DYNAMIC RESOURCE ALLOCATION FOR ENERGY MANAGEMENT IN DATA CENTERS

A Dissertation

by

CESAR AUGUSTO RINCON MATEUS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Industrial Engineering

DYNAMIC RESOURCE ALLOCATION FOR ENERGY MANAGEMENT IN DATA CENTERS

A Dissertation

by

CESAR AUGUSTO RINCON MATEUS

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Industrial Engineering

ABSTRACT

Dynamic Resource Allocation for

Energy Management in Data Centers. (December 2008)

Cesar Augusto Rincon Mateus , B.Sc. Mathematics, Universidad de los Andes;

Ms.Sc. Mathematics, Universidad de los Andes

Chair of Advisory Committee: Dr. Natarajan Gautam

In this dissertation we study the problem of allocating computational resources and managing applications in a data center to serve incoming requests in such a way that the energy usage, reliability and quality of service considerations are balanced. The problem is motivated by the growing energy consumption by data centers in the world and their overall inefficiency. This work is focused on designing flexible and robust strategies to manage the resources in such a way that the system is able to meet the service agreements even when the load conditions change. As a first step, we study the control of a Markovian queueing system with controllable number of servers and service rates ($M/M_t/k_t$) to minimize effort and holding costs. We present structural properties of the optimal policy and suggest an algorithm to find good performance policies even for large cases. Then we present a reactive/proactive approach, and a tailor-made wavelet-based forecasting procedure to determine the resource allocation in a single application setting; the method is tested by simulation with real web traces. The main feature of this method is its robustness and flexibility to meet QoS goals even when the traffic behavior changes. The system was tested by simulating a system with a time service factor QoS agreement. Finally, we consider the multi-application setting and develop a novel load consolidation strategy (of combining applications that are traditionally hosted on different servers) to reduce the server-load variability and the number of booting cycles in order to obtain a better capacity allocation.

To Lilia (Mom), Lilia (Grandma), Vivi and Caro.

Gracias

ACKNOWLEDGMENTS

Going through a Ph.D. program is a difficult task in itself, but it would be almost impossible without the wonderful people that is there to provide personal and academic support. In first place I want to thank my wife and family for their love, support and patience during these years. I am immensely thankful with my academic adviser, Dr. Gautam, for giving me the opportunity to work with him, and for teaching me so much, not only scientifically, but personally. I would not be here without the help of Dr. Curry, who gave me wonderful advice and dependable support even in the most difficult times, and without Dr. Alberto Garcia-Diaz, who introduced me to the Operations Research discipline. My committee (Dr. Gautam, Dr. Curry, Dr. Feldman and Dr. Reddy) did a great job by helping me with valuable ideas and providing feedback and references. I would like to also thank Dr. Ding for the valuable discussions on the forecasting methodology. An extremely helpful person to me and all graduate students is Judy Meeks; without her help, the life of the graduate students in the department would be very difficult. A special recognition to her. Finally, I want to thank my amazing fellow graduate students and friends for being there to celebrate the good times and palliate the bad ones; especially Alberto Mariscal and Monica Ramos, Gabriel Tucci, Piyush Goel, Hoda Parvin, Ezgi Çan Eren, Julian Gallego and Eduardo Perez.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The developments of the computing and networking technologies during the last three decades have triggered a large and increasing desire for obtaining, processing and sharing data. Nowadays, it is customary to rely on computer systems and the internet for business and entertainment. At the same time, the availability of new applications and faster and more compact equipment have created new opportunities and challenges. For instance, large and successful corporations such as Yahoo and Google are based on business models entirely devoted to collect, host and organize data; and traditional industries such as retail and transportation rely on data analysis, e-commerce and web advertising to improve their operations. However, providing the enormous amount of electric energy to power the hardware and auxiliary equipment needed to host the increasing number of applications and the growing demand is becoming a challenge from the economic, environmental and energy policy (Koomey [1]) perspective.

Due to economies of scale and management complexities, many firms have outsourced the hosting of their internet and data processing servers to specialized companies that own and manage large data centers or ”server farms” (as the one shown in Figure 1). Today’s data center industry is the result of more than two decades of increasing computational resources use, and outsourcing strategies intended to reduce logistics and personnel costs and to increase efficiency. The existence of large facilities with numerous and relatively affordable resources has created an unprecedented demand for computing services, such as in-demand computing and web hosting due to their convenience and resulting competitive advantage. The huge amount of computing power condensed inside a modern data center

The journal model is *IEEE Transactions on Automatic Control.*

facility consumes a large amount of energy used not only for powering the servers, but also to cool down the facilities in order to prevent hardware damages.

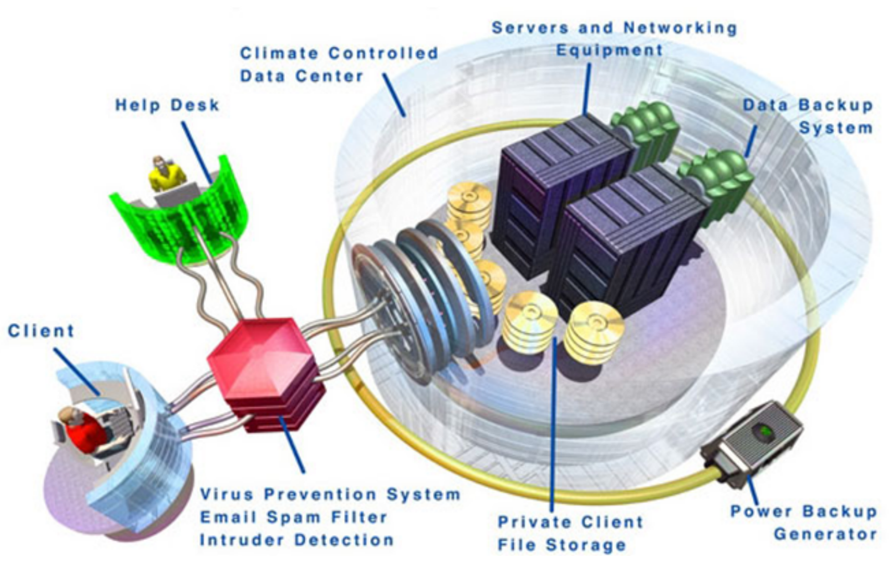

Fig. 1. A typical data center.

According to Chase et.al [2], Lefurgy et.al. [3] and Kumar [4], data centers can consume several megawatts for powering the equipment and cooling systems that protect the machines (Patel et.al. [5]). This has created a serious concern among interest groups and the companies that manage such facilities regarding the industry's impact on the environment, and the national electricity generation capacity in the USA. In fact, information technology electricity consumption is almost 0.5% of world production, and global IT electricity consumption generates more carbon emissions than entire countries such as Argentina or the Netherlands (See Garnet Report [6]). Thus, reducing energy consumption is a high priority in the data center industry, not only because high energy expenditures reduce their profitability, but for the public relations concerns that appear as a consequence of their environmental impact.

Contracts between a web-hosting services provider and its clients are usually based on a Service Level Agreement (SLA) that specifies a range of acceptable values of a quality of service (QoS). Higher QoS requirements produce higher revenues, so it is crucial for the data center administration to be able to provide enough computing capacity to meet the different contractual requirements. In many cases, traffic load received by the data center is not only irregular but also very sensitive to external environmental conditions. Because of those traffic characteristics and the imperious need of meeting the SLA, most data centers over-provision computing capacity to ensure providing an acceptable service even to the detriment of the facility's energy efficiency. This practice has a negative impact on energy usage and profitability, and produces a negative environmental impact. Furthermore, excessive capacity allocation, the high cost of real estate and the requirements of empty space needed for cooling some of the most powerful servers in the market can limit the growth of the data center industry (Kumar [4]).

Given the importance of data processing on today's economy and the increasing cost of energy, there has been some recent interest in improving the energy efficiency of the data center industry. Several studies claim that data center industry use of energy, hardware resources and physical space is quite inefficient. For instance, computational capacity and facilities are underutilized, and cooling systems are inefficient (Kumar [4]). Energy efficiency improvements can be obtained by means of strategic (building and design), tactical (hardware selection and assignments of tasks to servers) or operational (real time task management) decisions. In this dissertation we focus on the real time operational and some tactical decision-making processes. In other words, we propose to study the problem of operating a data center in the most efficient manner given that the facility, auxiliary systems and hardware are already in place. We refer to this problem as the Dynamic Resource Allocation for Energy Management in data centers Problem.

During the past decade, several solutions to this problem have been proposed. However, most practitioners have been reluctant to implement them, fearing that reducing the capacity may increase the risk of not meeting the SLA. In order to create a strategy that is more likely to be used in practice, we should provide an allocation scheme that is sufficiently simple to implement, effective enough to justify its costs, and flexible enough to adapt to load condition changes.

Besides resource allocation, other factors that may reduce energy consumption in data centers are: efficient cooling equipment, well designed facilities and the location of computer farms in regions with cool year-round temperatures. Additionally, hardware manufacturing companies have been developing web servers that are more energy efficient and more compact than their predecessors. However, even with efficient equipment and facilities, a scientific strategy to allocate the existing resources may save a significant amount of energy and money by reducing the amount of unused capacity.

In this dissertation we will present robust techniques based on dynamic programming, queueing theory, forecasting methods and variability reduction to improve energy efficiency and increase data centers' profitability without significantly reducing the quality of service. The proposed dynamic allocation techniques are suited to working with the current hardware and will have a significant potential to reduce the amount of energy used by the data center industry.

A. Problem Definition

In this dissertation, we consider a data center as the one in Figure 1, equipped with a fixed number of identical servers. Modern servers allow the user to select the processor service speed from a finite set of frequencies by using a dynamic voltage scaling (DVS) mechanism. Each server is used to process one or more types of requests, and it is assumed to be DVS capable. For this study, we assume that each request needs to visit only one machine to be served. Since the data center resources are fixed and the arrival process can not be controlled, we focus our attention in managing the data center resources in the most efficient manner by dynamically changing the server processor's frequency and the number of active servers.

The data center receives processing requests from the outside world, and it should process them according to a previously determined Serviced Level Agreement (SLA) and in the most economical way. When a request arrives at the data center, it goes to a free server if there is one available, but if all servers are busy, the incoming request is queued.

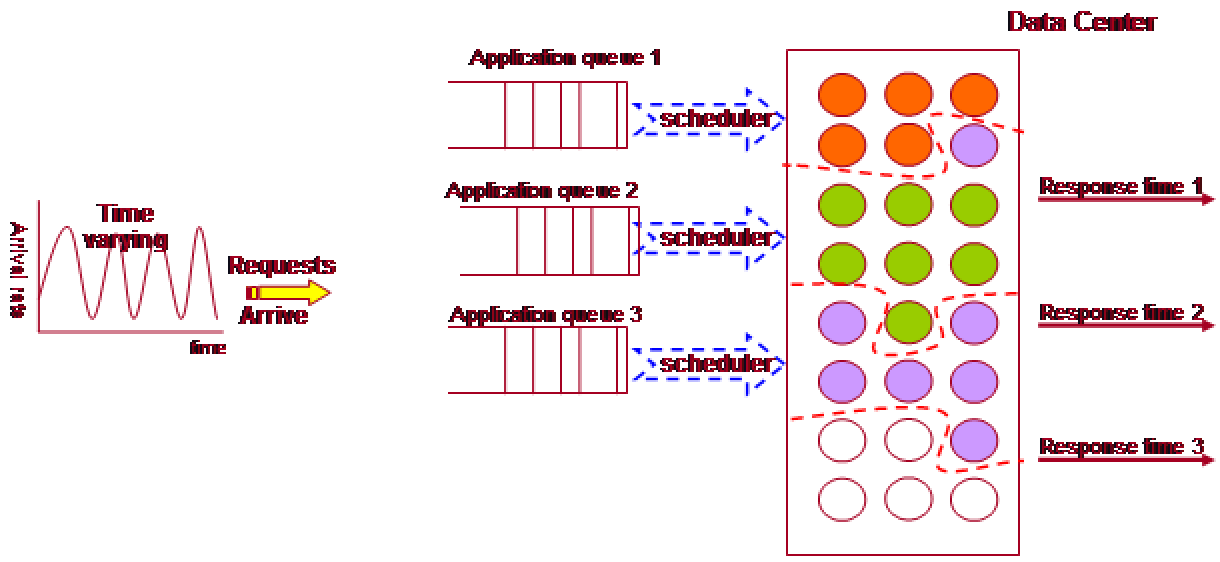

Fig. 2. Data center model.

We model the data center as the queueing system in Figure 2. We denote the total fixed number of servers by K, the number of request classes by M, the buffer capacity by

N and the set of possible processor frequencies by Γ. Both the arrival times and the sizes of the requests are unknown and are modeled as stochastic point processes. The processing time of the requests is proportional to the size of the file measured in bytes and inversely proportional to the server frequency. Additionally, we assume that a server cannot process more than one request at a time (however, the methodology would not be significantly affected if other work conserving disciplines such as processor sharing are used).

We will divide the overall analysis of the data center Operation Energy Management problem into three stages. Each stage emphasizes a different aspect of the allocation process, and the proposed solution of each of them is used in the analysis of the subsequent more general stage. First, we will study a theoretical problem to understand the control policy, thereby using the lessons in the more practical case using real traces. We consider a queueing control problem extension of the one found in George and Harrison [7]. This model represents a single request class data center with constant arrival rate under the assumption of exponential arrival and service times (see Figure 3). In this stage, we look for strategies to minimize the operational cost by adjusting the service rate and the number of servers. Every time a request arrives or leaves the system, the system manager can change the server frequencies and/or turn a new server on/off. Although the assumption of exponential arrival and service times is an approximation of the real distribution, its mathematical tractability allows us to obtain an elegant policy structure. Secondly, we are going to present a methodology suitable to be used with actual internet arrival information. We will propose a method to allocate capacity in a facility running each application independently. This method includes a problem-specific forecasting procedure, and a capacity selection rule that is adaptable enough to meet the QoS requirements even if the system behavior changes. Finally, we eliminate the assumption of requiring each application to run in different machines, and present a technique to exploit the economies of scale that arise due to the interaction between multiple request classes. This technique is based on selecting which groups of applications should be allocated to the same servers.

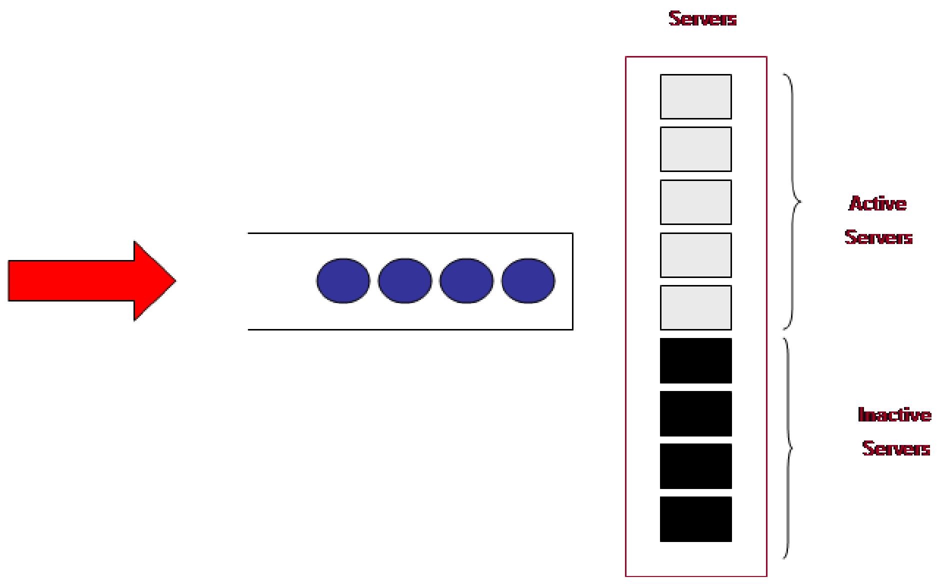

Fig. 3. Queueing system.

For the remainder of this document, we will refer to the three stages listed above as the *Queueing Control*, *Independent Provisioning* and *Consolidated Provisioning* sub-problems respectively.

B. Objectives and Expected Significance

This dissertation has the following objectives:

1. To propose simple and efficient dynamic capacity allocation policies for controlling multi-server Markovian queueing systems where the cost structure is non-submodular (see Figure 3). This is a mathematical simplification of the energy aware data flow

control in the data centers problem in Chen et al. [8]. This allows us to find efficient policy structures for the Markovian case that may be adapted and implemented into more realistic settings. These results are interesting from a pure queueing theory perspective.

2. To find suitable methods to forecast the traffic stream of arrivals to a data center in such a way that the resource allocation may be done more efficiently. The development of a quality forecasting method will allow us to allocate resources more efficiently and to reduce costs.

3. To develop suitable techniques to allocate capacity to single applications in such a way that the QoS requirements consigned in the SLA are met.

4. To propose schemes to assign applications to servers in such a way that the energy used by the entire data center and the number of booting cycles are reduced.

1. Importance

Everyday a large number of companies rely on the internet to advertise and sell products or services, customer service support and many other forms of interaction with the general public and employees; also, several industries rely on data processing and analysis to improve their business processes. As with other products and services, many firms contract specialized companies for managing and supporting a hardware and software structure that allows them to use the web properly; therefore, any operating cost reduction methodology is not only key for the success of the data managing companies, but also generates savings that may be passed to the overall national economy.

Generally, the contracts between a service provider and its clients include a service level agreement (SLA). Those agreements specify service quality and determine the revenue a web hosting provider will receive. In order to meet the contractual agreements most data center have over provisioned their computing capacity for all the running applications, so the SLA is met even if the load conditions change. While this strategy most likely would guarantee the desired quality of service, it is clearly inefficient in both energy usage and overall operating cost. However, data center managers will implement scientific allocation techniques only if these are able to perform acceptably even if the system conditions change rapidly. Therefore, it is very important that managers can be confident about continuing providing acceptable QoS in a dynamic load environment.

High energy consumption by the data centers also produces a negative impact on the environment. The recent growing interest for reducing contaminating carbon emissions highlights the importance of finding ways to reduce electricity expenditures without a significant reduction in the computing and data-processing capacity.

In fact, it has been estimated that energy consumption by data centers has doubled between 2000 and 2006, and today's IT energy consumption is around 0.5% of the worlwide energy production. Moreover, IT industry carbon emissions are comparable to those of Argentina and the Netherlands, and the United States will need 10 more electricity plants by 2010 if the industry sustains its rapid growth (See Garnet Report [6]).

Many SLA in the practice are based on bounding the fraction of requests that spend more than a certain amount of time in the system. This QoS constraint differs greatly from the classical average-time in the system constraint. This is one of the first studies to consider capacity allocation with this type of constraint.

Given that service demand is not constant, dynamic capacity provisioning, if done correctly, can reduce the amount of energy consumed with negligible lost of performance because the system reacts to changes in the load. In other words, provisioning capacity in such a way that the agreement will be met using less resources, when considering both energy and hardware costs, may save money and increase the profitability of the data center industry.

2. Challenges

The problem of providing capacity dynamically for a data center presents a set of challenges inherent to the nature of the data traffic, the information technology industry and the computational equipment. Data traffic over the internet is widely recognized as of fractal nature and is rapidly changing. Provisioning to serve this type of traffic should be done in such a way that the system is able to react to such sudden traffic variations. Additionally, from the business point of view, any provisioning strategy should not jeopardize achieving the contracted QoS requirements, because otherwise the energy savings may be overshadowed by loss of goodwill costs. Finally, from the practical point of view, any policy that mandates turning servers off should take into account the wear and tear on the data center hardware, and the energy and time consumed by this operation. Besides using time and energy, the routing tables would have to be updated, causing an even larger disruption to the system and running the risk of an unsuccessful boot. Furthermore, those costs and risks depend directly on the hardware technology the data center is using, as it may be possible that equipment in the future will have faster and possibly more reliable booting processes. In the following subsections, we are going to describe the problems and challenges specific to each one of the subproblems described above.

a. Queueing Control

The data center system may be represented as a multi-server, multi-frequency, queueing system (see Figure 3) where the frequencies and the number of active servers are to be determined, and which incurs in effort, holding and switching costs similar to the ones in Lu and Serfozo [9]. Although this problem extends the well known single server with adjustable service rate, and the multiple servers with constant service rate problems, it has not been addressed in the previous literature. It has been shown that for a single server with variable frequency and submodular costs, the optimal policy is monotone hysteretic, but no result is known for general switching costs mainly because non-submodular costs present several technical difficulties. The costs in the data center problem are not submodular, so there is not a known structure for the optimal policy. Additionally, for small cases the computation of the optimal policy can be done by an iterative procedure [10], but for large cases this may be computationally prohibitive. Since large data centers can contain thousands of computers, computing such a policy by iterative methods is not practical. We will find simple structure policies that, although possibly suboptimal, perform well in these systems and in many cases may be optimal.

b. Independent Provisioning

Given the self-similar characteristics of the internet traffic (Crovella [11]), forecasting of the load conditions is challenging. The arrival process of the requests to the data center is bursty and time varying. The distribution of the inter-arrival times and request sizes is heavy-tailed and has a high autocorrelation. These characteristics make capacity provisioning a difficult task because a sudden burst in the arrival traffic may cause a long delay for a large set of requests. To be able to make good proactive decisions, we need a forecaster that performs well in process of these characteristics and which provides us with high quality data to feed the allocation models. Among the several techniques of smoothing and forecasting, see for instance Basu [12] and Wang [13], it is necessary to find if those methodologies are accurate and appropriate for this problem.

Allocating capacity presents a challenge because the information on the system is not always complete. Details about the requests in the buffer are usually unknown until the requests arrive at a server. As a consequence, the scheduler only can observe the number of requests in the buffer, so it is not easy to estimate bounds to the approximate serving time of the requests in the queue.

c. Consolidated Provisioning

As the arrival rate per unit time is not constant, and different applications can have different arrival peak times, the data center may realize savings by running a set of applications in the same server in such a way that having to turn off that machine is less likely. Doing so may reduce the booting energy and hardware costs, and reliability risks, so as to improve the overall operational profit. Here we consider not only the decision as to how many servers we assign to each application, but also which applications should run on each server. Making such assignments effectively requires information about the arrival process and the interaction between them. Moreover, those assignments can be dynamically changed, but the changes may as well be costly and risky, so we will keep those assignments unchanged for considerable periods of time.

Although the problem of assigning tasks to servers has been studied in the scheduling literature, the Dynamic Capacity Allocation in data center problem presents some new challenges. First, we have a great uncertainty about the request arriving process. Secondly, we are charged a penalty cost for shutting down a server, but there is not a set up cost for changing the class of the request served by it, as long as that application was installed beforehand. As we are going to describe in Chapter V, this new problem structure has not been studied in the literature but presents some technical characteristics similar to the ones found in the portfolio optimization literature.

C. Organization of the Dissertation

This dissertation is composed of five chapters. Chapter II contains a concise review of the literature in the energy aware capacity allocation problem as well as the three sub-problems we are going to consider; we have also included a short State of the Practice section in which we describe the way most contemporary data centers allocate operational resources. Chapter III deals with the Queueing Theory portion of the dissertation; it includes theoretical results and numerical experiments. Chapter IV is about the Individual Allocation problem, and presents the forecasting and allocations techniques developed to serve independent applications, and experiments that use real internet traces. Chapter V contains details about the Consolidated Allocation sub-problem; it offers details on application load consolidation techniques as well as numerical experiments with simulated arrival data. Chapter VI contains concluding remarks and future research directions.

