The Pennsylvania State University
The Graduate School

MANAGING PERFORMANCE AND ENERGY IN LARGE SCALE DATA CENTERS

A Dissertation in
Computer Science and Engineering
by
Seung-Hwan Lim

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2012

The dissertation of Seung-Hwan Lim was reviewed and approved* by the following:

Chita R. Das
Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Mahmut Kandemir
Professor of Computer Science and Engineering

Bhuvan Urgaonkar
Professor of Computer Science and Engineering

C. Lee Giles
Professor of Information Sciences and Technology

Ji-Woong Lee
Professor of Electrical Engineering

Raj Archarya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Data centers are the computing facilities to process, store, and access information. Specifically, data centers serve as the key infrastructure for Cloud computing service providers. However, service providers have observed the trend of the under-utilization of production servers, which unnecessarily increases the total cost of ownership. The demand on managing the total cost of ownership is driving researchers to study both performance and energy issues in data centers, which are addressed in this thesis. For considering performance and energy, this thesis consists three contributions – design of a comprehensive multi-tier data center simulation platform; energy management of multi-tier data centers; performance model for predicting running times of applications in data centers.

Design and analysis of large and complex distributed systems like data centers often suffer from the lack of an available physical infrastructure due to the cost constraints especially in the academic community. With this motivation, this thesis proposes the design of a comprehensive, flexible, and scalable simulation platform for in-depth analysis of multi-tier data centers. The potentials of such a simulation platform is demonstrated by its ability to simulate and measure the performance and power consumption of data centers with high accuracy. The second contribution is towards increasing the energy efficiency of multi-tier data centers using a multifacet approach, namely Hybrid, consisting of dynamic provisioning, frequency scaling and dynamic power management (DPM) schemes to reduce the energy consumption of multi-tier data centers, while meeting the Service Level Agreements (SLAs). The energy management scheme consists of two heuristics that utilize the Mean Value Analysis by modeling data centers as closed queueing networks. This scheme manages the energy consumption at the global and local levels. The global level management determines the sufficient number of servers for a service. At the local level, the proposed scheme dynamically exploits energy management techniques in individual servers. The third contribution is estimating

the performance of data centers. This involves the development of performance models of shared service platforms with multiple resources contention given the fact that a typical data center is shared by multiple services that contend for multiple system resources. The proposed model can estimate the total completion time of jobs, which is the objective function of the scheduling problem. This work illustrates that an existing job scheduler can be enhanced by modifying its original objective function to the proposed model. To summarize, this thesis discusses the performance and energy implications in data centers, along with suggesting optimization techniques for improving performance and energy conservation in typical data centers.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data centers are computing facilities for processing, storing, and accessing data. In a 2011 survey [2], data center operators expressed their attentions on scalability due to the steady growth in the amount of data, which stretches to exabyte-scale (10^{18}) [3]. As of 2009, the entire Internet was estimated to contain around 500 exabytes [4] and expected to pass one zettabyte (10^{21}) in 2011 [5]. Indeed, designing an exascale system could become its own computation problem, which cannot be addressed just by adding more disks [6]. Unprecedent system scale prevents from accurately projecting system behavior before actual implementation [7], though it is essential for selecting more desirable design choices. Another challenge is delivering sufficient electrical energy to run all necessary equipments [2]. Obtaining predictable performance is also an unsolved problem [8]. In addition, we have to consider limited budgets to construct, operate, and maintain data centers. Thus, a scalable evaluation method, sizable reduction in energy requirement, and exerting control over performance variance should constitute the first steps to address data center scalability for handling growing demands.

Let us briefly overview the state of the art regarding scalable evaluation methods, energy management, and performance management in the context of large scale data centers.

Scalable evaluation method In order to accurately capture system behavior, we may have several options. For the greatest detail, we may employ a full system simulation platform such as SIMICS [9]. However, a full system simulator can evaluate systems in only limited scope of workloads and machines due to enormous computation time to obtain simulation results [10]. Thus, usually such a full system simulator is employed to analyze one server. In order to analyze a set of servers, we often rely on more abstracted comprehensive simulation platforms [11]. Still, the scale has been limited to a few tens of servers [11,12] and some of them are not able to capture inter-relationship among applications running on different server nodes [12]. A 2011 study, CloudSim, can simulate 1 million virtual machines [13], but it can consider less than 100 physical machines due to the limit of GridSim [11], which is employed to simulate the underlying physical nodes. Thus, this thesis starts with a scalable simulation platform, MDCsim, which simulated up to 128 physical machines and 4800 concurrent users.

Energy management Two important empirical findings to save energy are that idle machines consume almost 50% energy of fully utilized machines [14] and frequent, but short low-utilization periods in each server [15]. To address the energy wastage with idle machines, prior work proposed a variety of schemes for dynamically powering on/off systems [16–18] and they provide measurable reduction in energy consumption, while satisfing with the performance constraints. Often, those schemes are combined with power management techniques in CPU [16–20] or various components in the system [21]

While addressing frequent, short low-utilization periods, the authors in [15] proposed a method to save energy consumptions in a server, which puts a server into sleep states whenever a low-utilization period is encountered. However, it is not clear whether we can exploit such a phenamenon in large scale data centers without violating performance constraints. For example, too aggressively putting a server into sleep states may result in slow responses with following non-idle periods in workloads. For one server, those slow responses may create a graceful performance degradation. Without coordination with other servers, however, accumulated slow responses from a set of servers may result in fluctation of workloads at other active servers in a data center, which may cause unnecessarily long wait-

ing times. Therefore, this work proposes a novel energy management schemes for data centers instead of a single server, which proactively exploits frequent, short low-utilization periods in each server, while satisfying the performance constraints.

Performance management Performance management is often discussed in the context of a shared environment since typical data centers share system resources among multiple workloads. With sharing system resources, system administrators can enhance system utilization that can process more workloads without inducing significant additional costs. A variety of methods are available to share system resources: Sharing a data center by hosting multiple workloads [14]; Sharing a physical machine by hosting multiple Operating System (OS) instances [22–25]; Sharing an OS instance by concurrently running processes [26, 27]; and Sharing memory or I/O subsystems by increasing the number of processing cores [28, 29]. As a complment to resource sharing methods, performance isolation methods in hypervisor [30], OS [29], and hardware [31] are proposed to control undesirable performance interference among workloads.

Although performance isolation methods address performance interference, still performance variance is considered as serious issues [8]. It stems from the fact that some resources like cache are hard to isolate among concurrent workloads [32]. The only consensus is that the running time of a workload non-linearly slows down when a new workload arrives at the system [28, 32, 33]. A 2011 study proposed an empirical method to estimate this slow-down [32], but a generic method to efficiently estimate this non-linear slow-down is not available so far. The difficulty of understanding application running time creates unpredictable performance in shared environments like cloud platforms [8]. Consequently, we allocate dedicated resources to performance critical applications and, in turn, we create low system utilization. Therefore, it is critical to develop a generic model to capture workload slow-down due to sharing system resources and this thesis proposes a generic model –*Dilation Factor Model*, or D-factor Model.

In the following section, I will briefly overview three major contributions of this thesis for managing performance and costs in large scale systems.

1.1 Contributions

1.1.1 Simulation

I constructed a simulation platform for multi-tier data centers, called MDCSim. In a large scale system, the complexity of interaction among nodes create challenges to precisely predict the behavior of a system with a few parameters. Often, we may be interested in analyzing the interactions among nodes, which is not an easy task with analytical modeling [34]. Toward such a goal, simulation is a more reliable option than analytical models. However, a comprehensive simulation platform for large scale data centers, specifically, multi-tier data centers is not available. Thus, ahead of investigating productivity and cost of a large scale multi-tier data center, this work describes a simulation platform for large scale multi-tier data centers.Based upon MDCsim, we evaluate a dynamic energy management scheme for multi-tier data centers.

1.1.2 Energy Efficiency

I devised a dyanamic energy management schme of multi-tier data centers. Cost is a factor of the efficiency of a system, along with performance. Since, however, minimizing cost to process a workload involves minimizing the available resources to process the workload, cost efficiency often conflicts with satysfing required performance and availability [35]. Thus, in designing data centers, cost efficiency often owns lower priority than delivering high performance and high availability, which has been considered the most important factor in hosting services. However, due to the soaring scale of data centers, managing the cost efficiency can earn more profit than hosting more services. Challenges to manage the cost can be summarized as determining when to change the status of a system – from high performance states to energy saving states. I demonstrated a substantial reduction in the energy usage with an on-line algorithm that decides the pivotal point, equipped with queueing analysis [36].

1.1.3 Performance

I developed a model to estimate slow-down of applications in shared service platforms since a data center is an instantiation of shared service platforms. Since the time to complete required jobs is the primary measure of performance, offering acceptable performance means that a system can complete all the jobs within the time constraints. We can translate this into an optimization problem, known as the scheduling problem, which aims to minimize the total completion time of given jobs on an available machine [1, 37]. Since a scheduler evaluates candidate schedules with the total completion time, the accuracy of estimating total completion time is of importance. However, we usually obtain the total completion time from the sum of completion times of individual jobs without reflecting the actual interactions of jobs.

A typical data center concurrently executes many jobs that share multiple system resources such as processors, memory, storage systems, and network systems [29, 38, 39]. Thus, a job in data center often requests multiple system resources, which depends on the resource access pattern of other co-existing jobs. In order to explain the slow-down of applications in such a situation, prior work have developed analytical models [40] or measurement based methods [32]. Prior analytical models describes on resource contention in generic ways, but they often require detailed descripions on target workloads on the target systems. In contrast, measurement based methods provide simple and effective strategy to estimate the slow-down of applications in a system. However, measurement based methods are not generic strategies. Thus, I developed an analytical model that takes advantages of both sides – generic, but simple and effective. For developing such a model, I extended the linear sum model, the most simplistic model to estimate slow-down of applications in shared environments. Since the linear sum model is widely used in many existing scheduling mechanisms [37], the proposed dilation factor model can easily extend scheduling mechanisms, which is one of the primary benefits of the model.

In summary, this thesis addresses the energy and performance issues in data centers.

1.2 Organization of The Thesis

An overview of data centers is presented in Chapter 2. MDCSim, a simulation platform for multi-tier data centers, is described in Chapter 3. A dynamic energy management scheme for multi-tier data center is discussed in Chapter 4. Chapter 5 describes the quantitative performance model for shared service systems, which is followed by conclusions and future work in Chapter 6.

Chapter 2

Background

This chapter briefly overviews data centers, which includes a prototype data center designed in this work. A data center is a facility that houses computing systems that includes network systems, storage systems, compute servers, and software packages. Data center may store and process information as well as provide services through networks. Two types of data centers were constructed to represent common practices in actual data centers. The first is a data center that hosts three-tier enterprise internet applications based upon Linux. The second is a virtualized data center with Xen hypervisor, which is commonly used in cloud computing environment. The details of the two prototypes are presented in following sections.

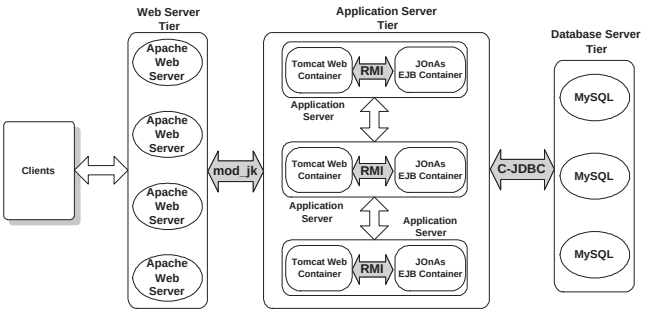


Figure 2.1: A Prototype Data Center

2.1 A three-tier data center

A three-tier data center prototype was designed and implemented as shown in Figure 2.1 for conducting actual measurements. The major components of this prototype include clients, a Web Server (WS) tier, an Application Server (AS) tier, and a Database Server (DB) tier. The WS tier serves as the front-end of the system to users and connects the data center to users. The AS tier processes complex computational operations for services, commonly written in Java or other programming languagues. The DB tier performs database queries to store and retrieve data.

As a representative of three-tier enterprise internet applications, we set up applications as follows. The web server nodes, in which Apache 2.0.54 [41] is installed, communicate with the application server nodes through the mod_jk connector module v.1.2.5. The maximum number of Apache processes was set to 512 to maximize the performance of the WS tier, which was found by trial and error. There are many application servers based on J2EE specifications, which define the architecture and interfaces for developing Internet server applications for a multi-tier data center. Among them, the JOnAS 4.6.6 [42], an open source application server, is installed in our prototype data center. The TomcatWeb container 5.0.30 is integrated with JOnAS as the web container. Remote Method Invocations (RMI) provides intra-cluster communication among the AS tier servers. Sun's JVM from the JDK 1.5.0_04 for Linux was used to run JOnAS. EJB session beans version were employed to achieve the best performance as reported in [43]. Communication between AS-tier and DB tier was achieved through C-JDBC v.2.0.1 [44], which allows the EJB container to access a cluster of databases. For the back-end tier, we used multiple machines running MySQL 5.0.15-0 Max [45].

Our prototype is implemented on varying number of dedicated nodes of a 96-node Linux kernel 2.6.9 cluster. Each of the these nodes has dual 64-bit AMD Opteron processors, 4 GBytes of system memory, Ultra320 SCSI disk drives and a Topspin host adapter, which is InfiniBand (IBA) v.1.1 compatible. The adapters are plugged into PCI-X 133Mhz slots. Each node has a 1Gig-Ethernet adapter of Tigon 3 connected to 64 bit, PCI-X 66Mhz slots. The default communication between all the server nodes is through a 1GigE connection. In order to make

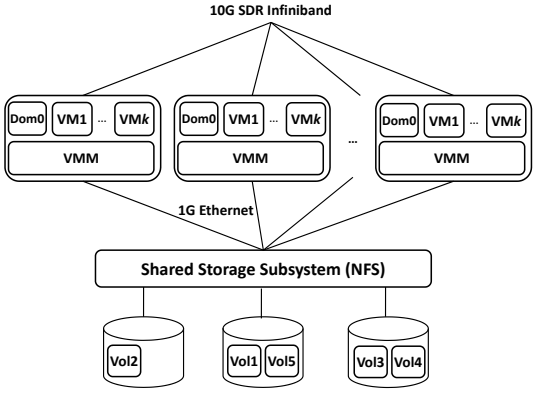


Figure 2.2: An overview of virtualized experimental environment.

IBA as our connection backbone instead of the Ethernet, we ran all of the server applications with the SDP library (provided by Topspin) in our experiment. We used RUBiS [46] benchmark workload to collect data from actual measurements. A round-robin web switch emulation was implemented to distribute client requests to the web server tier nodes.

2.2 A virtualized data center

We also implemented a vitualized environments in the Linux cluster as shown in Figure 5.3. A virtual machine is an independent operating system instance in virtualized environments. Thus, applications running on a virtual machine have an illusion that they exclusively access the virtualized resources. A special guest machine, Dom-0, can directly access the physical resources, especially the I/O devices. The hypervisor manages shared resources such as processors, memory, I/O subsystems and network devices in order to provide fair performance for each virtual machine instance. All the access requests to the hardware resources from applications running on a virtual machine flow into the hypervisor and then Dom-0, if necessary. The resource contention among collocated virtual machines is similar to the processes in non-virtualized environments [30].

The differences between the virtualized prototype and the unvirtualized bare metal prototype is described as follows. Each physical host machine is equipped

with dual Intel Xeon CPUs (3.4GHz) and 2GB RAM, running on Redhat 9 Enterprise edition with kernel version of 2.6.18-164.el5. We installed Xen v3.1.4 on all of these machines and created one Virtual Machine (VM) on each physical machine. Each VM runs on a Fedora core 4 having kernel version 2.6.18.8. VMs are given 512MB RAM and connected by a 1Gbps Ethernet. The four VMs are distributed across the three tiers in the following manner - the first tier consists of an Apache web server running on one VM, two JBoss 3.2.8SP1 application servers running on separate VMs comprise the second tier, and the MySQL 4.1 database server hosted on another VM forms the third tier. We used TPC-W benchmark that models a three-tier online book store and the J2EE implementation of TPC-W [47] was used. For the client emulator, we used a TPC-W [48] based workload generator.

In the following chapters, experiments are performed using the above prototypes. When required, we specify the difference in experimental settings from the prototypes.

Chapter 3

MDCSim: A Multi-tier Data Center Simulation Platform

3.1 Introduction

Design of high performance, power-efficient, and dependable cluster based data centers has become an important issue, more so with the increasing use of data centers in almost every sector of our society: academic institutions, government agencies and a myriad of business enterprises. Commercial companies such as Google, Amazon, Akamai, AOL, and Microsoft use thousands of servers in a data center environment to handle high volume of traffic for providing customized (24x7) service. A recent report shows that financial firms spend around 1.8 billion dollars annually on data centers for their businesses [49]. However, it has been observed that data centers contribute to a considerable portion of the overall delay for web-based services and this delay is likely to increase with more dynamic web contents. Poor response time has significant financial implications for many commercial applications. The increasing power consumption of data centers is also drawing much attention, leading to the concept of "Green Data Centers". As reported in [50], data centers in the U.S. consumed 61 billion kilowatt-hour of electricity in 2006 at the cost of $4.5 billion. It is estimated that data centers' power consumption will increase by 4% to 8% annually and is expected to reach 100 billion kWh by 2011. Thus, future data center's design must focus on three critical parameters:

high performance, power/thermal-efficiency and reliability.

Measurement, modeling, and simulation are common approaches for the analysis of a system. Measurement is a credible approach for analyzing the system, but the associated cost and time is high. Modeling offers a simple representation of the construction and working of the system of interest. However, developing a model with high accuracy and flexibility is not always feasible. Simulation fills in the above deficiencies by providing a flexible and operative working model of a system. A simulator can well outline a more detailed view of a system than modeling. It is also an economical and less time-consuming option as compared to the real implementation of a system.

Analysis of multi-tier data centers has been done using measurement and modeling in [34, 51, 52]. However, the scale and scope of analysis is restricted to small size clusters or to an individual tier. In most of the studies on data centers, the size of the system analyzed is limited because of practical difficulties encountered in testing large scale systems [51,52]. There is a growing need to analyze data centers on a large scale. In addition, despite the growing popularity of multi-tier data centers, to the best of our knowledge, simulating a multi-tier data center including the functionalities of all the tiers together has not been attempted. Keeping in view the importance of growing data centers and the above limitations in the analysis of them, this thesis propose a simulation platform for the design and analysis of large scale, multi-tier data centers.

The proposed simulation platform is comprehensive, flexible, and scalable. The comprehensiveness comes from the fact that important salient features of a multi-tier data center are implemented in detail. It is flexible as we can manipulate different features of the data center. For example, the number of tiers can be changed, the scheduling algorithms can be customized, the type of interconnection used can be altered and so also the communication mechanisms. Further, the layered architecture allows for the modification of individual layers without affecting the other layers. Simulation experiments are possible with large number of nodes across the cluster, making it scalable and apt to existing data centers' trends.

For the validation of our simulator, I first designed and implemented a prototype three-tier data center on an IBA and 1GigE connected Linux cluster. The major components of this prototype includes a web server (WS) tier, an application

server (AS) tier, and a database server (DB) tier. The web traffic was generated by RUBiS [46] workload, which gets distributed using a simple round-robin web switch. Since the prototype was limited in terms of the number of cluster nodes, system configurations, and driving workloads, I designed the proposed simulation framework to overcome these limitations. The simulator was validated with RUBiS benchmark measurements on the above mentioned prototype data center. The average deviation of simulation results from real measurements was found to be around 10% for the latency and 3% for the power consumption.

Using our simulator, this study experimented with three important applications. First, I performed a comparative analysis of IBA and 10GigE under different cluster configurations and with varying load. Next, I conducted power measurement in multi-tier data center. Finally, I presented a methodology for the determination of optimal cluster configurations in terms of performance (latency) through simulation experiments. Experimental results for the comparative analysis between 10GigE and IBA show that IBA outperforms 10GigE both with and without TCP Offload-Engine (TOE) with high traffic, though 10GigE performs better than IBA under low traffic. From the power consumption comparison among IBA, 10GigE with/without TOE connected data centers with various cluster configurations and with varying number of clients, I observed that the power consumption increases with the increase in the traffic across the cluster. Also, the power usage was found to be varying for different cluster configurations even for the same total number of nodes. Finally, simulations with several cluster configurations confirmed the fact that increasing the number of nodes in a multi-tier data center does not always guarantee better performance. There are specific configurations (optimal distribution of nodes across each tier) that can achieve the desired level of performance.

The rest of this chapter is organized as follows. Section 3.2 discusses the related works. The design of the simulation platform is described in Section 3.3. Prototype design, system configuration and validation of simulator are discussed in Section 3.4. Experimental results and discussions are presented in Section 3.5, followed by the concluding remarks and future directions in Section 4.7.

3.2 Related Work

A few researchers have addressed the modeling of a multi-tier data center. Modeling or simulation of each individual tier separately in a multi-tier data center has been studied in [43, 53–55]. Modeling of a multi-tier data center has been done in [34, 56]. [34] models a multi-tier data center as a network of queues and [56] models each service tier as a fine-grained component to capture practical implementations of each server. However, the design of a simulation platform for the analysis of multi-tier data centers taking into account the effects of all tiers together has not yet been addressed.

[57] demonstrates the benefits of IBA in multi-tiered web services. But, the system analyzed in [57] is restricted in size and system capability in the sense that it does not include the Java 2 Enterprise Edition (J2EE) [58] based application servers, which have become the *de facto* standard for designing data centers. Our prototype data center includes J2EE specifications. Further, they conducted the comparative analysis with low traffic and small cluster sizes. In this paper, as part of an application of our simulator, we perform the evaluations and comparisons of IBA and 10GigE with/without TOE on a much larger scale and under high load conditions.

