controlling the QoS at 0.95 for this experiment. The plots in Figure 6.22 and Figure 6.23 show the average value of the QoS measured in a time window of $40s$. We also show the confidence interval obtained for each window using the confidence interval of a proportion. The curves are plotted with an exponential smoothing average for better clarity. For high loads our algorithm resulted in a small drop of QoS, but this is acceptable. The exact QoS value is impossible to maintain, and even to measure with a narrow confidence interval. This variability must be included in the Service Level Agreement (SLA), so that QoS is specified as a small range, and not as a single value. Rusu's method,

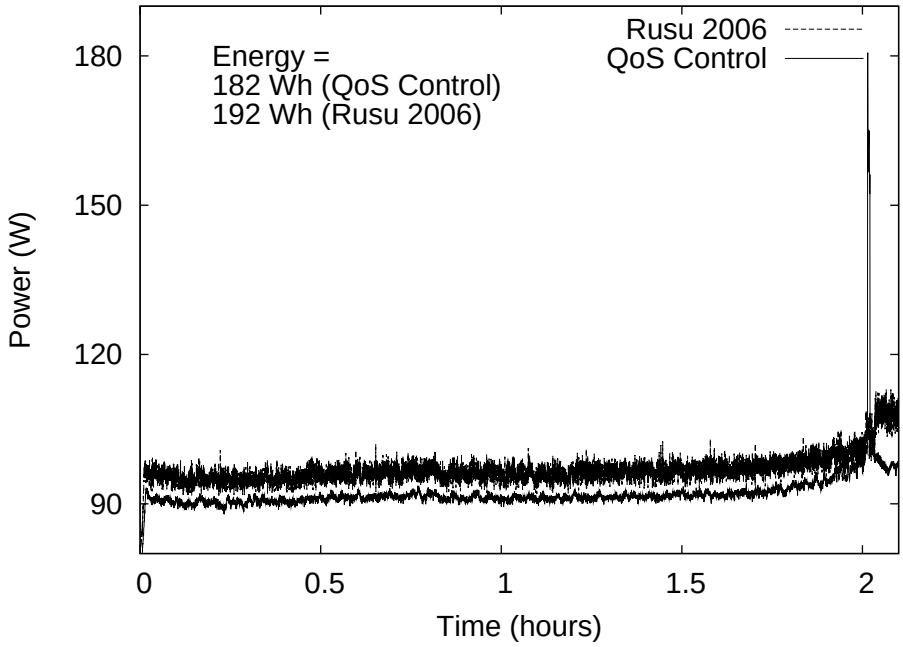

Figure 6.20: Power comparison: $t < 2h$

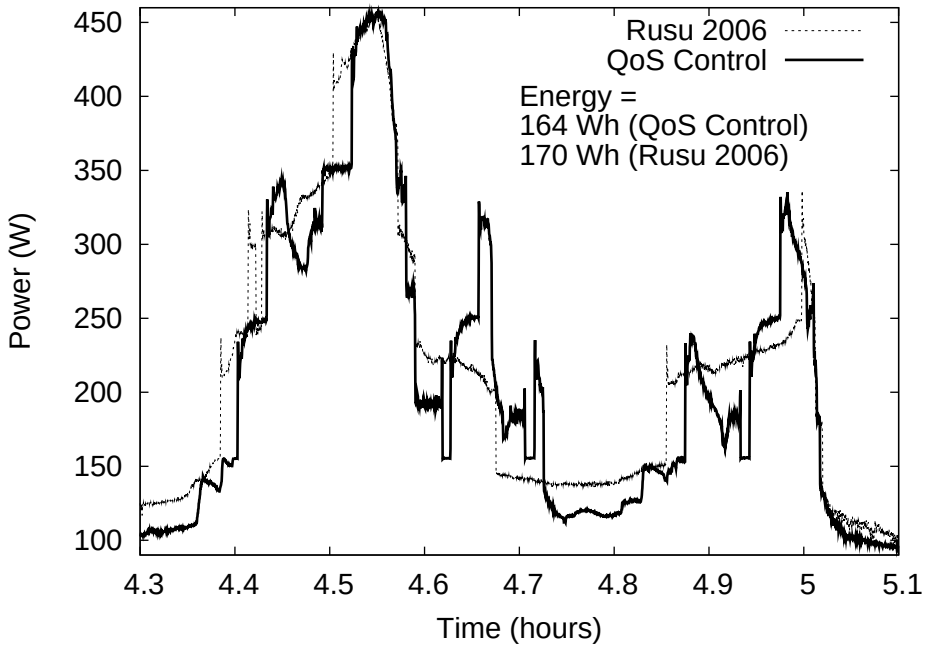

Figure 6.21: Power comparison: peak load

on the other hand, showed a smaller drop of QoS for the same load. This happens because Rusu's method overprovisions the system and spends more energy.