CHAPTER II

LITERATURE REVIEW AND STATE OF THE PRACTICE

A. Literature Review

Given the popularity of data center utilization and the existence of Service Level Agreements between the data center management and contractors, there have been several studies on QoS aware planning and provisioning (see Zhang [14], Ardagna [15], Shen [16]). Clearly, over-provisioning resources may result in significantly higher costs which will reduce the data center profit and which will have a negative environmental effect because of high energy consumption. However, the system should be able to perform when temporary overload situations occur and still reduce costs by decreasing performance when low loads are in effect. For these reasons dynamic capacity provisioning is necessary and advantageous. Moreover, (see Clark [17]) the arrival rate peaks are relatively frequent, and the ratio between the average load and the peak load may be very high. Several research studies have been performed on handling those peaks while meeting the SLA and maximizing the revenue, but most of them have not addressed the power consumption issue (Chen et al.[8]).

During recent years power management in data centers has gained attention (Carrera [18], Chase et al.[19], Elozahi et al.[20], Lefurgy et al. [3], Pinheiro et al. [21]) due to the large amount of energy consumed by the data center industry as a whole, steadily increasing energy prices and growing environmental awareness (Chase et al. [2]). One of the proposed alternatives was to turn off some of the servers following some predetermined strategy (Chase et al. [2], Pinheiro et al. [21]). Another mechanism for controlling the total energy consumption consists in modulating the CPU frequencies (Elnozahi et al. [20], Sharma et al.[22]) using a dynamic voltage scaling mechanism (DVS). The rationale for this approach is that a slower CPU consumes less energy, and the CPU is the largest energy expenditure of the server (Bohrer [23]).

Although DVS and shutting down servers provide improvements on energy consumption, several of the individual studies have not used both strategies at the same time or have not addressed the costs associated with the wear and tear on hardware because of repeated booting. A recent work by Chen et al. [8] estimates the booting cost and performs both a queueing and control analysis for multiple servers and multiple applications settings.

As seen in Chen et al. [8] with their hybrid scheme, the combination of DVS and shutting down servers produces opportunities for significant savings. On this work, they use S-ARMA prediction method and mention that it may be significantly improved. It is worth noting that queueing systems representing a data center with the mentioned energy savings mechanisms have not been studied in the queueing theory literature, and there is no a study on allocating applications to servers in such a way that it might represent economies in the system operation with the costs described above.

Uniprocessor Systems: Related dynamic resource allocation for energy efficiency has been done in the context of individual machines. For example, mobile devices, such as laptops or satellites, may become more energy efficient and have longer battery life if the performance of some of the components, such as the disk (Hembold et al. [24]), is reduced when there is no user activity or the current tasks do not require much processing. Other critical equipment, such as sensors or satellites, also require the implementation of energy management mechanisms to be able to complete longer missions. Several works have been centered on the development of software solutions for reducing power consumption of uniprocessor systems (See for instance Aydin et al. [25], Chandrakasan et al. [26] and Hsu et al. [27], Gurumurthi et al. [28], Lorch and Smith [29] and Yao et al. [30]).

Parallel Processors Systems: Similarly, parallel processing systems, such as multiple processor machines, may achieve significant energy efficiency by dynamically varying the processors speed and/or making them active or inactive (See Zhu et al. [31] and Merkel and Bellosa [32]). This type of power management may be done in distributed systems (See Mishra et al. 2003 [33]) as well as in individual chips with multiple processors (Juang et al. [34]). Those systems present similar characteristics to the data center models presented in this dissertation, but the negligible cost and time of activating a processor presents an important difference with the data center problem. Gruian [35] proposed two system level designs for architectures with variable voltage processors usable when the applications are fixed and the execution time is predictable. Yang et al. [36] proposed a two-phase scheme that minimizes electricity consumption under time constraints. Zhu et al. [31] used the concept of slack sharing to propose two algorithms that reclaim the time unused by other tasks to reduce the execution time of future tasks. Mishra et al. build schedules to process real time tasks with precedence constraints and propose static and dynamic power management schemes [33].

Bandwidth Allocation: Another IT area where dynamic provisioning is widely used is the bandwidth allocation problem on network switches. The objective is to provide bandwidth to different applications with different QoS constraints (See McGarry et al. [37]). According to McGarry et al. [37], traffic may be divided between bandwidth guaranteed traffic (such as multimedia) and best effort traffic, and the problem consists of allocating the amount of bandwidth dynamically to satisfy the service constraints. For instance Guerini at al. [38] and Assi et al. [39] present allcoation methods to satisfy QoS constraints, and Qiu et al. [40] present a price based approach to allocate the bandwidth to different applications. This problem differs from the data center energy problem in the absence of switching costs and times, as well as in the type of QoS constraints considered. Moreover, the problem setting generally did not require minimizing the total amount of bandwidth allocated.

1. Queueing Control

Queueing design and control has been extensively studied in the literature during several decades of the twentieth century and it is still active nowadays. The problem type consists of minimizing the costs incurred by a queueing system (or maximizing its profits) by choosing a given set of parameters either statically or dynamically. The static problems are known as design optimization, whereas the models where the parameters can be changed over time are referred to as dynamic control models. The problem studied in Chapter III is an example of a dynamic control model. Queue behavior may be manipulated, for instance, by admission control, service process control and control of the queue discipline (Crabill et al. [41]).

In Chapter III, we center our attention on the service process control that includes varying the number of servers and varying the service rate in order to minimize the operational cost of a system. The existing queueing literature contains several articles on two special cases of the system we are studying. Namely, several authors performed research on finding efficient strategies to vary the number of servers as it is found in the *call center* problem literature that has obvious applications in practice. In addition, a significant amount of research has provided results on the single server with variable rate problem, perhaps because of its mathematical tractability under the assumptions of exponential service and inter-arrival times. However, to our knowledge, no work has so far been done on the control of systems where both the rate of service and the number of servers can be chosen dynamically maybe because of the combined mathematical complexity and the lack of potential applications.

Single Server Queues: A compilation of relevant works before 1977 on single server queues can be found in (Crabill et al. [41]). Relevant recent literature includes Stidham [42], George and Harrison [7], Lu and Serfozo [9], and Kitaev and Serfozo [43]. George and Harrison [7] proposed a very efficient method for computing the optimal policy for an $M/M_t/1$ queue with adjustable service rate and infinite queue capacity, and Stidham [44] established that optimal policy for such queueing systems is monotone and threshold type when there are no switching costs. For the optimal control of $M/M_t/1$ queues with switching costs, it has been shown in Lu and Serfozo [9] and Kitaev and Serfozo [45] that the optimal policy is hysteretic monotone if the operational costs and switching costs are submodular.

Call Center Problem: The call center literature is rich and extensive. This literature studies the special case of our problem where the number of servers is controlled and no switching costs are incurred but the service rate is fixed. Some general surveys on the matter are Gans et al. [46], and Koole and Mandelbaum [47]. Some papers consider more complicated queueing disciplines involving inpatient customers and customer's observable information has been published as well (see Garnet et al. [48]). In addition, queueing control in multi-server queues have been studied in Crabill et al. [41] and Winston [49].

2. Independent Provisioning

There has been a considerable amount of work in resource provisioning in data centers. Several papers focus on either resource provisioning to meet stringent service level agreements or energy management but not both. Works of the first type are, for instance Appleby et al. [50], Ranjan et al. [51], Zhang et al. [52] and Zhou et al. [53]. Some of the works focused mostly in energy management are Pinheiro et al. [21], Sharma et al. [22], Elnozahi et al. [54] and Bianchini and Rajamony [55].

There are some research works that incorporate the Service Level Agreement constraints into the energy efficiency driven resource provisioning. Chase et al. [2], considers a data center that handles multiple applications with multiple servers and use turning on and off servers as the resource provisioning tool; this research work, acknowledges the fact that rebooting a computer requires a non-negligible amount of time, but it does not associate an energy or monetary cost to this operation. In Chen et al. [8], DVS, and turning servers on and off are used to handle a system with multiple applications, booting costs and with service level requirements in terms of the average response time. This work uses three different approaches: first, it presents a proactive approach to the decision-making based on steady state behavior (that will not be always reached in fine time granularities); second, it presents a reactive feedback control algorithm; and third, it proposes a hybrid approach that uses forecasting to determine the number of servers and the reactive decision-making process to decide the frequencies. Wang et.al [56] proposes a control methodology to manage the power consumption under QoS constraint of the maximum response time. Xue et al. [57] present an adaptive pool-based resource management mechanism to provide capacity on demand, and validate their results using simulated data with exponential inter-arrival times. Bertini et al. [58] measures the quality of service as the average ratio between the actual response time and the response time deadline, and uses a reactive control mechanism to make capacity provisioning decisions. Khargharia et al. [59] proposes the use of control methodologies on different components to reduce the power consumption such as the processor, memory and I/O. Our work in Chapter IV differs from previous papers in the type of SLA considered and the combination of proactive and reactive mechanisms at the same time.

Wavelet Based Forecasting: Forecasting data traffic has been regarded as a challenging task by time series researchers. Ethernet traffic was recognized to be *self-similar* and exhibit *fractal-type* behavior (Leland et al. [60], and Crovella and Bestavros [11]), which means that it is very bursty and does not becomes smoother quickly with the aggregation of sources and/or change of scales. Given the similar fractal-like characteristics of the wavelet decomposition, wavelet based tools have been proposed to estimate different parameter of self-similar time series (Abry and Veitch [61], Aussem and Murtagh [62] and Fryzlewicz et al. [63]). A good compilation on wavelet based time series forecasting may be found in Percival and Walden [64] and Renaud et al. [65]. In Chapter IV we will use wavelet mechanisms for the demand forecasting.

Wavelet-based Anomaly Detection: Wavelet applications to internet related time series go beyond forecasting. (Barrford et al. [66]). Standard signal processing data, such as wavelet filters, provide means to identify the presence of anomalies and their type; for instance, Barford et al. [67] present a signal analysis of four classes of network anomalies. Kwon et al. [68] present wavelet based methods to detect network anomalies effectively based on change point detection and test the procedure with simulated attack data, and Kim and Reddy [69] proposed a traffic anomaly detector that monitor packet headers and is based on IP address correlation.

3. Consolidated Provisioning

The allocation of tasks to servers has received interest in the operations research literature. For instance, in manufacturing, given a deterministic arrival stream of jobs, the problem of assigning those jobs to operators or machines have been extensively studied using integer (Pinedo [70]) and dynamic programming techniques (Bellman [71]). This problem has been studied in the context of real time scheduling and considers the possibility of unreliable machines (Maimon and Gershwin [72]). However, the kind of jobs each operator can perform is given in the problem and the decision maker can not modify that. In contrast, is the data center setting, the manager can install applications on the different servers.

In the queueing literature (Gomoluch and Schroeder [73]), similar problems have been discussed from the perspective of manipulating queue discipline and the assignment of jobs to specialized or well trained servers (see for instance Becker et al. [74]). Those studies have been centered in keeping the system balanced in order to not overflow any particular set of servers (Gardner and Liu [75]) and the issue of fairness (Park et al. [76]). However, it is worth noting that the concept of load balance and fairness should be treated differently in the data center context, because the concern of an unbalanced work load is valid only when it produces significant wear on the servers or disturbs the work environment. Clearly, because of the nature of the equipment, this is not the case. In our context, we are interested in load balancing only if it yields to savings. Moreover, most of the research studies in the area deal with homogeneous arrival processes.

Variance Reduction in Portfolio Analysis: We will allocate the tasks in such a way that the aggregated number of servers that should be shut down is reduced. In other words, besides avoiding a system overflow, we want to minimize the number of large changes on the aggregated arrival flows to each server to avoid paying the booting cost B_0. The problem may be described as allocating a finite amount of resources to a set of tasks. We want the total load of the classes allocated to a server to exhibit very little variance. Analogous work is done in the portfolio management literature when the concept of negative historic covariance is used to minimize the risks of large capital loss (see for instance Levy and Sarnat [77] and Markowitz [78]). There has been an immense amount of work on portfolio optimization since then. A different type of analysis is done in the so called portfolio analysis and market taxonomy. In this context, the correlation coefficient of all possible pairs of stocks is computed, and graph techniques such as minimum spanning trees are used to find strong positive correlation in stocks, and to partition the market into sectors (See Onnella [79]). Those techniques allow analysts to determine if a portfolio is well diversified and to detect strong market behavior disruptions such as recessions or crashes. Our task differs from the previously mentioned problems in two ways. First, we cannot choose to serve only a fraction of the requests, nor can we create more requests. Secondly, we do not consider the return of investment, but we are interested solely in the variability reduction.

B. State of the Practice

The number of data centers in the world has grown immensely in recent times. Energy consumption by data centers has doubled between 2000 and 2006, and today's IT energy consumption is around 0.5% of the worldwide energy production. However, not only the number, but the computational capacity of the data centers has been increasing. A modern average data center consumes the energy equivalent to the consumption of $25,000$ households. If this rapid growth continues, the United States will require between now and 2012 the equivalent of 10 additional power plants (Gartner Report [6]).

This huge energy consumption has a negative economic impact as well. Data center building and operation is a quarter of the total expenditures on today's IT, and the true costs of operating a server over 10 years is 4 to 5 times the cost of the hardware (Uptime Institute [80]).

Problems created by this rapid increase of the number of server farms include growing capital and operation spending, and a significant amount of greenhouse emissions. Moreover, all those energy resources are not being used efficiently. Several sources have established that the available computing power and sophisticated facilities are being severely underutilized. In particular, the average utilization of the servers is about 6%, and the facility utilization is about 56% (Gartner Report [6]). The inefficient resource usage is caused by poor capacity planning and application design, inefficient cooling systems, inefficient resource allocation and inappropriate hardware (Kumar [4]).

Facility underutilization is a consequence of poor capacity planning. It wastes real estate and induces unnecessary work on the cooling infrastructure. Low quality computer code produces software applications that should perform excessive computations to perform the desired tasks; these applications introduce non-essential load into the computer systems. Inefficient cooling systems require more energy than properly designed ones to achieve comparable across-facility temperatures. Finally, poor capacity provisioning makes a large number of active servers idle or running at higher frequencies than required to meet the performance criteria. The main reason for excessive capacity provisioning is the imperious necessity of meeting the SLA contracted with the data center customers.

As we just mentioned, there are several strategies to build more energy efficient and more economical data centers. However there exist the necessity and the possibility to take action to improve the operations of the facilities already in place. Implementing those techniques appears to be a viable solution to reduce the energy consumption in the IT industry in the short term.

CHAPTER III

QUEUEING CONTROL

Queueing control has been applied to a diverse number of real-life problems, such as optimizing manufacturing systems, call centers, banking operations, transportation, healthcare and computer networks. The general problem consists of making decisions regarding a set of eligible variables in order to minimize (maximize) the operational costs (rewards) under some pre-determined arrival process of the jobs to be processed. Different versions of this problem may be characterized by a particular choice of the number of servers, the arrival process, the service process, the factors under managers' controls and the costs incurred.

In this chapter, we study a generalization of two classical problem settings in the Markovian queueing control literature: unique server with adjustable service rate ($M/M_t/1$ where μ_t, the service rate is controlled), and the call-center problem with multiple fixed service rate servers that may be turned on or off ($M/M/k_t$ where k_t is controlled). In both cases, the server operation cost is a non-decreasing function of the service rate, and the penalty for having jobs waiting in the system is a holding cost that is a non-decreasing function of the number of jobs in the queue. To the best of our knowledge there are no queueing system studies that combine both the dynamic selection of the number of servers as well as the service rate (i.e. $M/M_t/k_t$ where μ_t, the service time parameter, and k_t are controlled).

Our problem of controlling a queue of jobs serviced by a fixed-size set of homogeneous servers that can run at a discrete set of service rates or that can be turned on and/or be off in order to minimize the operational cost per unit time has been motivated by applications in computer systems. The system incurs operating and holding costs (corresponding to energy and performance costs respectively) analogous to the ones in the previous queueing literature (see for example George and Harrison [7]), but an additional instantaneous cost is charged every time a server is activated corresponding to energy consumption and reliability costs. The study of those systems is rejuvenated by the recent interest in solving control problems in data center operations in which the servers can be both turned off and slowed down by means of *Dynamic Voltage Scaling* (DVS see Chen et al. [8]) to increase energy efficiency and reduce the operating costs of such facilities. If we assume that a predetermined set of servers is devoted to a fixed request type, we can construct a simplified model of a data center. This would be a set of independent single class queues with a FCFS (first come first served) queueing discipline, and homogeneous exponential arrivals being served by up to K servers with exponential service times than can run at variable service rates and can be turned on (incurred a booting cost B_0) or off.

In order to compute the optimal static policy, the problem may be modeled as a semi-Markov decision process (SMDP) to use uniformization and value iteration techniques when explicit model parameters are given and the queueing buffer is finite. However, those techniques are computationally very expensive and can be used only for cases much smaller than the ones found in practice (with several hundreds of servers). In this chapter, we show that for this system the optimal policy is hysteretic and show some structural results. In addition, we use those results to develop a computationally efficient heuristic approach allowing us to find policies that provide significant savings compared to systems with a constant number of servers running at a variable service rate, and a variable number of servers with constant service speed. In other words, we explore the potential savings of this new setting against the call center model as well as the single server with adjustable rate case. For smaller problem instances we compare the heuristic performance with the exact optimal policy, but for larger cases we limit the study to the savings against the traditional approaches.

There is a set of relevant challenging characteristics inherent to the structure of the problem. First, the SMDP has multi-dimensional state and action space. Secondly, since the running costs associated with our motivating problem are not sub-modular (Chen et al. [8]), we do not restrict our attention to the sub-modular case as in Lu and Serfozo [9], but we have to consider the more general case of non-decreasing holding and service costs. Third, the problem involves some non-linear and integer functions to optimize, and the realistic problem sizes are quite large (100 or more servers).

A. Problem Description

The queueing control problem has applications across several industries: banking services, call centers, manufacturing and port operations to name a few. However, this research has been motivated by an application in computer systems, namely, data centers. The general problem consists in deciding the value of a set of parameters (i.e. control variables) according to the state of the system, described by the state variables, in order to minimize the expected operational cost under some predetermined characteristics of the arrival process and the service times. Depending on the model, the costs, state and control variables are selected. In this chapter, we study a system with a set of K identical servers with variable service rate and the ability to be turned on (incurring a cost B_0) and off. We consider exponentially distributed inter-arrival and service times, and assume that the administrator, via software, has control over the service rates and the number of active servers.

The main challenges, in addition to those described in section 1, are the non-submodularity of the cost structure, the presence of an instantaneous cost of activating a server and the interaction between the chosen frequency and the number of servers. It is worth noting that the problem is multi-dimensional in both, the action and state spaces.

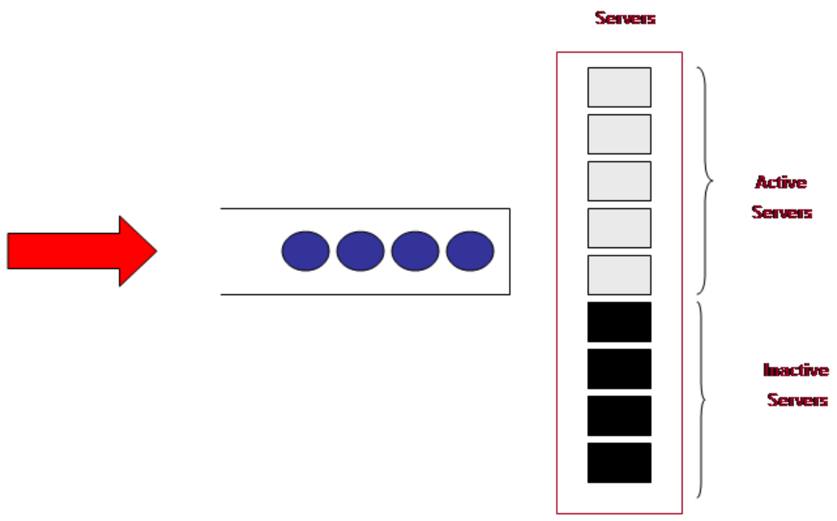

Fig. 4. Queueing system model.

1. Problem Description

Consider an $M/M_t/k_t$ queueing system where jobs arrive at a fixed arrival rate λ per unit time, and at time t there are k_t, $0 \leq k_y \leq K$, active servers, each running at rate μ_t. The values k_t and μ_t are under our control. Let $n(t)$ be the number of jobs in the system at time t. The main objective of this chapter is to develop a control policy to optimally select μ_t^* and k_t^* based on the state of the system an instant before, $(n(t-), \mu_t -, k_{t-})$, such that the average cost per unit time is minimized.

We assume that for all t, $\mu_t \in \Gamma = \{\gamma_1, \gamma_2, ..., \gamma_S\}$, i.e., there are K available servers that can run at a variable frequency, performing at a service rate picked from a finite set. The operating or effort cost function in terms of the number of active servers k and the frequency μ, $c(k,\mu)$ per unit time, is assumed to be non decreasing, but no further assumptions such as convexity are made. The system incurs holding cost $h(n)$ per unit time when there are n

jobs in the system. The function $h(n)$ is assumed to be non-decreasing. Moreover, there is a fixed instantaneous cost B_0 every time a server is turned on, and the decision of turning a server on, leaving as it is, turning it off is represented by an integer $b_0 \in \{-1,0,1\}$. Since inter-arrival and service times are assumed to be exponential, we can assume that the decisions about the control variables are going to be made only whenever a job arrives or leaves the system. In other words, by using the memoryless properties of the exponential distribution, we modify the continuous control problem into a discrete control problem with actions taken only during system state changes.

We assume that a period starts every time a job arrives or leaves the system; therefore, at each period the system state is constant. Thus, whenever a period starts if the service rate is γ_i, the number of servers is k, the number of jobs in the system is n and the booting decision is b_0, the incurred costs are given by:

$$r(i,n,b_0) = (c(k,\gamma_i) + h(n))\,x + B_0(b_0^+)$$

where x is the period length.

For our numerical experiments, we are going to use

$$c(k,\gamma_i) = k(F_{fix} + F_{var}\gamma_i^3) \tag{3.1}$$

where F_{fix} and F_{var} represent the fixed cost and a coefficient for the variable cost. We picked this cost structure following the considerations in Chen et al. [8] regarding data center energy costs. Observe that the energy cost structure implies that increasing the service speed has a tremendous cost on the energy consumption, and there is a fixed cost regardless of the server frequency. The booting cost B_0 is essentially a measure of both the energy spent in booting as well as the reliability costs, because the lifetime of a server depends on the number of boots. The holding cost $h(n)$ ensures that the jobs receive reasonable quality of service in terms of response time for requests.

B. Problem Formulation

Consider a queueing system where inter-arrival and service times are exponential, the maximum number of jobs in the system is N (the analysis of the special case of $N = \infty$ may be performed by considering large values of N), and the arrival rate is constant. Servers may be turned on and off and operate at a variable service rate. The system state is described by k_t, μ_t and $n(t)$ that represent the number of servers, the individual frequency and the number of jobs in the system at time t. Note that we assume that all servers run at the same frequency μ_t as we will justify in Proposition 1. As mentioned in section 2.1, we can restrict our decision epochs to the times when the system changes its state, and we can define time periods where the service frequency and number of servers are constant.

For a given period, the continuous costs are divided into holding costs per unit time $h(n)$ that is incurred when n jobs are in the system at any given time, and operating costs per unit time $c(k,\mu)$ that represent the effort cost of running the system with k servers at a service rate of μ. We assume that both, $h(n)$, and $c(k,\mu)$ are non-decreasing functions, so there is an extra price for serving the jobs at a faster rate or by having a longer queue. We assume that in particular, $c(k,\mu) = k\,(A + g(\mu))$, i.e., the sum of the fixed cost for operating k servers and a non-decreasing variable cost $g(\mu)$ (a special case was displayed in equation (3.1)). Additionally, switching a server from inactive to active state costs a fixed amount B_0. The decision epochs appear when a job arrives or leaves the system, and at every decision epoch, a frequency rate is chosen and the decision to activate one, deactivate one or keeping the same number of servers is taken.