Modeling of power consumption in computer systems has been done at various granularity levels, starting from large data center to individual components of a system either through real measurements, estimations from performance counters or using a combination of both [59, 60]. Power management for communication protocols has been studied in [12, 51, 61]. [51] compared the power based benefits of using Remote direct memory access (RDMA) adapters, such as IBA and 10 GigE with TOE against TCP/IP Ethernet. However, they conducted their experiments by using only two servers and with micro-benchmarks. Moreover, energy consumption in server farms using simulation is studied in [12], though the scale of simulation is restricted (results for only up to 32 nodes are shown). Our simulation results for power measurement across the servers of a data center span over cluster configuration with much larger nodes. Besides, there have been significant researches on power management methodologies that focus only on individual tiers such as a web server tier without taking into consideration the effects of all the tiers

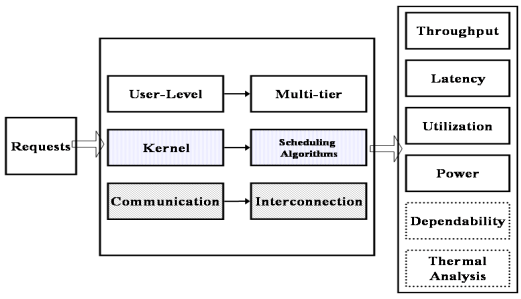


Figure 3.1: Functional Overview of Simulation Platform

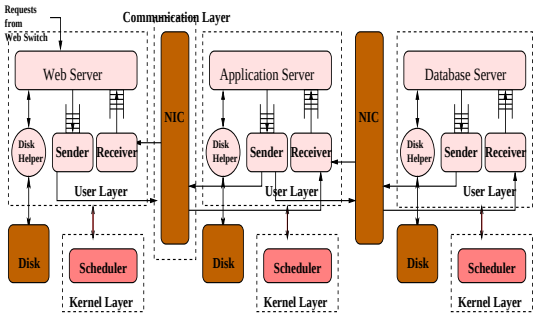


Figure 3.2: Architectural Details of the Three Layers of Simulation Platform

of a multi-tier data center. For example, [62,63] analyze only front end power and energy management techniques. [64] discusses local and cluster-wide power management techniques using just a web server simulator. The simulator proposed in this work considers the effects of all tiers and can accurately estimate power consumption across the servers of a multi-tier data center.

To the best of our knowledge, no previous works have looked at multi-tier data center simulator, which can analyze a cluster based data center with detailed implementation of each individual tier. Also, power measurement in a large consolidated environment like a multi-tier data center has not been analyzed using a comprehensive multi-tier simulator in previous literatures.

3.3 Design of Simulation Platform

In this section, we discuss the design and implementation issues of our simulation platform. The functional building blocks of our multi-tier data center simulator, written in CSIM [65], are shown in Figure 3.1.

The simulation is configured into three layers (a communication layer, a ker-

nel layer and a user-level layer) for modeling the entire stack starting from the communication protocols to the application specifics. Such a three layer abstraction provides the flexibility and scalability in analyzing various design details as described next. Intra-cluster communication is modeled at the communication layer. For intra-cluster communication, the simulator supports IBA and Ethernet over TCP/IP as the underlying interconnect technology. The IBA communication modules follow the semantics of the IBA specification [66] and support major functionalities of the IBA, including communication with Completion Queue (CQ) and Send/Receive Queues. Other communication paradigms/protocols can be incorporated into the communication layer by simply plugging in the appropriate timing parameters.

At the kernel layer, we modeled the Linux scheduler 2.6.9, which maintains a run queue for each CPU in the system. Since we assumed all the CPUs in a node as one single resource, we have just one run queue for each server node. Each time the scheduler runs, it finds the highest priority task among communication and application processes in the simulator. If there are multiple tasks with the same priority, those tasks will be scheduled by the round-robin scheme. When calculating the time slice, the scheduler punishes CPU intensive tasks and rewards I/O intensive tasks as Linux 2.6.x. kernel does.

The high level application/user layer captures the important characteristics of a three-tier architecture. The user level layer consists of the following categories of processes - WS, AS, and DB processes; auxiliary processes like disk helper for supplementing server processes; communication processes such as Sender/Receiver processes. The front-end tier (web server tier) is the closest to the edge of a data center and is responsible for serving either static requests from memory or disk or forwarding dynamic requests to application server nodes. In our case, we use Apache [41] as the front end web server interface. The middle tier (application server tier) is responsible for handling the dynamic web contents, and we use the publicly available JOnAS [42]. On receiving the dynamic requests, the application server nodes process them. Being in middle between the front-end and back-end in a data center, application servers include interfaces to both tiers: translating clients' messages in HTML to SQL and vice versa. The AS nodes' functionalities are based on J2EE specifications. To model EJB session beans [43] in our sim-

ulator, we implemented the main components - web container which takes care of the presentation logic, EJB container that handles business logic and controls interaction with the database. Finally, the database server, based on the SQL semantics [45], is dedicated for database transactions. It either primarily accesses the cached data in the database or sends a request to disks.

From measurements on our prototype data center (discussed later) with the RUBiS workloads, we observe that 97% of the database requests are satisfied from the cache, thus, only 3% of the requests are directed to the disks. We use these cache and disk access rates in simulating the database tier. Since we assumed mirroring as data sharing method among all the database servers, we modeled locking operation to ensure data coherence. We also assumed that there is one server process per node to simplify the implementation of each tier of our three-tier data center simulator.

The architecture of our simulator is shown in Figure 3.2. Here, Network Interface Card (NIC) and Disk are devices and others are processes that reside inside the CPU in a server node. We assumed that CPU, Disk, and NIC are basic computing resources in a node responsible to process requests. For simplicity, each resource is modeled as an M/M/1 queue in the simulator. In the simulator, nodes separately process the requests which they receive through the communication layer. After processing the requests, they send the responses back through the communication layer and this procedure follows for every node in the cluster. In addition, since this is an event-driven simulator, we can trace each individual request. Since we simulate each node independently, we can change the configuration of each tier easily, such as varying the total number of nodes in the entire cluster and the number of nodes for each tier.

The main advantages of our simulation platform is that it is comprehensive, flexible, and scalable. It is comprehensive as the critical behavior of an entire data center as well that of each node has been implemented in a lucid manner. For example, we implemented database synchronization and load distribution for web requests in the application layer. The task scheduler was deployed in the kernel layer and the communication primitives are captured in the low level communication layer. Its flexibility comes from the fact that the simulator provides the ease to modify the behavior of any entity in a simulator without affecting other parts

by simply plugging different parameters, policies, or even implementing different features. For instance, we need to only vary the latency timing parameters to analyze the behavior of both IBA and 10GigE network adapters without changing any other parts of the simulator. Also, we can analyze the behavior of several cluster configurations for optimizing various objective functions without having to rearrange or change our simulator. Moreover, although the current version of the simulator is used for performance and power analysis, it can be extended to study other aspects such as thermal behavior and reliability issues. The scalable aspect of the simulator enables it to simulate large data centers. There can be many possible applications of the simulator such as comparative performance analysis of high speed interconnects, power and thermal analysis, determination of optimal cluster configuration to meet a target performance level, fault tolerance and reliability of data centers.

3.4 Prototype Configuration and Validation

In this section, we present the system configuration, and the simulator validation. The design and implementation of the prototype data center used in this chapter is delivered in chapter 2.

3.4.1 System Configuration and Workload

Table 3.1 summarizes the system and network parameter values used in our experiments. These values were obtained from measurements on our prototype data center using micro benchmarks, hardware specifications, and estimation. System parameters, for example, context switch and interrupt overhead are obtained by running benchmarks. Disk read time is based on the hardware specifications. Network latency and CPU utilization is obtained by estimation, which will be presented in this section. Clients generate requests according to the RUBiS benchmark, which follows the TPC-W compatible specification [67]. For generating the traffic, we assumed the log-normal distribution for the file size as reported in [68] along with the parameters and think time interval of clients shown in Table 3.2.

In order to correctly simulate the characteristics of the AS and DB tier with

dynamic requests, estimating CPU service times for AS and DB servers is essential. We estimate the CPU service time of each tier node. From measurements on the prototype, CPU utilization and throughput of each node is obtained. We then use the Utilization Law [34] to obtain the service time as shown in equation (3.1).

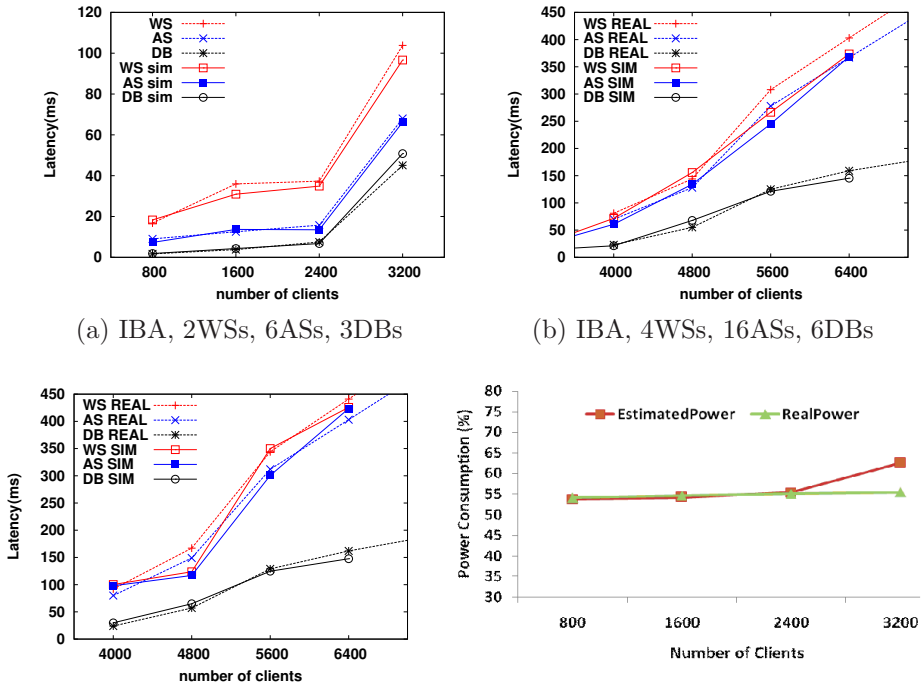
$$T = \frac{\rho}{\tau} \tag{3.1}$$

Table 3.1: Timing Parameters

Parameter from Measurement	**Value (ms)**
Context switch	0.050
Interrupt overhead	0.01
TCP connection	0.0815
Process creation overhead	0.352
Disk read time	$7.89 \times 2 + (0.0168 \times \text{Size}) + (\text{Size}/368 - 1) \times 7.89$
Network latency (IBA)	$0.003 + 0.000027 \times \text{Size}$
Network latency (1GigE)	$0.027036 + 0.012 \times \text{Size}$
Parameter from Estimation	Value (ms, KB)
Network latency (TOE)	$0.0005409 + 0.0010891 \times \text{Size}$
Network latency (10GigE)	$0.0009898 + 0.0017586 \times \text{Size}$

Table 3.2: Traffic Characteristics

Parameter		**Value**
File Size	WS $\rightarrow$ AS	$\alpha = 0.1602, \mu = 5.5974$
	AS $\rightarrow$ DB	$\alpha = 0.4034, \mu = 4.7053$
	DB $\rightarrow$ AS	$\alpha = 0.5126, \mu = 22.134$
	AS $\rightarrow$ WS	$\alpha = 0.5436, \mu = 56.438$
	WS $\rightarrow$ client	$\alpha = 0.5458, \mu = 50.6502$
Dynamic Req/Total Req		0.672
Avg. DB Req per Dynamic Req		2.36
Think time interval		Exponential with a mean of $7s$



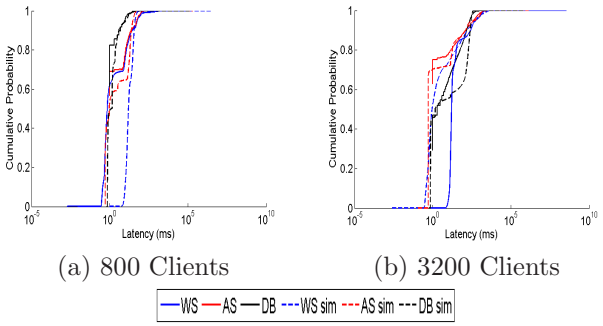
(a) IBA, 2WSs, 6ASs, 3DBs

(b) IBA, 4WSs, 16ASs, 6DBs

(c) 1G, 4WSs, 16ASs, 6DBs

(d) Validation of Power Estimation

Figure 3.3: Simulator Validation



(a) 800 Clients

(b) 3200 Clients

Figure 3.4: Service time CDFs from Measurement and Simulation

where, T is the service time, ρ is the utilization of the busiest resource (CPU) and τ is the throughput of each node.

Next for simulating 10GigE network adapter, we need the latency timing equation. Through linear regression analysis, we estimated the latency equations for two kinds of 10GigE implementations (with/without TOE) in terms of message size by obtaining latency from throughput [52] as follows:

$$Latency = \frac{Data\ Size}{Throughput} \tag{3.2}$$

According to the specifications of 10GigE, the hardware characteristics of 10GigE is almost the same as 1GigE. Further, the network latency for both 1GigE and 10GigE includes similar hardware overheads caused by switches and network interface card. Thus, we can safely assume that our simulator can be extended to capture the behavior of 10GigE.

Our simulator is capable of measuring power consumption across the servers of a multi-tier data center. To incorporate power measurements in our simulator, we measured the server utilization using some modifications to the heuristics given in [62]. The server utilization was measured as

$$U_{server} = \frac{N_{static}A_{static} + N_{dynamic}A_{dynamic}}{\text{Monitor Period} \times \#\ \text{WS nodes}}, \tag{3.3}$$

where N_{static} and $N_{dynamic}$ are the number of static and dynamic requests respectively; A_{static} and $A_{dynamic}$ are the average execution time of static and dynamic requests respectively; Monitor Period is the time interval over which we measure server utilization and power. We divided the numerator of the equation for server utilization from [62] by the total number of web server nodes in the cluster, since our simulator processes the client requests in parallel at the front end tier [69]. We then computed the server utilization using the above equation. Power consumption is expressed as percentage of the peak power across the data center. We used the power model of Wang *et al.* [70] to estimate power consumption. According to their power model, for a fixed operating frequency, both the power consumption of the server and the application throughput are approximately linear functions of the server utilization. The peak power consumption of the data center was obtained by simple linear regression between power and server utilization.

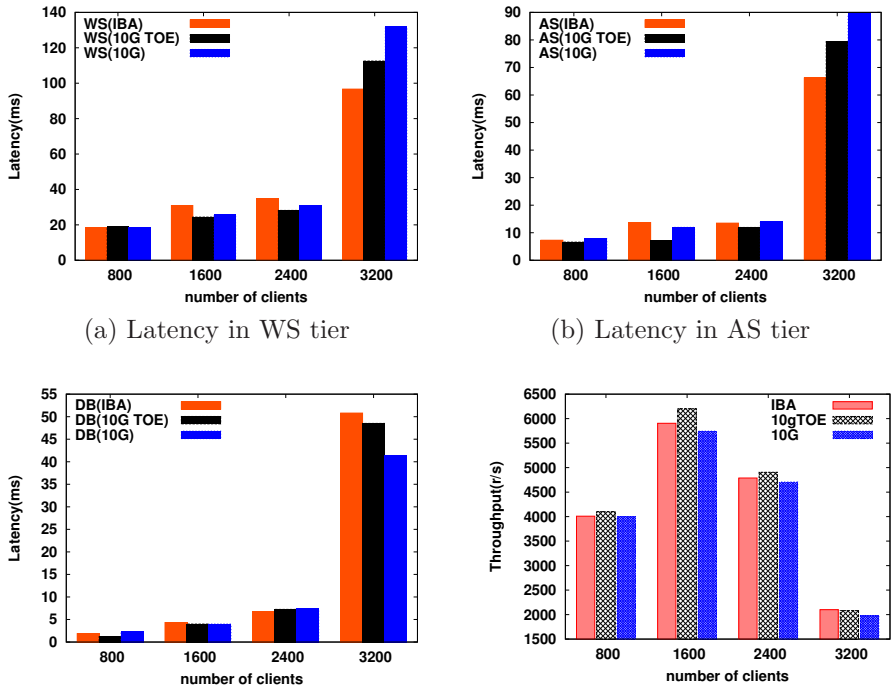
3.4.2 Simulator Validation

We validated our simulator with 1GigE and IBA based prototype data centers with different number of nodes for each tier. The average latencies of each requests from simulation results were compared with those from real measurements as shown in

Figure 3.3. Here, we selected the number of nodes for each tier in the prototype data center when the latency of requests was minimized. In Figure 3.3a and 3.3b, the underlying network architecture was IBA. Figure 3.3c shows the latencies over 1GigE from actual measurements and simulation. In the above results, the latency of each tier from simulation closely matched with the real measurements, which implies our simulator's underlying architecture captures critical paths in each tier. The average latencies from the simulator match closely with the average deviation of 10%. Note that the above deviation is independent of either the number of nodes or the number of clients used in experiments.

Figure 3.4 shows the CDF plots of service times of each tier for the simulation and real measurements. According to [68], service time of an actual machine for RUBiS workload follows the Pareto distribution. We notice that the service time of each tier from our simulator closely matched with the real measurements under both the cases of light and heavy traffic. It implies that the modeled behavior for each tier in the simulator is very close to that of actual servers. From both the latency and CDF comparison, we can conclude that our simulator captured accurately the critical behaviors to process requests and showed similar traffic characteristics to real environments.

We validated the estimated power consumption using our simulator with real power measurements. Figure 4.2 shows the variation of power consumption with the number of clients corresponding to real and estimated power measurements. We used a power measurement device WT210 [71] to measure power usage of nine (the power strip used could only accommodate nine connections at a time) randomly chosen nodes from our prototype data center. These nine nodes were configured as 2 web servers, 6 application servers and 1 database server. The power consumption from our simulator closely matched with that of real measurements with an average deviation of around 3%. This result indicates that our simulator is capable of accurately estimating power consumption across the servers of a multi-tier data center.



(a) Latency in WS tier

(b) Latency in AS tier

(c) Latency in DB tier

(d) Throughput

Figure 3.5: Comparison between IBA and 10GigE

3.5 Performance Evaluation

In this section, we demonstrate three important applications using our simulation platform to show the ability of our simulator for guiding design of data centers. First, comparison between two popular interconnection networks is done in Section 3.5.1. Second, power estimation using simulator is elaborated in Section 3.5.2. Finally, performance prediction according to various system configuration is described in Section 3.5.3.

3.5.1 Comparison between IBA and 10GigE

We conducted comparative performance analysis of 10GigE and IBA for three-tier data centers in terms of latency and throughput. The graphs in Figure 3.5 depict the average latencies and throughput of requests in terms of the number of clients for three configurations (IBA, 10GigE with TOE and 10GigE without TOE). We

used 2 WS nodes, 6 AS nodes, and 3DB nodes for this simulation. Notice that the latency of WS and AS includes the communication overhead between WS and AS and between AS and DB, respectively. We observe that 10GigE, regardless of TOE, has a better or very similar performance over IBA in latency measurement, provided the number of clients is under 3200. However, with the increase in traffic beyond 3200 clients, the latency of IBA comes out to be 14% smaller than 10GigE with TOE and 26.7% lesser than 10GigE without TOE.

Under high traffic, the communication cost which is determined by the mean number of jobs in communication layer is a dominant factor of the response time. Based on the queuing theory, the mean number of jobs in a queue J is obtained by

$$J = \lambda \times T, \tag{3.4}$$

where λ is the inter-arrival rate of requests and T is the mean service time of a single request. Accordingly, given the same inter-arrival rate of requests, the latency under high traffic is mainly affected by the mean service time of a single request, which is the average latency of a single file for the workload in our simulation. Even though the file sizes vary from 1.5KB to 16KB for the RUBiS workload, most of them are clustered around 2.5KB [68]. According to the latency estimation in Table 3.1, the latencies of 10GigE with TOE and without TOE are greater than that of IBA with 2.5KB file size. Hence, IBA has the smallest response time among all the three network interconnections under high communication workload.

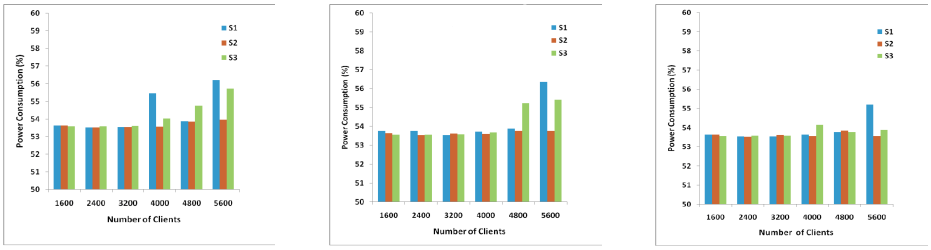
We observe that the latency for the DB tier in case of IBA is much higher than that of 10GigE with and without TOE under high traffic (3200 clients as shown in Figure 3.5a, 3.5b, and 3.5c). This can be explained due to the fact that the throughput of IBA in case of 3200 clients is higher than that of 10GigE , which increases the number of requests generated for the DB tier and thus degrades the response time of the DB tier. This is due to the fact that the blocking operation, which ensures consistency among databases, can cause system overheads in the database servers when the number of concurrent writing operation increases. We notice that the WS part of bar graphs is negligible in some results since the service time of WS which includes communication overheads between AS and WS is less than 10% of the overall latency in most experiments.