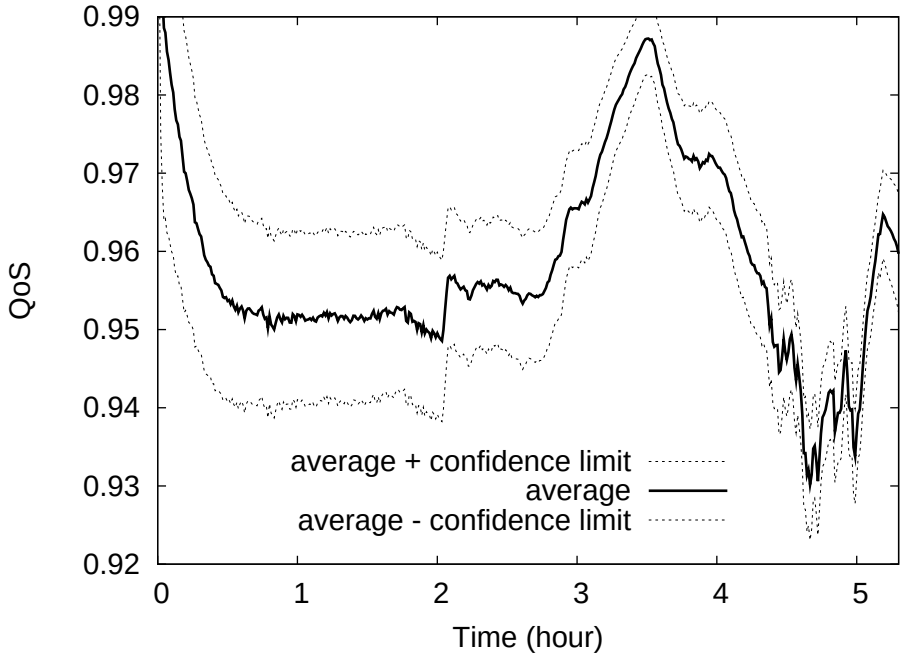

Figure 6.22: QoS for our method

In Figure 6.22, for the period before $t = 3h$, the QoS stays close to the setpoint, with a disturbance at $t = 2h$, when the first server switch occurs. Between $t = 3h$ and $t = 4h$, the QoS stays above the setpoint, because the performance of the new selected machine is too high, and even at the minimum frequency the server is faster than needed. Then, after $t = 4h$, during the peak load, there is a drop in the QoS caused by all the machine switching occurring in this interval, but note that this drop is close to 0.02. This small drop can be tolerated, due to the very stochastic nature of the system, and can be included in the QoS specification in the SLA.

Figure 6.23 shows the QoS for the running of Rusu's method. As expected the QoS stays very close to 1.0 because although it is QoS aware, it cannot control the QoS in a fine grain manner.

6.7 Conclusions

In this chapter we contributed to the state-of-the-art on dynamic cluster configuration by presenting an optimal solution to the cluster configuration, that is, by solving the problem of finding the combination of servers to be turned on/off and their speeds. We

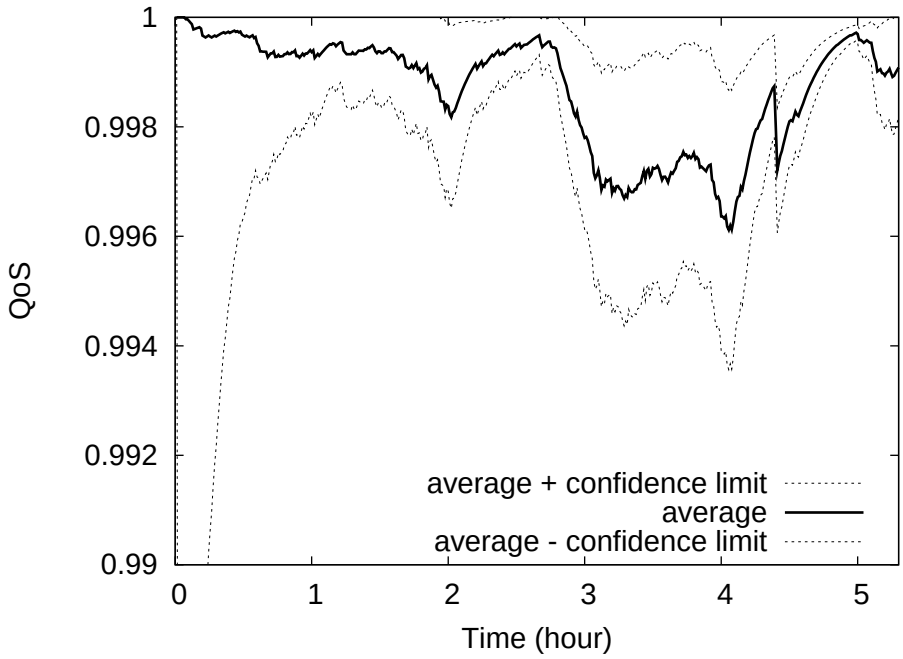

Figure 6.23: QoS for Rusu 2006 method

compared the technique with the work in [30] and for 10 machines it was shown that we can obtain power reductions of up to 40%. Our experiments show that the pseudo-continuous frequency approach is better than using only one discrete frequency per interval, without incurring in additional overhead. The former also allows for the use of a continuous actuator mechanism necessary for using feedback control theory. On the other hand, the discrete frequency method showed to be improper to be used as an actuator mechanism, because it does not provide proportionality between the control output and actuation in the system, generating instabilities from the control perspective.

When comparing the distributed control against the centralized with optimization, we achieved best QoS with lower power consumption in the latter case. The advantage of the distributed approach is the simplicity of the implementation and better scalability, because less tables are needed. However, scalability will not be an issue, because for up to 30 nodes the optimization can be done online, and beyond this cluster size it can be done offline.

We showed that, when using a real-life workload, our method chooses and changes servers with an overhead that is negligible. The experiment that run for $2h$ with one machine (Figure 6.20), and after that continued with one different machine, showed exactly the case when this change of configuration is desirable. If the system stays at a given load for a long period of time, and then the load increases to a value where there is a better

machine to handle the new load, and stays there for another long period, the advantage is clear. In this case, the fixed server sequencing proposed by earlier works cannot change servers.

Chapter 7

Conclusions and Future Directions

The direction in which education starts,
a man will determine his future life.
– Plato

Our premises for the development of this Thesis were based on the course of research related to power management in real-time systems. Initially, energy-aware systems were designed only for embedded applications, because lifetime of embedded devices is limited by its battery. Since the publication of the work in [24], showing the importance to reduce power consumption for web servers, energy-aware research is being directed also to high-end systems, motivated by the goal of reducing the cost of ownership and reducing environmental impacts. We then noticed that in data-centers and companies that depend on high-end servers, there is still a lack of confidence of systems administrators on energy-aware systems, because, intuitively, energy reduction is related to performance reduction. Thus, there is a need to develop dependable systems that move from the maximum performance approach to approaches that are able to tradeoff between energy and performance, and in this Thesis we contributed in this aspect to the state-of-the-art.

In this thesis we investigated local and cluster-wide energy management techniques for heterogeneous web server systems, addressing only the processor subsystem. We had in mind that heterogeneity is important, because systems become heterogeneous after they are deployed, either because of server replacement, or because of system scale up. Heterogeneity offers an extra opportunity to energy minimization, because it adds more variables to the problems, but it becomes more difficult to model and solve the optimization problems. Because of this additional level of complexity, the first papers on energy-aware server clusters, some of them quite recent, were all based on simplified models for homogeneous systems, like in [29, 30, 52, 61, 80, 109].

We based our study on a stochastic approach for modelling web servers performance. A web server is best modeled as a soft real-time system because the timeliness of the system is only related to the user satisfaction. The user satisfaction, in turn, is associated with the response time of the system, and if a single request is not executed before the expected response time, the user will not feel a poor response. A poor response will only happen if, statistically, the system shows a high response time. The better way to build a system that shows good response time in the average, is to measure some statistical characteristics of the system, and try to control them.