Since there is not a cost associated with changing the service rate, it does not have to be included in the state space; however, the service rate is a factor in determining the transition probabilities when we model this problem as a SMDP. In our model, the state is described by a two-dimensional state variable $(k,n) \in \mathbb{S} = \{0,1,...,K\} \times \{0,1,...,N\}$ and action space $\mathbb{A} = \Gamma \times \{-1, 0, 1\}$ (see Figure 2) where Γ is defined as in section 2.1. This is a two dimensional decision problem in which both dimensions are potentially very large because as N grows, the optimal policy of the finite system converges to the optimal policy of the infinite buffer size problem.

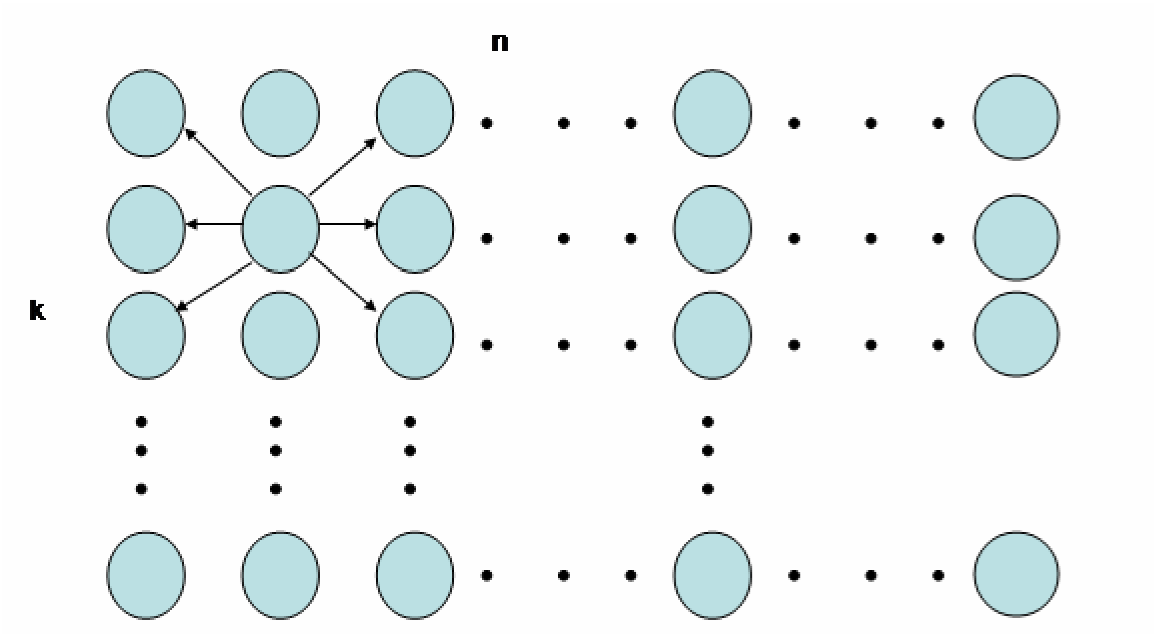

Fig. 5. State space.

Given the nature of the problem, there are two decisions at each epoch. First, we decide whether to turn on or off a server, which will determine a vertical movement on the state space and will limit the set of future states, and secondly, we pick a frequency that will determine the transition rates. A policy is defined by a function $\nu : \mathbb{S} \rightarrow \mathbb{A}$, that given the system state determines the action to take. In other words, for every state (k, n) we want to make a decision $\nu((k, n)) = (\mu, b)$. For a given ν, the associated average expected cost is:

$$\bar{C}(\nu) = \lim_{Z\to\infty} \frac{\sum_{z=1}^{Z}\left(c(k_z,\mu_z)t_z + h(n_z)t_z + B_0(b_{z0}^{+})\right)}{\sum_{z=1}^{Z} t_z},$$

where t_z is the length of the $z-th$ period and b_{z0} and μ_z are booting and frequency actions at the beginning of the $z-th$ period. We say that a stationary policy ν^* is optimal if $\bar{C}(\nu^*) \leq \bar{C}(\nu)$ for any ν in the class of stationary policies.

We denote the probability of going from state (k,n) to state (l,m) if the decision is (k,μ) as $P(n,k,m,l)(k,\mu)$. The policy ν^* is optimal if it satisfies the Bellman equation corresponding to the average cost criterion (see Bertsekas [81]):

$$\nu^*(n,k) = \min_{\Gamma\times\{1,\ldots,K\}} \left\{ B_0(b_{0,nk})^+ - \rho^*\bar{t}(k,\gamma_i) + \sum_{m,l} P(n,k,m,l)(k,\gamma_i)\nu^*(m,l) \right\}.$$

As we are going to explain in section 4.1, we need to find first the optimal policy structure for the discounted criterion case to find the average cost criterion optimal policy as the limit as the discount factor approaches 1. The Bellman equation for the discounted cost requires the definition of the value functions at each state analogous to the ν terms in the previous equation. Following the ideas in Serfozo [82], for the discounted factor β, we can write the value function $V(n,k)$ of the discounted criterion queueing decision process as:

$$V(n,k) = argmin_{(k,\gamma_i)\in\mathbb{A}} \left\{ B_0((b_{0,nk})^+) + \frac{c(k,\gamma_i)+h(i)}{\lambda + K\gamma_{|\Gamma|}} + \beta \sum_{(m,l)\in\mathbb{S}} P(n,k,m,l)(k,\gamma_i)V(m,l) \right\}, \tag{3.2}$$

Remark B.1 *When dealing with a finite buffer, most queuing control papers consider an additional cost for rejecting a job. This cost is incurred when a job arrives to the system and the buffer is full. Since we consider convex holding costs, we can define an adjusted* *holding cost*

$$\tilde{h}(n) = \begin{cases} h(n) & if \quad n < N, \\ h(n) + \lambda a & if \quad n = N, \end{cases} .$$

where a is the rejection cost. Notice that the convex cost structure is not going to be affected.

C. Solution Structure

In this section, we show that the optimal policy for the queueing system described above is hysteretic and we describe a set of special cases in which the solution structure is known. The following is a generalization of the definition of hysteretic policy given in Hipp and Holzbaur [83]:

Definition C.1 *For a Semi-Markov Decision Process with action space $\mathbb{A} = \{1,2,...,K\} \times \Gamma$ and state space $\mathbb{S} = \{1,2,...,K\} \times \{1,2,...,N\}$, a policy f is a hysteretic policy if $f(k,n,\mu_i) = (\bar{k},\bar{\mu})$ for some k,n,i implies $f(\bar{k},n,\bar{\mu}) = (\bar{k},\bar{\mu})$.*

1. Structure of the Optimal Policy

The solution for an SMDP that incurs a switching cost is often a hysteretic policy. The following is the proof of this fact. The strategy is the following: we will first show the result for the discounted cost criterion and then consider the average cost as the limit when the discount criterion tends to 1. Let us define a linear order in the action space using the lexicographic order of the coordinates, i.e.,, $(k_1,\gamma_1) \leq (k_2,\gamma_2)$ if and only if $k_1 < k_2$ or $k_1 = k_2$ and $\gamma_1 \leq \gamma_2$.

Theorem C.2 *If the costs are bounded by a polynomial, the optimal policy for the adjustable number of servers, adjustable frequency system with average reward criterion is hysteretic.*

Proof C.3 *By Lemma 1 (below), we know that the optimal policy in the discounted case is hysteretic. We need to show that the conditions in Theorem C.2 are met. Let $V(\beta, e)$ represent the value function for the discounted factor β and initial state $e = (k, i)$. From the cost structure, it is evident that $V(\beta, k, i) \geq V(\beta, k, i-1)$. Then it is clear that the minimum is achieved at some $e = (k, 0)$. Since the set of such states is finite, there exist a state $e = (k_0, 0)$ and a subsequence $\beta_{n_k} \to 1$ with $V(\beta_{n_k}, k_0, 0) = \min_{i \in E} V(\beta_{n_k}, k, i)$. Now, consider the policy $f(k_1, i) = (K, \mu_{|\Gamma|})$ if $i > 0$, and $f(K, 1) = (k_0, \mu_{|\Gamma|})$. Since we assumed $K \times |\Gamma| < \lambda$ it is clear that the conditions in the theorem are met. Since all the policies in the sequence are hysteretic, the limit policy is hysteretic as well.*

Lemma C.4 *Let $\beta \in (0, 1)$. The optimal policy for the multi-server adjustable frequency system with discounted reward criterion and discounted factor β is hysteretic.*

Proof C.5 *By using standard regularization procedures, see equation (3.2) in section 3, we can write the problem as (see Lu and Serfozo [9]) $v(i, a_1) = min(s(a_1, a_2) + w(s, a_2))$ where the a_j represent an action and s represents the state. Following the argument given in theorem 1 of Hipp and Holzbaur [83]. Let $s(k_1, \mu_1, k_2, \mu_2)$ be the switching cost. Observe that the switching costs of our problem satisfy the inequality $s(k_1, \mu_1, k_3, \mu_3) \leq s(k_1, \mu_1, k_2, \mu_2) + s(k_2, \mu_2, k_3, \mu_3)$. Using the linear order defined above, we can follow the same argument of Hipp and Holzbaur [83] to conclude that the policy is hysteretic.*

The following theorem appears in Weber [84] and Sennott [85] and state conditions under which the optimal policy for the average rewards criterion can be found as a limit of the optimal policies for the discounted cost criterion.

Theorem C.6 *Suppose there exist discount factors $\beta_n \to 1$ and a state $e \in \mathbb{S}$ such that $V(\beta_n, e) = \min_{i \in \mathbb{S}} V(\beta_n, i)$. Also, suppose there exists a stationary policy under which the expected time to go from i to e and the expected cost incurred during this passage are* *both finite for each $i \in S$. Then there exists a subsequence $\beta_{n_k} \to 1$ such that the limits: $g = \lim_{k\to\infty}(1-\beta_{n_k})V(\beta_{n_k}, e)$ $v(i) = \lim_{k\to\infty}[V(\beta_{n_k}, i) - V(\beta_{n_k}, e)]$ exists and satisfies the the optimality inequality: $g + v(i) \geq \min_{a\in A}\left\{C(i,a) + \sum_{i\in E} P_{i,j}(a)v(j)\right\}$.*

Moreover, g is the optimal average cost and any minimizer of the right side determines an average-cost optimal policy. This expression is an equality when the process can immediately move only to finitely many states from each i.

Remark C.7 *Due to the structure of the problem and the numerical experimentation, we believe that the optimal policy is monotone hysteretic to determine the number of servers and monotone threshold type to determine the frequency space (see Figure 3). We have not, however, proved this fact about monotonicity to be true due to the multi-dimensionality of the state and action spaces as well as the cost structure.*

2. Some Structural Results

In this section we want to list a set of results concerning the structure of the optimal policy under some additional assumptions. In particular we consider the case when the set of possible frequencies is continuous and when the booting cost is zero.

One of the assumptions made in this chapter is that all the servers run at the same frequency. The motivation for this assumption is explained next.

Proposition C.8 *Assume the effort cost function that depends on the frequency $g(\mu)$ is an increasing convex function in μ and the set of possible frequencies is connected. Then, it is optimal to run all active servers at the same frequency. In other words, if $\sum_{i=1}^{k}\mu_i = \mu$ then the minimum cost is reached when $\mu_i = \frac{\mu}{k}$ for all i.*

Proof C.9 *In order to obtain a contradiction, assume that there are two indexes i and j such that $\mu_i < \mu_j$. By definition of a convex function $g(x+\Delta x) - g(x)$ is increasing in x;*

therefore, $g(\mu_j) - g\left(\frac{\mu_i+\mu_j}{2}\right) \geq g\left(\frac{\mu_i+\mu_j}{2}\right) - g(\mu_j)$ *and*

$$2g\left(\frac{\mu_i+\mu_j}{2}\right) \leq g(\mu_i) + g(\mu_j).$$

Since the other terms do not change, this new frequency assignment has lower cost. Therefore it is optimum to run all servers at the same frequency.

The opposite result is true when the function is concave. In that case it is better to have servers running at the highest possible frequency even if one server has to run at a different lower pace.

Proposition C.10 *Assume that the cost function* $c(k,\mu)$ *is increasing and convex. If there are no booting costs, i.e.,* $B_0 = 0$*, then for every number of active servers* k*, there are two bounds* $\mu_{low}, \mu_{high} \in [0,\infty]$ *such that if* $\mu \leq \mu_{low}$ *we should turn off a server, and if* $\mu \geq \mu_{high}$ *then the instantaneous cost will be lower if we turn on a server. In particular, for the cost structure in equation (3.1) those bounds can be computed explicitly.*

Proof C.11 *Let us assume that there is some aggregated frequency* μ_0 *such that* $c(k,\mu_0) < c(k+1,\mu_0)$*. We want to show that there exist* μ_{high} *such that* $\mu > \mu_{high}$ *implies* $c(k,\mu) > c(k+1,\mu)$*. In order to do this, it is enough to show that the function* $c(k+1,\frac{\mu}{k+1}) - c(k,\frac{\mu}{k})$ *is non-increasing. In order to do that, observe that for any non-decreasing convex function* $f(x)$*,* $a > b > 0$*,* $\Delta = a - b$*, and* $t > 1$ *we have*

$$t(f(a) - f(b)) \leq f(ta) - f(tb).$$

Because

$$f(ta) - f(tb) = \sum_{i=0}^{\lfloor t \rfloor - 1} (f(at - i\Delta) - f(at - (i+1)\Delta)) + f(at - (t - \lfloor t \rfloor)\Delta) - f(bt)$$

$$\geq \sum_{i=0}^{\lfloor t \rfloor - 1} (f(a) - f(b)) + (t - \lfloor t \rfloor)(f(a) - f(b)) = t(f(a) - f(b)).$$

We need to show now that if $\mu > \omega$, and $c(k,\omega) = kg(\omega)$

$$(c(k+1,\mu) - c(k,\mu)) - (c(k+1,\omega) - c(k,\omega)) \leq 0.$$

Or equivalently $(k+1)\left(g\left(\frac{\mu}{k+1}\right) - g\left(\frac{\omega}{k+1}\right)\right) - k\left(g\left(\frac{\mu}{k}\right) - g\left(\frac{\omega}{k}\right)\right) \leq 0$. *Dividing by k we obtain:*

$$\frac{k+1}{k}\left(g\left(\frac{\mu}{k+1}\right) - g\left(\frac{\omega}{k+1}\right)\right) - \left(g\left(\frac{\mu}{k}\right) - g\left(\frac{\omega}{k}\right)\right).$$

But by our previous observation, making $t = \frac{k+1}{k}$ this is less than or equal to

$$g\left(\frac{\mu}{k}\right) - g\left(\frac{\omega}{k}\right) - g\left(\frac{\mu}{k}\right) - g\left(\frac{\omega}{k}\right) = 0.$$

An analogous analysis applies for μ_{low}.

Proposition 2 implies that there are frequency strips that define the optimal number of machines that should be active.

3. Special Cases

As mentioned before, the problem studied in this chapter is a generalization of several previous works in the queueing literature. Many of these problems result in optimal policies that are monotone threshold type or monotone hysteretic. To formalize those concepts we introduce the following definitions.

Definition C.12 *A policy is monotone threshold type if for the finite linearly ordered action space $\mathbb{A}$, and finite linearly ordered action space $\mathbb{S}$, there exist thresholds $0 = T_0 < T_1 < T_2 < ... < T_{|\mathbb{A}|} = \infty$ such that the action a_i is taken if and only if the system is on state s, such that $T_{i-1} \leq s < T_i$*

Definition C.13 *A policy is monotone hysteretic if for the finite linearly ordered action space $\mathbb{A}$, and finite linearly ordered action space $\mathbb{S}$, there exist thresholds $0 = L_0 < L_1 <$*

$L_2 < ... < L_{|\mathbb{A}|} = \infty$ and $0 = U_0 < U_1 < U_2 < ... < U_{|\mathbb{A}|} = \infty$ such that the action a_i is taken if and only if the system is on state s, such that $L_{i-1} \leq s < U_i$

The following are special cases worth noting:

- If K=1, this is the classical single server with adjustable service rate problem. In this case, the solution structure is a threshold monotonic policy described in George and Harrison [7].
- If K=1, the frequency switching costs are positive (although in our problem they are 0) and the cost functions are sub-modular, we have the case studied in Serfozo and Lu [9]. In this case the optimal policy is monotone hysteretic.
- If $|\Gamma| = 1$ and $B_0 = 0$, this is the call center control problem. In this case, the solution structure is a threshold monotonic policy.
- If $|\Gamma| > 1$ and $K > 1$, and the costs associated with the system are sub-modular, we conjecture that the optimal policy is hysteretic monotone for the number of servers and monotone threshold for the frequency.

Notice that in this chapter we assume $|\Gamma| > 1$, $K > 1$ and the costs are not sub-modular.

D. Policy Computation

We are interested in not only the structure of the optimal policy but also in finding an efficient method to compute it. In this section, we describe the methodology we use to find efficient policies for the multi-server with an adjustable service rate system. For small cases, it is possible to compute the optimal policy explicitly by means of a value or policy iteration procedure. However, real life data center problems routinely involve a very large number of servers, and the value iteration becomes a computationally infeasible approach.

Motivated by the solution structure of the special cases mentioned in section 4.2, we propose a simulated annealing based heuristic to determine hysteresis curves for the number of servers combined with a simultaneous local search for determining the thresholds for the frequencies. Finally, we compare the performance of our solution with the optimal solution for small cases and with other policies for larger cases.

1. Computation of Efficient Policies

It is well known that the optimal policy for a single server queueing control problem with variable service rate and without switching costs has a monotonic threshold structure. In fact, this algorithm to compute the thresholds and hysteresis curves can also be used to compute the policies mentioned in section 4.5. Additionally, in the presence of switching costs, if the holding and running costs are sub-modular the optimal policy is monotonic hysteretic Serfozo and Lu [9]. Moreover, for the case of a fixed running frequency system with dynamic number of servers and zero server set up cost, the optimal policy without a switching cost is also threshold type.

As the system combines both features, we consider policies that are monotone threshold type to determine the frequencies and monotone hysteretic for choosing the number of servers. A policy ν (see Figure 6) is completely determined by two vectors $\mathbf{L}$ and $\mathbf{U}$, and one matrix $\mathbf{T}$. The vectors $\mathbf{L} = \{l_1, l_2, ..., l_K\} \in \mathbb{R}^K$ $l_i \leq l_{i+1}$, and $\mathbf{U} = \{u_1, u_2, ..., u_K\} \in \mathbb{R}^K$, $u_i \leq u_{i+1}$; $l_i \leq u_i$ are of size K and represent the lower and upper bounds in the number of jobs in the queue for which we are going to turn off and on the servers respectively. The matrix $\mathbf{T} = \{\mathbf{t}_1, \mathbf{t}_2, ..., \mathbf{t}_{|\Gamma|}\} \in \mathbb{R}^{(|\Gamma|-1)\times K}$, $t_{ij} \leq t_{i,j+1}$ is of size $|\Gamma| \times K$ and represents the thresholds in which each frequency will be used.

By using the memoryless property, we may limit ourselves to taking decisions only when the state changes, i.e., when a job leaves or arrives at the system. At any decision epoch, if there are k active servers and n jobs in the system, a server is turned on (off) if

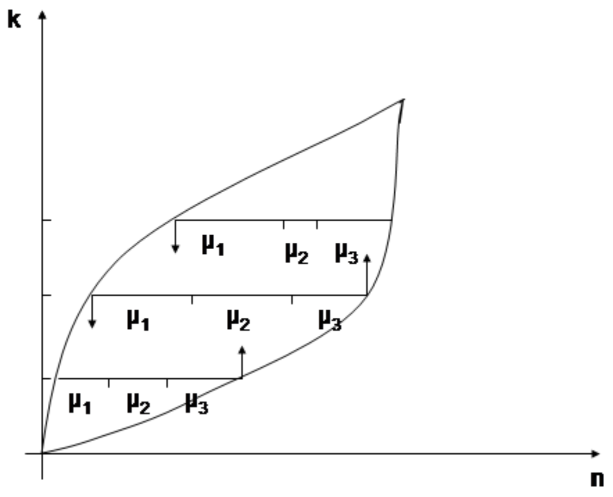

Fig. 6. Policy diagram for $|\Gamma| = 3$ and $K = 3$.

$n > u_k(< l_k)$ and the frequency is set to μ_s if $\mathbf{t}_{k,s-1} < n < \mathbf{t}_{k,s}$. Therefore the policy ν is determined by **L**, **U** and **T**.

As an example, consider the case in which there are $K = 2$ servers available and each server can run at $|\Gamma| = 3$ service rates $\{\mu_1, \mu_2, \mu_3\}$ and $N = 10$. If $L = (0,3)$ and U=$(5,10)$ it means that if there is one server running, we will turn the other server on if the number of jobs in the system n is larger than 5 (the second component is 10 meaning that no more servers will be turned on) and if there are two servers running we are going to turn off a server if $n < 3$. Additionally, consider the matrix **T** given by:

$$\mathbf{T} = \begin{pmatrix} 2 & 3 \\ 4 & 6 \end{pmatrix}.$$

This means that, for instance, having one active server, the system will use the first frequency (μ_1) with 2 or fewer jobs in the system, the second frequency (μ_2) for more than two, and the third (μ_3) for more than 3 jobs in the system. If there are two active servers, the first frequency is going to be used when there are 4 or fewer jobs in the system, the second frequency when there are 5 or six jobs, and the third frequency when there are more than 6 jobs. The individual running frequencies of the system, with the number of jobs in the columns and the number of active servers in the rows, will be given by

	$n(t)$									
$k_t = 1$	μ_1	μ_1	μ_2	μ_3	μ_3	μ_3	μ_3	μ_3	μ_3	μ_3
$k_t = 2$	μ_1	μ_1	μ_1	μ_1	μ_2	μ_2	μ_3	μ_3	μ_3	μ_3

2. Queueing Control Methods

In order to evaluate the quality of our solution, we compute the performance of other solution schemes selected because of their simplicity or because those are extensions of well studied special cases. Additionally, we implement a value iteration procedure to find the exact optimal policy for small cases under the assumption of a finite buffer. In order to do that, we transform the Semi-Markov Decision Process into an equivalent Markov Decision Process using the uniformization techniques in Beutler and Ross [86].

The methodologies used are the following:

Maximum number of servers[All servers] Here we compute the cost of operating the system having all the servers available turned on all the time (i.e.,, $k_t = K$ for all t) and varying only the service rates. This policy is sometimes used in practice and does not take advantage of the dynamic number of servers. Here the frequencies in **T** are determined by the George and Harrison [7] algorithm under the assumption of a unique server with the aggregated service rate, and **L** and **U** are fixed. We expect this policy to be outperformed by a policy that adjusts the number of servers dynamically.

Just enough servers[Min.Serv.] Here we compute the operating costs using a fixed number of servers that is chosen to be the minimum number necessary to have a stable system if the buffer were infinite ($k_t = L \leq K$ for all t) and the frequencies were $\gamma_{|\Gamma|}$. This approach does not use the dynamic number of servers as well but may be effective if the fixed cost of operating a server is large compared to the variable cost of operating the system. Here the frequencies in **T** are determined by the George and Harrison [7] algorithm under the assumption of a unique server with the aggregated service rate, and **L** and **U** are fixed.