3.5.2 Power Measurement

As validated in section 3.4, our simulator can estimate power consumption across the servers of a data center. The flexibility offered by our simulator allows one to measure the power consumption for any selected number of clients and with any chosen cluster configuration. From Figure 4.2, we notice that the power consumption across the cluster nodes increases with the increase in the traffic from 800 to 3200 clients for both the prototype data center as well as the simulation platform. Figure 3.6 shows the estimated power consumption using our simulator across 64 nodes clustered data center connected with IBA, 10GigE with TOE and 10GigE without TOE respectively. Here, S1, S2, and S3 represent three different cluster configurations with varying number of nodes in the WS tier, AS tier, and DB tier respectively for the same total number of nodes. In S1, 8 web servers, 32 application servers, and 24 database servers were used. In S2, 8 web servers, 40 application servers, and 16 database servers were used. S3 consisted of 16 web servers, 32 application servers, and 16 database servers. We expressed power as percentage of peak power. As we observe from Figure 3.6, power consumed when the number of clients is less, comes out to be more or less same for different cluster configurations in all the three cases (IBA, 10GigE with TOE, 10GigE without TOE). This is because when the traffic is less, the servers are under utilized, and so do not exhibit large power usage. However, with the increase in the traffic, server utilization is enhanced, causing the servers to consume more power. As we observe, the power consumption varies with different cluster configurations even for the same total number of nodes. This implies the dependency of some specific tier in deciding the power dominance. These results can provide one with the insight into determining the optimal cluster configuration, in terms of reduced power consumption across the cluster (this can be especially important for the designer of a multi-tier data center, where he has to determine the distribution of the number of nodes across each tier based on power efficiency).

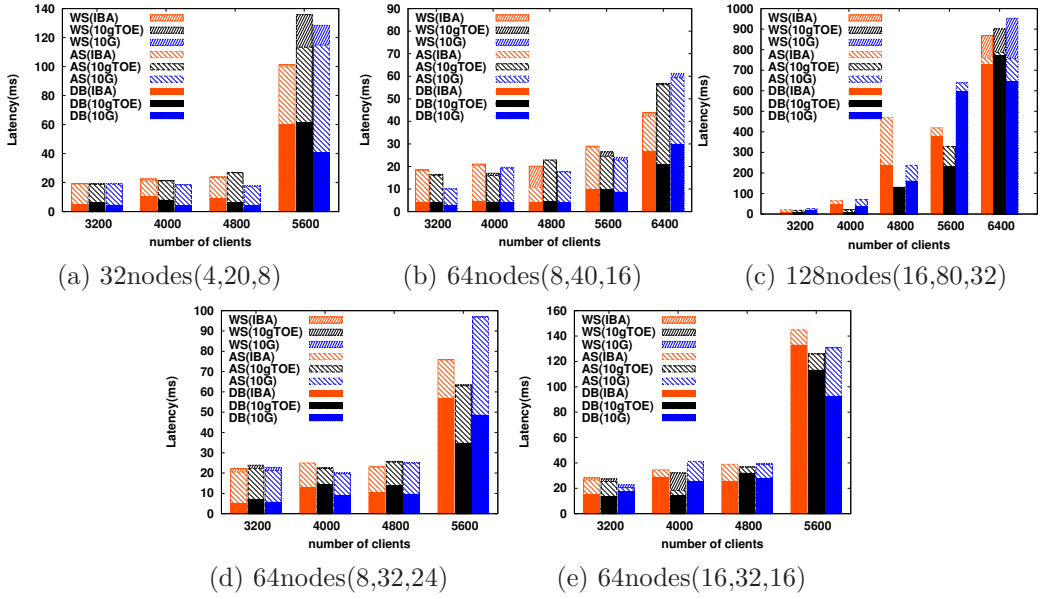
3.5.3 Server Configuration

To demonstrate the scalability and flexibility of our simulator, we conducted performance analysis of a three-tier data center with increased number of nodes and



(a) IBA, 64 nodes (b) 10G TOE, 64 nodes (c) 10G, 64 nodes

Figure 3.6: Estimated Power Consumption



(a) 32nodes(4,20,8) (b) 64nodes(8,40,16) (c) 128nodes(16,80,32)

(d) 64nodes(8,32,24) (e) 64nodes(16,32,16)

Figure 3.7: Latency with varying number of nodes for each tier (WS, AS, DB)

with different interconnection networks. Figure 3.7a, 3.7b, and 3.7c shows the variation of latency with different number of clients for 32, 64, and 128 nodes respectively. The combination of the number of nodes in each tier was selected randomly. We observe that the performance of the data center in terms of latency improves (with the decrease in latency) with an increase in the total number of nodes in the system. This is due to the efficient distribution of system load across multiple nodes of the tiers. However, we observe that this trend is violated for the case with 128 nodes as shown in Figure 3.7c. This is due to the large overhead caused by the DB tier locking operation, which causes the total response time to

be dominated by the DB portion of latency. These results imply that the performance might not be enhanced in proportion to the number of total nodes. The underlying mechanisms in each server may cause unexpected results if the scalability increases. The above results demonstrate that without real measurement, we can suggest, for given system characteristics, the right choice among possible combinations of server configuration by the simulation.

Next we examine latency variations for a fixed number of total nodes in a data center for different cluster configurations. Figure 3.7d, 3.7b, and 3.7e depicts the results. We observe that performance of a data center in terms of latency varies based on the different configuration of nodes selected for each tier and also with the type of interconnection used. Also, notice that a better configuration of the system for minimizing the response time can be achieved by selecting more number of AS nodes than WS and DB nodes. This is because AS nodes have CPU intensive jobs and interfaces with two other ends, thus increasing them efficiently distributes the load across the cluster. Thus, our simulator can be used to change various configurations to determine the desired distribution of nodes across the cluster for optimizing performance.

3.6 Conclusions

In this thesis, I first proposed the design and implementation of a prototype three-tier IBA and 1GigE connected the data center. Since the prototype was limited in terms of the number of cluster nodes, system configurations, and driving workloads, this study presented a comprehensive, flexible and scalable simulation platform for the performance and power analysis of multi-tier data centers. The simulator was validated with RUBiS benchmark measurements on the above mentioned prototype data center. Using three application studies, this thesis demonstrated how designer of a multi-tier data center might use the proposed simulation framework to conduct performance and power analysis. The first study detailed a comparative analysis of the performance of IBA and 10GigE under varying cluster sizes, network load and with different tier configurations. The results from this experiment indicated that 10GigE with TOE performs better than IBA under light traffic. However, as the load increases, IBA performs better than 10GigE both with and without

TOE. Next, this work introduced a methodology to perform power measurement in a multi-tier data center using MDCsim. Finally, using simulation platform, confguration analysis for performance optimization in terms of reduced network latency was demonstrated.

As a part of the future work, I intend to test MDCsim with diverse workloads besides RUBiS. Further, I plan to extend the proposed simulation platform for conducting thermal analysis, reliability measurement and servers' provisioning (*i.e.,* determination of the optimal number of servers in each tier based on performance and power) in a multi-tier data center. Also, I am working towards making MDCsim a publicly available tool for use in research community in the near future.

Chapter 4

A Dynamic Energy Management Scheme for Multi-tier Data Centers

4.1 Introduction

Multi-tier data centers are being increasingly used by Internet service providers for hosting modern applications with dynamic Web contents. A typical three-tier architecture with front-end Web servers, middle-level application servers and back-end database servers provides a modular, flexible and scalable environment for Web hosting [69]. However, the surging energy consumption of these data centers has become a serious concern from the economic and environmental standpoints. Server farms in U.S. are expected to consume 100 billion kWh at a cost of $ 7.4 billion per year by 2011 [72]. Service providers such as Google, Microsoft, Amazon, Akamai and Yahoo! spend millions of dollars annually to power on and cool their data centers. Therefore, instead of focusing on only high and scalable performance, energy efficiency has become a first order goal to minimize energy consumption and reduce the operating budget of data centers.

Most of the techniques for managing energy consumption in data centers fall into three broad categories. The first methodology involves dynamically turning on/off servers to save energy [16, 17]. The second approach uses Dynamic Voltage Frequency Scaling (DVFS), where a system dynamically adjusts the frequency/voltage to lower power usage [17]. The third technique uses Dynamic

Power Management (DPM) [15], which utilizes sleep states of various components of a server and decides when and for how long each component should be put to sleep to save power. Many proposed mechanisms combine some of the above techniques to enhance energy efficiency in data centers [16, 18, 19]. However, a systematic coordination of all the three techniques to boost energy efficiency without sacrificing performance is not trivial. Since dynamic provisioning is employed at the global/system level while the DVFS and DPM are employed at the local/component level and, worse, they are agnostic to each other [21]. We are not aware of any prior work that has combined dynamic provisioning of the number of servers, DVFS and DPM to balance the performance and energy in data centers.

Dynamic provisioning of resources *i.e.,* allocation and deallocation of servers to applications through analysis of arriving workload patterns adapts well to the dynamic Internet traffic, compared to static provisioning [16, 69]. The authors in [18] propose a coordinated dynamic provisioning scheme with DVFS for a single-tier data center, where a central controller decides the CPU speeds of the servers at different time frames and then the servers run at the same speed. The globally determined speed or sleep period might be reasonable, but it may not utilize local workload patterns such as frequent short idle periods as reported in [15]. On the other hand, we may conservatively use a local energy management strategy such as immediately waking up a system from a sleep mode (DPM) as a request arrives [15]. However, it may not fully exploit the potential to enhance energy efficiency without the global knowledge of a system. Therefore, we need to investigate how to coordinate dynamic provisioning with local energy management schemes, which is of interest in this paper.

In this work, I propose a three-prong approach, called Hybrid, for optimizing the energy consumption of multi-tier data centers, while satisfying the user specified SLAs. We develop a comprehensive mathematical model considering the performance (SLA) constraints and energy consumption. The model consists of two parts; the global energy consumption of a data center and the energy consumption of individual servers in each tier. Since the problem is NP-complete, I propose two heuristics for solving the global and local optimizations based on queueing theory. The first heuristic, for global optimization, dynamically provisions the required number of servers across each tier. The number of servers is

obtained by modeling the multi-tier data center as a queueing network and by using the Mean Value Analysis (MVA) [73] for estimating the performance of each tier. The second heuristic proactively decides the CPU speed (DVFS) and the duration of sleep states of a server (DPM) using the results from MVA for local energy optimization. The proposed scheme is evaluated with a multi-tier data center simulator, which in turn is validated with real measurements in terms of energy consumption and response time using a prototype three-tier data center of 25 servers. We used two benchmarks, RUBiS [74] and TPC-W [48] for both validating our simulator and evaluating the proposed heuristics.

The experimental results indicate that our proposed three-pronged Hybrid solution could save energy up to 50%, compared with the base case of static provisioning of a multi-tier data center without utilizing DVFS and DPM. We demonstrate the energy efficiency of the proposed approach compared to three other dynamic provisioning techniques: dynamic provisioning without local energy management, dynamic provisioning with DVFS, and dynamic provisioning with the DPM scheme in [15]. For the RUBiS workload, our approach provides an additional energy saving by 46% compared to dynamic provisioning only and by 42% compared to the recently proposed PowerNap technique [15]. This significant amount of energy savings stems from the fact that the proposed scheme creates longer total sleep durations per server and provisions less number of servers to handle any sudden increase in the workload. All the dynamic schemes exhibit similar energy behavior for the TPC-W workload since this workload generates many small requests, thereby making it difficult to utilize the local DVFS and DPM techniques effectively. This study also shows that the proposed scheme, Hybrid, incurs less number of sleep transitions than the PowerNap [15] scheme.

The rest of this chapter is organized as follows: Section 4.2 discusses the related work. Relevant backgrounds are described in Section 4.3. Section 4.4 and 4.5 presents the formulation of the problem and the methodology used for solving it. Our experimental results and discussions are presented in Section 5.4, followed by concluding remarks and future work in Section 4.7.

4.2 Related work

Resource capacity planning and dynamic provisioning for QoS control have been explored in the past. Chase et al. [16] presented an approach using economic theory for resource management in hosting centers with emphasis on energy and performance. They, however, considered only a single web tier with small number of web servers for their experiments and did not leverage the benefits of using DVFS and sleep states for saving energy. An analytical model of a multi-tier data center was proposed in [34]. Dynamic provisioning of a multi-tier data center has been explored in [69]. Both of the above works have focused on modeling and dynamic provisioning with respect to only performance (response time and throughput) of multi-tier data centers without considering their energy/power. One of the contributions of this paper is to consider dynamic provisioning for optimizing both performance and power/energy consumption in multi-tier data centers.

Power management in server systems adopts mostly three techniques, namely, shutting down selected servers, modulating CPU operating frequency/voltage and dynamic power management, which puts a system to sleep states to save power. Initial studies [16, 19] have used the methodology of shutting down the servers that are either not in use, or have low load to conserve power. Subsequent studies [17, 20] have looked at optimizing power efficiency by monitoring system load and dynamically modulating the CPU frequency/voltage through DVFS. The authors in [17] have also used a combination of DVFS and turning on/off servers approaches for power management. Also, finding the required number of servers to minimize energy consumption in a single Web tier has been done by Rajamani et al. [75]. They use a load distribution scheme to schedule the requests on a subset of servers and turn others off. However, our work is different from [75] in two ways. First, we propose a dynamic provisioning scheme to determine the required number of servers in all the three tiers together. Second, we make use of DVFS and DPM for obtaining better energy savings.

Power/energy consumption in server farms has been looked at with different perspectives. Gandhi *et al.* [76] used a queuing model for allocating a given power budget across the servers so as to maximize performance. Chen *et al.* [18] made the first formalism for the problem of reducing server energy consumption in hosting

centers running multiple applications, while meeting user specified performance bounds. They optimize energy efficiency by considering both static server provisioning and DVFS. This study differs from theirs in three aspects. First, we consider dynamic server provisioning in a multi-tier data center environment unlike their evaluation for single tier of servers. Second, we optimize energy savings by considering all three techniques: dynamic provisioning, DVFS, and DPM. The work done in [18] does not consider the benefits of sleep states for servers in achieving further energy savings. Third, they decided CPU frequencies for servers by estimating the performance of a server farm against all the available CPU frequencies. We propose a more efficient technique to find an energy optimal frequency at which a server selected by dynamic provisioning should run to minimize energy usage. Meisner *et al.* [15] proposed PowerNap to transition a system back and forth between a high performance state and a low power sleep state in response to instantaneous load to conserve power. The proposed heuristics in this work avoid a server's idle period by proactively putting the server to sleep much before the start of an idle period, thereby, replacing the periods of low server utilization with power saving sleep states. The authors in [77] proposed multi-mode energy management schemes for clusters running multi-tier applications. The authors made use of both DVFS and multiple sleep modes available in today's server systems to provide heuristics for power savings. Although our work has the same inspiration, it differs from [77] in the following two ways. First, dynamic provisioning of servers is not considered in their work. Second, we explicitly address the impact of power management techniques on user perceived SLAs and perform a detailed tradeoffs analysis between them using both simulations and a mathematical model.

4.3 Background

In this section, we describe system assumptions and our problem formulation along with definitions for subsequent sections.

We define an *active* server as one which is either in the turned-on or sleep state and an *inactive* server to be one in the turned-off state. We attempt to address the following two questions. First, given a total of N homogeneous servers in a multi-tier data center, how do we allocate $n \leq N$ active servers to process requests

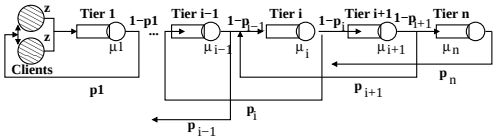


Figure 4.1: A closed queueing network

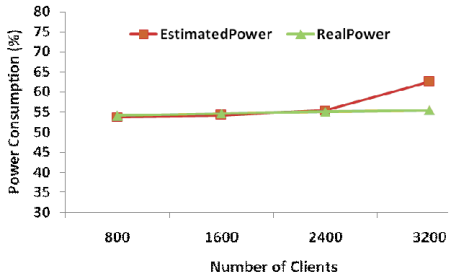


Figure 4.2: Estimating power consumption with CPU utilization (results from the RUBiS benchmark)

from users? Second, how do we schedule each active server's CPU speed s and decide on the parameters for DPM that minimize the energy consumption while satisfying the SLAs?

To model the performance aspects in our problem, we view a multi-tier data center as a closed queuing network (Figure 4.1) [34]. A user sends a new request to the front-end tier after waiting for a certain amount of time after getting the response for the previous request back. Requests served in a tier i is either forwarded to tier $i+1$ with a probability of $1-p_i$ or responded back to tier $i-1$ with a probability of p_i. We define the visit ratio to capture the phenomenon that a request received at a tier i can generate multiple requests to the next tier $i+1$, which commonly occurs between the Application server (AS) tier and the Database server (DB) tier. The servers in the same tier are assumed to have negligible intra-tier traffic [68]. The dynamic characteristics of the Internet traffic is captured by varying the number of users at the beginning of each time frame. Although a request consumes multiple resources in a server, the overall power consumption of a server can be estimated by the analysis of CPU utilization [14, 76]. In this work, the CPU utilization refers to the utilization of a server unless otherwise specified [14]. However, we consider all other resources in a server, especially for the sleep states.

We model the power consumption of a system as a function of its CPU utiliza-

tion and speed, which follows the observation reported in [14, 76].

$$P(s, \rho) = \rho \left[a(s - s_{min})^{\alpha} + b \right], \alpha \geq 1 \,, \tag{4.1}$$

where ρ is the utilization of a system, a is a coefficient which depends on the system, s_{min} is the lowest CPU speed, α is a factor that generalizes the CPU technology and b is the power consumption when the CPU is operated at s_{min}. Then, the peak power consumption is $P(s, 1.0) = 1.0 \times [a(s - s_{min})^{\alpha} + b]$. As reported in [14], we assume that the power consumption of a system at a given CPU speed is linearly proportional to the CPU utilization of the system. Our assumption is further supported in [70] where the authors show the power consumption of a server is approximately linear to the server utilization which in turn can be estimated from CPU utilization. We also verified this by comparing actual power consumption from our prototype data center to the estimated power consumption based on CPU utilization as shown in Figure 4.2.

The energy consumption of a system is the integral of the consumed power over the measured time.

$$E = \int P(s, \rho)\, dt \tag{4.2}$$

We establish the relationship among the energy consumed for processing requests ($E_{dynamic}$) , idle period (E_{idle}), and sleep period (E_{sleep}) in the following corollary.

Corollary 1. *For a given CPU speed s, CPU utilization ρ, time interval of length T, the ratio of idle power consumption to the peak power consumption $0 < k < 1$, and sleep power consumption to peak power consumption $0 < k' < 1$, the energy consumption E of a server is given by*

$$E = P(s, 1.0) \left[(T - t)(\rho(1 - k)) + k) + tk' \right] \,, \tag{4.3}$$

where t is the total amount of time when the server is in sleep state during the interval $[t_0, t_0 + T]$.

Reasoning for the above corollary is given as follows. The energy consumption of an active server for a given time frame is the sum of the energy consumed when

the system is in the dynamic, idle, and sleep states *i.e.,*

$$E = E_{dynamic} + E_{idle} + E_{sleep}$$

However, the right side of the previous equation is equal to

$$\int_{t_0}^{t_0+T} \{(\rho(1-k))P(s,1.0) + kP(s,1.0) + k'P(s,1.0)\}\ dt$$

Since we assume that the CPU speed remains constant for a certain time frame, the energy consumption is the product of elapsed time and consumed power according to Equation 4.3.

The energy consumption for a server is a function of the CPU speed and utilization. Thus, for a given CPU utilization, we can determine the energy-optimal CPU speed s_0 that minimizes the server energy consumption.

Corollary 2. *Given the average number of instructions m required to process a request and the predicted number of requests r, the energy consumption E of a server is minimal at a certain speed s_0.*

Proof. Let ρ be the utilization during T and $P(s,\rho)$ be the power consumption given in Equation 4.1. The energy consumption E thus becomes

$$E = \int_{t_0}^{t_0+T} P(s,\rho)\, dt = \rho P(s,1.0)T \tag{4.4}$$

The total number of instructions required to process the requests during an interval T for a CPU speed s and CPU utilization ρ is $T \times s \times \rho$. A simple formula that relates the number of instructions during T to the average number of instructions required for a request is $Ts\rho = mr$. Therefore, the energy consumption E of a server is given by

$$E = P(s,1.0)mr/s \tag{4.5}$$

To find a CPU speed s_0 that minimizes E, we differentiate E w.r.t. CPU speed s.

$$\frac{d\,E}{d\,s} = \frac{mr(sP'(s,1.0) - P(s,1.0))}{s^2} \tag{4.6}$$

In $P(s,1.0) = a(s - s_{min})^{\alpha} + b$, if $\alpha = 1$, E is a strictly decreasing function

since $\frac{dE}{ds} < 0$. Thus, the maximum CPU speed s_{max} minimizes the total energy consumption of a server, therefore $s_0 = s_{max}$. However, if $\alpha = 3$, there exists a $s_{min} \leq s_c \leq s_{max}$ such that $\frac{dE}{ds}(s_c) = 0$. Since CPU speeds are discrete, the CPU speed s_0 closest to s_c minimizes the energy consumption. When the idle and sleep state are considered, the proof remains similar. Hence, for processing a request, the energy consumption of a server is minimal at s_0. □

Since the energy consumption is the product of processing time and consumed power, we should consider both of these factors for potential energy savings. Decreasing CPU speed lowers power consumption, but extends the processing time of a job. Increasing CPU speed shortens the processing time, but increases power consumption during that interval, which again may not save significant energy. However, when we cannot meet the performance goal with the energy optimal CPU speed, we have to run the CPU at the maximum possible speed to achieve the desired performance. In short, the CPU speed of a server should be greater than or equal to s_0 when we consider the trade-off between performance and energy.

4.4 Problem formulation

The problem of optimizing energy consumption with performance constraints can be formulated as follows: Using Corollary 1, for a given average CPU speed s_i of servers in a given time frame T, the cost C (energy consumption per server) can be defined as

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}, \tag{4.7}$$

where $c_i = P(s_i, 1.0)[(T - t)(\rho_i(1 - k) + k) + tk']$ and ρ_i is the utilization of a server in a tier $1 \leq i \leq n$. k and k' are hardware-dependent constants, which can be determined through real measurements. Variables in the cost function are s, ρ_i and t. Different values of the function $P(s_i, 1.0)$ can be evaluated by measuring the power consumption of server at different speeds. The objective function representing the energy consumption of a multi-tier data center is given

by

$$CV = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}, \tag{4.8}$$

where v_i in $V = \begin{bmatrix} v_1 & v_2 & v_3 & \cdots & v_n \end{bmatrix}^T$ is the number of active servers in tier i in a given time frame. Thus, our goal is to determine a vector V and the variables in the cost functions which will minimize the overall cost function (Equation 4.8) under the following constraints.