Following this reasoning, we first presented a scheme to relate QoS to tardiness in a high-end system given by a multi-tiered e-commerce environment, based on the statistical distribution of the tardiness of web interactions. This QoS metric was shown to be very useful because some practical difficulties arose when we tried to use in the control of the system, the measured QoS by counting deadline misses. On the other hand, tardiness is a continuous value that can be calculated for each web interaction, and its value depicts how close the execution was to the deadline.

We proved that measuring QoS by calculating tardiness is better than by counting deadline misses. One way to see the difference is by comparing two cases: if a request is finished after the deadline, and before the deadline, both with a very small difference, that is, right before or right after the deadline. In the second case, there will be one more deadline miss, while in the first case, the tardiness value will not change significantly from that value of the second case. This shows that our QoS measure based on tardiness is more representative of the real state of the system. In the second case, by counting deadline misses, the same QoS would be reported if the request had finished after a long interval from the deadline, but tardiness would show a bigger value.

Our proposed TQM scheme, considering that tardiness has a Pareto probability distribution, was shown to be better than existing schemes like [85] and [90], because it meets with precision the real-time specification, not overprovisioning the system, and thus saving energy. The method in [85] is QoS aware, because it turns new machines on when the system is showing a QoS below the specified value, but cannot reduce energy consumption generally, for example when the load is lower than the needed value to switch a new machine. In this case, the QoS will be close to 1.0. The TQM improved the QoS awareness by providing a QoS metric that, together with the control closed loop, can put the QoS in the specified value in a much broader range of the load spectrum.

A shortcoming of the TQM approach is when the goal is to meet all deadlines. The

tardiness would have an upper bound of 1.0, and thus the assumption on the tail distribution, where we measure the area below the probability distribution curve and greater than 1.0, would not hold. If all deadlines are met, the real distribution cannot be approximated to a tail distribution, and the tardiness value will have no meaning. However, if the goal were to meet all deadlines, like in a hard real-time system for example, then other algorithms would be recommended, such as earliest deadline first.

Although efficient, the major drawback of the TQM method is that it is based on predetermined probability distributions: Pareto and Log-normal. The approximations showed to be good, but the first questions that arise are: what if the workload is not shaped as Pareto and not Log-normal? Is it possible to generalize? In Chapter 5 we answered this by using a stochastic approximation algorithm that can measure some characteristics of the random variable tardiness regardless of its probability distribution function. This means that the method will work for any kind of workload, with no assumptions.

We then built the Generalized TQM method by using the on line convergent sequential process proposed in [82] that is defined from a Markov chain. We derived quantile estimations that does not depend on the shape of the tardiness probability distribution, so that the metric can be used in any workload. To evaluate the new metric, we showed practical results in a three-tier web cluster with QoS control in an e-commerce environment.

The GTQM results showed a good response even for QoS values closer to 1.0, where the TQM were not so effective. The results also showed a very good precision between the setpoint of QoS and the observed value, for two kinds of workload. The deterministic and repetitive maid up of a fixed execution time and a fixed deadline, and also for the real-world workload given by TPC-W. Note that although the first workload were made of fixed execution time, fixed period, and fixed deadline, the complex execution of the requests inside the several server process make this also a nondeterministic process, with a tail probability distribution of tardiness.

We also showed a practical implementation of a feedback control loop in a multi-tier web server system for e-commerce. Practical issues that arise in the implementation of a controller in a real web cluster application were discussed. The experiments showed that the parameterized controller is easy to tune, because tuning has a limited degree of freedom, which helps stability. Our experiments showed an analysis of sensitivity to the controller parameters that can help in achieving the best performance for the controlled system. The fine-grain QoS control showed in this work is useful in achieving extra energy

savings for interval based DVS schemes, where the goal is to meet all deadlines, avoiding overprovisioning the system according to the real-time specifications.

In Chapter 6 we contributed to the state-of-the-art on dynamic cluster configuration by presenting an optimal solution to the cluster configuration, that is, by solving the problem of finding the combination of servers to be turned on/off and their speeds. We compared the technique with the work in [30] and for 10 machines we can obtain power reductions of up to 40%.

Our experiments showed that the pseudo-continuous frequency approach is better than the use of only one discrete frequency per interval, without incurring in additional overhead. We proved in Appendix B that energy consumption is smaller, considering that only efficient frequencies are used. We also showed that pseudo-continuous frequency is more appropriate for using with the control loop, because the control logic relies on a continuous actuator to work well. The overhead of switching DVS is in fact smaller than in other works. For example, [85] uses a 10ms period to change frequency, and we showed that 10 times this value is still small to provide a fine grain result.

When comparing the distributed control against the centralized with optimization, we achieved best QoS with lower power consumption in the latter case. The advantage of the distributed approach is the simplicity of the implementation and better scalability, because less tables are needed. However, scalability will not be an issue, because for up to 30 nodes the optimization can be done online, and beyond this cluster size it can be done offline.

The studies on cluster reconfiguration with the MySQL cluster implementation allowed us to identify that the performance scalability and the CPU boundness of the application are the two most important factors necessary for effective energy savings, respectively for dynamic reconfiguration, and for applying DVS. The former, performance scalability, is important because as machines are turned on, we need to scale up the cluster capacity. Using one front-end, for example, is a limiting factor for the performance scalability, because it cannot handle an infinite load. Network bandwith may also become a limiting factor. The latter, CPU boundness, is important to provide the maximum energy saving from the DVS techniques, as reducing energy from disks are much more difficult. Techniques of spinning down idle disks, or and using lower rotating speeds than the maximum speed, are being considered [92].

For future work, workload forecasting is a challenging branch of research that we must further investigate. We have a preliminary work [89] where we apply DVS control

through the forecast of the future system state, and then prepare the system to deliver a specified quality of service (QoS) that reduces the energy consumption. In that work, the prediction was based on the Holt's linear method of forecasting. A good forecasting would allow, for example, the decision on whether to change or not the configuration of the cluster, taking into account the energy overhead of turning nodes on and off.

Another future direction is to study how to optimize the cluster power if we consider further means of reducing energy in the database machines. This is a difficult practical problem because turning on and off databases is not simple, as there are consistency issues. We have to deal with long boot times caused by the resynchronizations of the databases, and thus it has to be done in a much bigger time granularity. The MySQL cluster architecture is a good reference to consider for this implementation, because of some of its desirable properties. As it is designed for high availability with no single point of failure, the solution is already designed to afford a machine being turned of. A database cluster node automatically restart, recover, and configure itself in case of failure (or an intentional turning off action). Furthermore, it can run only in main memory, providing the high performance that we need to apply DVS and expect high energy savings. However, performance scalability, and availability of memory are still unsolved issues.