Naive George and Harrison extension[NGH] In this approach we compute the optimal frequency given by George and Harrison algorithm for any fixed number of servers and use Simulated Annealing to find the bounds at which the servers are turned on and off. This approach uses both the dynamic frequencies and the dynamic number of servers but does not take advantage of the interaction between both parameters since it "optimizes" the frequencies first and then finds a good set of bounds to turn the servers off and on. Here **T** is determined by the George and Harrison [7] algorithm and **L** and **U** are found afterward by simulated annealing.

Simultaneous optimization[Sim.Opt.] We propose this heuristic procedure that performs a simultaneous search of both the frequency bound matrix **T** optimizing the complete policy, and taking advantage of both adjustments and the interaction between them. Here **T**, **L** and **U** are found by a heuristic procedure described in Section D.3 that takes into account the combined effects of the three policy parameters $\mathbf{U}, \mathbf{L}, \mathbf{T}$. This procedure finds a hysteretic policy and makes use of the conjecture that the optimal policy is monotone hysteretic to determine the number of servers and monotone threshold type to determine the frequencies.

3. Simultaneous Optimization Algorithm

Our objective is to find the best policy that is monotone hysteretic to determine the number of servers and monotone threshold type to determine the frequencies, and to compare its performance with the policies used in practice or the ones with a constant number of servers. We use a Simulated Annealing (SA) procedure on the number of servers combined with a neighborhood sampling for the vector **T** domain so all three vectors are adjusted at the same time. The neighborhood functions are built so that at each iteration, one of the components of the vectors is increased or decreased by one.

The intuition behind the procedure is as follows. Given **L** and **U**, there is an optimal **T** to make the average cost minimal. On the other hand, given **T**, there is a pair of vectors **L** and **U** that minimize the cost. Since on each iteration of the SA procedure the bounds of the hysteretic policy are changed in only one component by at most one unit, it would be reasonable to expect that the optimal frequencies for the neighbor are similar to the ones on the original policy; therefore, sampling the frequencies at each iteration allows us to improve the chosen frequencies. Moreover, given that as the variable temperature in the SA procedure decreases, the transition from one solution to the next is less likely, it is clear that as the temperature becomes small the neighborhood of frequencies will be sampled more extensively.

A scheme of the algorithm is:

```
start
Generate random vectors L,U,T
currentcost=cost(L,U,T)
t=Initial temperature
i=0
while t>0
{
        generate random neighbor (L,U)
        newCost=cost(neighL, neighU, T)
```

```
      for i=1:sample size
     {
            Generate random neighbor (T)
     neighNewCost=cost(neighL ,neighU ,neighT)
     if  (neighNewCost < newCost)
           {
               T = neighT
                         newCost =  neighNewCost
           }
      }
      if (newCost < currentCost)
    {
               currentCost=newCost
      (L,U,T)=(neighL ,neighU ,T)
    }
else
    {
      if (criterion(currentCost-newCost)==true )
      {
               currentCost=newCost
               (L,U,T)=(neighL ,neighU ,T)
      }
  }
  if(i mod parameter ==0)
  t=t-Delta_t
```

```
}
end
```

The function *criterion(currentCost-newCost)* is a randomized function that decides whether the algorithm should move to the new solution when this new solution is not better than the current one. As in the standard implementation of the Simulated Annealing meta-heuristic, the probability of obtaining a true value decreases with the temperature and increases with the value of *currentCost-newCost*.

Remark D.1 *The algorithm was implemented in Matlab 2007b. The function that computes the cost includes the solution of the steady state equations of a Markov chain. It is worth noting that an appropriate manipulation of the very large system of equations solved repeatedly in the algorithm is necessary. Matlab sparse equation solver was very efficient due, at least in part, to the almost band diagonal structure of the matrix. Moreover, a clever initial choice of the T, L and U proved to be useful. The neighborhood functions picked random values for T, L and U that differ from each component by at most one unit.*

E. Numerical Results

In this section, several numerical experiments are performed in order to determine the accuracy of the proposed heuristic. For a set of small cases with different booting costs, buffer sizes, arrival rates and fixed costs, we present the value of the exact solution, the value obtained after running our simultaneous optimization procedure twice, and the value obtained by the other three methods. For larger cases, we present the value obtained by our simultaneous optimization procedure compared with what can be obtained by running a fixed number of servers. In the data tables, K corresponds to the total number of available servers, N is the buffer size, λ is the arrival rate and the minimum cost is calculated by

Table I. Results for $c(k,\mu) = k(10+\mu^3)/8$; $h(n) = n^2$; $\Gamma = (2,3,4,5)$.

K	**N**	**B_0**	**λ**	**Exact**	**NGH**	**Sim.Opt.**	**All**	**Min.Serv.**
3	8	0	6	43.5342	52.9263	45.2321	58.1916	55.7968
3	8	0	8	59.145	68.0813	61.3232	69.0514	76.1616
3	8	0	10	78.2077	84.4089	80.0515	82.5612	98.5937
3	8	50	6	53.1451	62.5225	53.1451	58.1916	55.7968
3	8	50	8	65.7002	115.9786	65.7002	69.0514	76.1616
3	8	50	10	82.6939	114.5833	82.6939	82.5612	98.5937
3	8	100	6	55.7327	62.5225	55.7327	58.1916	55.7968
3	8	100	8	65.7002	88.2315	65.7002	69.0514	76.1616
3	8	100	10	82.6939	84.788	82.6939	82.5612	98.5937
3	13	0	6	43.6453	57.5814	43.6453	58.3248	58.9227
3	13	0	8	60.2498	70.7001	60.2498	70.3489	100.2718
3	13	0	10	85.0477	91.871	86.5307	89.6231	175
3	13	50	6	53.2506	64.2549	53.2506	58.3248	58.9227
3	13	50	8	66.8907	70.7001	66.8907	70.3489	100.2718
3	13	50	10	88.7711	196.25	88.7711	89.6231	175
3	13	100	6	53.2506	58.2071	53.2506	58.3248	58.9227
3	13	100	8	66.8907	217.0246	66.8907	70.3489	100.2718
3	13	100	10	88.7711	292.3178	88.7711	89.6231	175
3	18	0	6	43.6476	52.0059	43.6476	58.3277	59.4647
3	18	0	8	60.3468	109.0131	60.3468	70.4716	116.8011
3	18	0	10	86.8897	94.1907	90.4258	91.6551	283.9583
3	18	50	6	53.2527	58.2089	57.9172	58.3277	59.4647
3	18	50	8	66.9965	113.9257	66.9965	70.4716	116.8011
3	18	50	10	90.4258	94.1907	90.4258	91.6551	283.9583
3	18	100	6	53.2527	65.0493	53.2527	58.3277	59.4647
3	18	100	8	66.9965	70.7821	66.9965	70.4716	116.8011
3	18	100	10	90.4258	295.2985	90.4258	91.6551	283.9583

means of an exact value iteration procedure, the naïve George and Harrison [7] extension, the proposed heuristic and using a fixed number of servers. The heuristic finds a solution really close to the optimal in a considerably shorter time.

Table 1 was computed with a low fixed cost for running a server; this encourages the use of several servers running at low frequencies. In contrast, Table 2 corresponds to a large fixed cost of running the servers; this cost structure encourages the use of fewer servers running at higher frequencies. As seen in tables 1 and 2, the average error obtained by the simultaneous optimization algorithm compared to the optimal policy is 3.8% while the NGH method gives a much larger error (around 30%). The cost of the other two methods may be significantly higher depending on the cost structure. In Tables 1 and 2, the average error for using all the servers is around 10% if the fixed operating cost is low, but it increases to three times the optimal if the fixed operating cost is relatively high. It is worth noting that when the exact solution coincides with the solution found on the simultaneous optimization algorithm, the exact solution is of the type described in Figure 2. This happened in more than half of our experiments. However, for the other cases, the solution found by the simultaneous optimization algorithm is very close to optimal.

Tables 3 and 4 consider a more realistic number of servers where a exact solution is computationally infeasible. Hence we compare the average cost obtained by the simultaneous optimization method against the costs obtained by having all servers running or only enough servers to make the system stable. We can observe that as the arrival rate increases, the cost obtained by having all servers active is closer to the cost obtained by the simultaneous optimization algorithm. Similarly, as the arrival rate decreases, running the minimum number of servers becomes more attractive. The average savings of using the policy obtained by simultaneous optimization against using a fixed number of servers is more than 60% for the cost structure inTablee 3 and around 40% for the cost structure in Table 4. Moreover, the savings obtained by controlling the frequencies and the number of

Table II. Results for $c(k,\mu) = k(90+\mu^3/8)$; $h(n) = n^2$; $\Gamma = (2,3,4,5)$.

K	**N**	**B_0**	**λ**	**Exact**	**NGH**	**Sim.Opt.**	**All**	**Min.Ser.**
3	8	0	6	164.7232	168.8039	165.6511	298.1916	215.7968
3	8	0	8	207.1682	214.8897	207.1682	309.0514	236.1616
3	8	0	10	236.9143	237.8716	237.843	322.5612	258.5938
3	8	50	6	188.255	193.2978	188.255	897.6014	215.7968
3	8	50	8	223.4461	223.5757	223.4461	1104.439	236.1616
3	8	50	10	236.9143	237.8716	237.843	1302.26	258.5938
3	8	100	6	192.5947	193.2978	192.5947	1497.011	215.7968
3	8	100	8	223.4461	223.5757	223.4461	1899.826	236.1616
3	8	100	10	236.9143	237.8716	237.843	2281.959	258.5938
3	13	0	6	165.0179	169.398	167.0621	298.3248	218.9227
3	13	0	8	208.7663	212.1422	208.7663	310.3489	260.2718
3	13	0	10	261.3777	265.8561	263.096	329.6231	335
3	13	50	6	193.3958	196.837	194.6091	898.3187	218.9227
3	13	50	8	253.2312	392.8818	253.278	1110.151	260.2718
3	13	50	10	304.0764	447.6455	304.0764	1327.041	335
3	13	100	6	207.6338	210.2212	207.9962	1498.313	218.9227
3	13	100	8	257.4377	264.4077	257.4377	1909.953	260.2718
3	13	100	10	315.0271	407.9029	315.0271	2324.458	335
3	18	0	6	165.0255	173.4747	165.0255	298.3277	219.4647
3	18	0	8	208.927	248.0801	208.927	310.4716	276.8011
3	18	0	10	263.9956	267.9813	263.9956	331.6551	443.9583
3	18	50	6	193.4407	197.3434	193.4407	898.3277	219.4647
3	18	50	8	253.6424	256.617	253.6424	1110.463	276.8011
3	18	50	10	307.7404	647.6207	307.7404	1331.317	443.9583
3	18	100	6	207.7837	210.669	208.7475	1498.328	219.4647
3	18	100	8	258.0483	284.0655	265.8442	1910.455	276.8011
3	18	100	10	319.5356	396.0882	319.5356	2330.978	443.9583

Table III. Results for $c(k,\mu) = k(10+\mu^3)$; $h(n) = n$; $\Gamma = (2,3,4,5)$.

K	**N**	$\mathbf{B_0}$	λ	**Sim.Opt.**	**All**	**Min.Serv.**
50	70	0	150	609.33	4412.5	4034.1
50	70	0	200	789.33	5585	5367.2
50	70	20	150	609.33	10412	4034.1
50	70	20	200	789.33	13585	5367.2
50	120	0	150	1018	4412.5	4080.5
50	120	0	200	1019	5585	5420.7
50	120	20	150	659.33	10412	4080.5
50	120	20	200	839.33	13585	5420.7
100	150	0	150	1804	5311.4	4101.4
100	150	0	200	1887.7	6481.7	5443.6
100	150	20	150	689.33	11311	4101.4
100	150	20	200	869.33	14482	5443.6
100	200	0	150	1797.6	5311.4	4132.4
100	200	0	200	1887.7	6481.7	5476.5
100	200	10	375	9567.5	18079	10181
100	200	10	425	10141	20252	11525
100	200	20	150	4108.5	11311	4132.4
100	200	20	200	5442.5	14482	5476.5

servers decrease as the booting cost B_0 increases.

Table IV. Results for $c(k,\mu) = k(10+\mu^3)$; $h(n) = n^2$; $\Gamma = (2,3,4,5)$.

K	**N**	$\mathbf{B_0}$	**λ**	**Sim.Opt.**	**All**	**Min.Serv.**
200	349	0	700	11603	19999	59628
200	349	0	800	16005	22365	62321
200	349	20	700	36578	47999	59628
200	349	20	800	41922	54365	62321
500	699	0	2100	44540	58212	219850
500	699	0	2300	56426	63120	225240
500	699	20	2100	111960	142210	219850
500	699	20	2300	148430	155120	225240

Remarks: The solutions given by the different algorithms provide some insight on the influence of the cost parameters on the solution structure. In the following paragraphs, we are going to comment on the effect of changing the values of the input constants.

Arrival Rate: As the arrival rate increases, the solution that keeps all the servers on all the times becomes closer to optimal. Analogously, as λ becomes smaller, a solution with a fixed smaller number of servers is closer to optimal since with a low arrival rate, the likelihood of large holding costs is reduced and the optimal solution will use a large number of servers less often.

Booting cost: A large booting cost B_0 makes unattractive the decision of turning on machines. As the policy is static, it makes the decision of turning off machines unattractive as well. Therefore, the larger B_0 is, the more separation we find between L_i and U_i on the policy parameters making turning on the servers less likely.

Fixed running cost per server: A large fixed running cost will make unattractive to have a large number of machines turned on. Since in the experimental case the variable running cost is a cubic expression, then running several machines at a low frequency is cheaper than running one server at a high frequency. If the fixed cost is large enough, the solution will require running fewer servers at large frequencies.

Holding cost: As expected, the larger the holding cost, the higher the running frequencies in the optimal solution.

Optimal solution structure: Although we do not have a formal proof, we expected the solution structure to be monotone hysteretic for booting bounds and monotone threshold for the frequency bounds. The numerical experiments support this hypothesis.

F. Conclusions

We consider the $M/M_t/k_t$ queueing control problem where the system manager controls the number of servers and the operating service rate. We showed that the optimal control policy for this system is hysteretic for both, the discounted and average criteria. We proposed a numerical method to compute a near optimal policy that is monotone hysteretic to determine the number of servers and monotone threshold type to determine the running frequencies. Our experiments suggest that the optimal policy has this structure in almost all instances.

As it is evidenced by our numerical results, the possibility of varying both the service rates and the number of servers can produce significant savings. We proposed a numerical algorithm to find solutions to the $M/M_t/k_t$ that represent advantages with respect to widely used policies. Additionally we found that a policy that takes into account the interaction between those two factors performs better than using sequential optimization.

CHAPTER IV

INDEPENDENT PROVISIONING

Powering down servers and dynamic voltage/frequency scaling (DVS) provide two very important power management strategies. Powering down a server reduces the energy it consumes to zero, but in order to make the server available again, the system not only spends power in rebooting the server but this action wears the machine down and needs some time to be completed. DVS mechanisms reduce the total amount of energy consumed by the server by slowing down the speed of the processor, but all the other server components keep consuming the same amount of energy; in this state, the server may still process data at a slower pace, and speeding the processor up is quickly achieved with negligible cost.

Although considerable work has been done on using these capacity allocation techniques, these have not been widely implemented in the industry partly because practitioners fear that their use may increase the risk of not meeting the QoS requirements. Bad results may be a consequence of one or more of the following factors:

- Proactive capacity provisioning schemes that provide lower-than-needed capacity during a specific period, and do not have mechanisms to react to unexpectedly high loads.
- Purely reactive mechanisms that only consider the current state of the system and do not use the potentially useful historic information about the requests stream.
- Low quality load forecasts.
- No real time decision-making.

As a consequence, practitioners have been reluctant to implement resource allocation schemes that reduce the allocated capacity, preferring to assume the additional electricity and goodwill costs. However, it will be optimal to serve the arriving requests using the least amount of resources and meeting the requirements in the SLA.

In this chapter, we present a methodology that allows us to take capacity decisions in real time based on a combination of demand forecasting and system status. The methodology is flexible enough to adapt to sudden behavioral changes or forecasting errors without failing to meet SLA requirements. The proposed scheme of Adaptive Resource Allocation (ARA), includes a novel decomposition of the system load time series, along with a wavelet based forecasting methodology that estimates the near future system load and relies on the fact that the demand to a data center is ordinarily nearly cyclical. The controller we developed takes into account the forecasted system load and the current buffer conditions to make decisions regarding the system capacity. The combination of these reactive and proactive control techniques forms an efficient scheme to reduce the energy expenditures without compromising meeting the SLA. To the best of our knowledge, this is both the first study combining feedback control and proactive decision-making at each decision epoch, and the first study that does data center dynamic provisioning using TSF constraints and real data.

The main contribution of this chapter is the introduction of a scheme that is efficient enough to save a significant amount of power, but flexible and adaptable enough to meet the service level conditions even when the system has sudden behavior changes. Moreover, the scheme factors in the reliability consequences of changing the service capacity. In summary, we propose an effective novel wavelet based forecasting methodology that uses the nearly cyclical behavior of the load to the data center. We combine reactive and proactive mechanisms at the same decision epoch in order to make decisions regarding both the number of servers and the frequency to be used.

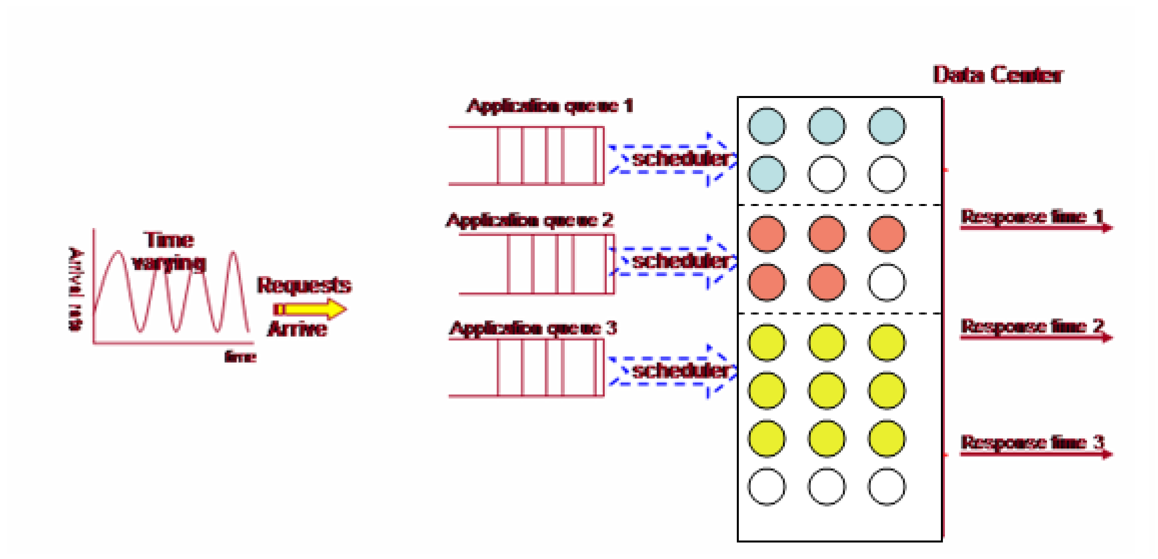

Fig. 7. Data center model.

A. Problem Description

Let us consider an abstract data center as the one in Figure 7, where a fixed number of servers and associated equipment are entirely devoted to host a particular application that serves requests from the outside world. In the data center in Figure 7, clear circles represent inactive (turned-off) servers and shaded circles represent active servers devoted to the corresponding application. Moreover, the running frequency of each application may be different. For instance, the first application has six servers available but at this particular point of time only four are active and running at a (potentially) different frequency than the five active servers running application two. In this model, we do the system control independently, so we can analyze the general problem as the aggregation of several single application facilities (illustrated in Figure 8). In this context, a *single* application may be an aggregation of several individual applications to be served at the same time in the servers. For instance, if a set of servers runs two applications, we consider the aggregation as a single *meta-application* for the purposes of this chapter. This aggregation may be designed in order to obtain additional energy and reliability savings as will be shown in Chapter V. Assume moreover, that the data center management does not have knowledge of the future load except the information regarding load history. Given the competitive environment in the data center industry, the objective of the data center administration is twofold: first, given the facility and equipment, it should reliably serve customer's requests using the lowest possible amount of energy to reduce costs and environmental impact, and secondly, it should provide the customers with a service that meets the quality standards consigned in the SLA.

We model this problem as a queueing system with K homogeneous servers that have the capacity of being turned on and off as well as the ability to run at any of a finite pre-determined set of frequencies $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_{|\Gamma|}\}$. The costs incurred by the data center are represented by the energy required to run the servers, and an energy, reliability and risk cost associated to turning on and off the machines. According to Elnozahi et al. [54] and Chen et.al. [8], the relation between the energy consumption and the frequency is of the form $P(\gamma) = P_{fix} + P_f \gamma^3$, where P_{fix} and P_f are equipment specific constants, and the cost of turning a server on is assumed to be a constant B_0.

The request arrival process, as well as the requests processing time is assumed to be arbitrary and time varying. In particular, as described in Crovella and Bestavros [11], in practice the arrival process is self-similar and the time needed to process each job is highly variable. The main objective is to operate this queueing system using the lowest possible amount of energy by controlling the number of active servers and their frequencies.

In order to control the queue efficiently, we divide the problem into two parts. First, we need to forecast the arrival process in order to make good proactive capacity allocation decisions; secondly, we should design a decision-making scheme that takes advantage of the forecasting data and the information about the state of the system.

B. Problem Formulation

Let us consider a First Come First Served queueing system as the one in Figure 8, with K identical available servers. At any given time t, each server may be in on (active) state or off (inactive) state; moreover, when a server is on, it can run at any frequency $\gamma \in \Gamma = \{\gamma_1, \gamma_2, ..., \gamma_{|\Gamma|}\}$, but all active servers should run at the same frequency. Given the convexity of the cost functions, it can be shown that if the set of eligible frequencies is an interval, it is optimum to run all the servers at the same frequency. Although it is possible to handle non-identical servers and different running frequencies, for this chapter we make this assumption for convenience in terms of the presentation of the algorithms. Let us denote by $k(t)$ the number of active servers at time t, so $k(t) \in \{1, 2, ..., K\}$, and by $F(t) \in \Gamma$ the frequency of the active servers.

Requests arrive at the system according to an arbitrary process. Let A_n be the arrival time of the $n-th$ request, and S_n the amount of work required to serve it. We assume that S_n is not observable until the service starts, so all the information available about the arrivals to the system at time t is the number of requests in the buffer $n(t)$. This assumption may be relaxed without having to make significant changes in our capacity allocation algorithm. The objective is to select $k(t)$ and $F(t)$ dynamically based on the observed $n(t)$ so that the long-run average energy consumed is minimized, subject to satisfying a Quality of Service constraint and ensuring system reliability. The number of active servers and the running frequency determines the service time for each request, so the departure time of the $n-th$ request D_n is a function of the capacity allocation decisions.

Since our objective is balancing energy efficiency, reliability and QoS, we will convert both energy and reliability into dollar costs. Later in the chapter we will convert the QoS to a monetary term by adding an appropriate factor.