Suppose

- T : response time of a multi-tier data center
- T_{SLA} : response time constraint specified in SLA
- N : total number of servers
- τ_i : throughput of a server in tier $1 \leq i \leq n$
- r_i : expected number of requests for tier i per *time frame*

then,

$$\begin{aligned} T &\leq T_{SLA} \\ v_i &\geq 1 \\ \sum_i^n v_i &\leq N \\ T_{SLA} \cdot \vec{\tau} \cdot V &\geq \begin{bmatrix} r_1 & r_2 & \cdots & r_n \end{bmatrix}^T, \end{aligned} \tag{4.9}$$

where $\vec{\tau}$ is a $n \times n$ matrix given by

$$\begin{bmatrix} \tau_1 & 0 & & \cdots & & 0 \\ 0 & \tau_2 & 0 & \cdots & & 0 \\ 0 & 0 & \tau_3 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & 0 \\ 0 & & & \cdots & & \tau_n \end{bmatrix}.$$

These constraints mean the following: the total response time should be bounded by the SLAs, each tier has at least one active server, the sum of the active servers should be less than or equal to the total number of servers, and each tier should process the required number of requests within the target response time.

By Little's Law [78], the average number of requests in a system is equal to the product of throughput and the average response time of each request. Thus, the throughput of our system, $\vec{\tau} \cdot V$ multiplied by the target response time T_{SLA} must be larger than or equal to the expected number of requests to satisfy the performance bounds and save energy. For example, if we let r_i be 12 and τ_i 3, then we get $v_i = 4$. This means that 4 servers are sufficient to handle all 12 requests. However, if T_{SLA} is 2 seconds, we can reduce v_i to half (i.e. $v_i = 2$) and still manage to meet T_{SLA}. This is why we need to multiply T_{SLA} in the last constraint.

4.5 Methodology

In this section, we describe the methodologies used to solve the problem formulated in Section 4.4. Specifically, we describe how to estimate the model parameters and the heuristics used to solve the problem. The notations for the parameters used in our model along with the methods to obtain them are summarized in Table 4.1.

4.5.1 Estimation of model parameters

The parameters of interest include the peak power consumption, server utilization, number of requests in the next time frame and server throughput. Since the plate peak power consumption is far from the real peak power consumption [14], we use the real *peak power consumption* in our model. The CPU *utilization* of a server is obtained from monitoring tools supported by operating systems, for example, `sar` package in Linux. The *number of requests* received by a server in the next time frame is measured by observing the number of open sockets. A server opens a socket when a request arrives and keeps it open until the response reaches to the client as per the HTTP1.1 protocol specification. We determine the number of open sockets by analyzing operating system logs. We predict the number of requests in a tier for a given time frame as a weighted moving average of the

Table 4.1: Summary of model parameters

Parameters	Description	Methods to obtain
T_{SLA}	The maximal response time specified in SLA	Given
N	Total # of Servers	Given
ρ	CPU utilization	Actual Measurement
M	The # of requests	Actual Measurement
τ	Throughput of a server	Actual Measurement
T	Response time	Actual Measurement
$1/\mu$	Service Time of a request at current CPU speed	Estimated by ρ/τ
m	The Avg. # of instructions per requests)	Estimated in Section 4.5.1
t_0	Service time at energy optimal CPU speed	Calibrated in Section 4.5.1
$\vec{r}$	The # of requests for each tier in next time frame	Predicted by weighted moving average from actual measurements
$Power(s, \rho)$	Power consumption	Estimated by linear regression from actual measurements
τ_{max}	Throughput of a server to meet the response time	Estimated by MVA
$\bar{T}$	expected response time of incoming requests	Estimated by MVA
$\vec{v}$	# of servers for each tier in the next time frame	heuristic
s	CPU speed for processing incoming request	heuristic
t	The amount of time to sleep before processing incoming request	heuristic

number of requests during the previous time frames. It is to be noted that more accurate prediction techniques such as auto-regressive time series could also have been used. However, simple moving average serves as a good prediction technique in our experiments.

A server in a tier may not always operate at the energy optimal speed s_0. Therefore, we calibrate a server's service time to the service time at the speed s_0. To calibrate the *service time* of a request when a CPU is operated at the speed s_0, we first calculate the mean number of instructions requested for a request in a server at the current CPU speed s_c and then relate it to the target speed s_0. The mean number of instructions required to process a request, m_i is obtained as follows. For a time period of duration t, CPU utilization ρ, average CPU speed $\bar{s}$, number of requests r_i, and number of servers v_i in a tier i during the previous

time frame, m_i is given by

$$m_i = \frac{\bar{s}\rho t}{\frac{r_i}{|v_i|}} = \frac{\bar{s}\rho t|v_i|}{r_i}\,, \tag{4.10}$$

where ρ, $\bar{s}$, and r can be obtained from operating system supported tools. Now, let us derive a simple formula to relate m_i to the energy optimal speed s_0. When a CPU is operated at the speed s_0, the average service time t_0 of a request is given by

$$t_0 = \frac{m_i}{s_0}\,, \tag{4.11}$$

where m_i is the average number of CPU instructions for a request at tier i. Also, the total number of instructions for a request remains the same even though we vary CPU speeds. This calibration procedure aims at estimating the service time of a request in a server before changing the frequency in the dynamic provisioning algorithm.

In our model, we assume that each tier is modeled as a queue in a closed queuing network. We use Mean Value Analysis (MVA) algorithm [73] to determine the maximal throughput and marginal response time of each tier. MVA takes as input the number of clients, visit ratio of each tier, number of tiers, and user think time. The parameters obtained from MVA algorithm determine the elements of $\vec{\tau}$ in Equation 4.9. The elements of vector V are all integers, leading to the above problem formulation as an integer linear programming problem, which is known to be NP-complete. Hence, we propose heuristics to solve the above problem, though they may not always guarantee an optimal solution. In the next section, we describe our heuristics.

4.5.2 Heuristics

We propose two heuristics to solve our proposed optimization problem. The first heuristic involves a global optimization for finding the desired number of active servers in each tier for reducing energy consumption by using just sufficient number of servers. The second heuristic relates to a local optimization at individual server levels. It determines the energy optimal CPU speeds and sleep states.

The first heuristic algorithm (Algorithm 1) works as follows: We get the esti-

Table 4.2: Parameters used in Algorithm 1

Parameters	Descriptions
T_{SLA}	Target response time
M	# of requests from clients
t_{0_i}	Service time of tier i at CPU speed s_0
$\vec{r}$	# of incoming requests for each tier
z	Think time interval
t	Duration of time frame
$\vec{V}$	# of servers for each tier
L_i	# of queued requests for tier i
$\bar{T}$	Total average response time
$\bar{T}_i$	Estimated average response time of tier i
$\bar{\tau}_i$	Estimated throughput of tier i
N	Total # of servers

Algorithm 1 Heuristic for selecting the optimal number of servers for each tier of a multi-tier data center

Require: T_{SLA}, M, $\vec{r}$, s_0, $\bar{s}$, ρ, $\vec{V}_{cur}$, t, z, k, n
1: $m_i = \frac{\bar{s}\rho_i t|v_i|}{r_i}$
2: $t_{0_i} = \frac{m_i}{s_0}$
3: $(\bar{T}, \bar{T}_i, \bar{\tau}_i) = MVA(M, t_{0_i}, \vec{r}, z, T_{SLA}, n)$
4: $L_i = \tau_i \times T_i$
5: $v_i = \frac{r_i + L_i}{\bar{\tau}_i T_{SLA}}$
6: **if** $v_i > v_{i_{cur}}$ **then**
7: $\quad v_i = v_{i_{cur}} + 0.9|v_i - v_{i_{cur}}|$
8: **else if** $v_i < v_{i_{cur}}$ **then**
9: $\quad v_i = v_{i_{cur}} - 0.2|v_i - v_{i_{cur}}|$
10: **end if**
11: **if** $\sum_{i=1}^{n} v_i \geq N$ **then**
12: $\quad v_i = \frac{N v_i}{\sum_{i=1}^{n} v_i}$
13: **end if**

mated throughput, response time, and queue length for each tier from the MVA algorithm. As described in section 4.4, the number of servers v_i for tier i is obtained by

$$v_i = \frac{L_i + r_i}{T_{SLA} \times \bar{\tau}_i} \tag{4.12}$$

The performance of this algorithm might be sensitive to the time interval between provisioning. However, the algorithm itself does not need to specify the time interval.

The intuition behind this heuristic is that we can save energy with agreeable performance degradation by minimizing the number of servers. Since the throughput τ_i is the maximum possible throughput that can satisfy the target response

time, the number of servers v_i, given by our heuristic is the minimum number of servers that can meet the given SLA and achieve substantial power/energy savings. Note that our method does not search the entire solution space to find the appropriate number of servers. When the number of active servers required is greater than N, we allocate all of these N servers to process the given workload. We use the simple Round-Robin scheme to distribute workload across the servers. Further, to prevent frequent fluctuation of the number of servers in each tier, we gradually change the number of servers as described in lines 6 - 9 of Algorithm 1.

Next, Algorithm 2 describes the heuristic for dynamically determining the duration of sleep states and CPU speed to account for the dynamic nature of the Internet traffic. This problem is considered NP-hard [79]. Our approach may put a CPU into the sleep state while delaying the incoming and already queued requests, provided that we can process the delayed requests at an energy optimal CPU speed s_0 without violating the given time limit. However, the scheme presented in [15] makes a system sleep whenever it detects an idle state and then wakes up on the arrival of the next job. Our approach is motivated by the fact that a server sleep power consumption is around 10% of peak power consumption [15].

In order to find the duration for the sleep period, we first estimate the response time of an incoming request. Since an incoming request will be processed after all other queued requests are completed in a First-Come-First-Serve discipline and the service rate of a CPU is assumed to be the CPU speed, the response time of an incoming request is given by

$$\frac{(L_i + 1)m}{s},$$

where L_i is the number of requests in the queue, m is the average number of instructions required to process a request, and s is the CPU speed. To minimize energy consumption, we need to run the server at a speed $s = s_0$ as given in Corollary 2. Thus, the current incoming request will be completed at $(L_i+1)m/s_0$.

If we can finish processing the current incoming request within the marginal response time of a request in tier i, $(L_i + 1)m/s_0 < \bar{T}_i$, we can put the system into the sleep state while keeping the response time within T_{SLA}. When a request arrives, we inspect the expected response time of the incoming request as above.

If $\bar{T}_i - (L_i + 1)m/s_0 > 0$, we can transit the server into the sleep state for $\bar{T}_i - (L_i + 1)m/s_0$ duration of time. After that duration, the server wakes up and runs at the energy optimal speed. If the incoming request cannot be completed within $\bar{T}_i$, we run the CPU at least at the energy optimal speed without transitioning into the sleep state. Since the overhead of transition from/to the sleep state in terms of performance is negligible [15], the above approach does not hurt the response time, specified in SLA.

As shown in Corollary 2, we do not need to run the system under the energy optimal CPU speed s_0 even when we can process the incoming request with a slower CPU speed than s_0. However, when the given workload cannot be processed within $\bar{T}_i$ at the speed s_0, we need to run the CPU at a speed s, where $s_0 < s \leq s_{max}$. The speed s is the closest available CPU speed to $(L_i + 1)m/\bar{T}_i$. When $s > s_{max}$, we need more servers to process the current workload and the number of servers will be increased in the next time frame as explained in Algorithm 1.

Our heuristic for selecting the minimal number of servers picks the servers for each tier without searching the solution space exhaustively. A single tier data center environment does not take into consideration the chain reaction that may occur in a multi-tier environment when the number of servers in a tier is changed. Unlike a single tier case, in a multi-tier data center, one has to consider the behavior of other tiers while selecting servers for a tier. Also, both DVFS and DPM should be considered together based on the estimated results from an entire tier granularity and not from an individual server in a tier. This leads to the optimal allocation of servers across all the tiers as well as avoids the global performance degradation caused by local optimization technique such as DVFS and DPM done at the granularity of a server.

4.5.3 Intuitions behind possible energy savings

Next, we analyze the trade-off between performance and possible energy savings with our proposed heuristics. For the predicted number of requests for next time frame r, the number of servers in tier i v_i is equal to $(r_i + L)/\bar{\tau}_i T_{SLA}$ in our heuristics (in Algorithm 1). That is, we use enough number of servers to meet the target response time as explained in section 4.5.2. However, our heuristics for

Algorithm 2 CPU speed scheduling of a server in tier i

Require: T_{SLA}, $\bar{T}_i$, $\bar{\tau}_i$, N, r, s_0, ρ, t, $\bar{s}$
1: $m = \frac{\bar{s}\rho t}{r}$
2: $L_i = \bar{T}_i \times \bar{\tau}_i$
3: **if** $\frac{(L_i+1)m}{\bar{T}_i} < s_0$ **then**
4: sleep for $(\bar{T}_i - (L_i+1)m/s_0)$
5: run the CPU at s_0
6: **else**
7: **if** $(L_i+1)m/\bar{T}_i < s_{max}$ **then**
8: run the CPU at the speed closest to $\lceil (L_i+1)m/\bar{T}_i \rceil$
9: **else**
10: run the CPU at s_{max}
11: **end if**
12: **end if**

the CPU speed considers the energy optimal CPU speed rather than the maximal CPU speed as given in Corollary 2. The performance degradation of our proposed scheme is graceful since we can adjust the speed dynamically to avoid violating the SLA.

Let us now discuss the possible energy savings that can be obtained with our schemes. According to Equations 4.8 and 4.9 in section 4.4, the total energy consumption of a data center increases linearly with the increase in number of active servers. From experimental results, it was confirmed that the number of servers provisioned by our scheme is almost similar to that of dynamic provisioning with maximal CPU speed. Thus, when the traffic fluctuates, we can achieve energy savings by reducing the number of active servers under light traffic.

Towards this end, the individual servers also utilize DPM which further reduces their energy consumption. The system utilization of a server with our scheme is at least

$$\rho = \tau \times \text{Service Time} = \frac{L}{T} \times \frac{m}{s_0}, \tag{4.13}$$

where L is the average queue length, T is the target response time, m is the average number of instructions per request, and s_0 is the energy optimal CPU speed. Since, we put the system including CPU into the sleep state when

$$T_i s_0 > (L_i + 1)m.$$

(line 3 of Algorithm 2), we can yield the lowest utilization of servers with our

scheme as follows:

$$\begin{aligned} \rho &= \tau \times \text{Service Time} && \text{(Utilization Law)} \\ &= \frac{(L_i + 1)}{T_i} \times \frac{m}{s_0}. && \text{(Little's Law)} \end{aligned}$$

Thus, we replace the period of idle state as well as low utilization with the sleep state which consumes 10 % of peak power consumption as reported in [15].

Finally, since the power consumption is a cubic function of the CPU speed, if we operate a server with the energy optimal speed rather than maximal or minimal speed, we achieve more energy savings. As an example, suggested in [76], suppose the relation between the power consumption and the CPU speed is given by

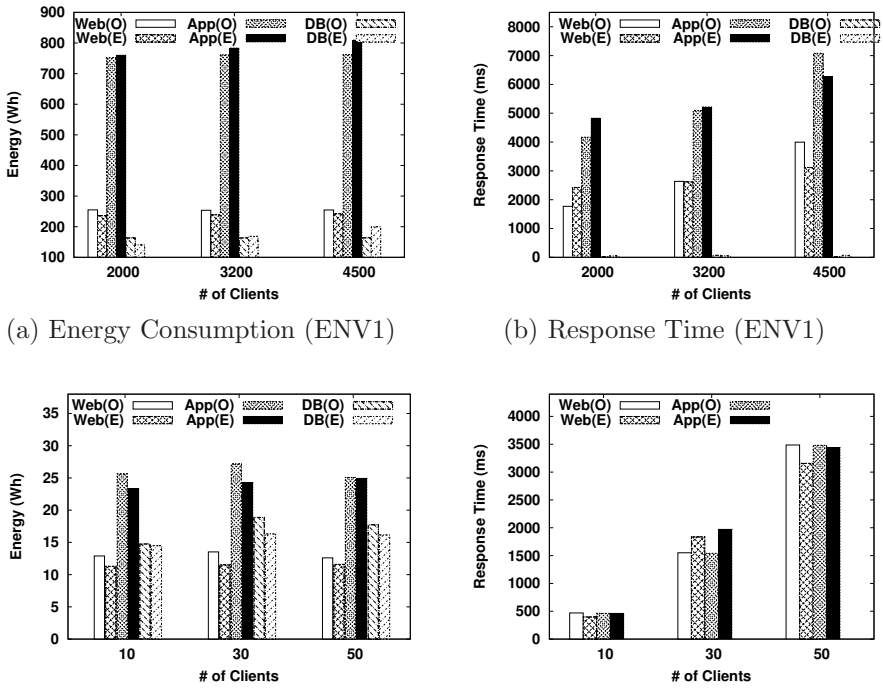
$$P(s) = \left(\frac{100}{39}\right)^3 (s - 1.2)^3 + 150\,, \quad 1.2 \leq s \leq 3.0\,.$$

The energy consumption of a server is minimized for $s = 2.8$ GHz according to Corollary 2. We assume that there is no job dependency with other machines. Using the above power-to-frequency equation, the power consumption for $s = 3.0GHz$ is 250W and for $s = 2.8GHz$ is 219W. Therefore, the ratio of energy consumption to process unit cycle for $s = 2.8GHz$ to $s = 3.0GHz$ is

$$219W/2.8GHz : 250W/3.0GHz = 0.938 : 1.0$$

Thus, by only using the energy optimal speed, we could reduce the power consumption of each machine by 6.2 %, compared to running them at maximal CPU speed.

To summarize, our proposed schemes can achieve substantial energy/power savings by reducing the number of servers required, using energy optimal speed instead of running servers at either the maximal CPU speed or any randomly picked CPU speed, and utilizing dynamic power management with the sleep state. We obtain potential energy savings, while satisfying SLAs.



(a) Energy Consumption (ENV1)

(b) Response Time (ENV1)

(c) Energy Consumption (ENV2)

(d) Response Time (ENV2)

Figure 4.3: Validation results of the model and simulator

4.6 Experimental results

4.6.1 Validation of the simulator

We validate the correctness of our simulator from real measurements on a prototype three-tier data center consisting of 25 servers using two multi-tier application benchmarks. For validation, we compared the response time and energy consumption obtained from simulation with those collected from real measurements. Baseline technique for comparison with our schemes uses static provisioning without either DVFS or DPM. For simulation, we used a modified version of a multi-tier data center simulator [7], written in CSIM [65]. The parameters used in the simulation like service time, peak power consumption, duration of time frame, etc are all derived from real measurements (Table 4.1).

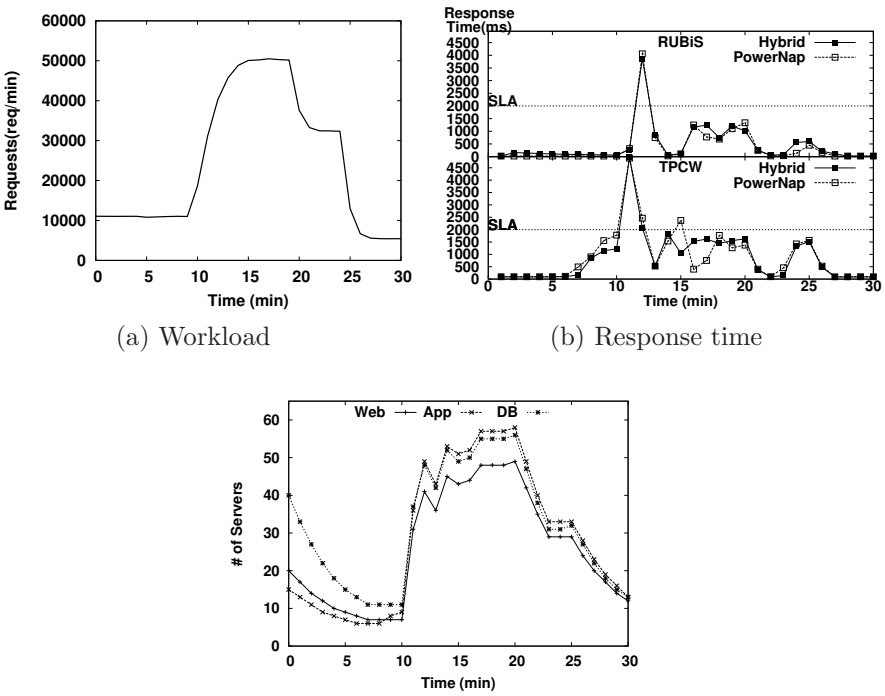
We perform experiments with the both unvirtualized and virtualized platforms presented in Chapter 2. The power consumption of machines was recorded by Yokogawa's WT210 power meter in the following experiments.

Figure 4.3 shows the validation results with respect to energy consumption and response time. Here, (O) represents the observed results from real measurements and (E) represents the estimated results from our simulator. (The energy numbers represent the total energy consumption across the cluster). From these figures, we observe that the graphs for the response time and energy consumption corresponding to real measurements and simulation results closely match with each other for different number of clients in both RUBiS and TPC-W benchmarks. We also notice that the height of the bar graph for DB tier in Figure 4.3b is negligible due to the small response time of DB tier. Moreover, since the version of MySQL that we used does not provide response time information, we were unable to validate response time of the DB tier in Figure 4.3d for TPC-W. (We, however, expect similar trend for the DB tier as was observed for the WS and AS tiers.) However, in ENV1, we were able to log DB tier response times by modifying the RUBiS benchmark.

Since response time and energy consumption are well captured by our simulator, according to Corollary 1, it implies that CPU utilization and, hence, throughput can also be correctly estimated using our simulator. We use our multi-tier data center simulation platform for detailed analysis of the proposed schemes in the following section. Due to the unavailability of sufficient number of servers in our experimental cluster, we obtained results from the simulator.