Virtualization is also a promising research area for future attention. We are seeing now a tendency on aggregating and virtualizing of servers which can help in solving the energy problem, by allowing a better use of CPU resources, avoiding underutilized machines. Power-efficient control for data-centers built up of virtual machines has to be multivariable, because hardware components subject to power management affect all VMs [105]. Transparent and energy-efficient live migration from one VM to another may be a good solution for implementing dynamic configuration in server clusters. The problem of how to do power management in a system with several VMs is still an open question.

The EPA report [1] mentioned in Chapter 1 gives some directions of research, some of them are not in the scope of this thesis, for example research related to energy-efficient cooling systems and efficient heat removal. Some others we have addressed in this Thesis. The relevant future work cited in that report are:

- Improve energy performance of hardware-based virtualization technologies (reduce virtualization overhead).

- Develop, validate, and demonstrate the effectiveness of virtualization software-based system power management.
- Improve software development tools and techniques to allow software to more efficiently use chip-level multiprocessing (improved parallelization).
- Develop lower power states for use at lower utilization levels.
- Improve power management for storage systems, to allow many disks to remain off most of the time with little impact on performance; investigate impact of storage latency.
- Investigate application of solid-state (non-mechanical) storage technologies to data centers.
- Develop improved computing control strategies (such as statistical machine learning or control theory) to allow better power management of IT equipment at the system, cluster, and data center level.
- Develop active power management strategies for high-performance computing systems, e.g., taking advantage of workload imbalances to reduce the power of lightly loaded system components.
- Develop standard communication protocols to allow continuous energy monitoring and interoperability among IT equipment and data center infrastructure products.
- Develop best practices for improving energy efficiency through storage optimization and server virtualization.

Virtualization is indeed being considered in the EPA report for investigation, and also several research needs related to the development of better hardware to reduce overhead and increase parallelism. But note that one item is related to the development of improved computing control strategies with statistical machine learning or control theory, which was the focus of this Thesis.

APPENDIX A - TPC-W Benchmark

Experience is the teacher of all things.
– Julius Caesar

TPC-W is a transactional web benchmark, produced by the Transaction Processing Performance Council [98], where the workload is performed in a controlled Internet commerce environment. The workload tests several system components associated with this environments, such as multiple on-line browser sessions, dynamic page generation, secure connections, and a database consisting of many tables with a wide variety of sizes, attributes, and relationships.

A possible environment for the TPC-W is depicted in Figure A.1. The workload is generated by the remote browser emulator, responsible for managing the emulated browsers (EB) and the emulated sessions. The EBs access the web server using HTTP and HTTPS connections. The system under test is composed of three components, the web server for static pages, the application server for the execution of the application (e.g., using PHP), and the database server.

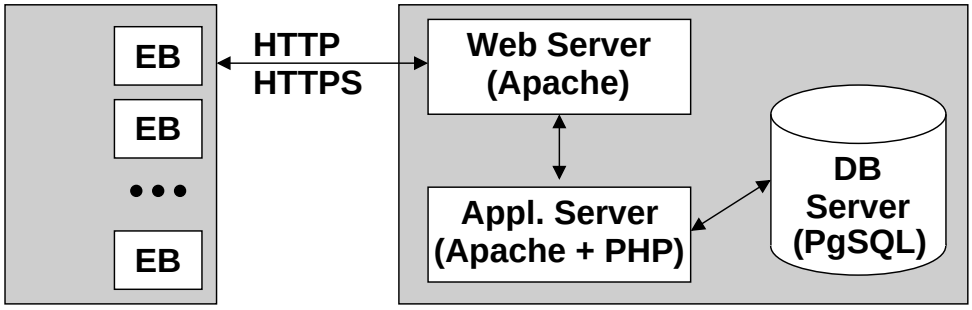

Figure A.1: TPC-W environment

The performance metric reported by TPC-W is the number of *web interactions per second* (WIPS). TPC-W specifies 14 different interactions necessary to simulate the activity of a retail store, and each interaction is subject to a deadline that must be met with a

specified QoS (as a percentage of deadlines met). There are three different profiles for the test, with a mix of interactions for shopping, browsing and ordering. The primary metric (WIPS) is intended to reflect an average shopping scenario with a mix of 80% of browsing interactions and 20% of ordering interactions (for a review about the TPC-W benchmark see [36]). Besides the primary metric WIPS, there is also a specification for the associated price per WIPS ($/WIPS), and the availability date of the priced configuration

One aspect of the TPC-W specification is that one web interaction is not composed of a single request, but of a request to a dynamic page followed by several static requests for the embedded objects that are part of the dynamic generated page. The *web interaction response time* (WIRT) is defined by the time elapsed between sending the dynamic request until receiving the last byte of the last embedded object. This specification makes it impossible to measure the QoS locally in one server node, because each embedded object request may be sent to different nodes.

In the TPC-W real-time specification, each class of web interaction has a different deadline, as shown in Figure A.2, with a minimum deadline hit ratio defined by the standard as 90% for all classes. Although it is not specified in the standard, a system should not a priori discard 10% of the requests just because the goal is to service 90%, but rather it should attempt to service all requests and provide for all an equal probability of meeting the deadline. Also, it is worth to note that these values for the deadlines include only local area access, according to the TPC-W specification, so that the Internet access can be disregarded.

	Admin Confirm	Admin Request	Best Sellers	Buy Confirm	Buy Request	Customer Regist.	Home	New Products	Order Display	Order Inquiry	Product Detail	Search Request	Shopping Cart	Search Results
90% WIRT Constraint (deadline in seconds)	***20***	***3***	***5***	***5***	***3***	***3***	***3***	***5***	***3***	***3***	***3***	***3***	***10***	***3***

Figure A.2: Deadlines as defined by TPC-W

The TPC-W was approved in February 2000, and a new standard, the TPC-App was released in April 2005, as a more general application server and web services benchmark. The new TPC-App benchmark focuses on commercial application server environments set in a B2B Web services workload. In addition to the differences between both, some complaints was reported about TPC-W that it was flawed with respect to being able to

web cache too much of the data. In despite of that, TPC-W fits very well for the purpose of this work, and has the advantage of bigger availability of open source implementations.