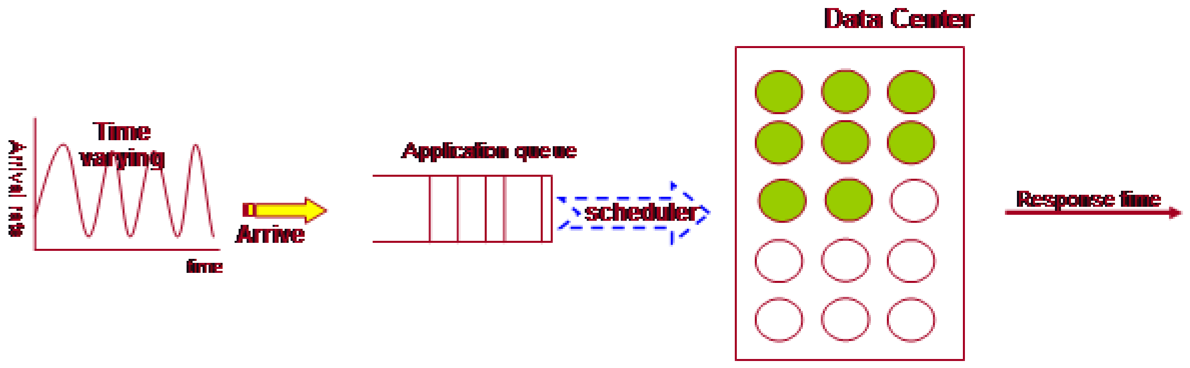

Fig. 8. Single application data center model.

Assume that powering up a server requires T_{boot} seconds, and has a fixed cost B_0 caused by the wear and tear on internal components due to the booting process and the energy expended in booting the machine up. Operating k servers at frequency γ, costs $c(k,\gamma)$ per unit time, and the function $c(k,\gamma)$ is assumed to be non-decreasing in both components. Moreover, the system manager is required to meet a previously specified SLA that bounds the fraction of the requests that may stay in the system longer than (previously determined) R seconds.

Powering up a server not only spends energy, but induces a significant hardware wear and tear, and risks possible failures such as unsuccessful boots or routing tables errors; therefore, is undesirable to keep powering servers up and down arbitrarily frequently. In order to avoid that, we define fixed time intervals of length t_0 in which we are going to make decisions regarding frequency and number of server changes. Therefore, every t_0 seconds, the system administrator will fix values of $n(t)$ and $k(t)$, that will remain constant during the next t_0 seconds, with the objective of minimizing the energy consumption in the overall system without violating the quality of service constraint.

Using this notation, and using z as an index for the time periods, the optimization problem may be written as:

$$\min \sum_{z=1}^{Z} \left(c(k_z,\gamma_z)t_0 + B_0(b_{z0})^+\right)$$
$$\text{s.t. } \forall n \left(P\left(D_n - A_n > R\right) < \alpha\right),$$

where t_0 is the time length of the periods, and $b_{z0} = (k(zt_0) - k((z-1)t_0))$ represents the number of servers to be turned on/off in $z-th$ period, and α is a constant previously designed in the SLA.

This system may be described as a dynamic system, where the state is determined by the number of jobs in the queue $n(t)$, and the number of active servers $k(t)$. If at time t a new request arrives at the system, and there are fewer than $k(t)$ requests being served (i.e. there is an empty active server), then the request goes directly to service, otherwise it goes to the last place in the queue. When a service is completed, the first request in the queue starts service with a completion time $\frac{S_n}{\beta\gamma(t)}$ where β is a constant that represents the proportion between the amount of work performed and the running frequency. At an arbitrary time t, the system state will change to $k(t)$ and $n(t)+1$ if the next request comes to the system before a completion is achieved. Otherwise, the system will go to state $k(t)$ and $n(t)-1$, after a time period equal to the minimum completion time. Whenever a server should be turned off, it is allowed to finish the service of the request present at the time of decision before becoming inactive.

As described earlier, the system dynamics depends, on the values of $n(t)$, $F(t)$ and $k(t)$ as well as on the arrival process. As will be explained in section 4.1, large values of $n(t)$ increase the likelihood of some requests' time in the system being larger than R, but higher values of $k(t)$ and $\gamma(t)$ may decrease the value of $n(t)$. However, increasing the number of servers and/or the frequency, implies larger energy expenditure of the overall system. Hence, our objective is to optimize the trade off between the energy expenditures and the QoS constraint by controlling this highly variable system.

C. Methodology

As mentioned in the literature review, most data center dynamic capacity provision schemes are based on either reactive mechanisms or proactive mechanisms. A purely reactive scheme makes the same provisioning decision without using any knowledge of the future system behavior; in contrast, a purely proactive mechanism uses, usually imperfectly, information about the future or past behavior of the system but does not have the capacity to adapt when the forecasting fails. We intend to overcome these difficulties by combining three complementary techniques. First, we modify the formulation in order to obtain an approximate, more tractable optimization problem; second, we propose a problem-specific forecasting method to make proactive decisions; and third, we add a reactive mechanism to allow the system to recover from unforeseen fluctuations in demand.

Given that request sizes and inter-arrival times are uncertain and non-stationary, solving this stochastic optimization problem is not feasible. We can, however, proceed in a Lagrangian-relaxation fashion and define an appropriate term that will be added to the objective function in order to obtain a closely related unconstrained problem. Therefore, we will define below an appropriate non-decreasing holding cost $h(n,k)$ per unit time when there are n jobs in the system and k active servers. Decisions regarding the frequency and the number of servers will be taken every t_0 seconds.

The new problem may be stated as:

$$\min \sum_{z=1}^{Z} \left(c(k_z, \gamma_z) t_0 + B_0 (b_{z0})^+ + H_z \right) \tag{4.1}$$

where t_z and b_{z0} remain the same as above, and H_z is the cumulative holding cost realized during the z-th period.

1. Selection of the Holding Cost

Quality of Service (QoS) refers to the ability to devote necessary resources to serve a data flow generated by a group of users in order to guarantee a certain level of performance. The performance is measured in terms of, for example, bit rate, delay, jitter or packet dropping. In this chapter, we assume that the statistical QoS constrained is defined in such a way that the probability of having a delay longer than a certain time is bounded, i.e.,

$$P\left(D_n - A_n > R\right) < \alpha.$$

This type of constraint provides a bound on the proportion of requests receiving unsatisfactory service, and contrasts with the average delay or standard deviation constraints found in other studies. Among the features of this type of constraint, we can mention that the system will be indifferent between two different values of $R-(D_n-A_n)$ as long as those have the same sign, except for the obvious fact that a large response time in the system for a specific request will likely imply delays for the subsequent arrivals. This, in particular, reflects a concern about providing the individual users with a good experience most of the times an individual sends a request, and not only about the average long run response time.

As mentioned in the formulation, the energy management data center problem with QoS constraints is then given by:

$$\min \sum_{z=1}^{Z} \left(c(k_z, \gamma_z) t_0 + B_0 (b_{z0})^+ \right)$$

$$\text{s.t. } \forall n \left(P\left(D_n - A_n > R\right) < \alpha \right),$$

Let us consider the expression $P\left(D_n - A_n > R\right)$. Let us denote the mean and standard deviation of the service time of the last element in the buffer by μ and σ. Given that we expect α to be a small proportion of the total arrival stream, we can write $R = \mu + A$ where A is a positive constant that may be expressed as $A = k\sigma$. Under these assumptions, we can write:

$$P\left(D_n - A_n > R\right) = P\left(D_n - A_n > \mu + k\sigma\right) = P\left(D_n - A_n - \mu > k\sigma\right).$$

The last expression is very close to the left hand side of the one sided Chebyshev's inequality (Cantelli's inequality), $P(X - \mu > k\sigma) < \dfrac{1}{k^2+1}$ [87].

Defining $k = \sqrt{\frac{1}{\alpha} - 1}$, we obtain:

$$P\left(D_n - A_n - \mu > \sqrt{\frac{1}{\alpha} - 1}\sigma\right) < \alpha.$$

In order to obtain a decision-making rule regarding $k(t)$ and $\gamma(t)$, we need to rewrite this expression in terms of the decision variables, and the state of the system at time t. To do that, let us denote the average request size by μ_s and the requests' sizes standard deviation as σ_s. Define a positive constant β that represents the proportional relation between the frequency with the amount of work processed by the server. The expected waiting time for the last job in the buffer given n(t) will be equal to the expected waiting time in the buffer $W = \frac{(n(t)-1)\mu_s}{k(t)\beta\gamma(t)}$, plus the expected service time $S = \frac{\mu_s}{\beta\gamma(t)}$. Therefore,

$$\mu = S + W = \frac{(n(t)-1)\mu_s}{k(t)\beta\gamma(t)} + \frac{\mu_s}{\beta\gamma(t)}$$

The variance of the time in the system will be the variance of the waiting time in the buffer plus the variance of the service time:

$$\sigma^2 = \frac{(n(t)-1)\sigma_s^2}{(k(t)\beta\gamma(t))^2} + \frac{\sigma_s^2}{(\gamma(t)\beta)^2}.$$

The holding cost will be defined as follows:

$$h(n(t),k(t)) = \begin{cases} 0 & if \quad \mu + \sqrt{\frac{1}{\alpha} - 1}\sigma < R, \\ \eta n/k & if \quad \mu + \sqrt{\frac{1}{\alpha} - 1}\sigma > R, \end{cases}$$

where η is a constant larger than the cost of operating one server at its maximum frequency between two decision epochs. This holding cost penalizes larger loads and decreases in the number of active servers as illustrated in Figure 1. The unconstrained problem is as follows:

$$\min \sum_{z=1}^{Z} \left(c(k_z, \gamma_z) t_0 + H_z + B_0 (b_{z0})^+ \right). \tag{4.2}$$

Notice that, for a given time period, given a decision on $k(t)$ and $\gamma(t)$, the operating and booting costs until the next decision time are immediately determined. However, the holding cost still needs to be estimated. In order to assess H_z for the immediately following period, we need some information about the arrival process. To obtain that, we propose to use the wavelet based forecasting procedure described below.

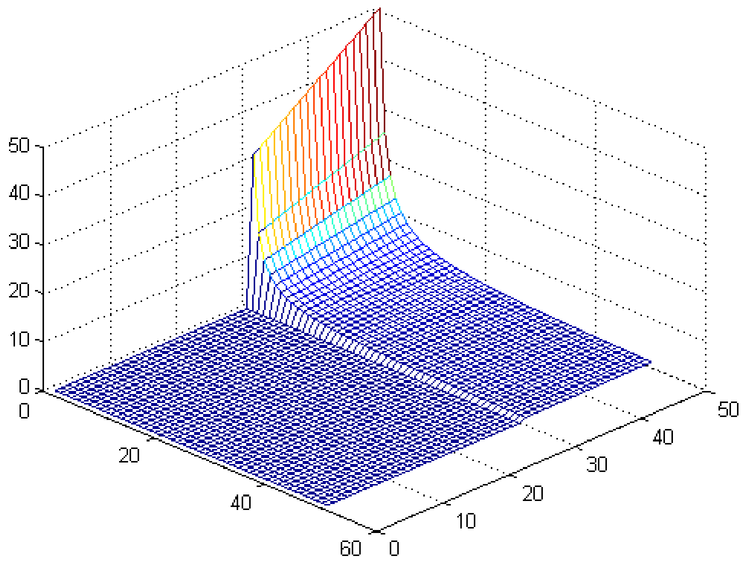

Fig. 9. Holding cost.

2. Forecasting

Individual application traffic is usually approximately periodical in terms of number of arrivals per unit time, with period size varying depending on the specific application. For instance, even if traffic to a business application such as stock trading is not perfectly identical during two consecutive days, there are some behavioral patterns that remain constant. For example, a decrease in traffic during the night, and predictable peak hours are usually encountered. Therefore it is reasonable to assume that we have a system that processes an arrival stream that exhibits nearly periodic behavior, but may be highly variable and self-similar. As the frequency and the number of servers may be adjusted every previously determined amount of time t_0, let us denote by $t = 0$ the beginning of the process, and let $t_i = it_0$ be the decision epoch for the $k(t)$ and $F(t)$ adjustments. Assume that the arrival process is nearly periodical with period Nt_0. Moreover, let us call λ_i the sum of the sizes of the requests between t_i and t_{i+1}. Our objective is to forecast λ_i in order to feed a decision-making scheme that provides capacity dynamically using the forecasting information and observations of the system load.

The idea is to divide the forecasting into two parts. First, a *global*, long term prediction that is computed every Nt_0 seconds, and is left unchanged during that time. The vector of global forecasts $\hat{g}$ provides information about global trends (for instance, a rate drop every night or a rate increase every Monday). Since this behavior tends to be relatively stable over time, it may be predicted by averaging the last few periods or by some other type of smoothing forecasting such as moving averages or ARMA. Secondly, we are going to compute a *local*, short term forecast that will be updated every t_0 seconds and contain information only about the following few periods. This local term, $\hat{\alpha}$, represents a deviation from the global trend (for instance, a period of increasing popularity of a certain website or service).

The main idea of the forecasting method is to divide the arrival process into periodic and a non-periodic factors, i.e., for the $i-th$ arrival during a specific period, let us denote our estimator:

$$\hat{\lambda}_i = \hat{g}_i \hat{\alpha}_i.$$

The justification to decompose the arrival rate using factors instead of additive terms comes from the fact that the variation in the traffic of a server most of the time features some type of cyclic behavior. For instance, even in the presence of a sudden increase in traffic for a web-based news site because of an important event, it is very likely that traffic late at night will be much smaller than traffic around 8:00 p.m.. In this scenario, the percentage of traffic increase is likely to be more stable, thus easier to predict, than the difference between the average and the actual rate. Forecasting a factor has the inconvenience of the possibility of obtaining negative meaningless values; in order to avoid such inconvenience, we propose to analyze the time series $\log(\alpha_i)$ instead, with the additional advantage that the forecasting will be equally sensitive to a reduction of a factor of the traffic than to an increase of the same factor of the traffic. Finally, even if for long enough periods, the likelihood of a negative infinity term (when the α_i is zero) is very small, we add an artificial lower bound to our forecasts and observations.

Let us assume that for the $i-th$ epoch during the $j-th$ period, we have observed the system and want to predict α for the next few periods $i+1,...,i+k$. According to Renaud et al. [65] the use of wavelet based forecasters for self-similar or long range dependent phenomena produce better results than the traditional time series forecasting methods. Therefore, we implement a Haar wavelet based forecaster (described in section 4.2.1) to obtain information about α by forecasting each frequency separately. Given the presence of noise in the α_i time series, we add a *safety factor* obtained from the data that will keep the system from being overloaded during the time slot. Figure 10 illustrates the forecasting procedure for a sample trace. The top graph is the actual arrival rate to a real life server during the third week of data. The second graph is the plot of $\hat{g}$, the global forecasting that in this case is the denoised (see section 4.2.2) average of the first two weeks of data. The third graph is the forecasted arrival rate, obtained by the procedure described in detail in section 4.2.1, and the last graph is the difference between the forecasted and the actual values. As seen in Figure 10, the forecasting error is small compared to the total load of the system. For instance, the $90-th$ percentile of the error is a small fraction of the average load of the system.

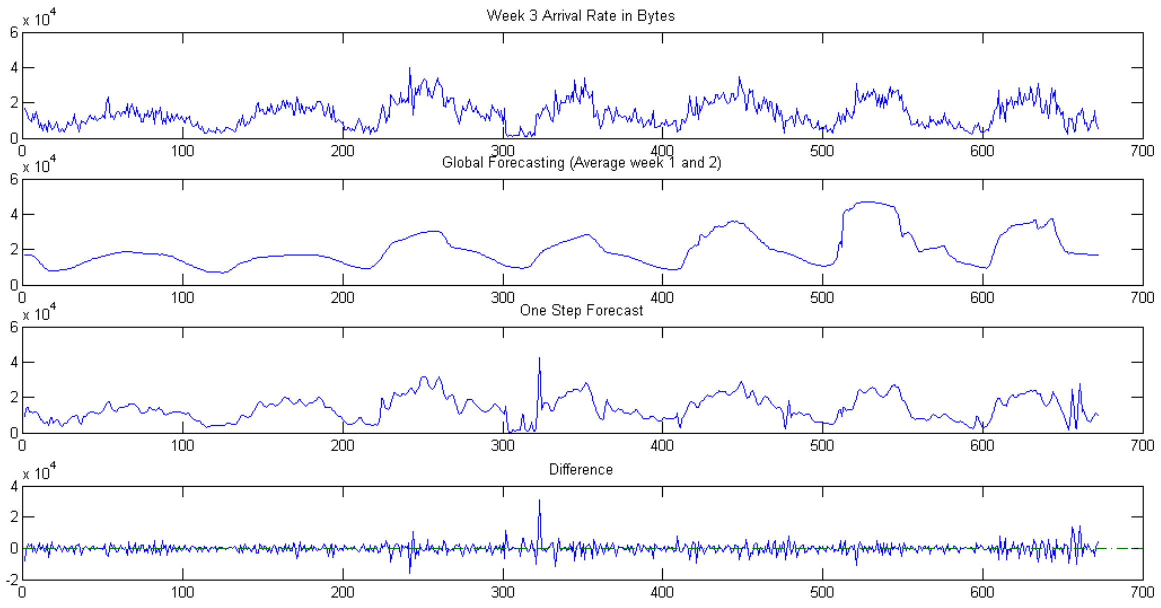

Fig. 10. NASA Trace Forecast. 90th perc.=3000 95th Perc.=4250.

a. Forecasting with the Haar Wavelet

Given a vector **X** that represents the arrival rate in the previous M time segments, where $M = 2^J$, for some j. The redundant wavelet transform, decomposes the signal **X** as:

$$X_i = c_{J,i} + \sum_{j=1}^{J} w_{j,i},$$

where $\mathbf{c}_J$ is a coarse or smooth version of the original signal, and $\mathbf{w}_j$ represents the details at scale j. If we denote $c_0 = \mathbf{X}$, then the Haar wavelet approximations at each level are given by:

$$c_{j+1,i} = \frac{1}{2}(c_{j,i-2^j} + c_{j,i}),$$

and

$$w_{j+1,i} = \frac{1}{2}(c_{j,i-2^j} - c_{j,i}).$$

Notice that at any i, the algorithm uses only the information of previous entries in order to compute the decomposition.

The key idea in this method of forecasting is that we will forecast the smooth approximation as well as the details at each level separately and combine that information to obtain a prediction of the original time series. This forecasting method is particularly well suited for self-similar processes such as data arrival at data centers because the frequency decomposed streams are smoother than the complete time series.

Figure 11, presents the decomposition of the first week of the NASA Trace. The top graph is the actual arrival rate per period. The coarse approximation, labeled A5 (standing for approximation at the fifth level), presents the coefficients found in $\mathbf{c_5}$ and is much smoother than the original time series. Subsequent graphs present the details contained in the different levels (w_i, $i = 1,...,5$), and provide more information about the behavior of the process in shorter periods of time. It can be seen that the details in levels 3, 4 and 5, labeled D3, D4 and D5, have a smooth behavior as well. In contrast, the high frequency levels

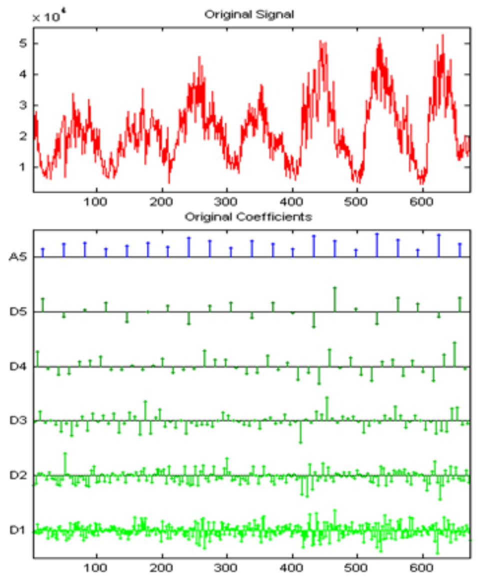

Fig. 11. NASA trace coefficient decomposition.

(labeled D1,D2) do not have a clear trend, but most of the information contained in them is noise that will be removed by a denoising procedure described in the next subsection.

b. Signal Denoising Using Wavelets

Given a signal **X**, we are interested in distinguishing real trend changing behavior against random uncorrelated variability. This process is known as *signal denoising* (a detailed exposition on this matter may be found in Percival [64]). The main idea is that the noise in a signal produce a large amount of small details coefficients contained in the vectors $\mathbf{w_i}$, $i = 1, \ldots, J$. Therefore, the method eliminates small coefficients from the decomposition to obtain a smother signal. Wavelet researchers have developed thresholds that will determine how small are the coefficient that should be suppressed at each scale in order to obtain a signal that reflects the trend more accurately as seen in Figure 12 [88]. In this Figure, we have the original signal information in the left column and the denoised version in the right column. The lower graphs represent the signal decomposition into approximation and details, as in Figure 5, but in the left coefficient graph, we see dashed horizontal lines that represent the established thresholds. On the right side, all values between the horizontal lines have been eliminated, and the reconstructed signal presents a much clearer trend. The central graph is the difference between the original and denoised signals. It is worth noting that the mean of the difference is approximately zero, and its value, at least seems to be independent of time.

It is to be determined, however, how this threshold should be established. There are some standard methodologies that are more or less effective depending on the type of time series. In particular, there are two basic types of thresholding: the hard non-smooth thresholding that eliminates all the coefficients below a certain level and the, smooth but biased, soft thresholding that eliminates the small coefficients, but which also reduces the larger ones by the amount given by the threshold. In our experiments we use hard level independent thresholding.

3. Control Algorithm

The solution we present here for the dynamic capacity allocation problem is a myopic stochastic-dynamic programming algorithm, which outputs the number of servers and frequencies that should be used in the next two periods by finding the optimal decision that minimizes the expected cost over the next three periods. The algorithm uses the information about the state of the system (number of requests in the buffer), the predicted load arrival rates and the number of active servers. Thus, for any period z_0, we define the following reduced version of the problem in equation 4.2:

$$C^{l*}_{z_0 i} = \min \sum_{z=z_0+i}^{z_0+l} \left(c(k_z, \gamma_z) t_0 + H_z + B_0 (b_{z0})^+ \right),$$

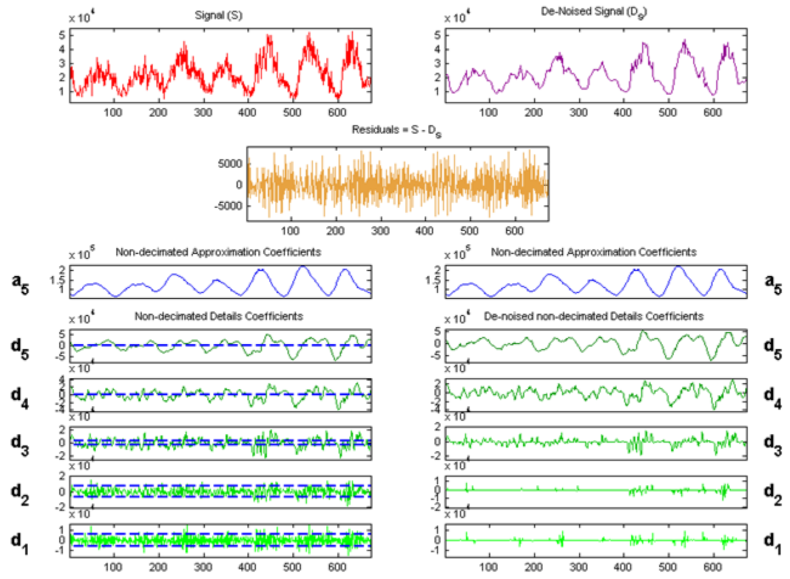

Fig. 12. NASA Trace. Denoised signal.

where $i \leq l$ is a positive integer. Clearly, $C^{l*}_{z_0 i} = min\{c(k_{z_0+i}, \gamma_{z_0+i})t_0 + H_{z_0+i} + B_0(b_{z_0+i0})^+ + C^{l*}_{z_0 i+1}\}$, which provides us with the familiar dynamic programming formulation, and, as explained at the end of this section, allows us to estimate $C^{l*}_{z_0 i}$ for consecutive time intervals.