4.6.2 Evaluation results

In this section, we present the simulation results to compare our heuristics, Hybrid, with four other schemes, STATIC, DYN, DVFS and PowerNap. The baseline is the static provisioning at maximal CPU speed without invoking DVFS or DPM. For the static provisioning, we found the optimal number of servers for each tier using the MVA algorithm to guarantee an average response time of 2 seconds (which is the SLA response time limit in our experiments). The dynamic provisioning (DYN) decides the number of servers for each tier based on heuristic 1 and then the CPUs run at the maximal speed. DVFS is the scheme that conflates heuristic 1 and heuristic 2 without using DPM. PowerNap employs heuristic 1 with the DPM mechanism proposed in [15].



(a) Workload

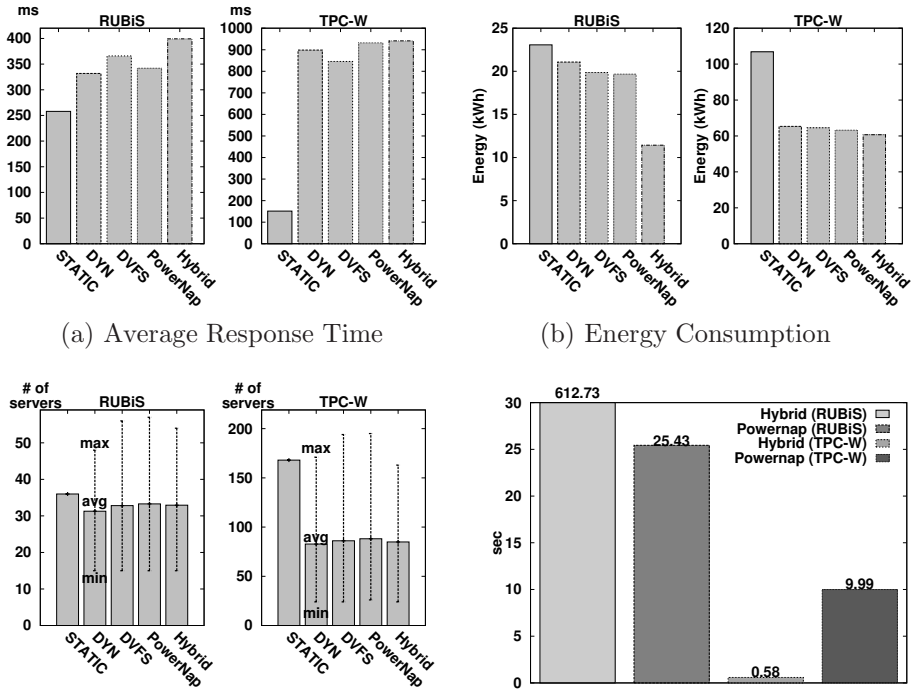
(b) Response time

(c) # of servers of each tier for TPC-W

Figure 4.4: The effect of dynamic provisioning in Algorithm 1

Experimental details We varied the CPU speed from 1.2 GHz to 3.0 GHz and estimated power consumption of a node using the equation in [76]. In our simulations, servers are modeled as M/M/1 queues for simplicity. From real measurements with the RUBiS and TPC-W benchmarks, we obtained various simulation parameters like service time, ratio of dynamic to static requests and number of database queries generated per dynamic request. We varied the number of requests for the two benchmarks dynamically over a 30 minutes duration and the request variation with time is shown in Figure 4.4a. In our experiments, the dynamic provisioning scheme decides the number of servers for each tier during the next time at the granularity of one minute interval.

Simulation results Figure 4.4b shows the variation of response time of Hybrid and PowerNap for the RUBiS and TPC-W workload for each time frame. The number of servers provisioned by our heuristics across all the tiers in a multi-tier data center is shown in Figure 4.4c. We observe that except the spike in these



(a) Average Response Time (b) Energy Consumption

(c) The number of powered-on servers (d) Total sleep duration per server

Figure 4.5: Evaluation of the proposed method

graphs during the 10-12 minute time interval (corresponds to flash crowd), Hybrid keep the response time below 2000ms and thus adheres to the response time bounds of SLA. Although the response time of our scheme hikes up to 5000 ms during the flash crowd, within the next couple of intervals, our scheme could allocate enough number of servers across the tiers to keep the response time within the SLA limit of 2000ms. The PowerNap scheme showed similar trend. The results demonstrated that the proposed scheme, Hybrid can adapt to both static and burst Internet traffic. Although not shown in these graphs, other schemes also exhibited similar behavior for both RUBiS and TPC-W.

Figure 4.5 provides comparative analyses of the average response time, energy consumption, the number of of powered-on servers, and the total sleep time per server. Figure 4.5a represents the average response time of all requests for the RUBiS and TPC-W workloads, respectively. As expected, STATIC has the minimum latency due to overprovisioning and Hybrid incurs the maximum latency since Hybrid can make a system sleep longer than PowerNap. For the TPC-W

workload, one dynamic request generates around 55 small and frequent database queries, leading to little idle periods to exploit DPM. Thus, the sleep state cannot be utilized as much as for the RUBiS workload, which makes the average response times of all dynamic provisioning schemes similar to each other. We note that the response time of DVFS is 7% higher than PowerNap for the RUBiS workload because the servers in DVFS process the requests at a lower speed than PowerNap. For the TPC-W workload, however, the response time of DVFS is lower than other dynamic schemes.

Figure 4.5b shows that Hybrid can reduce the total energy consumption by about 50% and 44% for RUBiS and TPC-W, respectively, when compared to STATIC. Hybrid could save 42% more energy compared to PowerNap for the RUBiS workload. The energy saving is attributed to the longer sleep period and energy optimal CPU speed in Hybrid than PowerNap, which is described in Section 4.5. We observe that the total sleep period of a host with Hybrid is 24 times longer than that of PowerNap for the RUBiS workload (Figure 4.5d). The average duration of one sleep period with Hybrid is 18.66 *ms*, compared to 0.48 *ms* with PowerNap. Hybrid resulted in 32836 transitions to the sleep state per server on average, while PowerNap had 52512 transitions on average. With PowerNap, the system wakes up whenever a request arrives, creating longer low utilized periods in the system than with Hybrid. Also, we took the advantage of DVFS by running the active servers at the energy optimal speed instead of at the maximal speed. Compared to static provisioning, the energy saved by PowerNap, DVFS, and DYN for the RUBiS workload is about 15%, 14%, and 10%, respectively.

Since the behavior of a system varies according to the workload, the results from the TPC-W workload are different from those of RUBiS. In Figure 4.5b, all dynamic provisioning schemes reduce energy consumption for the TPC-W workload by more than 49% relative to STATIC. Since the number of servers to process requests during the peak traffic period is significantly larger, compared to the case with low traffic, dynamic provisioning schemes can save significant energy. However, because of many small database queries in TPC-W workload, the idle period between requests is short. Thus, the opportunity for utilizing the sleep state is less for the TPC-W workload, although the overall energy consumption decreases compared to the STATIC provisioning.

The number of active servers for both the workloads is shown in Figure 4.5c. In static provisioning, a total of 36 and 168 servers were used during all time frames for the RUBiS and TPC-W workloads, respectively. For the RUBiS workload, the number of active servers in STATIC is close to the average number of active servers in all dynamic schemes. However, for the TPC-W workload, the number of servers with STATIC is close to the maximal number of active servers in dynamic schemes due to high variability in number of requests generated between tiers. In all the dynamic provisioning schemes, the number of active servers for all the tiers decreased under light traffic, which corresponds to the minimum number of active servers (a total of 17 servers across all the tiers in Figure 4.5c). However, we used a total of 36 servers in STATIC. 168 servers were provisioned with STATIC for the TPC-W workload, but only 25 servers were allocated with all dynamic provisioning approaches. This is because STATIC reserves adequate number of servers for handling peak traffic even under low load conditions. At the eleventh time frame (Figure 4.4a), the dynamic provisioning schemes increase the number of servers to maintain the response time of requests due to the sudden increase of the workload in the previous time frame. During heavy traffic periods, for the RUBiS workload, PowerNap and Hybrid provisioned 16% and 12% more servers than DYN, respectively. However, for the TPC-W workload, the number of active servers in PowerNap is 13% more than DYN, while that of Hybrid is 5% less than DYN. This implies that the server utilization in Hybrid is higher than that of PowerNap under heavy traffic. To summarize, our method creates more opportunities for energy savings, while maintaining the SLA since it incorporates dynamic provisioning with the energy management techniques for individual servers.

Some insights drawn from the experimental results are summarized below. First, our proposed scheme, Hybrid, may have less potential to save energy where the DPM technique cannot be leveraged much; for example, in the TPC-W workload, which generates many small requests between tiers. Second, experimenting with different response time thresholds, we observe that the behavior of all the considered dynamic schemes does not significantly change. With a stricter SLA constraint, the overall energy consumption increases by allocating more number of servers to keep up with the performance bound in dynamic schemes. Similarly, with a relaxed SLA constraint, the energy usage reduces since dynamic schemes

can achieve the desired performance with less number of servers. We experimented with 1000 and 3000 ms SLA thresholds, but the results are not shown due to space constraints.

4.7 Conclusions

This thesis proposed a novel multi-level approach for increasing the energy efficiency and maintaining the performance bounds of a multi-tier data center. I formulated the trade-off of performance and energy as an optimization problem and introduced two heuristics based on queuing theory to solve it. The first heuristic dynamically provisions sufficient number of servers across each tier to conserve energy, while satisfying the SLAs. The second heuristic schedules the above selected servers to run at energy optimal CPU speeds and utilizes the DPM mechanism involving sleep states to boost further energy savings. This work has constructed a prototype three-tier data center to validate our simulator and obtain parameters for solving the proposed model. Experimental results with the RUBiS and TPC-W benchmarks demonstrated that the proposed Hybrid technique can provide up to 50% more energy savings compared to the base scheme of static provisioning, where the servers run at the maximal speed without DPM. Extensive simulation results demonstrated that the combined Hybrid approach to energy saving is more effective than other compared dynamic techniques such as DYN, DVFS and PowerNap for workloads that provide the opportunity to exploit DVFS and DPM techniques at individual servers. For the RUBiS workload, our integrated approach provided additional 42% energy savings and turned on less number of servers under heavy traffic compared to the most recently proposed PowerNap technique. The proposed approach delivers higher energy efficiency by facilitating longer sleep durations and/or reducing the number of active servers.

As future work, I plan to extend the proposed scheme to environments with dynamic workloads consisting of multiple types of applications, heterogeneous servers and shared resources among servers.

Chapter 5

D-Factor: A Quantitative Model for Application Slow-down in Multi-resource Shared Systems

5.1 Introduction

Sharing a system with multiple workloads enhances system utilization, thereby lowering the economic [14] and environmental dent [72]. Recent advents in server virtualization bolster this sharing policy since it allows multiple operating system instances to share a physical machine [30]. However, sharing a system comes at a price – contention for system resources. Contention for system resources leads to high variance in performance, which would deter hosting latency sensitive applications that require tight performance guarantees [8]. The high variance in performance is often considered as unpredictable performance [8] due to the lack of the understanding of performance variation of a workload from contention for multiple resources [28, 32]. This in turn would make the usual, but costly over-provisioning design more attractive, thereby defeating the whole concept of resource consolidation and cloud computing. Therefore, performance quantification of jobs (or performance degradation) in a multi-resource shared platform is essential to both facilitate co-hosting of applications and meet the service requirements. However, performance prediction in such an environment is admittedly a challenging problem

and to our knowledge, no efficient technique is available today.

We may capture the performance variation of jobs in shared systems with slow-down due to resource contention. Let *dilation factor* denote the slow-down of a job due to resource contention. Prior efforts to estimate dilation factor falls into two groups – modeling based solutions [40, 80] or measurements based solutions [32, 81–83]. Modeling based solutions often use resource usage statistics of each job to construct the model. For example, queuing theory based approaches use system parameters like job arrival rates and service rates of resources [80] and control theory based approaches use the relationship between allocated amount of resources and workload behavior [40]. However, resource access behavior of a job depends on co-located jobs and specific hardwares, especially when both jobs and machines are heterogeneous [28,84,85]. For instance, co-located jobs can alter cache hit ratio of a job in multi-core environments [28] or disk access latencies in shared storage systems [84], which can change resource access statistics of a job. Therefore, in order to improve the accuracy of a model, prior modeling based approaches require detailed resource access behavior of each job.

In order to address such challenges, measurement based approaches have been proposed [32, 81–83]. Koh *et al.* predicted the performance degradation of co-located workloads using recorded similar workloads in terms of their resource usage statistics [82], similar to program similarity analysis [81]. Recently, Govindan *et al.* [32] and Mars *et al.* [83] independently proposed techniques that employ probe jobs to infer behavior of workloads when they are co-located, which does not require resource usage statistics. However, those approaches do not provide generic strategies to estimate dilation factors of co-located jobs. Therefore, we desire to develop a simple, but generic model to estimate dilation factors of co-located jobs, without depending on resource usage statistics.

In order to develop such a model, we focused on a simple method to obtain dilation factor, linear sum of original completion times of jobs, which has been used to obtain makespan of jobs in scheduling algorithms [37]. With linear sum, we may obtain dilation factor of each job from dividing the makespan by individual completion time of each job. The benefit of the linear sum model is that it only relies on the completion times of jobs. However, linear sum may not be able to provide desired accuracy due to non-linearity between individual completion times

and makespan in multi-resource shared environments [28, 29, 33]. For example, completing two co-hosted jobs – one 100 *sec* CPU-job with one 100 *sec* I/O-job – will take less than 200 *sec*, contradict to estimation from the linear sum of completion times of both jobs. The deficiency of linear sum model is the lack of ability to consider multiple resources at the same time. Thus, we propose a model that extends the linear sum model to consider multiple shared resources, which only relies on the completion times of jobs instead of resource usage statistics. With the proposed model, we overcome challenges in using prior analytical model based approaches to estimate dilation factors of co-located jobs. Also, we provide a mathematical foundation on using well-known workloads, probe jobs in [32, 83], to infer behavior of co-located workloads.

To provide accurate estimation of slow-down of jobs due to resource contention in shared environments like data centers/cloud computing platforms, we propose a generic quantitative model for dilation factors of n-jobs contending for m-resources, called the *dilation factor model* (*D-factor model*). We view *a job* as a sequence of accessing one of the system resources and a shared system as a collection of resources. For this, we characterize the resource access statistics of a job by a vector-valued loading statistics, *a loading vector*. We obtain the factor of dilated completion time, the *dilation factor*, of each job from a quadratic relation of loading vectors for jobs in the system, which can be utilized in optimization problems such as job scheduling. In addition, the proposed D-factor model can profile a job by only measuring completion time of a job, which does not require any instrumentations such as monitoring tools and modifying software/hardware infrastructures as have been used in prior studies [32]. This is an important characteristic for cloud environment, where application operators on virtual machines often cannot monitor physical resources.

We validated the proposed model with actual measurements on cluster infrastructures running native Linux and Xen Hypervisor [30], with synthetic workloads, FileBench [86], and MapReduce. The experimental results from synthetic workloads indicate that the average error rates of the estimations from the proposed model are 7% for a system on native Linux and 10.3% for a virtualized system. However, the estimation error from the linear completion time model could be up to 98% in the worst case. For the FileBench benchmark, the proposed model

estimates the completion time of workloads within a 6.0% error. We used three popular applications to benchmark MapReduce– `Sort`, `Grep`, and `PiEstimator`. In experiments with MapReduce, we increased the number of instances of each MapReduce application up to four and co-hosted three different MapReduce applications. The margins of errors are about 16% for identical MapReduce applications and 11% for heterogeneous MapReduce applications. We also demonstrate how the D-factor model can extend a scheduler for single-resource systems into a scheduler for multi-resource systems for reducing the makespan of workloads by about 20%.

This paper is organized as follows: Section 2 provides the motivation of this work. In Section 3, we present the underlying theoretical concepts for developing the D-factor model. Section 4 describes the experimental settings of this study. The validation results with both synthetic and realistic test samples are presented in Section 5, followed by an example application to a scheduling algorithm in Section 6. Related work is presented in Section 7, followed by concluding remarks in Section 8.

5.2 Motivation

Interference due to sharing resources results in slow-down of workloads [38], depending on the characteristics of co-located workloads [28, 32]. Prior findings have suggested non-linear dilation factors when we consider multiple system resources; co-located applications on multi-core processors [29, 33], co-located processes in an operating system [38], co-located virtual machines in a physical machine [22], and co-located workloads in a data center [14].

Figure 5.1 highlights the non-linearity in slow-downs of workloads due to multiple resource contention. Here, we created two different types of jobs – a CPU-bound job, which consists of arithmetic operations and an IO-bound job, which randomly reads two 2GB files. Both jobs take $100sec$ without the presence of other jobs. Experiments are done in a machine with two single-core CPUs that run Linux. However, we turned off one CPU to ensure that both jobs are accessing the same CPU. More details of the experimental platform are described in Section 5.4. We measured completion time of each job with another job in order to

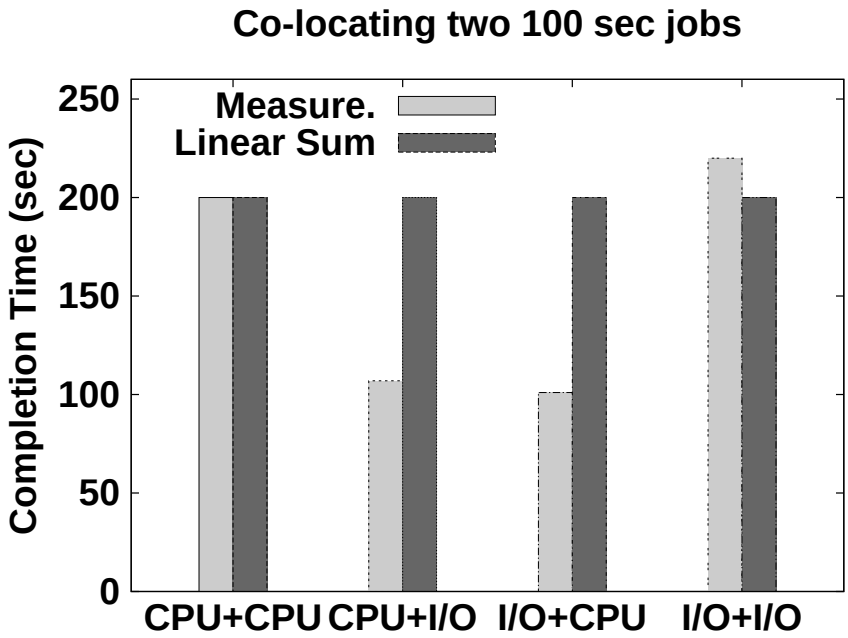


Figure 5.1: Non-linearity in slow-down of a job in a multi-resource system. CPU represents a CPU-bound job and IO represents an IO-bound job.

understand the interference between jobs. When the same type of jobs coexist in the system, we observe a linear relationship between total completion time and individual completion times. However, for different types of jobs in the system, a non-linear relationship is observed. This example suggests that we may reduce the performance interference among co-located jobs by considering multiple resources in a system. An empirical study on a large scale data center has shown that multiplexing workloads leads to higher efficiency since each workload utilizes different system resources [14].

A simple truth is that a better estimation results in better performance, which brings in the first fundamental question to be answered in this work:

Question 1. *What is the general quantitative model for performance of systems with multiple-resource contention?*

The previous simple example implies that, in a realistic environment, a machine or a system consists of multiple resources and hosts jobs that can request any of the resources, as shown in Figure 5.2. Recall, when we assume that a job accesses a single resource such as a processing unit, we may characterize a job by its completion time to estimate interference as shown in Figure 5.1. For multiple

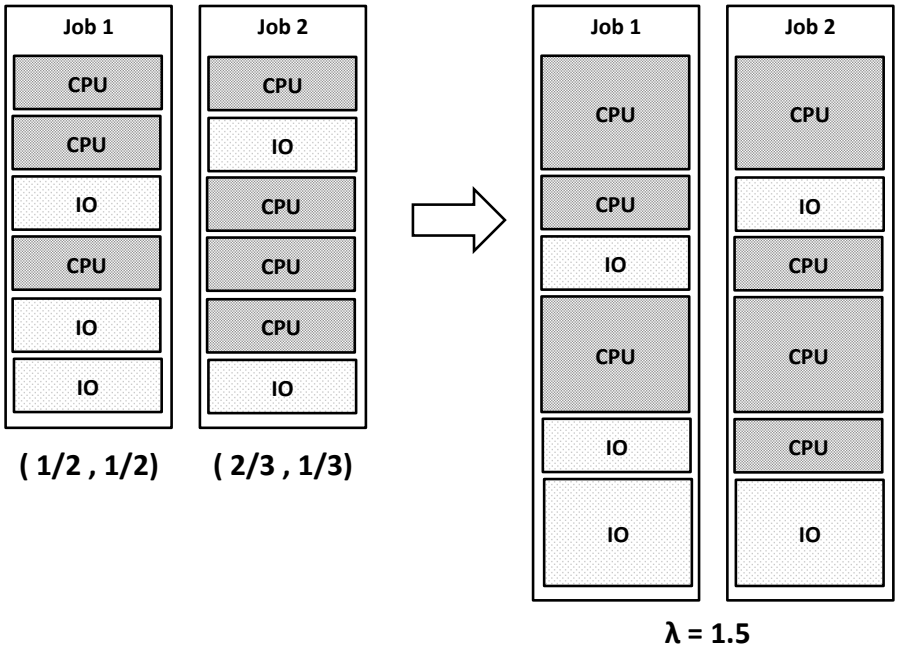


Figure 5.2: Dilation factor model describes slow-down of each job when multiple jobs are contending for multiple resources. (Left) Job-slices in their neutral states, where loading vectors are (1/2, 1/2) and (2/3, 1/3), respectively for CPU and I/O. (Right) The processing times of job slices will be dilated when they request the same resource at the same time. As described in Property 2, when two jobs are competing, their dilation factors, λ, are identical.

resources, on the contrary, a job cannot simply be measured by the time duration between the start and the end of the job; we need more information on the request distribution of the job for each of the resources. Thus, we need to find the answer of the following question:

Question 2. *How a job that requests multiple resources can be quantitatively modeled?*

In this work, we consider a job as a scheduling unit such as a process. We modeled a job as a sequence of hypothetical *job slices*, each of which is devoted to a single resource in order to approximate the behavior of real workloads. A job, *i.e.* the entire sequence of job slices, is statistically characterized by the probability of requesting/accessing each resource by a (random) single slice. The result is a vector-valued probability of resource-requests. Each job will be characterized by this *loading vector*. Note that the loading vector represents the portion of time to

access each resource during the execution of a job, which should be dependent on the hardware that a job runs.