APPENDIX B - Efficacy of the Switched DVS Scheme

Restlessness and discontent are the first
necessities of progress.
– Thomas A. Edison

We will show that the efficacy of this switched DVS scheme is better than using the lowest available discrete value higher than the necessary frequency. Picture B.1 show the power consumption for both schemes. The dashed line is the scheme that switches to the lowest available frequency higher than the load. The continuous line shows the switching DVS scheme.

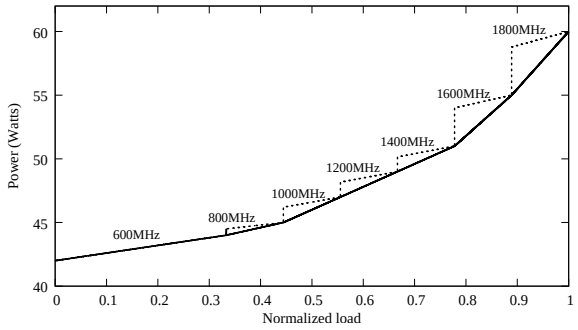

Figure B.1: Power consumption for the Pentium M processor using two different DVS schemes.

However, the benefit shown in Figure B.1 only happen for a processor that satisfies the criteria of having only efficient frequencies, and it does not consider the DVS overhead. The definition of inefficient frequencies were given in [70], and the concept was extended for different frequencies with different idle power in [108]. A frequency f_1 is inefficient with respect to a higher frequency f_2 if running any task at $f1$ consumes more energy than running in $f2$. In that technical report [108], the authors showed that the condition for this to happen is given by:

$$\frac{P(f_1) - I(f_2)}{f_1} \geq \frac{P(f_2) - I(f_2)}{f_2} \tag{B.1}$$

We can now formulate this simple theorem about the switching DVS technique.

Theorem B.1 (Switching DVS) *Given a load of s cycles per second, the technique of switching between the frequencies $||s||^-$ and $||s||^+$, with a duty cycle α so that $\alpha||s||^+ + (1-\alpha)||s||^- = s$, is more power efficient than using always $||s||^+$, if and only if the processor has only efficient frequencies.*

Proof Let f_1 be the lower frequency $||s||^-$ and f_2 be the higher frequency $||s||^+$. The power consumption of the switching DVS at a load of f_1 cycles per second will be exactly the power consumption at full load in frequency f_1 ($\alpha = 0$). Let this value be $P(f_1)$. The same for f_2, but with $\alpha = 1$, and the power consumption will be $P(f_2)$. In contrast, the power consumption of the DVS scheme with always $||s||^+$, at a load $f_1 + \varepsilon$, will have a power consumption given by $\frac{f_1}{f_2}P(f_2) + \left(1 - \frac{f_1}{f_2}\right) I(f_2)$, where $I(f_2)$ is the idle power consumption in f_2. The better scheme will be the one with lower power consumption at the load f_1 cycles per second. Thus, we need:

$$P(f_1) < \frac{f_1}{f_2}P(f_2) + \left(1 - \frac{f_1}{f_2}\right) I(f_2)$$

rearranging this inequation, we obtain inequation B.1. ∎

References

[1] Report to congress on server and data center energy efficiency public law 109–431. Tech. rep., U.S. Environmental Protection Agency EPA, East Lansing, Michigan, August 2007.

[2] Abdelzaher, T. F., Shin, K. G., and Bhatti, N. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. on Parallel and Distributed Systems 13*, 1 (2002), 80–96.

[3] Apache. Apache mod_proxy extension for load balancing, 2007. http://httpd.apache.org.

[4] Arlitt, M., and Jin, T. A workload characterization study of the 1998 world cup web site. *IEEE Network 14*, 3 (May/June 2000), 30–37.

[5] Aydin, H., Mejía-Alvarez, P., Mossé, D., and Melhem, R. Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *22nd IEEE Real-Time Systems Symposium (RTSS '01)* (London, UK, December 2001), pp. 95–105.

[6] Aydin, H., Melhem, R., Mossé, D., and Mejía-Alvarez, P. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput. 53*, 5 (2004), 584–600.

[7] Backhand Project. http://www.backhand.org.

[8] Baryshnikov, Y., Coffman, E., Pierre, G., Rubenstein, D., Squillante, M., and Yimwadsana, T. Predictability of web-server traffic congestion. In *10th International Workshop on Web Content Caching and Distribution* (Sophia Antipolis, France, September 2005), pp. 99–103.

[9] Benini, L., and de Micheli, G. System-level power optimization: Techniques and tools. *ACM Transactions on Design Automation of Electronic Systems (TODAES) 5*, 2 (April 2000), 115 – 192.

[10] Bertini, L., and Leite, J. C. B. Um breve survey: Escalonamento em sistemas de tempo real com otimização do consumo de energia. In *VI Workshop de Sistemas Tempo Real (WTR'04)* (Gramado - RS, May 2004).

[11] Bertini, L., Leite, J. C. B., and Mossé, D. Coordinated DVS for QoS control in energy-efficient web clusters. In *9th Workshop on Real-Time Systems* (Belém, Brazil, May 2007), pp. 73–80.

[12] Bertini, L., Leite, J. C. B., and Mossé, D. SISO PIDF controller in an energy-efficient multi-tier web server cluster for e-commerce. In *2nd IEEE Intl. Workshop on Feedback Control Impl. and Design in Computing Systems and Networks* (Munich, Germany, May 2007), pp. 33–38.

[13] Bertini, L., Leite, J. C. B., and Mossé, D. Statistical QoS guarantee and energy-efficiency in web server clusters. In *19th Euromicro Conference on Real-Time Systems* (Pisa, Italy, July 2007), pp. 83–92.

[14] Bertini, L., Leite, J. C. B., and Mossé, D. Dynamic configuration of web server clusters with qos control. In *20th Euromicro Conference on Real-Time Systems* (Prague, Czech Republic, july 2008). Work in Progress Session.

[15] Bertini, L., Leite, J. C. B., and Mossé, D. Optimal dynamic configuration in web server clusters. Tech. Rep. RT-1/08, Instituto de Computação – UFF, January 2008.

[16] Bertini, L., Leite, J. C. B., and Mossé, D. Generalized tardiness quantile metric: Distributed dvs for soft real-time web clusters. In *21th Euromicro Conference on Real-Time Systems* (Dublin, Ireland, July 2009). To appear.