As mentioned in the introduction, we are going to assume that the behavior of the requests arrival stream is nearly cyclical with a period equaling one week. This is a natural assumption since in many cases the expected behavior changes on different days of the week but the weekly pattern is reasonably close. As we mentioned in the problem statement, the difference in behavior between weeks because of an increasing (decreasing) trend and/or sudden spikes should be reflected in the local forecast term. The forecasting macro-algorithm is as follows:

```
for week_number=1:N {
  Update_Global Forecasting();
        for i=1: Number_of_periods_per_week {
         Update Local Forecast(Global Forecast, Obs Rate, i);
         Observe load of the system();
         Make_Decision(load);
         }
}
```

The following is the description of the proposed local forecasting algorithm:

```
Update Local Forecast(Global_Forecast, Obs_Rate, i) {
  Compute observed alpha=Observed_Rate/global;
        if i<2^{Forecasting level}){
         Extend alpha left to 2^{Forecast level};
         }
        if (i>=2^{Forecasting level})){
         Take the last 2^{N} terms;
    }
        Decompose log(alpha) at the selected level;
        Linear regression at each level for the last 3 terms;
        Reconstruct log(alpha);
        Take the last term;
        Return global*alpha;
}
```

The decision-making rule is given by:

```
Make_Decision() {
        for (frequency,b) in (Gamma x Number_Servers)^p{
                ComputeMinimumCost(P,load);
        }
}
```

The function ComputeMinimumCost(P) estimates the minimum expected operating cost for P periods in the future given a sequence of decisions. For each period and decision, the cost is computed as the sum of the booting cost, the operating cost, and estimate of the holding cost. This estimate is constructed as follows. First we computed the holding cost from the beginning of the period until the load becomes zero for the first time. This is estimated as:

$$\int_0^T h(n_0 + (\lambda - k\gamma)t)^+ dt.$$

Secondly, if $T > \frac{n_0}{k\gamma - \lambda}$, we add the holding cost of the average number of requests in the system times the remaining time after the load reaches zero, i.e., for $\frac{T - n_0}{k\gamma - \lambda}$ seconds. This number of requests is computed as:

$$E[n(t)] = W\frac{\phi}{\beta\gamma} = \left(\frac{\phi}{\beta\gamma} + \frac{\alpha_m\phi}{\beta\gamma}\left(\frac{1}{1-\rho}\right)\left(\frac{C_a^2 + C_s^2}{2m}\right)\right)\frac{\phi}{\beta\gamma},$$

where $\alpha_m = \rho^{\frac{m+1}{2}}$, $\rho = \frac{\lambda\phi}{m\beta\gamma}$, ϕ is the mean request size (see Bolch et.al. [89]), and C_a^2 and C_s^2 are the observed squared coefficient of variation of the inter-arrival times and service times respectively. After estimating the minimum cost of any sequence of three decisions, the controller picks the minimum cost decision and determines the proper frequency and number of servers.

D. Numerical Evaluation

The numerical evaluation was made with the available web requests traces in the Internet Traffic Archive [90] that are longer than two weeks. Unfortunately, the number of available traces of this length is very low since most publicly available traces are only a few hours long. For each of the traces, the first week was used as training data for the algorithm. From the first week data, the mean and standard deviation of the request size and inter-arrival times were computed. Additionally, a denoised version of the first week data was taken as the global forecast. The experiments were performed using the second week data and several allocation strategies were evaluated. For each of the traces, we computed the energy expenditures of having a different number of servers turned on all the time, our algorithm performance using only global forecasting, and our algorithm using the forecast methodology proposed in this chapter. The eligible decisions made at each decision epoch are increasing $k(t)$ any value between 0 and $K-k(t)$, but decreasing its value at most one unit. The frequencies can be chosen arbitrarily.

For each experiment we report the total energy cost as well as the proportion of requests served in the target time. Table V contains the parameters used for the numerical experiments.

In the Table VI, K stands for the total number of available servers. The costs presented here are average amount of dollars per second. For analyzing the α time series, we used hard thresholding, with the Heursure procedure described in the Donoho-Johnstone methods [64].

As we claimed earlier, after the numeric experimentation, we can see that our capacity provisioning scheme addresses two major points in capacity provisioning for data centers. First, it reduces the energy consumption by utilizing the well known mechanisms of DVS and powering down the servers; second, it has the capacity to adapt to sudden changes

Table V. Settings of the numerical experiment.

Parameter	**Value**
Available Frequencies	$(1.4, 1.57, 1.74, 1.91, 2.08, 2.25, 2.42, 2.6)$
Power	$(60, 63, 66.8, 71.3, 76.8, 83.2, 90.7, 100)$
Cost (cents/KWH)	10
B_0 (cents/Boot)	0.525
T_{boot}	90 sec
Decision Period (t_0)	900 sec

in traffic behavior and forecasting mistakes. We would like to emphasize that any energy saving scheme that is not highly adaptable is not likely to be adopted in practice because data center managers do not want to risk failing to meet the contractual terms with their customers.

We compare the performance of our scheme against fixed capacity schemes in terms of consumed energy and observe that for the first trace, the consumed energy is more than 40% lower than the minimum of the acceptable fixed capacity schemes. In this case we want to highlight that the energy savings are significant without compromising on the customers' perception of the system operation. We divide the energy used by each method by the energy used to operate all the servers during the whole experiment, and substract this quantity from 1 to obtain the energy savings. Therefore, using the total available capacity will lead to zero energy savings, and reducing capacity will increase the energy usage, but if the reduction is high enough, the QoS constraint will not be met as in rows three and four.

For the second trace, the difference in energy consumed is very low compared to the energy used by 4 servers. However, we can note that even if for this experiment we

Table VI. Numerical results.

Trace	**Week**	**K**	**Method**	**Energy Savings**	**QoS met**
NASA [90]	2	12	6 Servers	**0**	TRUE
	2	12	5 Servers	**0.166546**	TRUE
	2	12	4 Servers	**0.332103**	FALSE
	2	12	3 Servers	**0.49584**	FALSE
	2	12	ARA	**0.36613**	TRUE
	2	12	Global Forecasting	**0.38382**	FALSE
ClarkNet-HTTP [90]	1	12	6 Servers	**0**	TRUE
	1	12	5 Servers	**0.166667**	TRUE
	1	12	4 Servers	**0.317556**	TRUE
	1	12	3 Servers	**0.489062**	FALSE
	1	12	ARA	**0.374921**	TRUE
	1	12	Global Forecasting	**0.252702**	TRUE

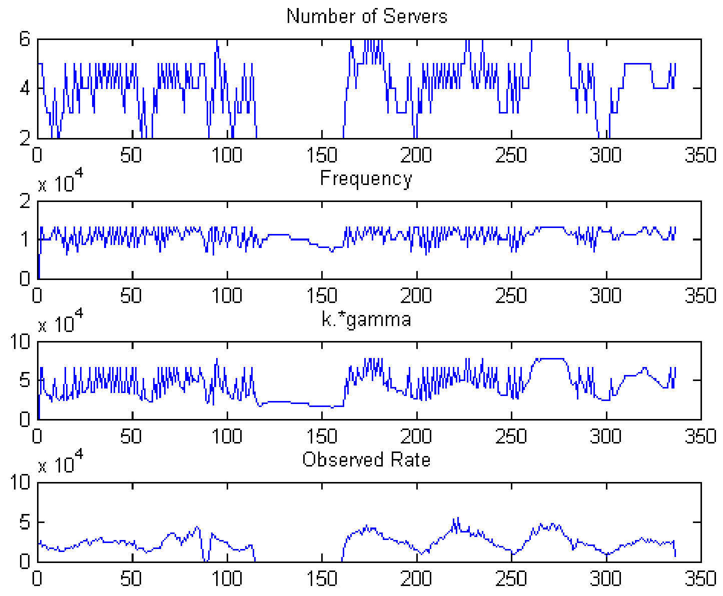

Fig. 13. Allocated capacity.

can obtain acceptable results with a tight fixed capacity, a system manager would not risk operating the system with a tight capacity because a sudden traffic change would potentially not allow the data center to meet the SLA.

The global forecasting schemes performance, in the first case, fails to meet the QoS constraint, and in the second case, the desired performance is met, but the energy consumed is considerably higher than in ARA. This is caused by poor quality forecasting that leads the scheduler to make bad decisions.

Figure 13 illustrates how the system behaves during the experiment with the NASA trace. The first part of the first figure shows the number of servers during each decision period. The second represents the individual frequencies. The third picture is the aggregated capacity allocated and is calculated as the number of servers times the frequency. The last graph is the load of the system. It is clear from the picture that the behavior of the allocated capacity resembles the system load; therefore, the allocation scheme increases capacity when the load increases and vice versa.

One more interesting fact is the impact of the forecast quality in the overall provisioning performance. In the experiments, we compared the performance of our scheme against a similar decision-making rule but with a lower quality, long term forecast. While in the first trace, the spent energy is lower for the global forecast methodology, it does not satisfy the SLA. Similarly, in the second trace the SLA is met, but the energy is higher than the cost of the algorithm we proposed.

E. Conclusions

With the methodology presented in this chapter we addressed two significant problems in the data center industry by presenting a sensitive resource provisioning strategy. The strategy reduces the energy consumption in the operation, and is safe enough to be implemented without the concern of application collapses when requests arrival patterns present sudden changes. This problem is particularly challenging because the system behavior is highly variable, and the QoS constraints are TSF type. The scheme is the result of the combination of three techniques: wavelet data analysis, proactive resource provisioning, and feedback control.

We considered and analyzed the advantages and drawbacks of alternative provisioning policies. Static policies may provide acceptable results for specific cases, but by their very nature, those polices fail to adapt to system changes or load variations. Purely proactive policies do not react well to overloads of the system caused by forecasting errors, and purely reactive policies fail to provide capacity for potentially foreseeable load changes. We tested the performance of our decision-making scheme being fed by purely global forecasting data, and observed that the lack of immediate behavior information has a negative effect on performance and energy expenditures.

As seen in the numerical experimentation, the combination of reactive and proactive decision-making may produce significant savings in the operation of data centers providing acceptable performance and reliability. Moreover, the implementation of quality forecasting techniques into the decision-making process yield additional savings. In this chapter we presented a novel forecasting techniques tailored specifically for (potentially self-similar) processes that are not cyclical but exhibit some type of cyclic behavior even when there are pattern deviations. This makes the decision scheme very adaptable to arrival stream behavior changes and/or disruptions.

CHAPTER V

CONSOLIDATED PROVISIONING

Chapters III and IV focus on real time provisioning strategies to solve the single application energy aware capacity allocation problem in data centers from a theoretical and practical perspective. Besides the operational decision-making, we may consider two more ways to further improve the system's energy efficiency. First, there are decisions, such as data center maximum capacity, hardware selection, and cooling equipment purchase that are made mainly when the facility is built. Secondly, there are decisions, such as the ones we analyze in this chapter, that should not be made in an hour to hour or day to day basis, but that once implemented, will not usually be changed for a considerable period of time, such as weeks or months.

In this chapter, we allow several applications to be run in each of the servers. We are going to focus on selecting the applications that are going to be available to run in each of the servers. This is not considered as an operational decision because enabling a server to run a new application may require a long installation procedure. For allocating applications, we partition the application set and identify the elements of the partition as a single composed application, or meta-application. This portion of the dissertation is the one that makes direct use of the potential economies of scale, resulting of having all the different applications being hosted in the same facility. This may allow us, for instance, instead of having three different applications running in three separate servers (as is the common practice in the industry), we may have all the applications running in all three servers. This would mean that in the original case all three servers have to be powered up all the time, but in the latter case we only need to power up as many servers as necessary, thus creating immediate energy savings. In addition, this strategy would reduce the computational capacity in times of low combined demand, thus saving energy. As in the previous chapter, we assume that the data center should meet a stringent SLA that may be homogeneous or heterogeneous across the different applications, and we propose techniques that make use of and complement the algorithms developed in Chapter IV. Similarly, our ultimate objective is to reduce the data center energy consumption, but in this case we achieve that objective by solving a different type of optimization problem: reducing the aggregated load variance by properly partitioning the set of applications.

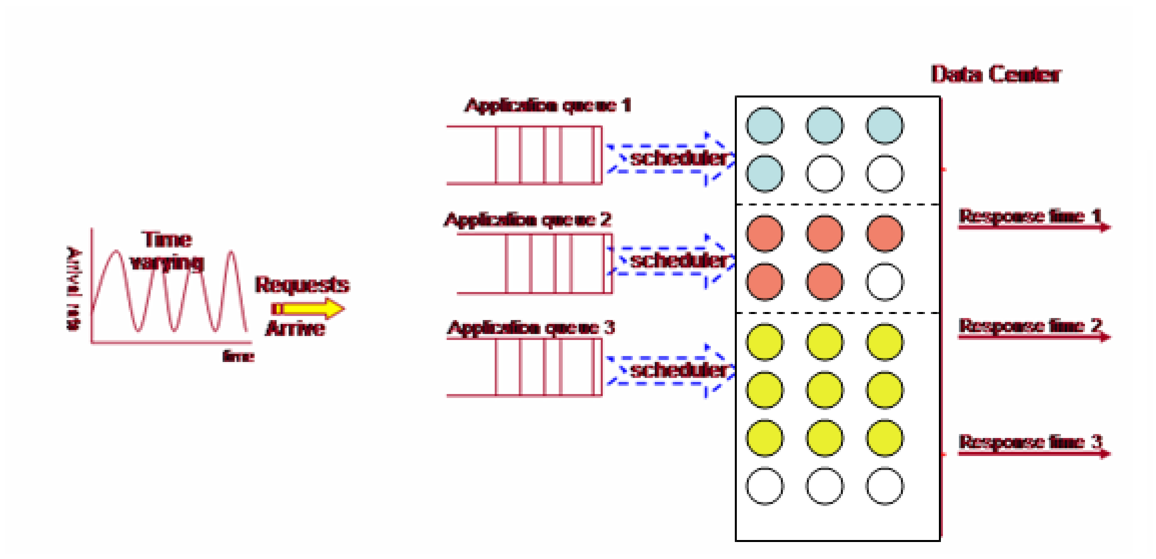

Fig. 14. Data center model.

A. Problem Description

Let us consider a data center that serves a certain fixed number of applications, as the one in Figure 14. Assume that the servers in the data center are identical, and may be deactivated or activated to run at a finite set of eligible frequencies. Suppose, without lost of generality, that any subset of the applications may be hosted in the same machine (otherwise, we can divide the problem into facilities that include only applications that may run with each other and analyze them separately). Assume moreover, that the data center management does not have knowledge of the future load except observed load history. The objective of the data center administrators is: first, given the facility and equipment, it should serve requests balancing energy usage and hardware reliability for reducing the facility costs and environmental impact, and secondly, it should meet the service standards consigned in the SLA.

We model this problem as a queueing system with K homogeneous servers able to host any number of applications, able to being turned on and off, as well as the ability to run any of the hosted applications at any of a finite predetermined set of frequencies $\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_{|\Gamma|}\}$. The costs incurred by the data center are represented by the energy required to power the servers, and an energy, reliability and risk cost associated with booting the machines. According to Elnozahi et al. [54] and Chen et al. [8], the energy consumption as a function of the frequency has the form $P(\gamma) = P_{fix} + P_f \gamma^3$, where P_{fix} and P_f are equipment specific constants, and the cost of turning a server on is assumed to be a constant B_0.

It is well known (see for instance Chen et al. [8] and Chapter IV) that dynamic capacity allocation, using powering servers up and down and DVS for individual applications, has the potential of reducing significantly the energy consumption of the overall system. However the frequent changes in capacity may increase the likelihood of not meeting the SLA and causes wear and tear on the equipment due to the repeated booting cycles. Both phenomena share a common cause: the high variability of the inter-arrival times and requests sizes that come into the system. Our objective is to properly combine subsets of applications in such a way that the variability of the aggregated load is reduced, and the capacity allocated to serve those applications will become more stable. This way, the dynamic capacity allocation scheme will reduce the number and size of capacity adjustments, and reduce the risk of not meeting the service constraints as a consequence of capacity allocation errors or sudden load changes. In other words, we attempt to achieve an improvement in the operation cost by solving an auxiliary problem that determines how to cluster applications in such a way that the aggregated variability is reduced.

B. Problem Formulation

Let us consider a set of M single class First Come First Served queueing system as the one in Figure 14, where each of the classes represents a specific application. The system has K identical available servers able to process any of the requests classes. At any given time t, each server may be in on (active) state or off (inactive) state; moreover, when a server is on, it can run at any frequency $\gamma \in \Gamma = \{\gamma_1, \gamma_2, ..., \gamma_{|\Gamma|}\}$, but all active servers devoted to the same group of applications should run at the same frequency. This assumption, analogous to the one made on Chapter IV, is used for mathematical tractability and presentation convenience. Let us denote by $k_m(t)$ the number of active servers dedicated to (meta-)application m at time t, so $k_m(t) \in \{1,2,...,K\}$ and $\sum_{m=1}^{M} k_m(t) \leq K$, and by $F_m(t) \in \Gamma$ the frequency of the active servers running the application m.

Requests arrive at the system according to an arbitrary process. Let A_n be the arrival time of the $n-th$ request, S_n the amount of work required to serve it, and $C_n \in \{1,2,...,M\}$ its class. Let us divide the time horizon into contiguous time intervals of length t_0, and define the load the system receives from the $m-th$ application during the $z-th$ time interval as

$$L_m(z) = \sum_n S_n 1_{\{n:(z-1)t_0 < A_n < zt_0\}} 1_{\{n:C_n=m\}}.$$

This produces M time series that reflect the amount of work the system receives from each application during the different time slots. We define the covariance of the time series L_{m_1} and L_{m_2} as (see Onnella et al. [79]):

$$cov(L_{m_1}, L_{m_2}) =< L_{m_1} \cdot L_{m_2} > - < L_{m_1} >< L_{m_2} >,$$

where the operator $< \cdot >$ returns the expected value of the argument.

We are interested in aggregating several applications, i.e., several time series, in order to reduce the variability of the system. In the rest of this Chapter we define the variance of the system as the sum of the variances of the meta-applications formed when the individual applications are grouped. Formally, for the subset $S = \{m_1, m_2, ..., m_{|S|}\}$, we define the variance of the aggregated time series $\sum_{i=1}^{|S|} L_{m_i}$ as:

$$var(S) = \sum_{i=1}^{|S|} \sum_{j=1}^{|S|} cov(L_{m_i}, L_{m_j}),$$

and the variance of the system will be the sum of the variance of the element of the partition.

If all the applications are served separately, the total variance of the system is simply the sum of the variances of the individual applications. However, given that some covariances may be negative, it is possible that $var(S)$ is lower than the sum of the variances of the individual applications. For instance, suppose there are eight applications hosted in a particular data center with ten servers $\{a, b, c, ..., j\}$. We can assign four applications, say $\{1, 2, 3, 4\}$, to servers $\{a, b, c\}$, two applications, say $\{5, 6\}$, to servers $\{d, e, f, g\}$, and four applications, $\{7, 8, 9, 10\}$ to servers $\{h, i, k\}$. This way, it will not be necessary to have at least 8 servers running all the time, and if we can make the load over each group of servers as stable as possible, we can provision energy more efficiently. In order to determine if this grouping or any other is appropriate, we need to find the optimal partition of the applications such that the total variance of the system is minimized:

$$\min_{\mathcal{P} \in \mathbb{P}} \sum_{S \in \mathcal{P}} var(S),$$

where $\mathcal{P}$ is a partition in the set $\mathbb{P}$ of all possible partitions of the set $\{1, 2, ..., K\}$.

We assume that S_n is not observable until the service starts, so all the information available about the arrivals at the system at time t is the number of requests of each class m in the buffer $n_m(t)$. Let us suppose that there is a rule (as the one in Chapter IV) to select individual class $k_m(t)$ and $F_m(t)$ dynamically based on the observed $n_m(t)$ and load history, so that the long-run average energy consumed is minimized subject to satisfying a Quality of Service constraint for individual applications.

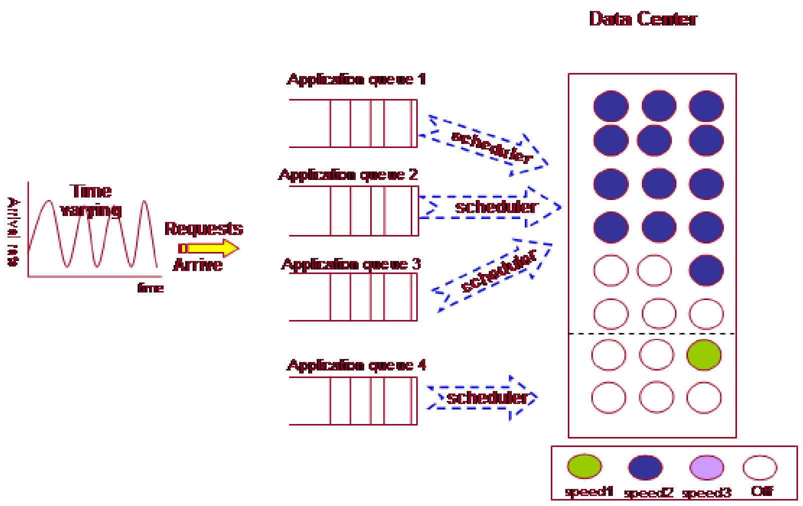

Fig. 15. Some applications are grouped.

Assume that powering up a server needs T_{boot} seconds, and has a fixed cost B_0 caused by the wear on physical components, such as the hard drives, due to the booting process and the energy required to boot the server up. Running k servers at frequency γ, costs $c(k,\gamma)$ per unit time, where the function $c(k,\gamma)$ is assumed to be increasing in both components. Moreover, the system manager is required to meet a previously specified SLA that bounds the fraction of the requests of type m that may stay in the system longer than (previously determined) R_m seconds.

Powering up a server requires energy, produces significant hardware wear and tear, and risks possible failures such as unsuccessful boots or machine failures; therefore, it is undesirable to allow the algorithm to turn servers on and off arbitrarily often. In order to avoid that, for the remaining of this chapter (as we did in Chapter IV), we assume that both the booting and frequency decisions are going to be made and executed every t_0 seconds. Therefore, every t_0 seconds, the system administrator chooses the values of $n_m(t)$ and $k_m(t)$, and those remain constant during the next t_0 seconds.

As we mentioned earlier, we are assuming that for each period z, and each value of $n_m(t)$ and possible predictions of the future load, there is a decision $k_m(t)$, $F_m(t)$ that is intended to minimize the individual application energy consumption. Therefore, for all applications $\{m_1, m_2, ..., m_{|S|}\}$ contained in a given element of a partition $S \in \mathcal{P}$, let us define the aggregated frequency dedicated to that group as $\mathcal{F}_S = \sum_{i=1}^{|S|} k_{m_i} F_{m_i}$, and the proportion of the aggregated frequency allocated to application m as $p_m = \frac{k_m F_m}{\mathcal{F}_S}$. Those proportions represent the fraction of the overall system effort used to serve the requests belonging to each individual type, and are going to be used by the algorithm described in Chapter IV, section C.4, where we will allocate the aggregated frequency to a group of k_S servers running at identical frequencies F_S.