As for approximating a real job with job slices and characterizing a real job with a loading vector, three major issues need to be addressed: simultaneous access of resources; dynamically varing resource access patterns; and multiple resources of the same type such as multi-core CPUs. Since loading vector represents the statistical characteristics to access each resource – average intensity to access each resource, simultaneous resource acceses can be approximated. Specifically, we show that our model reasonably captures a mixture of IO workloads and CPU workloads in Section 5.5. For dynamic workloads, we may approximate a dynamic workload by a series of static workloads, each of which can be represented by a loading vector. In this study, we will focus on static workloads since treating time-varying parameters requires additioal effort. As for n-core processors, we may reserve one element for each core in a loading vector. More details will be discussed in Section 5.3.

Finally, in order to formulate the *model* for question 1, the behavior of a machine with multiple resources should be concretely defined.

Question 3. *What is the quantitative model of a machine with multiple resources?*

We model a machine as a collection of *resource-queues* – one queue for each shared resource. A slice of a job on a machine is assumed to be queued in one of the resource-queues with the probability given by the loading vector.

Multiple jobs on a machine will populate the resource-queues, hence, a job-slice can be slowed down due to the waiting time for a resource-queue. A quadratic function of loading vectors will be established to determine the dilation of jobs. Eventually, the resource contention by a job set on a machine will be described in terms of *dilation factors* – referred to as slow-down of job completion time under the presence of other jobs.

5.3 Mathematical modeling

As illustrated by the simple example in Section 5.2, if jobs on a single machine compete over multiple shared resources, the dilation of their completion times can

exhibit complicated behavior. Recall, for a single-resource machine, time-sharing jobs slow down uniformly and proportionally to the number of competing jobs, which is not true any more if they compete over multiple jobs; jobs can even have different dilation factors. To clearly describe this phenomenon, we need the modeling of machines with multiple resources.

5.3.1 Modeling: machines and jobs

A machine with multiple resources can be viewed as a collection of multiple resource-queues; each resource corresponds to a queue and requests from jobs running on the machine compete for the resources. Let a machine have m resources, i.e. m-queues. We assume a resource-request from a job can be issued only after the previous request from the same job is completed. In order to describe this behavior, we introduce the concept of *job-slices* – a hypothetical atomic unit of work which is characterized by the following assumptions; (i) a job-slice requests for and can be processed by only a single resource, thus, it is also an atomic unit of request for the resource queues. (ii) It takes 1 (hypothetical) time-unit for a job-slice to be processed by the resource in which it is queued, independent of the type of resource.

A job is considered as a sequence of job-slices each of which requests a single resource. If a job consists of ℓ job slices, its *neutral* (which means undisturbed by other jobs) completion time is ℓ time-units. A request is viewed as the submission of a slice into one of the resource-queues. The job can proceed to the next slice only if the current slice in a queue is completed. For n concurrent jobs on the machine, we make the following two assumptions: (iii) there are at most n slices (including the one in service) in any one of the m queues. (iv) At any time, there are total n slices in all the queues. Assumption (iii) implies that the time for a slice to be completed is at most n time-units.

***Remark* 1.** *Notice that the actual amount of work during a unit time such as the amount (in MB) of file processed during the time depends on the system configuration; for example, if the operating system utilizes I/O buffer cache, some of the I/O requests turn effectively into memory requests. Thus, it might not be feasible to generally estimate the actual amount of work for multiple-resource workloads*

by investigating the program source and hardware specification. Characteristics of workloads will be obtained by probe processes or by regression analysis of their completion times, which will be illustrated in this paper.

Based on assumption (ii), we are actually proposing a unified measure of work done by different resources; Instead of counting the amount of work done by resources by their native measure such as the number of instructions (for CPU) and the file size in MB (for I/O), we represent the amount of work by the the time spent by any resource for the task. Thus, one unit of work is the amount of work that can be done by any resource in unit time. This allows to compare the workloads of different kinds using a common unit.

Recall the only required information for the linear sum model for single-resource machines is the neutral completion time of the job. For a multiple-resource model, additional information is required: the request–resource-queue correspondence. The amount of information increases with the size of the system, the number of resource types to be considered, and the number of workloads if we want to obtain the complete description as with queuing model approaches. Hence, we take a statistical approach.

5.3.2 The loading vector

In this paper, we characterize a job by the statistical pattern of its resource requests. In general, a deterministic resource-access pattern in time is difficult to obtain except for very specific applications of known structures. Instead, we assume a job has, for each resource, a unique probability of a slice to be queued for the resource; that is, we view that the number of job-slices queued for each resource divided by the number of total slices of the job is a statistical invariant of a job. Thus, in an m-resource model, the workload of a job is characterized by an m-dimensional vector, which we call the *loading vector* $\boldsymbol{p}$ of the job.

***Remark* 2** (Multicore systems)**.** *In order to avoid possible confusion, we present the theory and experimental results for single-core systems since there exists a variety of architectures for multi-core processors. However, the application of the presented model to multicore systems would be possible. For example, as many operating systems treat, a dual-core system can be considered as a machine with*

2 CPU-resources independent of each other. Thus, if we consider CPU and I/O resources in the model, the resulting loading vector can be 3 dimensional – namely, CPU1, CPU2, and I/O. Suppose an application is characterized by a workload given by a loading vector $\boldsymbol{p} = (CPU, I/O) = (0.2, 0.8)$. *Depending on the scheduling policy and cache architecture, the loading vector can be (0,0.2,0.8), (0.1,0.1,0.8), etc.*

Thus, without any other competing job (in our terms, in the neutral condition), p_i are given by

$$p_i = \frac{\text{the time spent at the resource-}i}{\text{the total completion time}} \geq 0. \tag{5.1}$$

Also, p_i can be interpreted as the probability of a job-slice being queued for queue-i. Obviously,

$$\|\boldsymbol{p}\|_1 = \sum_{i=1}^{m} p_i \leq 1. \tag{5.2}$$

Notice the inequality; we allow a job to have slices that do not request any resource. Such an *idling* slice simply spends a time-unit without populating any of the m queues. We define two classes of jobs: a job is

1. *idle* if $\sum_{i=1}^{m} p_i < 1$ and
2. *busy* if $\sum_{i=1}^{m} p_i = 1$.

Given a set of n jobs, we denote $\boldsymbol{p}_j$ as the loading vector of job-j and p_{ij} as the ith component of $\boldsymbol{p}_j$. An m by n matrix can be formed with p_{ij} as its elements. We call this matrix the *loading matrix* of the job set. Each column vector of the loading matrix is the loading vector of the corresponding job. The loading vector and the loading matrix are our statistical characterizations of the workload of a job and a job set, respectively.

5.3.3 The dilation factor

Consider n jobs on an m-resource machine characterized by a given m by n loading matrix. Suppose a slice of job-j is queued at queue-i. Then, the expected waiting time of the job-slice (of job-j) is the expected number of job-slices (from other

jobs) in the queue. Hence, the conditional expectation of the dilated completion time of the single slice of job-j, on the condition that it's in queue-i, is given by

$$1 + \sum_{\substack{k=1 \\ k \neq j}}^{n} p_{ik} = 1 + \sum_{k=1}^{n} p_{ik} - p_{ij} \,, \tag{5.3}$$

where the additional 1 time-unit is the service time.

Since p_{ij} is the probability of queuing a slice of job-j at queue-i, the expectation of the dilated completion time T_j of job-j is given by

$$\begin{aligned} T_j &= \tau_j \left(\left(1 - \sum_{i=1}^{m} p_{ij} \right) + \sum_{i=1}^{m} p_{ij} \left(1 + \sum_{k=1}^{n} p_{ik} - p_{ij} \right) \right) \\ &= \tau_j \left(1 + \sum_{i=1}^{m} p_{ij} \sum_{k=1}^{n} p_{ik} - \sum_{i=1}^{m} p_{ij}^2 \right) \end{aligned} \tag{5.4}$$

where τ_j is the neutral completion time of job-j. Notice, the term $(1 - \sum_{i=1}^{m} p_{ij})$ represents the probability of idling that causes no dilation. Let us define the total loading vector $\overline{\boldsymbol{p}}$ by $\overline{\boldsymbol{p}} = \sum_{j=1}^{n} \boldsymbol{p}_j$. The above relation can be rewritten in vector notation by $T_j = \tau_j \, (1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j)$. Thus, a job j is slowed down by the factor of T_j/τ_j due to resource contention. We summarize the result by introducing the dilation factor $\lambda_j = T_j/\tau_j$ as follows:

Definition 1. *Given a job set on a machine characterized by the loading vectors $\boldsymbol{p}_j$ $(j = 1, \ldots, n)$, the dilation factors $\lambda_j = T_j/\tau_j$ $(j = 1, \ldots, n)$ are given by*

$$\lambda_j = 1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j \,. \tag{5.5}$$

***Remark* 3.** *The above formula distinguishes our model from other applications of queuing theory. Combined with the representation of loading statistics by the loading matrix presented above, we can describe the average slow-down of a job in a closed formula. Also, the formula can be viewed as a statistical description of the interaction between jobs running on the same machine in terms of mutual interference.*

For identical jobs, the formula can be simplified as follows:

Property 1. *If all of n jobs are identical ($\boldsymbol{p}_j = \boldsymbol{p}$), the dilation factors are identical ($\lambda_j = \lambda$) and are given by $\lambda = 1 + (n-1)\,\boldsymbol{p} \cdot \boldsymbol{p}$.*

The above property can be utilized to obtain experimentally the loading vector of a job; (1) we obtain the loading factor by measuring the dilated time of n identical jobs for $n = 1$ to a sufficient number. (2) The data for various n can be analyzed by regression to obtain $\boldsymbol{p}$.

The dilation factors are also identical in another situation. By applying $n = 2$ and $\overline{\boldsymbol{p}} = \boldsymbol{p}_1 + \boldsymbol{p}_2$ to Equation (5.5),

Property 2. *If there are 2 jobs, the dilation factors are identical ($\lambda_1 = \lambda_2 = \lambda$) and given by $\lambda = 1 + \boldsymbol{p}_1 \cdot \boldsymbol{p}_2$.*

If we create a synthetic process which requests only a single resource, this reference (or *probe*) process can be utilized to compute the loading vector of a job. The primary benefit of the probe processes is that we can identify the loading vector of a job when we cannot control the execution of a job in the system.

Property 3. *Suppose there are two jobs – one that we want to identify its loading vector p and the other that uses only resource-i, a probe process. By Property 2, they share the same λ and the ith component of p is given by $p_i = \lambda - 1$.*

5.3.4 Special case: 1-resource-busy jobs (the linear sum model)

In this section, we illustrate that the linear completion time model is actually a special case of our general model, where all the jobs compete for the same resource.

Lemma 1. *Assume 1-resource-busy jobs: $p_{kj} = 0$ for any $j = 1, \ldots, n$ and for every $k = 1, \ldots, m$ except an index $1 \le i \le m$, and $\|\boldsymbol{p}_j\|_1 = 1$. Then, the dilation factors are identical ($\lambda_j = \lambda$) for all jobs and given by*

$$\lambda = n \tag{5.6}$$

Proof. Since all the jobs are busy, $p_{ij} = 1$ and $p_{kj} = 0$ for any $k \neq i$. Hence, all jobs are identically given by $\boldsymbol{p}_j = \boldsymbol{p}$. Then, $\boldsymbol{p} \cdot \boldsymbol{p} = 1$ and, by Property 1 in the previous

section, the dilation factors are identical and given by $\lambda = 1 + (n-1)\,\boldsymbol{p}\cdot\boldsymbol{p} = 1 + (n-1) = n$ □

Let τ_j denote the neutral completion time of job-j in a job set of size n. We define the total completion time T of the job set as the time duration from the starting time of the first initiated job to the ending time of the last completed job. For 1-resource-busy jobs, $T = \sum_{j=1}^{n} \tau_j$.

Theorem 1. *Suppose during the total completion time, there is no idling gap, i.e. the machine is always populated by at least one job. Then, the total completion time is the sum of individual neutral completion times independent of the starting and ending times of the jobs, that is,*

$$T = \sum_{j=1}^{n} \tau_j \,. \tag{5.7}$$

Proof. For any overlapping of jobs, there is a corresponding refinement $[0, t_1, \ldots, t_l]$ such that $t_l = T$ and, in each subinterval $[t_{k-1}, t_k]$, the number of running jobs denoted by n_k is constant. Let $\Delta_k = t_k - t_{k-1}$. Let J_k denote the set of indices of active jobs in the k'th subinterval and by I_j the set of indices of subintervals, where job-j is active. Let τ_j^k is the completed amount of job-j in subinterval-k, that is, in the subinterval, $100\,\tau_j^k/\tau_j\%$ of job-j is completed. (1) Since the dilation in each subinterval is identical for all active jobs, $\Delta_k = n_k\,\tau_j^k$ for all $j \in J_k$ for any k. Adding this for all $j \in J_k$, we obtain $n_k\,\Delta_k = n_k \sum_{j\in J_k} \tau_j^k$, hence, $\Delta_k = \sum_{j\in J_k} \tau_j^k$. (2) Then, $T = \sum_{k=1}^{l} \Delta_k = \sum_{k=1}^{l} \sum_{j\in J_k} \tau_j^k$. (3) By rearranging the indices, $T = \sum_{j=1}^{n} \sum_{k\in I_j} \tau_j^k$. (4) Since all jobs should be completed, $\sum_{k\in I_j} \tau_j^k = \tau_j$, which completes the proof. □

Note that Lemma 1 and Theorem 1 assume that jobs are fully overlapped for estimating completion times.

***Remark* 4.** *In general cases, such a simple formula for total completion time does not exist. Consider a 2-job system with* $\boldsymbol{p}_1 = (1, 0)$ *and* $\boldsymbol{p}_2 = (0, 1)$*. Then,* $\lambda_1 = \lambda_2 = 1$*, that is, there is no dilation of completion time. Let* $\tau_1 = \tau_2 = 1$*minute. If the two jobs started at the same time, they will be completed at the same time after 1 minute. If one of them started first and the other job started at the time of*

completion of the first job, the total completion time will be 2 minutes. Depending on the overlap, the total completion time can vary from 1 minute to 2 minutes. But, still the linear model estimate becomes the upper bound of the total completion time in any case.

***Remark* 5.** *(1-resource-idling jobs) Even if there's only one requested resource, the linear model cannot be applied to a job set with an idling job. Consider a simple job set of n identical jobs requesting only resource-i. Then, the loading vectors can be represented by a single scalar parameter $p < 1$. Then, $\lambda = 1 + (n-1)\,p^2 = (1-p^2)\,1 + p^2\,n$, that is, λ is a linear interpolation of 1 and n with respect to p^2, since $p < 1$ and $\lambda < n$.*

5.3.5 Special case: 2-resource-busy jobs

The usefulness of this model comes from the fact that each loading vector $\boldsymbol{p}_j$ can be represented by a single scalar parameter p_j. Without loss of generality, we can remove non-requested resources from considerations and assume $\boldsymbol{p}_j = (p_j, 1-p_j)$. Then, $\overline{\boldsymbol{p}} = (\overline{p}, n-\overline{p})$ where $\overline{p} = \sum_{j=1}^{n} p_j$, and

$$\begin{aligned}\lambda_j &= 1 + p_j\overline{p} + (1-p_j)(n-\overline{p}) - p_j^2 - (1-p_j)^2 \\ &= 1 + n - \overline{p} - n\,p_j + 2\,p_j\,\overline{p} - p_j^2 - (1-p_j)^2\,. \end{aligned} \tag{5.8}$$

The result can be further simplified if the jobs are identical, i.e. $p_j = p$. Then, $\overline{p} = n\,p$ and, for any $j = 1, \ldots, n$,

$$\lambda_j = 1 + n - 2\,n\,p + 2\,n\,p^2 - p^2 - (1-p)^2 \tag{5.9}$$

$$= 1 + n(1 - 2\,p + p^2) + (n-1)\,p^2 - (1-p)^2 \tag{5.10}$$

$$= 1 + (n-1)\,\left(p^2 + (1-p)^2\right)\,. \tag{5.11}$$

To emphasize the convenience of the formula, we summarize the result as a theorem.

Theorem 2. *Assume 2-resource-busy identical jobs with the loading vector given*

by $(p, 1-p)$*. Then, the dilation factors are identically given by*

$$\lambda = 1 + (n-1)\ \left(p^2 + (1-p)^2\right) \tag{5.12}$$

In other words, if the dilation factor is known, we can obtain the loading vector by the formula:

$$p = \frac{1}{2}\left(1 \pm \sqrt{1 - 2\frac{n-\lambda}{n-1}}\right) \tag{5.13}$$

Proof. The derivation is given above. □

Notice that there are two possible solutions for p and the model does not distinguish them.

The above relation suggests a simple experimental strategy to obtain the loading vector using Algorithm 3. Notice that we can characterize jobs only from their completion times, which does not require resource monitoring or kernel instrumentation. However, by utilizing system resource monitor, we can enhance the accuracy of the model.

Algorithm 3 Constructing the loading vector of a job in 2-resource model.

1: Measure τ by running job j alone.
2: Measure T, the dilated completion time of job j when n instances of job j are running concurrently in the system.
3: Evaluate the dilation factor $\lambda_j = T/\tau$.
4: From Equation 5.13, obtain $p_i = \frac{1}{2}\left(1 \pm \sqrt{1 - 2\frac{n-\lambda}{n-1}}\right)$
5: Obtain loading vector $\boldsymbol{p}_j = (p_i, 1-p_i)$

5.3.6 The total dilation factor

One of the potential benefits of D-factor model is the capability to extend existing schedulers for single-resource machines into schedulers for multi-resource machines.

We define the total dilation factor $\overline{\lambda}$ by the sum of λ_j for all jobs. Then, by the formula given in Definition 1,

$$\overline{\lambda} = \sum_{j=1}^{n} \lambda_j = n + \|\overline{\boldsymbol{p}}\|_2^2 - \sum_{j=1}^{n} \|\boldsymbol{p}_j\|_2^2 \,. \tag{5.14}$$

Notice, this simple formula involves only the L^2 norms of the loading vectors; informally the absolute values of the loading vectors.

In order to present online version of the formula, assume a set of jobs are already running on a machine with known characteristic parameters such as $\boldsymbol{p}_j$, $\overline{\boldsymbol{p}}$, λ_j, and $\overline{\lambda}$. Consider the following two cases:

1. A new job with the loading vector $\boldsymbol{p}'$ is added to the job set:

$$\begin{aligned}\overline{\lambda}_+ &= (n+1) + \|\overline{\boldsymbol{p}} + \boldsymbol{p}'\|_2^2 - \sum_{j=1}^{n} \|\boldsymbol{p}_j\|_2^2 - \|\boldsymbol{p}'\|_2^2 \\ &= \overline{\lambda} + 1 + 2\,\boldsymbol{p}' \cdot \overline{\boldsymbol{p}} \end{aligned} \tag{5.15}$$

2. From the job set, a job-k is removed: identifying $\boldsymbol{p}' = \boldsymbol{p}_k$,

$$\begin{aligned}\overline{\lambda}_- &= (n-1) + \|\overline{\boldsymbol{p}} - \boldsymbol{p}'\|_2^2 - \sum_{j=1}^{n} \|\boldsymbol{p}_j\|_2^2 + \|\boldsymbol{p}'\|_2^2 \\ &= \overline{\lambda} - 1 - 2\,\boldsymbol{p}' \cdot \overline{\boldsymbol{p}} + 2\,\|\boldsymbol{p}'\|_2^2 \end{aligned} \tag{5.16}$$

Thus, the increment (decrement) is given by

$$\delta\overline{\lambda}_\pm = \pm\,(1 + 2\,\boldsymbol{p}' \cdot \overline{\boldsymbol{p}}) + \begin{cases} 0 & \text{for } + \\ 2\,\|\boldsymbol{p}'\|_2^2 & \text{for } - \end{cases} \tag{5.17}$$

Notice, the decremental formula (with $-$ subscript) is used only to reset the current state when a job is completed, hence, removed from the job set. The incremental formula (with $+$ subscript) is more crucial in job scheduling algorithms. Recall remark 4; a simple formula for the total completion time as in the linear model does not exist anymore for the general multiple-resource model. Even though it is technically possible to compute the total completion time from the thorough job overlapping information, the total completion time might not be that useful/convenient criteria for job scheduling.

In this paper, we suggest a new objective function for the optimization problem. Notice that the total dilation can be viewed as a measure of performance degradation. Hence, the minimization of the performance degradation, i.e. the

Table 5.1: Specifications for experimental environment

CPU	Two single-core 64-bit AMD 2.4Ghz
RAM	4GB
Shared Storage	NFS (disk images for Xen)
Local Storage	Ultra320 SCSI
Network	1Gbps Ethernet, 10Gbps Infiniband
vCPU (Dom0)	Runs on both CPUs
vCPU (VMs)	Runs on one CPU
RAM/VM	256MB
I/O (VM)	Tap:aio (bypasses buffer cache of Dom-0)
Kernel	Linux 2.6.18
Hypervisor	Xen 3.4.2

increase in the total dilation factor can be considered as an alternative way to utilize available resources optimally. If we accept this new criteria, the optimization problem can be stated as follows:

Definition 2. Given a new job with the loading vector $\boldsymbol{p}'$, find the assignment to a machine μ which minimizes

$$\boldsymbol{p}' \cdot \overline{\boldsymbol{p}}_{\mu} \tag{5.18}$$

where $\overline{\boldsymbol{p}}_{\mu}$ is the current total loading vector of machine-μ.