[17] Bertini, L., Loques, O., and Leite, J. C. B. Replicação de dados em redes ad hoc para sistemas de apoio em situções de desastres. In *23o Simpósio Brasileiro de Redes de Computadores (SBRC'05)* (Fortaleza - CE, May 2005), vol. 1, pp. 539–552. best paper award.

[18] Bertini, L., Loques, O., and Leite, J. C. B. Improving data availability in ad hoc wireless networks. In *ICIC (1)* (2006), D.-S. Huang, K. Li, and G. W. Irwin, Eds., vol. 4113 of *Lecture Notes in Computer Science*, Springer, pp. 1306–1313.

[19] Bertsimas, D., and Popescu, I. Optimal inequalities in probability theory: A convex optimization approach. *SIAM J. on Optimization 15*, 3 (2005), 780–804.

[20] Bhardwaj, M., Min, R., and Chandrakasan, A. Power-aware systems. In *34th Asilomar Conference on Signals, Systems, and Computers* (November 2000), vol. 2.

[21] Bianchini, R., and Rajamony, R. Power and energy management for server systems. *IEEE Computer 37*, 11 (2004), 68–74.

[22] Bíró, J., and Heszberger, Z. Comparison of simple tail distribution estimators. In *IEEE International Conference on Communications* (1999), vol. 3, pp. 1841–1845.

[23] Bodik, P., Armbrust, M. P., Canini, K., Fox, A., Jordan, M., and Patterson, D. A. A case for adaptive datacenters to conserve energy and improve reliability. Tech. Rep. UCB/EECS-2008-127, EECS Department, University of California, Berkeley, September 2008.

[24] Bohrer, P., Elnozahy, M., Kistler, M., Lefurgy, C., McDowell, C., and Rajamony, R. The case for power management in web servers. In *Power Aware Computing*, R. Graybill and R. Melhem, Eds. Kluwer Academic Publishers, 2002.

[25] Brasil Telecom. Cyber Data Center - CyDC. http://www.cydc.com.br/.

[26] Canuto, O. A insustentável leveza da economia. O Estado de São Paulo Newspaper, July, 13 1999. http://www.eco.unicamp.br/artigos/artigo87.htm.

[27] Cardellini, V., Casalicchio, E., Colajanni, M., and Yu, P. S. The state of the art in locally distributed web-server systems. *ACM Computing Surveys 34*, 2 (2002), 263–311.

[28] Chase, J., Anderson, D., Thakur, P., and Vahdat, A. Managing energy and server resources in hosting centers. In *18th Symp. on Operating Systems Principles* (Banff, Alberta, Canada, October 2001), pp. 103–116.

[29] Chase, J., and Doyle, R. Balance of power: Energy management for server clusters. In *Eighth Workshop on Hot Topics in Operating Systems* (Schloss Elmau, Germany, May 2001), pp. 1–6.

[30] Chen, Y., Das, A., Qin, W., Sivasubramaniam, A., Wang, Q., and Gauta, N. Managing server energy and operational costs in hosting centers. *SIGMETRICS Perform. Eval. Rev. 33*, 1 (2005), 303–314.

[31] Cheng, A. M. K. *Enterprise Information Systems II.* Kluwer Academic, 2001, ch. E-Commerce and its Real-Time Requirements: Modeling E-Commerce as a Real-Time System.

[32] Citigroup. Citi data center wins environmental award, December 2007. http://findarticles.com/p/articles/mi_m0EIN/is_2007_Dec_7/ai_n21148474.

[33] Citigroup. Citi Data Centre – Arup Associates Project, 2008. http://www.arup.com/germany/project.cfm?pageid=11969.

[34] Coskun, A. K., Rosing, T. S., Mihic, K., Micheli, G. D., and Leblebici, Y. Analysis and Optimization of MPSoC Reliability. *Journal of Low Power Electronics 2*, 1 (2006), 56–69.

[35] Crovella, M. E., and Bestavros, A. Self-similarity in world wide web traffic: Evidence and possible causes. In *ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems* (May 1996), pp. 160–169.

[36] Daniel F. Garcia, J. G. TPC-W e-commerce benchmark evaluation. *IEEE Computer 36*, 2 (February 2003), 42–48.

[37] Dantzig, G. B. On the significance of solving linear programming problems with some integer variables. *Econometrica 28*, 1 (January 1960), 30–44.

[38] Diao, Y., Hellerstein, J. L., and Parekh, S. Control of large scale computing systems. *SIGBED Rev. Special Issue on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2006) 3*, 2 (2006), 17–22.

[39] DiPippo, L. C., Fay-Wolfe, V., Nair, L., Hodys, E., and Uvarov, O. A real-time multi-agent system architecture for e-commerce applications. In *Intl. Symp. on Autonomous Decentralized Systems* (March 2001), pp. 357–364.

[40] Elnozahy, M., Kistler, M., and Rajamony, R. Energy-efficient server clusters. In *Second Workshop on Power Aware Computing Systems* (Cambridge, MA, USA, February 2002), pp. 179–196.

[41] Elnozahy, M., Kistler, M., and Rajamony, R. Energy conservation policies for web servers. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems (USITS'03)* (Seattle, WA, USA, March 2003), USENIX Association, pp. 8–8.

[42] Eyal, A., and Milo, T. Integrating and customizing heterogeneous e-commerce applications. In *Workshop on Technologies for E-Services* (September 2000), pp. 16–38. In cooperation with VLDB2000.

[43] Falkman, P., Vahidi, A., and Lennartson, B. Convenient, almost optimal and robust tuning of PI and PID controllers. In *15th IFAC World Congress* (Barcelona, Spain, July 2002).

[44] Fu, Y., Wang, H., Lu, C., and Chandra, R. S. Distributed utilization control for real-time clusters with load balancing. In *27th IEEE Intl. Real-Time Systems Symposium* (Rio de Janeiro, Brazil, December 2006), pp. 137–146.

[45] Garey, M. R., and Johnson, D. S. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[46] GLPK. GNU Linear Programming Kit, 2007. http://www.gnu.org/software/glpk/glpk.html.

[47] Gray, A. Data centers: Myths of energy efficiency. Building Operating Management, Setembro 2005. http://www.facilitiesnet.com/bom/article.asp?id=3295.

[48] Guerra, R., Bertini, L., and Leite, J. C. B. Managing energy and quality of service in heterogeneous server clusters. In *32nd Latin-American Conf. On Informatics* (Santiago, Chile, 2006).