We intend to find a partition such that the energy expenditures of running the grouped applications described by (see Chapter IV)

$$\sum_{S \in \mathcal{P}} \sum_{z=1}^{Z} \left(c(k_{zS}, \gamma_{zS}) t_0 + B_0 (b_{zS0})^+ \right)$$
$$\text{s.t. } \forall n \forall m \left(C_n = m \Rightarrow P\left(D_n - A_n > R_m\right) < \alpha \right),$$

is lower than the one for applications being served in independent servers, written as:

$$\min \sum_{m=1}^{M} \sum_{z=1}^{Z} \left(c_m(k_{zm}, \gamma_{zm}) t_0 + B_0 (b_{zm0})^+ \right)$$
$$\text{s.t. } \forall n \forall m \left(C_n = m \Rightarrow P\left(D_n - A_n > R_m \right) < \alpha \right),$$

where t_0 is the time length of the periods, $b_{zm0} = (k_m(zt_0) - k_m((z-1)t_0))$ represents the number of servers to be turned on/off in $z-th$ period, α is a constant previously designed in the SLA, and P is the fraction of the requests that meet the condition in parenthesis. The furthest left sum on the first equation is indexed by $S \in \mathcal{P}$, the subsets of the partition, so the frequency and number of server allocations are made subset-wise. In contrast, in the second equation, the index is m, the application number, so here both frequency and number of servers are allocated individually. Although we used the quality of service constraint of the types described in Chapter IV, the same equations with different types of constraints are valid.

We will restrict our attention to a specific group of applications $\{m_1, m_2, ..., m_{|S|}\}$, and assuming there is a rule to select k_S and F_S. This queueing system may be described as a dynamic system, where the state is described by the number of jobs of each class in the queue $n_{m_i}(t)$, and the number of active servers $k_S(t)$. If at time t a new request arrives at the system, and there are fewer than $k_S(t)$ requests being served (i.e., there is an empty active server), then the request goes directly to be served. Otherwise it goes to the last place in the queue. When a service is completed, the first request in the queue starts service with a completion time $\dfrac{S_n}{\beta F_S(t)}$ where β is a constant that represents the proportion between the amount of work performed and the running frequency. At an arbitrary time t, the system state will change to $k_S(t)$ and $n_m(t)+1$ if the next request is of class m and comes to the system before a completion is achieved. Otherwise, the system will go to state $k_S(t)$ and $n_m(t)-1$, after a time period equal to the minimum completion time.

As described, the system dynamics depends on the values of $n_m(t)$, $F_S(t)$ and $k_S(t)$ as well as the arrival process. As explained in Chapter IV Section B, large values of $n_m(t)$ increase the likelihood of some request's time in the system being larger, making more likely The violation of the QoS constraint, and higher values of $k_S(t)$ and $F_S(t)$ may decrease the value of $n_m(t)$. However, increasing the number of servers and/or the frequency implies greater energy expenditure of the overall system. Hence, our objective is to balance the trade off among the energy expenditure, reliability and the QoS constraints by controlling this highly variable system.

C. Methodology

We propose an algorithm that allows us to find an optimal or near optimal partition of the applications arriving at the data center. In order to do so, we propose two approaches. The first one applies whenever the quality of service constraints are homogeneous among all the applications; the second one applies when the quality of service requirement varies from one application to the other. Since requests arrival at a data center present some type of cyclic behavior (see Rincon Mateus and Gautam [91]) with period size T, we will assume that we have a training period of length T used to obtain information regarding the load time series behavior and interactions.

1. Computation of the Observed Covariances

Given the training data, the observed covariance during the training period is computed as:

$$cov(m_i, m_j) = < L_{m_i} \cdot L_{m_j} > - < L_{m_i} >< L_{m_j} >,$$

if the quality of service constraints is homogeneous. Otherwise, we proceed in the following way: assuming that the individual application capacity allocation method receives perfect information about future system loads (we do this using the training data, so the information on the previous time slots is perfect), we obtain the allocated aggregated frequency for each period z and each individual application m. With this information, we can form M new time series $\mathcal{F}_m = k_m F_m$, $m = 1,...,M$, with which we will compute the variances in a similar fashion:

$$cov(m_i,m_j) =< \mathcal{F}_{m_i}, \mathcal{F}_{m_j} > - < \mathcal{F}_{m_i} >< \mathcal{F}_{m_j} > .$$

Computing the covariances of the aggregated capacities, instead of computing the covariances of the loads, allows us to compare applications with heterogeneous requirements. In other words, even if the loads are equal, one application with a very high quality of service requirement will have a different capacity allocation than one with lower service level; as a consequence, a direct load comparison will not necessarily reflect well the relation between the capacity allocated to both applications, and we are interested in obtaining stable allocated capacity. Clearly, we could apply the second algorithm when the QoS constraint are homogeneous across the different applications, but computing the variance of the arrival rates is much less computationally expensive. This methodology training data analysis has a much longer run time because it must determine the capacity allocated to each time slot in the training period.

2. Design of the Data Structures

Let us consider the regular variance covariance matrix obtained by either procedure described in subsection C.1:

$$\Sigma = \begin{bmatrix} Var(1,1) & Cov(1,2) & \cdots & Cov(1,m) \\ Cov(2,1) & Var(2,2) & \cdots & Cov(1,m) \\ \vdots & \ddots & \cdots & \vdots \\ Cov(m,1) & Cov(m,2) & \cdots & Var(m,m) \end{bmatrix}$$

If the applications are served separately, the total variance of the system, denoted by $Var(D)$, is simply the trace of the matrix:

$$Var(D) = Tr(\Sigma) = Var(1,1) + Var(2,2) + ... + Var(M,M).$$

If the applications were grouped, for instance, as $\{1,2\}$,$\{3\}$ and $\{4,5\}$, then the total system variance would be given by the sum of the elements on the diagonal block entries of the following matrix:

$$\Sigma = \left[\begin{array}{cc|c|cc} Var(1,1) & Cov(1,2) & Cov(1,3) & Cov(1,4) & Cov(1,5) \\ Cov(2,1) & Var(2,2) & Cov(2,3) & Cov(2,4) & Cov(2,5) \\ \hline Cov(3,1) & Cov(3,2) & Var(3,3) & Cov(3,4) & Cov(3,5) \\ \hline Cov(4,1) & Cov(4,2) & Cov(4,3) & Var(4,ChapterIV) & Cov(4,5) \\ Cov(5,1) & Cov(5,2) & Cov(5,3) & Cov(5,4) & Var(5,5) \end{array}\right],$$

which equals to $Var(1,1) + Var(2,2) + Var(3,3) + Var(4,4) + Var(5,5) + 2Cov(1,2) + 2Cov(4,5)$.

In order to obtain different partitions we define a sorted boundary vector of length m, $\mathbf{b} = (b_1, b_2, ..., b_M)$, where $b_1, b_2, ..., b_M$, that defines the placement of the horizontal and vertical lines of the matrix. For instance, the first set in the partition is $\{1,2,...,b_1\}$ and the second is $\{b_1 + 1, ..., b_2\}$. In the previous example $\mathbf{b} = (3,4,4,4,4)$, but this representation is not unique. Additionally, we define a permutation of the rows and columns of the matrix, represented by a permutation matrix Q, and implemented as $Q^t \Sigma Q$. This permutation allows us to change the order of the rows and columns so we can obtain subsets of non-consecutive elements. Using this notation, each possible grouping of the arriving applications may be represented (not uniquely) by a matrix P and a vector $\mathbf{b}$. If for instance

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and $\mathbf{b} = (3,4,4,4,4)$, then the resulting grouping will be:

$$\Sigma = \left[\begin{array}{cc|c|cc} Var(5,5) & Cov(5,2) & Cov(5,3) & Cov(5,4) & Var(5,1) \\ Cov(2,5) & Var(2,2) & Cov(2,3) & Cov(2,4) & Cov(2,1) \\ \hline Cov(3,5) & Cov(3,2) & Var(3,3) & Cov(3,4) & Cov(3,1) \\ \hline Cov(4,5) & Cov(4,2) & Cov(4,3) & Var(4,4) & Cov(4,1) \\ Var(1,5) & Cov(1,2) & Cov(1,3) & Cov(1,4) & Var(1,1) \end{array}\right].$$

With this notation the problem of finding an optimal partition to minimize the variance may be stated as:

$$\min_{Q,\mathbf{b}} Var(D),$$

where $Var(D)$ represents the sum of the variance of elements of the partition, and is calculated by adding the elements of the diagonal blocks resulting from process just described.

3. Optimization Procedure

The optimization is performed using a standard simulated annealing algorithm, searching in the space of boundary vectors and permutation matrices. The procedure searches in the space of all possible representations of partitions in order to find the one that minimizes total system variance. Once it is found, it returns the partition elements that represent the applications that should be installed on each machine. The neighborhood function for the

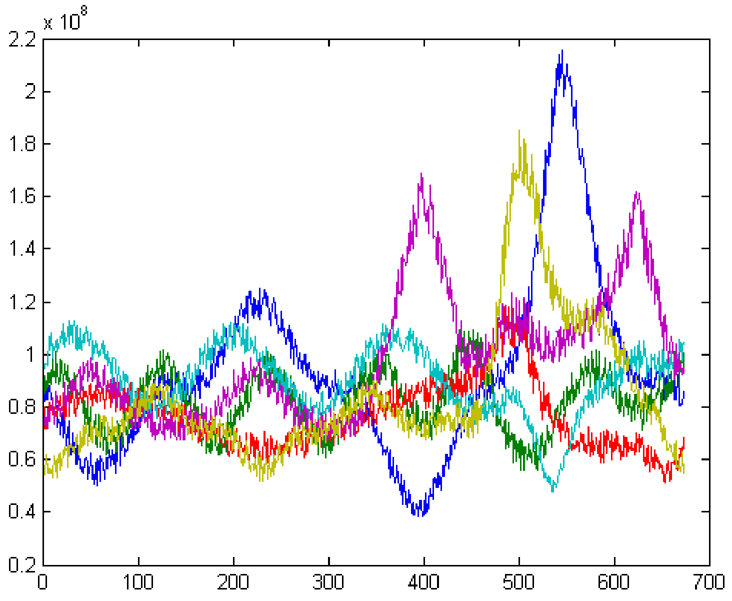

Fig. 16. Arrival data.

boundaries vector simply picks a random entry and adds or subtracts one to it. The resulting vector is sorted again in ascending order.

```
Neighb(b){
    r=random number between 1 and M;
    s=random number in {1,−1};
    b(r)=b(r)+s;
    sort(b);
}
```

The neighborhood function for the permutation matrices picks two different indexes in $\{1, \ldots, M\}$ and switches the rows and columns of the identity matrix. The resulting matrix is multiplied by the original Q to obtain the output matrix.

```
Neighb(Q){
    r=random number between 1 and M;
```

```
  s=random number between 1 and M;
  N=Permute rows r and s of the m by m identity matrix;
  Q=NQ
}
```

The simulated annealing macro-algorithm is a standard SA procedure that generates partitions, evaluates their variability and decides if the algorithm should move to the new element based on a dynamic criterion:

```
Generate random vector b
Generate random matrix P
Compute Cov(S);
currentVar=Var
t=Initial temperature

while (t>0) {
        generate random neighbor (P,b)
        newVar=Var(neigh(S))
        if (newVar < currentVar){
                currentVar=newVar;
        Update P,b;
      }
  else {
        if (criterion(currentVar-newVar)==true ){
                                            currentVar=newVar;
                        Update P,b;
```

```
                                }
    }
    if(i mod parameter ==0)
    t=t-Delta_t
}
```

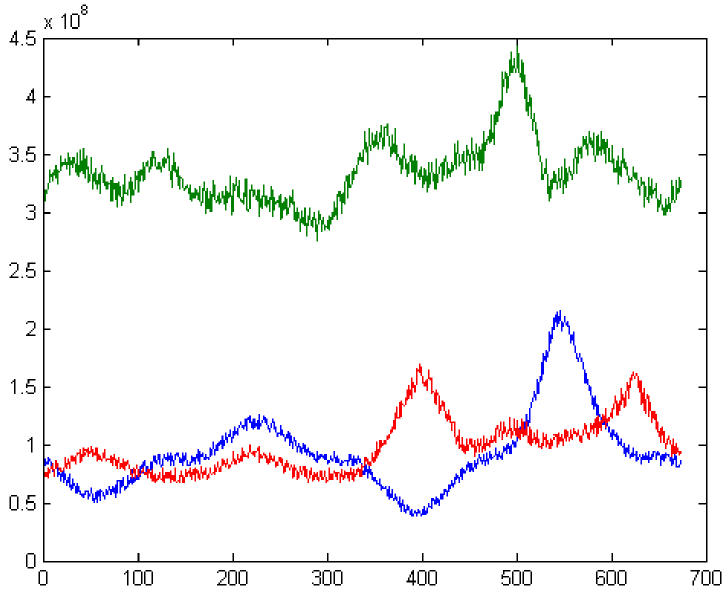

Fig. 17. Consolidated arrival data.

Figure 17 shows the resulting consolidated time series after the procedure just described. For this 5 application example, the algorithm returned three separate groups. Two of the groups contain only one application, and the third is the sum of the three remaining applications. The original data is plotted in Figure 16. The resulting partition has two one-element subsets and one three-element consolidated time series.

4. Application Load Consolidation

It remains to decide how to allocate and distribute the capacity among the applications allocated to the same group of machines. After feeding the individual capacity allocation with the values of $n_m(z)$ and the individual load history, the system receives a set of m couples $F_m(z)$ and $k_m(z)$. Now we compute the aggregated frequency $\mathcal{F} = \sum_{m\in S} F_m(z)k_m(z)$, and find the lowest possible number of machines $k(z)$ and lowest frequency $F(z)$ (in that order) such that $k(z)F(z) \geq \mathcal{F}$. Then we execute a local search for minimal cost $k_S(z)$ and $F_S(z)$ without having a capacity lower than $\mathcal{F}$. With the values of $F_m(z)$ and $k_m(z)$, it is possible to compute p_m as defined in section 2. Then whenever a server is free, the machine will serve the first request in the queue of application m with probability p_m. The macro-algorithm are the following:

```
Find_k_F(k_m, F_m, AggFreq){
        Find min_k(min_F (k*F>AggFreq));
        Find min_{k,F:k*F>AggFreq} (Cost(k,F));
        Compute p_m;
}

Choose Application to be Served(buffer, p){
        flag=0
        while flag==0 {
                Generate  m in {1,...,|S|} with prob. p_1,...,p_s
                if n_m(t)>0{
                flag=1;
                }
        }
}
```

D. Numerical Evaluation

In order to evaluate the effectiveness of the algorithm, we generate M individual application load time series, L_m, in such a way that the behavior on the second half of the sequences resembles the behavior on the first part, but incorporates some randomly placed disruptions and white noise. The first half of the arrival information is used as training data in order to prepare the individual application capacity allocator, and to compute the correlation of the load or aggregated capacity time series. Once the information is processed, the Simulated Annealing algorithm will return a partition of the applications set indicating which applications should be installed together. We will take the elements of the partition with cardinality larger than one, and simulate the system for both the newly created meta-application, and for the individual applications to compare the amount of energy used in each case. The objective of this numerical experimentation is to show that a proper application grouping will result in energy savings for the applications on that specific subgroup; therefore there is no additional information on running experiments with a much larger number of applications because the final comparison will take place on the elements of the partition. For the experiments, we assume that there are 60 servers in the facility, and allocation to each group will be proportional to the combined average load. It is worth noting that the algorithm never used all the available servers.

1. Data Generation

For each time slot, the total load of the system is divided by a constant Y and then a Poisson random variable with mean $\frac{L_m(z)}{Y}$ determines the number of arrivals. The individual arrival times are obtained by a uniform sampling of an interval of length t_0, and the file sizes are sampled from a Pareto distribution with squared coefficient of variance between 10 and 100.

Table VII. Data generation parameters.

Load Trend	$a\sin^d(bz+c)+eZ+f$
Disruptions	$\max(ud,-0.25)coef\cdot sech(\frac{z-loc}{scl})$
Noise	$Unif(0,1)Z$
Number of arrivals	$\frac{Load}{Y}$
Arrival Times	Uniform Sampling
Arrival Sizes	Pareto

Table VII contains detailed information about the parameter generation procedure. The load trend is an arbitrary integer power of a general sine function. The disruptions are modeled as scaled and shifted hyperbolic secants, and the noise is generated by independent normal sampling.

The generated data will look like the Figure 16, where there is some resemblance between the first and second half of the data, but clearly there is not an exact repetition. Those disruptions in the data reflect the changes in behavior that are natural in data center operations.

2. Numerical Results

We present several experiments to illustrate the effectiveness of the proposed methodology. We performed 15 simulations with data corresponding to 5 applications and homogeneous QoS constraints, and 10 corresponding to heterogeneous constraints. We assume that the load cycle is 80 periods, so we generated 160 periods of experimental data, where the periods 81 to 160 reflect the behavior of the first half of data, but independent noise and random disruptions are added (See Table VII). Notice that the hypothesis we are testing are that the serving grouped applications consume less energy than serving them individually; therefore, designing experiments with a large number of applications will not provide additional

Table VIII. Running time for the grouping algorithm.

Number of Applications M	**Running Time seconds**
5	2.37
10	2.45
100	32.81
1000	850.42

information. Larger sets of applications would result in larger and/or more groups with results similar to the small experiments performed here. We should notice that the algorithm is computationally simple enough to handle a data center with thousands of applications. As a point of comparison, the more complex optimization algorithm in Chapter II solved linear systems with more than a million variables on each iteration and run in a few hours; in contrast, the procedure in this Chapter will only perform sums on an $M \times M$ matrix, so it is much less complex. For reference, Table VIII contains the running times for different values of M in a Windows Vista computer with 1.83 GHz. Intel Core Duo 2 processor and $2Gb$ RAM.

For each one of the experiments we report the total cost of running all the applications independently, the cost of running the system with the partitions we proposed, and the fraction of requests that required longer than R_m time units to be served. Here, as in Chapter IV, we use a TSF constraint, where the requirement gives a bound on the fraction of the requests of each class that lasts more than R_m seconds in the system.

Observing the results in Table X, we can conclude that there are savings produced by the consolidation of the capacity allocated to the selected applications. It is important to notice that the comparisons in the tables are against hosting individual applications using the methodology presented in Chapter IV, so the savings are on top of the ones obtained

Table IX. Settings of the numerical experiment.

Parameter	Value
Available Frequencies	$(1.4, 1.57, 1.74, 1.91, 2.08, 2.25, 2.42, 2.6)$
Power	$(60, 63, 66.8, 71.3, 76.8, 83.2, 90.7, 100)$
Cost (cents/KWH)	10
B_0 (cents/Boot)	0.525
T_{boot}	90 sec
Decision Period (t_0)	900 sec
Heterogeneous conditions (Table XI)	$90 + 8 * rand()$

there. So, if the system may save around 35% by using individual capacity allocation and 13% by consolidating applications, then the total amount of savings will be around 45%.

Table X contains the results of 15 experiments serving 5 applications. We assume a TSF QoS constraint with $R = 0.05$, and the last two columns represent the maximum fraction of requests across the applications that violate the maximum time in the system constraint for both the individual and grouped cases. The average savings in Table X is 13%, and the maximum amount of savings obtained is 29.7%. The variability on the amount of energy saved is due to the correlation between the arrival time series. Since the parameters of the arrival process are randomly generated, it is possible to obtain settings in which the grouping will lead to low savings, or even worse, as in the experiment number 4 in Table X, where the performance is worse for a very small margin. The reduction in the energy cost has two causes. First, obtaining an optimal value for $k(z)$ and $F(z)$ globally outperform the aggregation of individual capacity optimization. Secondly, the stabilization of the load reduces the number of boots in the system, lowering both the energy expenditure and reliability cost. The proposed algorithm meets the QoS specifications in all the cases when the individual algorithm is able to do so.

Table X. Homogeneous QoS allocation results.

Data	**# Gr.**	**Max.Size**	**Cost/s I.**	**Cost/s G.**	**Savings**	**Max Fr. I.**	**Max Fr. G.**
1	4	2	0.0126	0.0104	17.1601%	0.0323	0.0313
2	2	4	0.0065	0.0054	16.9093%	0.0487	0.0174
3	3	3	0.0078	0.0069	12.5096%	0.0467	0.0265
4	3	3	0.0066	0.0066	-0.7135%	0.0991	0.0189
5	3	3	0.0115	0.0095	17.0115%	0.0461	0.0427
6	3	3	0.0156	0.0112	28.1852%	0.0490	0.0453
7	3	2	0.0106	0.0083	21.8762%	0.0477	0.0472
8	4	2	0.0054	0.0053	2.9763%	0.0647	0.0066
9	3	3	0.0120	0.0084	29.6990%	0.0477	0.0401
10	2	4	0.0063	0.0057	9.4447%	0.0848	0.0363
11	2	3	0.0067	0.0056	16.6270%	0.0379	0.0353
12	3	2	0.0079	0.0074	5.8151%	0.0207	0.0508
13	2	4	0.0070	0.0063	10.3833%	0.0425	0.0295
14	4	2	0.0080	0.0078	1.9371%	0.0477	0.0219
15	4	2	0.0108	0.0101	6.6499%	0.0283	0.0283
	Max Sav.	29.7%	**Ave Sav.**	13.1%	**Min Sav.**	-0.7%	

Table XI contains 10 experiments with heterogeneous TSF quality constraints. The changes between quality constraints are in the fraction of the requests that should be in the system for fewer than 30 units of time. The table contains the requirements for each one of the application classes. We also report the operation cost per second if the applications are hosted individually, and the cost per second if the applications are grouped according to our algorithm. The last two columns show whether the system met the quality of service constraint for all the applications. Average savings were 7.4%, and the maximum savings were 22.6%. Only in one case (experiment number 7) was the operating cost larger for the grouped applications. This may happen when the disruptions are so large that the

Table XI. Heterogeneous QoS allocation results.

	#Gr.	**Max.Sz.**	**Cost/s I.**	**Cost/s G.**	**Savings**	**Requirement**					**I.**	**G.**
1	3	3	0.024877	0.021305	14.3586%	93	95	95	96	93	T	T
2	3	3	0.023651	0.019336	18.2445%	96	93	93	94	97	T	T
3	3	3	0.021853	0.019133	12.4468%	92	95	92	93	96	T	T
4	3	3	0.020922	0.020536	1.8449%	95	92	93	93	96	T	T
5	2	4	0.022464	0.017384	22.6140%	92	97	94	93	95	T	T
6	3	3	0.02172	0.020203	6.9843%	93	96	94	93	96	T	T
7	4	2	0.008031	0.01003	-24.8910%	94	93	92	95	94	T	T
8	4	2	0.026951	0.022913	14.9827%	96	95	94	96	94	F	T
9	3	3	0.026343	0.022472	14.6946%	95	97	92	95	94	T	T
10	3	3	0.030614	0.032603	-6.4970%	96	97	96	93	95	T	T
	Max. S.	22.6%	**Ave. S.**	7.4%	**Min. S.**	-24.8%						

correlation structure is severely affected. It is worth noting that in experiment number 8 the system did not meet the QoS constraint when the applications were run individually, but it did when the applications were grouped. This is an illustration of the fact that more stable loads increase the likelihood of providing capacity properly.

We observed that there are moments in which no grouping is desirable. For instance if all the applications are positively correlated, grouping applications may cause the variability of the system to increase, and as a consequence more servers should be turned on and off, and the risks of miscalculating the necessary capacity grows. Our algorithm is capable of identifying most of those cases and suggests a partition where all the elements are unitary sets. We present here only the instances for which the algorithm found grouping applications to be beneficial. It happened in around 75% of the times we generated data.

It is worth noting that even if the algorithm suggests that grouping applications is beneficial, the random disruptions introduced, if large enough, may alter the correlation structure significantly. These changes in behavior may produce a performance decrease or even, as in experiment number 4 in Table X or experiment number 7 in Table XI, a setting in which grouping the applications will not produce any benefit at all.