5.4 Experimental Environment

This section describes the experimental settings designed to validate the proposed D-factor model.

5.4.1 Target system overview

We experimented with clusters in two environments – native Linux and Xen-based virtualized environments [30]. Table 5.1 shows the detailed specifications of experimental settings for the native Linux and virtualized environment. In the native Linux environment, each node is running only one Linux operating system (OS). On the contrary, in the VM environment, each node can host multiple OS instances. The overview of the VM environment is shown in Figure 5.3. Applications running on a virtual machine have an illusion that they exclusively access the virtualized resources. A special guest machine, Dom-0, can directly access physical resources, especially I/O devices. The Virtual Machine Monitor (VMM) manages the shared

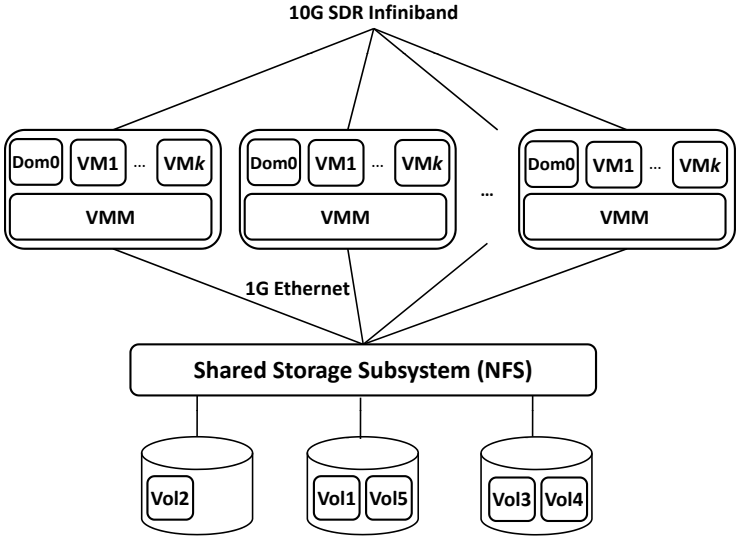


Figure 5.3: Virtualized experimental environment

resources such as processors, memory, I/O subsystems and network devices. All the access requests to the hardware resources from applications running on a virtual machine flow into the VMM and then Dom-0, if necessary. Each node in the native Linux environment accesses local SCSI disks whereas the virtual hard disks of guest virtual machines are located in a Network File System (NFS) partition. Thus, I/O requests from guest machines may invoke network traffic. We employed varying number of applications on each node in the native Linux environment. In the VM environment, we run one application per virtual machine, but we varied the number of virtual machines.

5.4.2 Description of workloads

We used a mixture of synthetic and realistic workloads to validate the proposed D-factor model on a variety of workload environments. For synthetic workloads, we employed *standard jobs* (the details of which will be described later) and file compressions that use both CPU and I/O resources. For realistic workloads, we used I/O-bound workloads (FileBench [86]) and MapReduce workloads. Table 5.2 shows resource usage profile of workloads in this study.

Table 5.2: Resource Usage Profile of Workloads

Type	Name	CPU	I/O
Synthetic	`STD-CPU`	High	Low
Synthetic	`STD-IO`	Low	High
Synthetic	`FileComp`	High	Med
FileBench	`FileServer`	Low	High
FileBench	`VarMail`	High	Med
MapReduce	`PiEstimator`	High	Low
MapReduce	`Sort`	Med	Med
MapReduce	`Grep`	Med	Med

Synthetic workloads – standard jobs and file compression We have created standard jobs for two purposes. The first is to demonstrate that the proposed model can capture completion times of synthetic workloads using only one system resource. The second is to illustrate a method to profile workloads. *Standard job* is a workload that uses only one resource without any idle period. The completion time of a standard job should be consistent, provided that the standard job is running alone in the system. For example, the completion time of a standard job for disk I/O should not depend on the current size of free memory or the current status of buffer cache. In this study, we created two standard jobs – a standard job for CPU (`STD-CPU`) and a standard job for disk I/O (`STD-IO`). `STD-CPU` is a workload that repeats arithmetic calculations. `STD-IO` is a workload that opens two 4GB files on the local disk in the synchronous mode and, then, reads 1MB from random positions of both files, while controlling the buffer cache using the POSIX library, specifically `posix_fadvise`.

We have created a workload, `FileComp` to show that we can profile a CPU-I/O workload with standard jobs. `FileComp` compresses 2,560 files, each of 256KB size. The neutral completion time of `FileComp` τ is $78.08sec$. With `STD-CPU` of the duration of $200sec$, the completion time of `FileComp` T is $123.67sec$. If we assume that a system consists of two resources – CPU and I/O, we may consider `FileComp` as a two-resource-busy job. Since `STD-CPU` is a CPU-only job, the loading vector is $(CPU, IO) = (1, 0)$. Let the loading vector of `FileComp` be $\boldsymbol{p} = (p, 1-p)$. According to Property 2 in Section 3, the dilation factor $\lambda = 1 + p$. λ can be calculated by $T/\tau = 123.67/78.08 = 1.58$, which yields $p = 0.58$. Hence, the loading vector of `FileComp`, $\boldsymbol{p} = (0.58, 0.42)$. We used `FileComp` for evaluating our model in the native Linux environment. In this example, we profiled a job using

standard jobs. However, it is non-trivial to implement these standard jobs due to their dependence on the system configuration. Thus, with FileBench workloads, we demonstrate an alternative way to profile a job – Algorithm 3.

Realistic workloads – FileBench and MapReduce We experimented with FileBench in a virtualized environment. We employed two predefined workloads in FileBench, `fileserver` and `varmail`. FileBench is an application benchmark that mimics the typical behavior of workloads. We employed one FileBench workload per VM and varied the number of VMs. According to Algorithm 3, both workloads are profiled by $\boldsymbol{p}_{file} = (0.02, 0.98)$ and $\boldsymbol{p}_{mail} = (0.10, 0.90)$.

We experimented with three MapReduce workloads : `Sort`, `Grep`, and `PiEstimator` in the native Linux environment. MapReduce job consists of multiple tasks that are hosted in parallel on different cluster nodes. We run these MapReduce workloads on a dedicated 17-node native Linux cluster with Hadoop 0.20.1 [87]. Each workload generates two map-and-reduce tasks on each node. Each task accesses the local disk and exhibits insignificant communication between each other. The completion time of an application is determined by the completion time of the slowest task. For these experiments, we confirmed that the variance of completion times between tasks for one application is insignificant. Since our experimental platform consists of two single-core CPUs, increasing the number of instances of an application creates resource contention for each CPU. We conducted two experiments; (i)increasing the number of instances of each application from 1 to 4 and (ii) co-locating three different MapReduce workloads in the system. Applications are initiated at the same times and we used the Hadoop fair scheduler [87] to allocate system resources to these applications.

In order to estimate the completion times with the D-factor model, we profiled each workload with `STD-CPU`, similar to the `FileComp` workload. The loading vector, $\boldsymbol{p} = (CPU, \textit{other resources})$, and the neutral completion time, τ, of each application are $\boldsymbol{p}_{Sort} = (0.9, 0.1)$, $\tau_{Sort} = 56.0sec$, $\boldsymbol{p}_{Grep} = (0.5, 0.5)$, $\tau_{Grep} = 95.0sec$, $\boldsymbol{p}_{Pi} = (0.5, 0.5)$, $\tau_{Pi} = 90.0sec$. Then, the loading matrix is given by

$$P = \begin{bmatrix} 0.9 & 0.5 & 0.5 \\ 0.1 & 0.5 & 0.5 \end{bmatrix}.$$

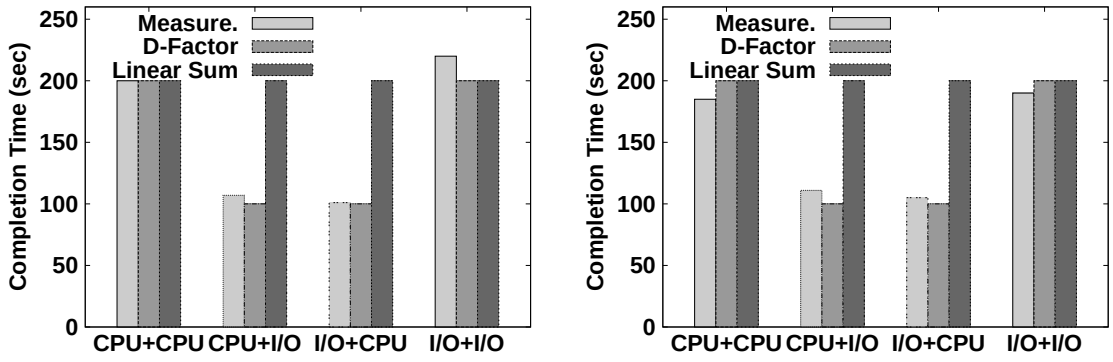
Note that the loading vector represents the probability of accessing each resource instead of the utilization. Thus, resource usage does not necessarily match with the loading vector. For example, regardless of high CPU usage, some other factors may dominate the execution time of `PiEstimator`, which is captured in the loading vector representation.

Methodologies We compare the estimated total completion time from the D-factor model and the linear sum model with actual measurements. We compare D-factor with linear sum since both methods do not require information of request arrival rate and service rate as in queuing analysis. For two jobs i and j when $\tau_i > \tau_j$, we can calculate the actual completion times of two jobs, T_i and T_j, from the dilation factors, λ_i and λ_j, obtained by $T_i = \lambda_j \tau_j + \left(1 - \frac{\lambda_j \tau_j}{\lambda_i \tau_i}\right) \tau_i$, $T_j = \lambda_j \tau_j$. Since job j finishes at $\lambda_j \tau_j < \lambda_i \tau_i$, $1 - \frac{\lambda_j \tau_j}{\lambda_i \tau_i}$ of job i will be executed without interference with job j. Similarly, we can calculate the actual completion times of more than two jobs. Note that in Remark 4, we have already discussed difficulty with calculating completion time of jobs when jobs are partially overlapped during their executions. We believe that an efficient method for such a general case deserves a seperate effort.

What our model computes is the dilation factor, λ_j for job j, which is given by $\lambda_j = 1 + \boldsymbol{p}_j \cdot \overline{\boldsymbol{p}} - \boldsymbol{p}_j \cdot \boldsymbol{p}_j$, where $\boldsymbol{p}_j$ is the jth column vector of the loading matrix and $\overline{\boldsymbol{p}} = \boldsymbol{p}_1 + \boldsymbol{p}_2$ in the two-resource-busy model. To compute the dilation factor, we obtain the loading vector or loading matrix using either Algorithm 3 or Property 2 (Refer to Section 3).

5.5 Model Validation

In this section, we validate the proposed D-factor model while illustrating the procedure to profile a job and to estimate the completion time of a job when it is co-hosted with other jobs. Experimental results in this section indicate that the D-factor model captures the behavior of each job in shared systems. The D-factor model provides (1) more accurate estimation of the completion times of co-hosted jobs than the linear sum model, (2) more efficient utilization of the system resources and (3) better predictable performance with existing scheduling



(a) Native Linux (b) Virtualized Environment

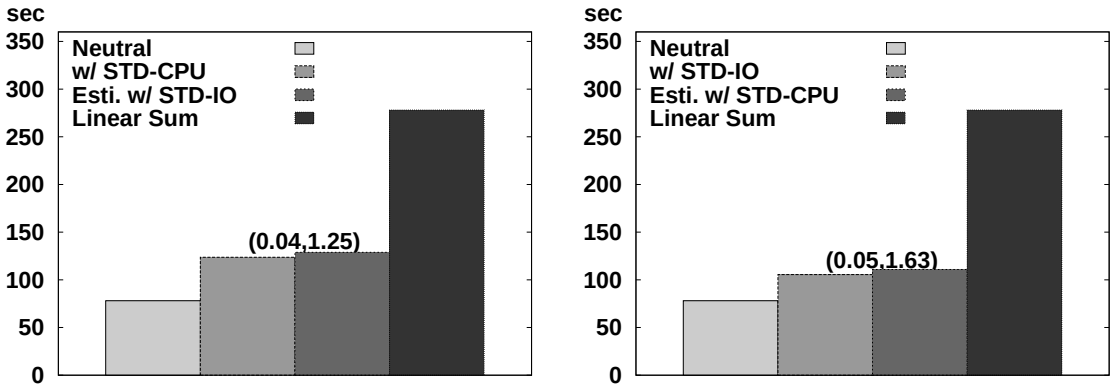
Figure 5.4: Experiments with standard jobs show the average errors from dilation factor are 7% for the native Linux and 10.3% for the virtualized environment. CPU represents `STD-CPU` and I/O represent `STD-IO`. We hosted two processes in (a) and two VMs in (b).

algorithms than with the linear sum. Note that all numbers are averaged over 40 runs since dilation factor model actually aims to predict average completion time of jobs, not to predict variance of completion time of jobs. Numbers in parenthsis represent the relative error of the estimation given by $|(\hat{T}-T)/\hat{T}|$, where $\hat{T}$ is from measurement and T is from estimation.

5.5.1 Synthetic workloads

Standard jobs As shown in Figure 5.4, the experiments with both `STD-CPU` and `STD-IO` confirm that the proposed model estimates the slow-downs of completion times of jobs due to multiple resource contention in shared systems. Since each standard job uses only one resource, we may consider `STD-CPU` characterized by $\boldsymbol{p} = (1, 0)$ and `STD-IO` by $\boldsymbol{p} = (0, 1)$. With the given loading vectors, we obtain $\lambda = 1$ for two different standard jobs and $\lambda = 2$ for two identical standard jobs according to Theorem 1. When we set the duration of each standard job to $100sec$, the total completion time of two different standard jobs will be $100sec$ and that of two identical standard jobs will be $200sec$. Hence, we may observe that the completion time of a job in a shared multi-resource system is not linearly proportional to the completion times of individual jobs.

The gist of the dilation factor theorem is that multiple resource contention results in a non-linear waiting time, proportional to their probabilities of accessing



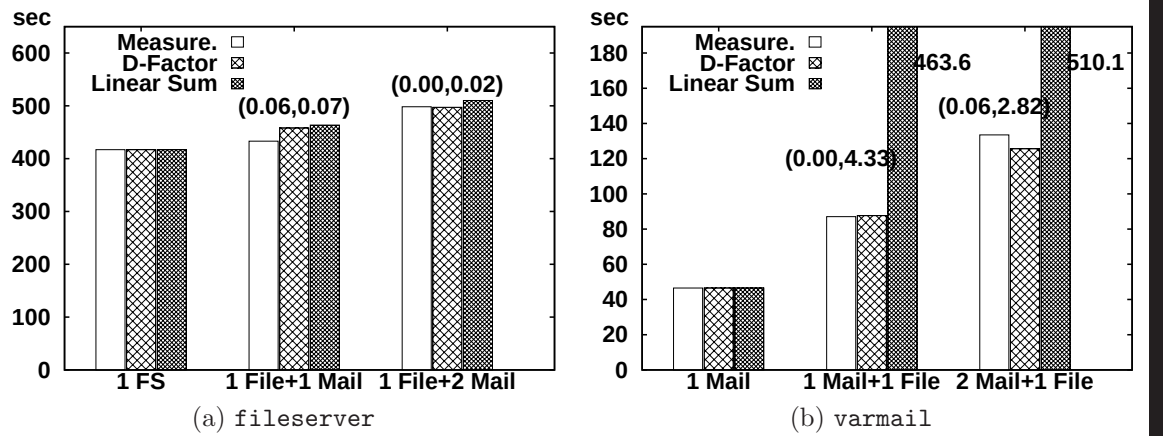
(a) `FileComp` with `STD-CPU`. (b) `FileComp` with `STD-IO`.

Figure 5.5: Estimated completion times of `FileComp` (a) using the loading vector of `FileComp` from measurements with `STD-CPU`; and (b) using the loading vector to predict the completion time with `STD-IO`. The numbers in parenthesis represent relative errors from D-factor and linear sum, respectively.

the system resources. In contrast, the linear sum model estimates the total completion time as the linear sum of completion times of individual jobs. The linear sum fits when the system owns one resource or serves each job after completing the previously assigned jobs. Experiments from standard jobs indicate the average errors from D-factor model are 7% for the native Linux and 10.3% for the virtualized environment. However, the linear model shows a 98% error in the worst case.

Figure 5.4a shows completion time of each co-located workload in the native Linux (two processes) and Figure 5.4b for the virtualized environment (two VMs). These experiments confirm that D-factor model accurately captures the completion times of jobs with multiple shared resources. In addition, we show that we can construct a standard job to profile workloads according to the proposed model in actual systems. However, we observe a significant error for two identical `STD-IO`, which is attributed to variance in disk access latencies to perform the same operations according to the sequence of access requests [88].

File compression Now, we demonstrate that the proposed model can estimate the completion time of a workload with both CPU and disk I/O accesses, `FileComp`. In these experiments, we set the duration of `STD-CPU` and `STD-IO` to 200*sec*. For the loading vector of `FileComp`, $\boldsymbol{p} = (0.58, 0.42)$, the estimated completion time



(a) `fileserver` (b) `varmail`

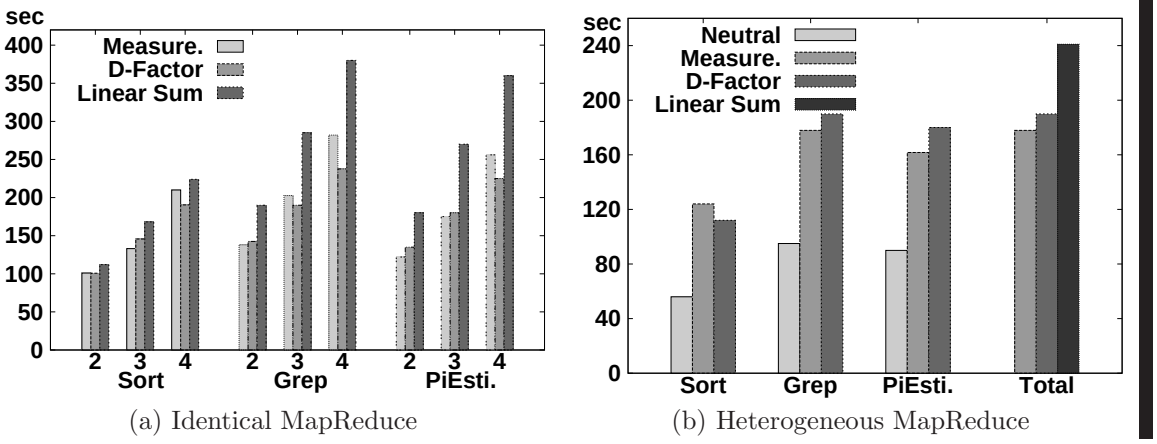
Figure 5.6: Completion times for various combinations of FileBench workloads on one physical machine with up to 3VMs. The loading vectors are obtained as to Algorithm 3.

will be $78.1 \times (1 + 0.42) = 110.9sec$ since we assume that the loading vector of the I/O standard is $(0, 1)$. From actual measurements for `FileComp` with the I/O standard, we obtain the completion time as $105.50sec$, resulting in a 5.1% error (refer to Figure 5.5). We have shown the accuracy of estimation from D-factor model with synthetic workloads. Next, we show the validation results with realistic workloads.

5.5.2 Realistic workloads

FileBench in a virtualized environment Experimental results with FileBench are shown in Figure 5.6. We confirm that the D-factor model well explains the dilated completion times for collocated I/O workloads in a virtualized environment as well.

With the provided loading vectors for both workloads, we estimate the completion time as $458.1sec$ when we co-locate one `varmail` and one `fileserver` in the same physical machine, compared to $433.1sec$ with actual measurements. In this situation, we estimate the running time of `varmail` to be $87.62sec$, which shows only a 0.65% error with actual measurements. However, the linear model cannot estimate the completion time of each workload. With the linear model, if we consider the total completion time of all the workloads in the system as the completion time of each job, the completion time of `varmail` becomes $463.6sec$.



(a) Identical MapReduce (b) Heterogeneous MapReduce

Figure 5.7: (a) Total completion times of each MapReduce application against the number of instances. The maximal errors are 9.4% for `Sort`, 15.78% for `Grep`, and 12.1% for `PiEstimator`. (b) With the loading vectors of each of three applications, we can estimate the completion times of each of hetorogeneous MapReduce applications. Both experiments used a 17-node cluster.

From Figure 5.6, we can observe that for one instance of `fileserver` and two instances of `varmail`, the respective average running times are 133.5*sec* and 498.3*sec*. The corresponding average running times from estimation are 133.48*sec* and 497.2*sec*. From the experimental results with FileBench, the estimation from the D-factor model shows 6.0% error. Next, we show more complex examples of identical MapReduce applications and heterogeneous MapReduce applications in a system.

Identical MapReduce applications Experiments with identical mapreduce applications on a 17-node cluster demonstrate the accuracy of the D-factor model as the number of instances increases up to four. In Figure 5.7a, the estimation from dilation factor theorem shows 9.4% error for `Sort`, 15.78% for `Grep`, and 12.1% for `PiEstimator` in the worst case. Compared with the D-factor model, the linear model shows larger margin of errors for all the considered workloads, 26%, 38%, and 54% for `Sort`, `Grep`, and `PiEstimator`, repsectively. We observe that applications with substantial I/O accesses tend to show larger errors as we increase the number of instances in the system. We speculate that this trend stems from the variance in disk latencies similar to what we observed for `STD-IO` cases. We plan to model this behavior in future work.

Heterogeneous MapReduce applications Figure 5.7b shows that the D-factor model estimates the completion times of three co-hosted MapReduce applications within 11% error. Loading vector and neutral completion time of each application remain the same as identical MapRecude experiments.