[49] Havinga, P. J. M., and Smit, G. J. M. Energy-efficient wireless networking for multimedia applications. In *Applications in Wireless Communications and Mobile Computing, Wiley* (2001), pp. 165–184.

[50] Heath, T., Diniz, B., Carrera, E. V., Jr., W. M., and Bianchini, R. Energy conservation in heterogeneous server clusters. In *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming* (Chicago, June 2005), pp. 186–195.

[51] Hill, T., and Lewicki, P. *Elementary Concepts in Statistics.* StatSoft, Inc., 2006, ch. Distribution Fitting. http://www.statsoft.com/ textbook/stathome.html.

[52] Horvath, T., Abdelzaher, T., Skadron, K., and Liu, X. Dynamic voltage scaling in multitier web servers with end-to-end delay control. *IEEE Transactions on Computers 56*, 4 (April 2007), 444–458.

[53] Ishihara, T., and Yasuura, H. Voltage scheduling problem for dynamically variable voltage processors. In *Intl. Symp. on Low power electronics and design* (Monterey, California, United States, 1998), pp. 197–202.

[54] Jain, R. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

[55] Jones, C. E., Sivalingam, K. M., Agrawal, P., and Chen, J. C. A survey of energy efficient network protocols for wireless networks. *Wireless Networks 7*, 4 (August 2001), 343–358.

[56] Kephart, J. O., and Chess, D. M. The vision of autonomic computing. *Computer 36*, 1 (2003), 41–50.

[57] Kolinski, J., Chary, R., Henroid, A., and Press, B. *Building the Power-Efficient PC - A Developer's Guide to ACPI Power Management.* Intel Press, 2002.

[58] Kristiansson, B., and Lennartson, B. Robust tuning of PI and pid controllers. *Control Systems Magazine 26*, 1 (2006), 55–69.

[59] Kushner, H. J., and Yin, G. G. *Stochastic Approximation and Recursive Algorithms and Applications*, 2nd ed. Spring-Verlag, 1997.

[60] Lehoczky, J. P. Real-time queueing network theory. In *Proceedings of the 18th IEEE Real-Time Systems Symposium* (San Francisco, CA, USA, December 1997), p. 58.

[61] Lien, C.-H., Bai, Y.-W., Lin, M.-B., and Chen, P.-A. The saving of energy in web server clusters by utilizing dynamic sever management. In *12th IEEE Intl. Conf. on Networks* (Singapore, November 2004), vol. 1, pp. 253–257.

[62] Liu, X., Heo, J., and Sha, L. Modeling 3-tiered web applications. In *13th Intl. Symp. on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems* (September 2005), pp. 307–310.

[63] Lu, C., Lu, Y., Abdelzaher, T. F., Stankovic, J. A., and Son, S. H. Feedback control architecture and design methodology for service delay guarantees in web servers. *IEEE Trans. Parallel Distrib. Syst. 17*, 9 (September 2006), 1014–1027.

[64] Lu, C., Stankovic, J. A., Son, S. H., and Tao, G. Feedback control real-time scheduling: Framework, modeling, and algorithms. *Real-Time Syst. 23*, 1-2 (2002), 85–126.

[65] Lu, Y., Abdelzaher, T. F., Lu, C., Sha, L., and Liu, X. Feedback control with queueing-theoretic prediction for relative delay guarantees in web servers. In *IEEE Real Time Technology and Applications Symposium* (2003), pp. 208–218.

[66] Mazzeo, L. M., Pantoja, S., and Ferreira, R. Evolução da internet no Brasil e no mundo. Tech. rep., Ministério da Ciência e Tecnologia - Secretaria de Política de Informática e Automação, http://www.mct.gov.br/Temas/info/Palestras/EvolInter.pdf, Abril 2000.

[67] Mejia-Alvarez, P., Levner, E., and Mossé, D. Adaptive scheduling server for power-aware real-time tasks. *Trans. on Embedded Computing Sys. 3*, 2 (2004), 284–306.

[68] Mishra, R., Rastogi, N., Zhu, D., Mossé, D., and Melhem, R. G. Energy aware scheduling for distributed real-time systems. In *17th International Parallel and Distributed Processing Symposium (IPDPS 2003)* (Nice, France, Abril 2003), p. 21.

[69] Mitchell-Jackson, J. Energy needs in an internet economy: A closer look at data centers. Master's thesis, Energy and Resources Group, University of California at Berkeley, EUA, July 2001.

[70] Miyoshi, A., Lefurgy, C., Hensbergen, E. V., Rajamony, R., and Rajkumar, R. Critical power slope: Understanding the runtime effects of frequency scaling. In *16th Intl. Conf. on Supercomputing* (2002), pp. 35–44.

[71] Mosberger, D., and Jin, T. httperf – a tool for measuring web server performance. *ACM SIGMETRICS Perform. Eval. Rev. 26*, 3 (1998), 31–37.

[72] Mossé, D., Aydin, H., Childers, B., and Melhem, R. Compiler-assisted dynamic power-aware scheduling for real-time applications. In *Workshop on Compilers and Operating Systems for Low Power (COLP'00)* (Philadelphia, October 2000).

[73] Mudge, T. Power: A first-class architectural design constraint. *IEEE Computer 34*, 4 (April 2001), 52–58.

[74] National Instruments. NI USB-6008/6009 user guide and specifications, 2007. http://www.ni.com.

[75] Petrucci, V., Loques, O., and Mossé, D. A framework for dynamic adaptation of power-aware server clusters. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing* (Honolulu, Hawaii, USA, 2009), ACM, pp. 1034–1039.

[76] PgFoundry. TPC-W PHP Implementation Project Page. http://pgfoundry.org/projects/tpc-w-php.

[77] Pillai, P., and Shin, K. G. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th Symposium on Operating System Principles (SOSP 2001)* (Banff, Alberta, Canada, 2001), pp. 89–102.

[78] Pinheiro, E., Bianchini, R., Carrera, E., and Heath, T. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on Compilers and Operating Systems for Low Power* (Barcelona, Spain, 2001).

[79] Pinheiro, E., Bianchini, R., Carrera, E. V., and Heath, T. Dynamic cluster reconfiguration for power and performance. In *Compilers and Operating Systems for Low Power*. Kluwer Academic Publishers, 2003.