Finally, we observed that grouping the applications did not cause a significant performance decrease. For a large majority of the cases, the maximum fraction of the requests staying in the system for longer than the amount of time consigned in the SLA is not increased by using the proposed methodology (See last two columns in Table X).

E. Conclusions

In this chapter we proposed the implementation of tactical decisions to complement the real time allocation strategies presented in Chapter IV. We developed and implemented an optimization procedure to find groups of applications that should be run in the same machines in order to reduce energy and reliability costs. These savings are a consequence of a reduction in the overall system effort variability and the consolidation of the effort across the different applications. Moreover, the consolidation strategy still operates the system in such a way that it meets the QoS agreements. The proposed solution is versatile enough to handle applications with heterogeneous QoS service constraints.

We can conclude that a scientific applications allocation to machines has the potential to increase the savings obtained by the single-application methods of Chapter II. Moreover, these variance reduction techniques address further the main concern of industry practitioners because it reduces the risk of a drastic performance reduction due to miscalculations of the capacity.

CHAPTER VI

CONCLUSIONS AND FUTURE RESEARCH

This dissertation presents a study of the Dynamic Resource Allocation for Energy Management in Data Centers from the operational and tactical point of view. This study responds to increasing economical and environmental concerns about data centers gigantic energy consumption by practitioners and interest groups. We assumed that a facility layout, auxiliary equipment and hardware are given and found strategies to increase the energy efficiency and hardware reliability. The strategies presented were designed to reduce the energy usage without jeopardizing the achievement of the QoS requirements. The solution strategy developed here is robust and simple enough to be implemented without major disruptions in the facility operations.

A. Conclusions

Given the immense energy expenditure and its economic and environmental consequences, data center energy consumption has been a subject of study for the past decade. During our study we found that implementing sophisticated methodologies to allocate capacity dynamically will result in significant energy savings and will not cause significant performance degradation.

We performed a study of this problem divided into three steps. First, we presented a mathematical abstraction of the data center system that is a generalization of two classic queueing control problems, the single server with variable rate setting, and the call center setting. We showed that if the inter-arrival and service times are exponential, and the average arrival rate is constant, then the optimal policy is hysteretic. We designed an algorithm to find explicit hysteresis curves to obtain the optimal monotonic hysteretic policy. After numerical experimentation, we conjectured that the optimal policy is monotone hysteretic in the number of servers and threshold type in the frequency. This research resulted in an article submitted for publication (Rincon Mateus and Gautam [92]).

We presented an allocation method for a single application system with TSF QoS agreement. To the best of our knowledge, there is not a previous published capacity provisioning strategy that deals with TSF constraints. Our interest in this type of contract comes from its popularity among practitioners. The allocation method is composed of reactive and proactive components. The reactive portion of the algorithm is a function of the number of requests in the buffer; to make proactive decisions, we rely on a wavelet based forecasting algorithm that takes advantage of the nearly cyclic nature of the requests arrival process. This forecasting procedure combines long and short term data to obtain results reflecting the periodicity and sudden load changes or disruptions. Our methodology not only reduces the amount of energy used by the system, but provides the system manager with a high degree of certainty that the QoS requirements will be met. In Chapter IV, we simulated the system with real internet data, and in Chapter V, we used artificially generated data that includes random traffic disruptions. These results are contained in an article submitted for publication (Rincon Mateus and Gautam [93]).

Finally, we developed an analytic methodology to group applications in such a way that the variance of the system is reduced. We based our allocation rule on finding negative correlations between the arrival load time series. This way of allocating applications has two important benefits: it reduces the number of booting cycles, and reduces the risk of not meeting the QoS constraint when the forecasting is incorrect. We coupled this methodology with the single application allocation algorithm and found that the load consolidation reduces the energy consumption compared to our method of independent resource allocation. This material is contained in an article currently in preparation (Rincon Mateus and Gautam [94]).

B. Future Research

Future directions of research include:

- A deeper study of structure of the optimal policy of the $M/M_t/k_t$ queueing control system to confirm or disprove the conjecture regarding the optimal policy monotonicity.
- A study of policies of the $M/M_t/k_t$ queueing control system that may observes and make changes to the number of active servers and frequencies only every $p > 1$ number of periods.
- A study of structured policies to more realistic data center models with non exponential arrival processes.
- A detailed study of the theoretical properties of the decomposition in the forecasting methodology presented in Chapter V.
- Development of dynamic allocation techniques that allow the system to exploit different correlation structures at different stages within the periods.
- Allowing the possibility of splitting application arrival streams to divide the load between two or more sets of servers to achieve further variance reductions.
- A study on optimal capacity data center design.

REFERENCES

[1] J.G. Koomey, "Estimating the total power consumption by servers in the US and around the world," [Online]. Available: http://www.koomey.com/research.html, 2007.

[2] J.S. Chase, D.C. Anderson, P.N. Thakar, A.M. Vahdat, and R.P. Doyle, "Managing energy and server resources in hosting centers," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5, Chateau Lake Louise, Banff, Alberta, Canada, 2001, pp. 103-116.

[3] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T.W. Keller, "Energy management for commercial servers," *Computer*, vol. 36, no. 12, pp. 39–48, Dec. 2003.

[4] R. Kumar, "A message from data center managers to CIOs: Floor space, power and cooling will limit our growth," Gartner RAS. G [Online]. Available: http://www.greenercomputing.com/files/document/CustomO16C45F77407.pdf, 2006.

[5] C.D. Patel, C.E. Bash, C. Belady, L. Stahl, and D. Sullivan, "Computational fluid dynamics modeling of high compute density data centers to assure system inlet air specifications," *ASME International Electronic Packaging Technical Conference and Exhibition (IPACK 01)*, Fountain Hills, AZ, 2001.

[6] "Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions," [Online]. Available: http://www.gartner.com/it/page.jsp?id=503867, 2007.

[7] J.M. George and J.M. Harrison, "Dynamic control of a queue with adjustable service rate," *Operations Research*, vol. 49, pp. 720–731, Sep. 2001.

[8] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1, 2005 Banff, Alberta, Canada, pp. 303–314.

[9] F.V. Lu and R.F. Serfozo, "M/M/1 queuing decision process with monotone hysteretic optimal policies," *Operations Research*, vol. 32, pp. 1116–1132, May 1984.

[10] L.I. Sennott, *Stochastic dynamic programming and the control of queueing systems*, New York: Wiley-IEEE, 1999.

[11] M.E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: Evidence and possible causes," *IEEE/ACM Transactions on Networking (TON)*, vol. 5, no. 6, pp. 835–846, Jun. 1997.

[12] S. Basu, A. Mukherjee, and S. Klivansky, "Time series models for internet traffic," *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 2, San Francisco, CA, 1996, pp. 611-620.

[13] X. Wang and X. Shan, "A wavelet-based method to predict internet traffic," *IEEE 2002 International Conference on Communications, Circuits and Systems*, vol. 1, Chengdu, China, 2002, pp. 690-694.

[14] L. Zhang and D. Ardagna, "SLA based profit optimization in autonomic computing systems," *Proceedings of the 2nd International Conference on Service Oriented Computing*, New York, 2004, pp. 173-182.

[15] D. Ardagna, M. Trubian, and L. Zhang, "SLA based resource allocation policies in autonomic environments," *Journal of Parallel and Distributed Computing*, vol. 67,

no. 3, pp. 259–270, Mar. 2007.

[16] K. Shen, H. Tang, T. Yang, and L. Chu, “Integrated resource management for cluster-based internet services,” *ACM SIGOPS Operating Systems Review*, vol. 36, San Francisco, CA, 2002, pp. 225-231.

[17] D. Clark and W. Lehr, “Provisioning for bursty internet traffic: Implications for industry and internet structure,” *Presented at the MIT ITC Workshop on Internet Quality of Service*, Cambridge, MA, 1999.

[18] E.V. Carrera, E. Pinheiro, and R. Bianchini, “Conserving disk energy in network servers,” *Proceedings of the 17th Annual International Conference on Supercomputing*, San Francisco, CA, Mar. 2003, pp. 86-97.

[19] J. Chase and R. Doyle, “Balance of power: Energy management for server clusters,” *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HotOS)*, Elmau, Germany, 2001, pp. 163-165.

[20] E.N. Elnozahy, M. Kistler, and R. Rajamony, “Energy-efficient server clusters,” *Power-Aware Computer Systems: Second International Workshop, PACS 2002*, Cambridge, MA, 2003.

[21] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath, “Load balancing and unbalancing for power and performance in cluster-based systems,” *Workshop on Compilers and Operating Systems for Low Power*, vol. 180, New Orleans, LA, 2001, pp. 182-195.

[22] V. Sharma, A. Thomas, T. Abdelzaher, and K. Skadron, “Power-aware QoS management in web servers,” *24th IEEEReal-Time Systems Symposium.*, Cancun, Mexico, 2003.

[23] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, *Power Aware Computing: The Case for Power Management in Web Servers*, Norwell, MA: Kluwer Academic Publishers, 2002.

[24] D. Hembold, D. Long, T. Sconyers, and B. Sherrod, "Adaptive disk spin-down for mobile computers," *Mob. Netw. Appl*, vol. 5, no. 4, pp. 285–297, Apr. 2001.

[25] H. Aydin, D. Melhem, D. Mosse, and Mejia-Alvarez P., "Dynamic and aggressive scheduling techniques for power-aware-real-time systems," *Proceedings of the 22th IEEE Real-Time Systems Symposium*, London, UK, 2001.

[26] A. Chandrakasan, V. Gutnik, and T. Xanthopolous, "Data driven signal processing: An approach for energy efficient computing," *Proceedings of the International Symposium on Low Power Electronic Devices*, Monterey, CA, 1996, pp. 347-352.

[27] C.H. Hsu, U. Kremer, and M. Hsiao, "Compiler directed dynamic frequency and voltage scaling," *Proceedings of the Workshop on Power Aware Computer Systems*, Monterey, CA, 2000, pp. 65-81.

[28] S. Gurumurthi, M. Sivasubramanian, M. Kandemir, and H. Franke, "DRPM: Dynamic speed control for power management in server class disks," *Proceedings of the International Symposium on Computer Architecture*, San Diego, CA, 2003, pp. 169-179.

[29] J.R. Lorch and A.J. Smith, "Improving dynamic voltage scaling algorithms with PACE," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 50–61, Jan. 2001.

[30] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*,

vol. 272, Los Alamos, CA, 1995, pp. 374-382.

[31] D. Zhu, R. Melhem, and B.R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel Computer Systems*, vol. 14, pp. 686–700, Jul. 2003.

[32] A. Merkel and F. Bellosa, "Balancing power consumption in multiprocessor systems," *Proceedings of the 2006 EuroSys Conference*, vol. 40, no. 4, Leuven, Belgium, 2006, pp. 403-414.

[33] R. Mishra, N. Rastogi, D. Zhu, D. Mosse, and R. Melhem, "Energy aware scheduling for distributed real-time systems," *International Parallel and Distributed Processing Symposium*, Nice, France, 2003, pp. 21-34.

[34] P. Juang, Q. Wu, L.S. Peh, M. Martonosi, and D.W. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," *Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, San Diego, CA, 2005, pp. 127-130.

[35] F. Gruian, "System-level design methods for low-energy architectures containing variable voltage processors," *Proceedings of the Workshop on Power Aware Computer Systems*, Cambridge, MA, Nov. 2000.

[36] P. Yang, C. Wong, F. Marchal, D. Catthor, D. Desmet, D. Versket, and Lauwereins R., "Energy-aware runtime scheduling for embedded-multiprocessor SOCS," *IEEE Design & Test of Computers*, vol. 18, pp. 46–58, May 2001.

[37] P. McGarry, M. Maier, and M. Reisslein, "Ethernet PONs: A survey of dynamic bandwidth allocation (DBA) algorithms," *IEEE Communications Magazine*, vol. 42, no. 8, pp. S8–15, Aug. 2004.

[38] R. Guerin, H. Ahmadi, M. Naghshineh, and Y. Heights, "Equivalent capacity and its application to bandwidth allocation in high-speed networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 7, pp. 968–981, Jul. 1991.

[39] C. Assi, Y. Ye, S. Dixit, and M. Ali, "Dynamic bandwidth allocation for quality-of-service over ethernet PONs," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 9, pp. 1467–1477, Sep. 2003.

[40] Y. Qiu and P. Marbach, "Bandwidth allocation in ad hoc networks: A price-based approach," *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE*, vol. 2, San Francisco, CA. 2003.

[41] T.B. Crabill, D. Gross, and M.J. Magazine, "A classified bibliography of research on optimal design and control of queues," *Operations Research*, vol. 25, no. 2, pp. 219–232, Feb. 1977.

[42] S. Stidham Jr, "Optimal control of admission to a queueing system," *IEEE Transactions on Automatic Control*, vol. 30, no. 8, pp. 705-713, Aug. 1985.

[43] M. Kitaev and R. Serfozo, "M/M/1 queues with switching costs and hysteretic optimal control," *Operations Research*, vol. 47, no. 2, pp. 310–312, Feb. 1999.

[44] S. Stidham Jr and R.R. Weber, "Monotonic and insensitive optimal policies for control of queues with undiscounted costs," *Operations Research*, vol. 37, no. 4, pp. 611–625, Apr. 1989.

[45] H.J. Plum, "Optimal monotone hysteretic Markov policies in an M/M/1 queueing model with switching costs and finite time horizon," *Mathematical Methods of Operations Research*, vol. 35, no. 5, pp. 377–399, May 1991.

[46] N. Gans, G. Koole, and A. Mandelbaum, "Telephone call centers: Tutorial, review, and research prospects," *Manufacturing and Service Operations Management*, vol. 5, no. 2, pp. 79–141, Feb. 2003.

[47] G. Koole and A. Mandelbaum, "Queueing models of call centers: An introduction," *Annals of Operations Research*, vol. 113, no. 1, pp. 41–59, Jan. 2002.

[48] O. Garnet, A. Mandelbaum, and M. Reiman, "Designing a call center with impatient customers," *Manufacturing & Service Operations Management*, vol. 4, no. 3, pp. 208–227, Mar. 2002.

[49] W. Winston, "Optimality of monotonic policies for multiple-server exponential queueing systems with state-dependent arrival rates," *Operations Research*, vol. 26, no. 6, pp. 1089–1094, Jun. 1978.

[50] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, DP Pazel, J. Pershing, and B. Rochwerger, "Oceano-SLA based management of a computing utility," *Proceedings of the 2001 IEEE/IFIP International Symposium on Integrated Network Management*, Seattle, WA, 2001, pp. 855-868.

[51] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "QoS-driven server migration for internet data centers," *10th IEEE International Workshop on Quality of Service.*, Monterey, CA, 2002.

[52] Q. Zhang, E. Smirni, and G. Ciardo, "Profit-driven service differentiation in transient environments," *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003*, Orlando, FL, 2003, pp. 230-233.

[53] X. Zhou, J. Wei, and C.Z. Xu, "Quality-of-service differentiation on the internet: A taxonomy," *Journal of Network and Computer Applications*, vol. 30, no. 1, pp.

354-383, Jan. 2007.

[54] E.N. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," *Proceedings of the Second Workshop on Power Aware Computing Systems*, Cambridge, MA, 2002.

[55] R. Bianchini and R. Rajamony, "Power and energy management for server systems," *Technical Report*, 2003.

[56] M. Wang, N. Kandasamy, A. Guez, and M. Kam, "Adaptive performance control of computing systems via distributed cooperative control: Application to power management in computing clusters," *IEEE International Conference on Autonomic Computing*, Dublin, Ireland, 2006.

[57] Z. Xue, X. Dong, S. Ma, S. Fan, and Y. Mei, "An energy-efficient management mechanism for large-scale server clusters," *The 2nd IEEE Asia-Pacific Service Computing Conference*, Tsukuba Science City, Japan, 2007, pp. 509-516.

[58] L. Bertini, J. Leite, and D. Mosse, "Statistical QoS guarantee and energy-efficiency in web server clusters," *19th Euromicro Conference on Real-Time Systems*, Luebeck, Germany, 2007.

[59] B. Khargharia, S. Hariri, and M.S. Yousif, "Autonomic power and performance management for computing systems," *Cluster Computing*, vol. 11, no. 2, pp. 167–181, Feb. 2008.

[60] W.E. Leland, M.S. Taqqu, W. Willinger, and D.V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 1–15, Jan. 1994.

[61] P. Abry and D. Veitch, "Wavelet analysis of long-range-dependent traffic," *IEEE Transactions on Information Theory*, vol. 44, no. 1, pp. 2–15, Jan. 1998.

[62] A. Aussem and F. Murtagh, "Web traffic demand forecasting using wavelet-based multiscale decomposition," *International journal of intelligent systems*, vol. 16, no. 2, pp. 215–236, Feb. 2001.

[63] P. Fryzlewicz, S. Van Bellegem, and R. von Sachs, "Forecasting non-stationary time series by wavelet process modeling," *Annals of the Institute of Statistical Mathematics*, vol. 55, no. 4, pp. 737–764, Apr. 2003.

[64] D.B. Percival and A.T. Walden, *Wavelet methods for time series analysis*, Cambridge, MA: Cambridge University Press, 2000.

[65] O. Renaud, J. Starck, and F. Murtagh, "Wavelet-based forecasting of short and long memory time series," [Online]. Available: http://www.unige.ch/ses/metri/cahiers/2002-04.pdf, 2002.

[66] P. Barford and D. Plonka, "Characteristics of network traffic flow anomalies," *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, San Francisco, CA, Sep. 2001, pp. 69-73.

[67] P. Barford, J. Kline, D. Plonka, and A. Ron, "A signal analysis of network traffic anomalies," *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment*, Marseille, France, Sep. 2002, pp. 71-82.

[68] D. Kwon, K. Ko, M. Vannucci, A. Reddy, and S. Kim, "Wavelet methods for the detection of anomalies and their application to network traffic analysis," *Quality and Reliability Engineering International*, vol. 22, no. 8, pp. 953–969, Aug. 2006.

[69] S. Kim and N. Reddy, “Statistical techniques for detecting traffic anomalies through packet header data,” *IEEE Transactions on Networking*, vol. 16, pp. 562–575, Jun. 2008.

[70] M. Pinedo, *Planning and Scheduling in Manufacturing and Services*, New York: Springer, 2005.

[71] R.E. Bellman, *Dynamic Programming*, Chelmsford, MA: Courier Dover Publications, 1957.

[72] O.Z. Maimon and S.B. Gershwin, “Dynamic scheduling and routing for flexible manufacturing systems that have unreliable machines,” *Operations Research*, vol. 36, no. 2, pp. 279–292, Feb. 1988.

[73] J. Gomoluch and M. Schroeder, “Market-based resource allocation for grid computing: A model and simulation,” *Proceedings of the 1st International Workshop on Middleware for Grid Computing*, Rio de Janeiro, Brasil, 2003, pp. 1-8.

[74] K.J. Becker, D.P. Gaver, K.D. Glazebrook, P.A. Jacobs, and S. Lawphongpanich, “Allocation of tasks to specialized processors: A planning approach,” *European Journal of Operational Research*, vol. 126, no. 1, pp. 80–88, jan. 2000.

[75] M.K. Gardner and J.W.S. Liu, “Performance of algorithms for scheduling real-time systems with overrun and overload,” *Proceedings of the 11th Euromicro Conference on Real-time Systems*, York, UK, 1999, pp. 287–296.

[76] J. Park, M. Ryu, and S. Hong, “Fair real-time resource allocation for internet end systems QoS support,” *Lecture Notes in Computer Science*, vol. 2713, pp. 764–769, 2003.

[77] H. Levy and M. Sarnat, "International diversification of investment portfolios," *The American Economic Review*, vol. 60, no. 4, pp. 668–675, Apr. 1970.

[78] H. Markowitz, "Portfolio selection," *The Journal of Finance*, vol. 7, no. 1, pp. 77–91, Jul. 1952.

[79] J.P. Onnela, A. Chakraborti, K. Kaski, J. Kertész, and A. Kanto, "Dynamics of market correlations: Taxonomy and portfolio analysis," *Physical Review E*, vol. 68, no. 5, pp. 56–110, 2003.

[80] "Uptime institute report," Uptime Institute [Online]. Available: http://www.uptime.org, 2008.

[81] D.P. Bertsekas, *Dynamic programming: Deterministic and stochastic models*, Upper Saddle River, NJ: Prentice-Hall, 1987.

[82] R.F. Serfozo, "An equivalence between continuous and discrete time Markov decision processes," *Operations Research*, vol. 27, no. 3, pp. 616–620, Mar. 1979.

[83] S.K. Hipp and U.D. Holzbaur, "Decision processes with monotone hysteretic policies," *Operations Research*, vol. 36, no. 4, pp. 585–588, Apr. 1988.

[84] R.R. Weber and S. Stidham Jr, "Optimal control of service rates in networks of queues," *Advances in Applied Probability*, vol. 19, no. 1, pp. 202–218, Jan. 1987.

[85] L.I. Sennott, "Average cost optimal stationary policies in infinite state Markov decision processes with unbounded costs," *Operations Research*, vol. 37, no. 4, pp. 626–633, Apr. 1989.

[86] F.J. Beutler and K.W. Ross, "Uniformization for semi-Markov decision processes under stationary policies," *Journal of Applied Probability*, vol. 24, no. 3, pp. 644–656, Mar. 1987.

[87] W. Feller, *An Introduction to Probability Theory and its Applications*, New York: Wiley-Interscience, 1970.

[88] A. Kovac and B.W. Silverman, "Extending the scope of wavelet regression methods by coefficient-dependent thresholding," *Journal of the American Statistical Association*, vol. 95, no. 449, pp. 172–183, Aug. 2000.

[89] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi, *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science applications*, New York: Wiley-Interscience, 1998.

[90] "The internet traffic archive," [Online]. Available: http://ita.ee.lbl.gov/, 2007.

[91] C. Rincon Mateus and N. Gautam, "Adaptive resource allocation in data centers for balancing energy conservation, quality of service and reliability," Tech. Rep., Texas A&M University, 2008.

[92] C. Rincon Mateus and N. Gautam, "Efficient control of an M/ M_t / k_t queue with applications to energy management in data centers," *Submitted to INFORMS Journal of Computing*, 2008.

[93] C. Rincon Mateus and N. Gautam, "Adaptive resource allocation in data centers for balancing energy conservation, quality of service and reliability," *Submitted to IEEE Transactions on Computers*, 2008.

[94] C. Rincon Mateus and N. Gautam, "Load consolidation for energy management in data centers," *In preparation*, 2008.

VITA

Cesar Augusto Rincon Mateus

Texas A&M University Department of Industrial and Systems Engineering

College Station TX, 77843-3131

(979) 845-5536 crincon@tamu.edu

Education

Ph.D. in Industrial and Systems Engineering	*Texas A&M University, 2008*
Master of Science in Mathematics	*Universidad de los Andes, 2003*
Bachelor of Science in Mathematics	*Universidad de los Andes, 2001*
High School Diploma	*Colegio Champagnat, 1996*

Experience

Teaching Assistant. Texas A&M University	*Fall 2003 to Summer 2008*
Enterprise Optimization Intern. United Airlines	*Summer 2007*
Mathematics Instructor. Universidad de los Andes	*Spring 2000 to Spring 2003*

Awards and Honors

Regents Fellowship. Texas A&M University	*Augusto 2003*
40 años Scholarship. Universidad de los Andes	*1997 to 2001*
Andres Bello Award. Republica de Colombia	*January 1997*

Activities

Treasurer. INFORMS Student Chapter. Texas A&M University	*2006*