[80] Rajamani, K., and Lefurgy, C. On evaluating request-distribution schemes for saving energy in server clusters. In *IEEE Intl. Symp. on Performance Analysis of Systems and Software* (Austin, TX, USA, March 2003), pp. 111–122.

[81] Rasmussen, N. Implementing energy efficient data centers, August 2006. White Paper, American Power Conversion.

[82] Robbins, H., and Monro, S. A stochastic approximation method. *The Annals of Mathematical Statistics 22*, 3 (September 1951), 400–407.

[83] Roychowdhury, D., Koren, I., and Krishna, C. A voltage scheduling heuristic for real-time task graphs. In *Performance and Dependability Symposium (IPDS 2003)* (2003), pp. 741–750.

[84] Rusnak, I. The generalized PID controller for stochastic systems. In *The 21st IEEE Convention of the Electrical and Electronic Engineers in Israel* (Tel-Aviv, Israel, 2000), pp. 145–148.

[85] Rusu, C., Ferreira, A., Scordino, C., Watson, A., Melhem, R., and Mossé, D. Energy-efficient real-time heterogeneous server clusters. In *IEEE Real-Time and Embedded Technology and Applications Symp.* (San Jose, CA, USA, April 2006), pp. 418–428.

[86] Rusu, C., Melhem, R., and Mossé, D. Maximizing rewards for real-time applications with energy constraints. *ACM Transactions on Embedded Computing Systems (TECS) 2*, 4 (November 2003), 537–559.

[87] Rusu, C., Melhem, R. G., and Mossé, D. Maximizing the system value while satisfying time and energy constraints. In *23rd IEEE Real-Time Systems Symposium (RTSS'02)* (Austin, Texas, USA, December 2002), pp. 246–255.

[88] Rusu, C., Xu, R., Melhem, R., and Mossé, D. Energy-efficient policies for request-driven soft real-time systems. In *16th Euromicro Conf. on Real-Time Systems* (July 2004), pp. 175–183.

[89] Sant'Ana, C. H., Bertini, L., Leite, J. C. B., and Mossé, D. Applying forecasting to interval based DVS. In *10th Workshop on Real-Time Systems* (Rio de Janeiro, Brazil, May 2008), pp. 1–8.

[90] Sharma, V., Thomas, A., Abdelzaher, T. F., Skadron, K., and Lu, Z. Power-aware QoS management in web servers. In *24th IEEE Real-Time Systems Symposium* (December 2003), pp. 63–72.

[91] Son, S. H., and Kang, K.-D. Qos management in web-based real-time data services. In *Proc. of the Fourth IEEE Intl. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems* (2002), pp. 129–136.

[92] Son, S. W., and Kandemir, M. Energy-aware data prefetching for multi-speed disks. In *Proceedings of the 3rd conference on Computing frontiers* (Ischia, Italy, 2006), ACM, pp. 105–114.

[93] SPSS. Statistical analysis software. http://www.spss.com/.

[94] Stankovic, J. A., Burns, A., Jeffay, K., Jones, M., Koob, G., Lee, I., and Etecetera. Strategic directions in real-time and embedded systems. *ACM Computing Surveys 28*, 4 (December 1996), 751–763.

[95] Stankovic, J. A., Lu, C., and Son, S. H. The case for feedback control real-time scheduling. In *11th Euromicro Conference on Real-Time Systems (ECRTS)* (York, England, 1999), pp. 11–20.

[96] Subramanian, S., and Singhal, M. Real-time aware protocols for general e-commerce and electronic auction transactions. In *ICDCS Workshop on Electronic Commerce and Web-Based Applications* (May 1999), pp. 65–70.

[97] Sukinik, D. First green data center, July 17th 2006. The LEED Guide online article http://www.edcmag.com.

[98] TPC. Transaction Processing Performance Council. http://www.tpc.org/.

[99] Unsal, O. S., and Koren, I. System-level power-aware design techniques in real-time systems. *Proceedings of the IEEE 91*, 7 (July 2003), 1055–1069.

[100] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. An analytical model for multi-tier internet services and its applications. In *ACM SIGMETRICS Intl. Conf. on Measurement and Modeling of Computer Systems* (2005), pp. 291–302.

[101] Venkatachalam, V., and Fran, M. Power reduction techniques for microprocessor system. *ACM Computing Surveys 37*, 3 (September 2005), 195–237.

[102] Wang, M., Kandasamy, N., Guez, A., and Kam, M. Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters. In *Intl. Conf. on Autonomic Computing* (June 2006), pp. 165–174.

[103] Wang, X., Jia, D., Lu, C., and Koutsoukos, X. Decentralized utilization control in distributed real-time systems. In *26th IEEE Intl. Real-Time Systems Symposium* (Washington, DC, USA, 2005), pp. 133–142.

[104] Wang, X., Lu, C., and Koutsoukos, X. Enhancing the robustness of distributed real-time middleware via end-to-end utilization control. In *26th IEEE Intl. Real-Time Systems Symposium* (Washington, DC, USA, 2005), pp. 189–199.

[105] Wang, Y., Wang, X., Chen, M., and Zhu, X. Power-efficient response time guarantees for virtualized enterprise servers. In *Proceedings of the 29th IEEE Real-Time Systems Symposium* (Barcelona, Spain, December 2008), pp. 303–312.

[106] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced CPU energy. In *First Symposium on Operating System Design and Implementation (OSDI'94)* (Monterey, California, November 1994), USENIX Association, pp. 13–23.

[107] Wright, J. Blades have the edge. *IEEE Spectrum 42*, 4 (Abril 2005), 24–29.

[108] Xu, R. About inefficient frequencies. Tech. rep., Computer Science Department, University of Pittsburgh, 2005.

[109] Xu, R., Zhu, D., Rusu, C., Melhem, R., and Mossé, D. Energy-efficient policies for embedded clusters. *SIGPLAN Notices 40*, 7 (2005), 1–10.

[110] Yao, F., Demers, A., and Shenker, S. A scheduling model for reduced cpu energy. In *36th Annual Symposium on Foundations of Computer Science (FOCS'95)* (Washington, DC, USA, 1995), IEEE Computer Society, pp. 374–382.

[111] Yuan, W., and Nahrstedt, K. Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *19th ACM symposium on Operating systems principles* (Bolton Landing, NY, USA, 2003), pp. 149–163.

[112] Zhu, D., Melhem, R. G., and Childers, B. R. Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. *Trans.on Parallel & Distributed System 14*, 7 (2003), 686–700.