This experiment contrasts the D-factor model with the linear model in terms of the ability of estimating completion times of individual workloads when they are hosted in a shared system. The linear model accounts for the total completion time without considering the completion times of individual jobs. However, the D-factor model obtains the total completion time based upon the completion times of individual workloads. The ability to estimate completion time of each workload is practically useful in shared systems. In our experiment, we observe that `Sort` completes in about 120*sec*. Using the linear model, a new job will be scheduled to the system after all the previous workloads are completed; 241*sec* later since we cannot estimate that `Sort` will be completed before that (refer to Figure 5.7b). However, the D-factor model can estimate that `Sort` will be completed in 110*sec*. Thus, we may schedule another workload at 110*sec* after initiating all the applications.

We notice that the measured completion times of `Grep` and `PiEstimator` are smaller than the estimation. In this analysis, we assumed two-resource-busy jobs, which means there is no idle period in processing the given workloads and consumes only two resources. Since `Grep` and `PiEstimator` have substantial I/O activities, we may have idle wait times for completing I/O accesses. When we multiplex I/O workloads with other types of workloads, operating systems may utilize idle wait times. Thus, the completion time of `Grep` and `PiEstimator` could be smaller than the estimation when they are co-hosted. We have briefly discussed how to handle jobs with idle periods in Section 5.3.4 (refer to Remark 5).

To summarize, through experiments, we demonstrated that the D-factor model can estimate the total completion time of jobs when they share multiple system resources. In this section, we experimented with synthetic workloads – standard jobs and file compression as well as realistic workloads – MapReduce and FileBench. We demonstrated that the D-factor model well describes the behavior of both the native Linux and virtualized systems since the model is a high-level statistical abstraction of system behaviors. We showed that the D-factor model does not only estimate the total completion time, but also estimates the completion times of

individual jobs. We can observe that the D-factor model can quantitize the performance of multiplxed workloads, given the workload profiles. Also, we showed the process to use the D-factor model to estimate the completion times of given workloads: construct standard jobs, probe the target workloads using the standard jobs, and estimates completion times of given workloads.

Experiments in this section suggest that we may predict I/O performance for cloud services [8] using the D-factor model. In following section, we demonstrate an example for exploiting the D-factor model to enhance actual system performance with a scheduling algorithm.

5.6 Extending Schedulers for Single Resource Systems

This section describes practical benefits from the proposed D-factor model by applying it to an existing scheduling algorithm. We selected the on-line parallel machine scheduling problem [37] that addresses the problem of scheduling jobs on parallel machines without the knowledge of the sequence and the size of future jobs. This problem can be translated into the on-line bin packing problem [89], which is most widely considered in assigning virtual machines to physical machines [24, 90, 91]. We acknowledge that it would be challenging to apply the D-factor model to other classes of scheduling problems that consider flow control of jobs or machines such as open-shop, flow-shop, and job-shop [92] since the model assumes independent jobs.

One of the initial studies on on-line parallel machine scheduling problems is the list scheduler [1], an on-line greedy scheduler, which decides schedule S of a new job j with size p_j by

$$min \left\{ p_j + \sum_{i \in J_\mu} p_i \right\}, \forall \mu \in M \,, \tag{5.19}$$

where J_μ is the set of jobs that is currently running in the machine μ and M is the set of machines. For breaking a tie, the lowest indexed machine is chosen. Although this algorithm is simple, competitive ratio of list scheduler is 2, while the best result known to date is 1.9201-competitive algorithm for deterministic

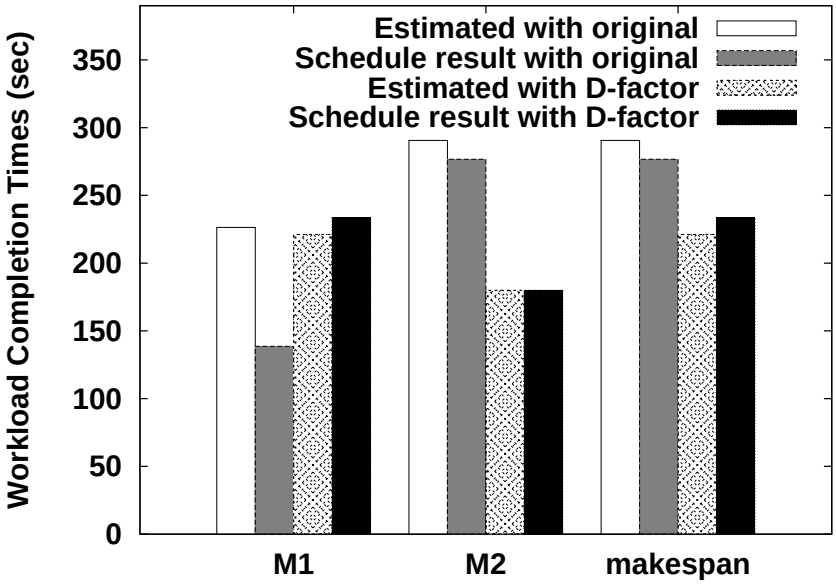


Figure 5.8: Adopting D-factor as objective function in Graham's on-line scheduler [1] shortens the makespan by 20.5%, compared with the original algorithm.

algorithms [93] and 1.58-competitive algorithm for randomized algorithms [94]. Thus, we selected this algorithm to demonstrate the performance enhancement by adopting the D-factor model in scheduling algorithms, though we do not claim any theoretical improvements in scheduling problems.

Here, we created four jobs using synthetic workloads, `STD-CPU` and `STD-IO`, which run on two machines on the same experimental platform described in Section 5.4. We consider a situation that three jobs (W_1, W_2, W_3) are already in the system and a new job, W_4, arrives at the system $60sec$ later. The size, completion time and loading vector, of each job are shown in Table 5.3. Experimental results shown in Figure 5.8 demonstrate that the makespan with D-factor is 20.5% shorter than that with the original list scheduler, which is a significant enhancement since the theoretical lower bound of deterministic on-line parallel machine scheduling problem is 1.88-competitive [95], which is a 12% enhancement (1.88 compared to 2). However, with the multi-resource contention model, performance enhancement may exceed the theoretical bound for single-resource machines.

Let us explain the rationale behind this performance improvement. The original list scheduler schedules the first three jobs as follows:

Table 5.3: Job profiles for the scheduling example.

workload	completion time	loading vector
W_1	110.56 sec	(1,0)
W_2	115.83 sec	(0,1)
W_3	180.0 sec	(1,0)
W_4	110.56 sec	(1,0)

$$\begin{array}{lllll} Machine1 & : W_1 & W_2 & (226.4sec) \\ Machine2 & : W_3 & & (180.0sec) \end{array}^{,}$$

where the numbers in the parenthesis represent estimated completion times of jobs in each machine. When the fourth job, W4 with job size of $p_4 = 110.56sec$, arrives $60sec$ later, list scheduler schedules the fourth job to Machine 2, expecting $180.0 + 110.56sec$ of total completion time. Thus,

$$\begin{array}{lllll} Machine1 & : W_1 & W_2 & (226.39sec) \\ Machine2 & : W_3 & W_4 & (290.56sec) \end{array}^{.}$$

By augmenting the objective function (Equation 5.19) to Equation 5.18, we can use vector-valued size, loading vectors. Then, the list scheduler with loading vectors schedules the first three jobs as follows:

$$\begin{array}{lllll} Machine1 & : W_1 & W_2 & (\overline{\boldsymbol{p}} = (1,1), 115.83sec) \\ Machine2 & : W_3 & & (\overline{\boldsymbol{p}} = (1,0), 180.0sec) \end{array}^{,}$$

where $\overline{\boldsymbol{p}}$ is the total loading vector and the numbers are estimated total completion times based upon D-factor model. Notice that, for the same configuration of machine 1, the estimated completion time with D-factor model is $115.83sec$, compared to $226.4sec$ with the original list scheduler based upon linear sum. Actual measurement of this schedule was $138.65sec$, which confirms that the D-factor model provides better estimation of total completion time. This will lead to a contrasting schedule when W_4 arrives $60sec$ later. For W_4 with $\boldsymbol{p}_4 = (1,0)$, the schedule becomes

$$\begin{array}{lllll} Machine1 & : W_1 & W_2 & W_4 & (\overline{\boldsymbol{p}} = (2,1), 221.2sec) \\ Machine2 & : W_3 & & & (\overline{\boldsymbol{p}} = (1,0), 180.0sec) \end{array}^{,}$$

which results in 20.5% enhancement in makespan.

In this example, we demonstrated that the same scheduling algorithm may show

significant performance gain by evaluating the total completion time of candidate schedules with the D-factor model. In general, we can expect better scheduling results with the D-factor model when workloads access multiple resources and they are independent since the D-factor model considers multiple resources contention among independent jobs. In order to consider multiple resources in a system, we may consider other types of scheduling problems such as open-shop, flow-shop, or job-shop scheduling problems. Instead, the D-factor model can augment existing on-line parallel machine scheduling algorithms that assume a machine as one resource, to consider multiple resources in a machine. Hence, when a system uses on-line parallel machine scheduler or on-line bin packing algorithms, we could augment the system without significant changes in software/hardware to reduce the makespan of workloads.

5.7 Related Work

Despite the importance of estimating dilation factor of each co-located job for providing predictable performance in shared environments, there is no established generic approach to estimate dilation factor of workloads in shared envionments [32, 83]. The main difficulty to estimate the dilation factor of each job is that it depends on the behavior of co-located jobs, which is often described as non-deterministic behavior [28] and a critical problem in Cloud platforms [8].

As an attempt to estimate dilation factor, Govindan *et al.* [32] and Mars *et al.* [83] proposed an empirical method for sharing memory subsystems. Especially, Mars *et al.* demonstrated that they could predict dilation factors up to three co-located Google's workloads in Google's infrastructure. Both studies created a probe program to measure the sensitivity of each workload for sharing memory/cache. Based upon the empirical sensitivity analysis, they estimated slow-down of workloads with different co-located workloads. With our model, both works can be considered as an example of two-resource busy model to express the slow-down of co-located applications. Their concepts of sensitivity of each application and pressure on shared resources are captured in our statistical job characteristics – loading vector. In addition, we suggested probe jobs for other resources such as CPU and disk I/O. We believe that loading vector can be con-

structed using probe jobs in [32, 83] in order to consider spatial characteristics of memory systems.

The closest problem setting to our model is the bin packing problem, specifically the multi-bin packing problem. Bin packing problem finds a solution to pack items into single or multiple bins so as to minimize the number of bins used [96]. In the bin packing problem, the total size of packed items is the linear sum of the sizes of individual items. However, this study mentions that obtaining the actual size of all the packed items might be smaller than the total size of each item. In multi-bin packing, we model bins and the sizes of items as vectors, but it does not consider the contention across resources when we place items to bins. Thus, it cannot express the non-linearity of dilation factors of workloads in shared environments. Since most of prior virtual machine placement methods are based upon on-line bin packing algorithms [24, 90, 91] or similar linear relations [23], they are inherently difficult to reflect non-linear slow-down of a job due to multiple resource contention.

Unlike the prior work, we provide a model that can predict the performance of applications in shared environments. With mixed workloads in data centers, we can achieve more graceful performance degradation [14, 25]. In addition, the issue raised from I/O-bound work in [8] implies that we need to consider the importance of multiple resource requests in processing a job. Therefore, we proposed a performance model that estimates the performance impact on a job by other collocated jobs running together in a server.

5.8 Conclusions

In this study, we derived a novel completion time model of jobs for shared service systems. We estimate the completion time of jobs on a system by considering that multiple shared resources may be involved to process a job. The resource usage of a job is represented by a loading vector, which in turn is used to estimate the dilation in individual job completion times. Based upon the proposed model, we profiled a job and estimate the overall completion time of jobs in shared service systems. In contrast to queuing theory based models that require parameters of each resource such as service rates and request arrival rates, the D-factor model only needs to the completion times to profile a job, from which the model can estimate the

completion times of co-located jobs. We validate the proposed D-factor model with experiments using synthetic and real workloads. From the validation results using synthetic workloads, the average relative errors are 7% in the native Linux environment and 10.3% in a VM environment. With realistic workloads, the D-factor model also predicts the completion time of co-existing workloads within a 16% error.Also, we presented an example to extend a scheduler based upon single-resource model, which reduces the makespan by 20.5%.

Calculating the overall completion times of the assigned jobs is the fundamental operation in designing or evaluating a scheduling algorithm. We can use the D-factor model to estimate the completion time more accurately or conveniently than previous models. The required overhead is to find the loading vector of a job to obtain the expected total completion time. As described in section 5.3, we can obtain the loading vectors by comparing the total completion time of multiple instances of identical jobs with the neutral completion time of the job. One of the assumptions made in this study is that the resource access pattern of each job is independent to each other. Otherwise, we cannot easily calculate the probability of the resource contention by the inner product of loading vectors.

Since sharing multiple resources among jobs is a common practice in computing systems, we can find a plenty of applications for the proposed D-factor model. Some of possible applications are as follows: Estimating the total completion time of jobs when we have heterogeneous processors such as CPU and GPU might be possible. CPUs and GPUs can be treated as different types of resources to be shared among jobs and the total completion time of a job depends on the portion of accessing times of each processor. Similarly, when we have multiple layers of caches and want to find the total completion time of a job, we can apply the D-factor model.

Although we successfully demonstrated the potential of D-factor model, several diretions are possible to further enhance the D-factor model. These include:

- Extending the model for the space-shared resources like memory systems; the working set behavior of each co-located job will modify the loading vector of each job. In order to use the model for the space-shared resources, we need to establish a model that identifies the modifier of the loading vector of each job due to the working set behavior. The authors believe such an extention

can consider in-processor cache and shared system memory, which will make the D-factor model more useful in modern many-core environments.

- An algorithm to compute the dilation factor for n partially overlapped jobs; this can be discussed in the context of job scheduling since the initialization and completion of jobs are dependent on job schedulers. As breifly mentioned in the section 5.6, developing a job scheduler with the D-factor model is one of the eventual goals of this study.
- Overcoming the dependency of the loading vector of a job on the measured platform; since the loading vector of a job is obtained from measurements, it is dependent on a specific hardware-operating system combination. However, the D-factor model requires relatively smaller effort than prior analytical models. One possible direction to work around this would be a method to convert the loading vector of a job in on system to other systems.

In a sense, although the proposed D-factor model is not the most perfect model to explain shared resource contention, the model provides an insight to understand contention for multiple shared resources. In addition, the D-factor model allows to extend many existing scheduling algorithms for single resources into scheduling algorithms for multiple resources.

Chapter 6

Conclusions

In order to address constant pressure on scalibility in data centers, this thesis studied a scalable management of performance and energy consumption in data centers. The contributions of this study encloses: a simulation platform; a dynamic energy management scheme; and a performance model for job slow-down due to co-located jobs. Scalbility and flexiblity are the coherant thread that connects those contributions.

The proposed simulation platform can increase the number of system nodes up to 128 server nodes as well as the traffic volume up to 4800 users, which was achieved by a modular design of the simulation platform. Each resource is modelled after a queue and, therefore, a queuing network is constructed for a server node. The entire data center is considered as a open queuing network of server nodes. A process is created to simulate a server node, which may limit the total number of servers to be simulated. If, however, we can construct a simulation platform that can run in parallel environment, we may overcome this deficit. In the future, we may extend or intergrate this simulator with network simulator in order to capture network behavior in a data center. This simulation platform has served a fair scientific method to evaluate the performance and energy consumption of data centers with the proposed energy management scheme in this study.

A formal description of energy consumption of a system is discussed, which leads to a dynamic energy management scheme based upon queueing anaylsis. The proposed energy management scheme dynamically and proactively adjusts the states of servers, which includes power on/off and sleep states. The proactive

exploitation of low-utilization periods in each server produced a 50% of energy saving with benign performance impact as shown by simulation results for RUBiS workloads. This energy management scheme can be implemented in software layer with system calls to exert controls over sleep states of each hardware component. We may further enhance the energy saving by introducing randomized algorithms instead of deterministic algorithms used in this scheme.

I developed a generic quantitative performance model for shared systems since large scale systems are often intended to be shared by multiple services. In Cloud computing, users are interested in stable performance of their applications instead of system performance. Thus, I propose D-factor model that can estimate application slow-down with other co-located workloads. The proposed D-factor model can consider multiple resources and workloads in a system without incurring system resource monitoring and kernel tracing. Such a feature is favorable in cloud environments, where users who run applications often cannot monitor physical resources. D-factor model successfully formulated the commonly observed, but not formulated two phenomena – non-linear slow-down of application with co-existing workloads and the slow-down is related to characteristics co-existing workloads. The enhanced accuracy of running times of workloads in shared systems may contribute to make performance variance due to sharing resources more prectable.

This study demonstrated that D-factor model offers more practical and accurate criteria for schedulers in shared systems since D-factor model respects that a system consists of multiple resources. We observed a 20.5% reduced makespan when an existing scheduler for a single-resource system, list scheduler, adopted D-factor model as the objective function to evaluate candiate schedules. Such a significant reduction is attributed to the capability of describing multiple resources in a system. If we can design a novel scheduler that fully utilizes the benefits from D-factor model, we may experience measurably enhanced system performance. It would be interesting to combine the proposed energy management scheme and a scheduler based upon D-factor model. Consequently, this study may lead to more energy efficient and productive data centers, which will result in more economic data centers.

Bibliography

[1] Graham, R. (1969) "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal on Applied Mathematics*, **17**(2), pp. 416–429.

[2] Normandeau, K. (2011), "2011 Data Center Market Insight Report," http://www.datacenterknowledge.com/archives/2011/09/13/2011-data-center-market-insight-report/.

[3] HPCwire.com (2010), "Scaling the Exa," http://www.hpcwire.com/hpcwire/2010-06-03/scaling_the_exa.html.

[4] Wray, R. (2009), "Internet data heads for 500bn gigabytes," http://www.guardian.co.uk/business/2009/may/18/digital-content-expansion.

[5] McKendrick, J. (2010), "Size of the data universe: 1.2 zettabytes and growing fast," http://www.zdnet.com/blog/service-oriented/size-of-the-data-universe-12-zettabytes-and-growing-fast/4750.

[6] Thibodeau, P. (2011), "Q&A: Exascale now a global race for tech," http://www.computerworld.com/s/article/9222022/Q_A_Exascale_now_a_global_race_for_tech?taxonomyId=13.

[7] Lim, S.-H., B. Sharma, G. Nam, E. K. Kim, and C. R. Das (2009) "MDCSim: A multi-tier data center siimulation platform," *Cluster Computing and Workshops, 2009. (CLUSTER '09). IEEE International Conference on.*

[8] Armbrust, M., A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia (2010) "A view of cloud computing," *Communications of the ACM*, **53**(4).

[9] Magnusson, P., M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner (2002) "Simics: A full system simulation platform," *Computer*, **35**(2), pp. 50–58.

[10] Shantharam, M., P. Raghavan, and M. Kandemir (2009) "Hybrid Techniques for Fast Multicore Simulation," in *Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, Springer-Verlag, Berlin, Heidelberg, pp. 122–134.
URL `http://dx.doi.org/10.1007/978-3-642-03869-3_15`

[11] Buyya, R. and M. Murshed (2002) "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," *Concurrency and Computation: Practice and Experience*, **14**(13), pp. 1175–1220.

[12] Freeh, V. W., F. Pan, N. Kappiah, D. K. Lowenthal, and R. Springer (2005) "Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster," in *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*.

[13] Calheiros, R. N., R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya (2010) "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*.

[14] Fan, X., W.-D. Weber, and L. A. Barrose (2007) "Power Provisioning for a Warehouse-sized Computer," in *ISCA '07: Proceedings of the ACM international Symposium on Computer Architecture*.

[15] Meisner, D., B. T. Gold, and T. F. Wenisch (2009) "PowerNap: eliminating server idle power," in *ASPLOS '09: Proceeding of the 14th international conference on Architectural support*.

[16] Chase, J. S., D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle (2001) "Managing energy and server resources in hosting centers," in *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*.

[17] Elnozahy, E., M. Kistler, and R. Rajamony (2002) "Energy-Efficient Server Clusters," in *Proceedings of the 2nd Workshop on Power-Aware Computing Systems*.

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

[19] Pinheiro, E., R. Bianchini, E. V. Carrera, and T. HEath (2001) "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," in *Workshop on Compilers and Operating Systems for Low Power*.

[20] Sharma, V., A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu (2003) "Power-aware QoS Management in Web Servers," in *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*.

[21] Raghavendra, R., P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu (2008) "No "power" struggles: coordinated multi-level power management for the data center," *SIGARCH Comput. Archit. News*, **36**(1), pp. 48–59.

[22] Bobroff, N., A. Kochut, and K. Beaty (2007) "Dynamic placement of virtual machines for managing SLA violations," in *10th IFIP/IEEE International Symposium on Integrated Network Management*.

[23] Hermenier, F., X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall (2009) "Entropy: a consolidation manager for clusters," in *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09.

[24] Verma, A., G. Dasgupta, T. K. Nayak, P. De, and R. Kothari (2009) "Server Workload Analysis for Power Minimization using Consolidation," in *USENIX '09 Annual Technical Conference*.

[25] Meng, X., C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis (2010) "Efficient resource provisioning in compute clouds via VM multiplexing," in *ICAC '10: Proceeding of the 7th international conference on Autonomic computing*.

[26] Park, S., W. Jiang, Y. Zhou, and S. Adve (2007) "Managing energy-performance tradeoffs for multithreaded applications on multiprocessor architectures," in *SIGMETRICS '07: Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ACM, New York, NY, USA, pp. 169–180.

[27] Ding, Y., M. Kandemir, P. Raghavan, and M. J. Irwin (2009) "Adapting application execution in CMPs using helper threads," *J. Parallel Distrib. Comput.*, **69**, pp. 790–806.
URL `http://dl.acm.org/citation.cfm?id=1576861.1577035`

[28] Iyer, R., L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt (2007) "QoS policies and architecture for cache/memory in CMP platforms," in *Proceedings of the 2007 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*.

[29] Fedorova, A., S. Blagodurov, and S. Zhuravlev (2010) "Managing contention for shared resources on multicore processors," *Communications of the ACM*, **53**(2), pp. 49–57.

[30] Barham, P., B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield (2003) "Xen and the art of virtualization," in *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*.

[31] Liu, C., A. Sivasubramaniam, and M. Kandemir (2004) "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," in *Proceedings of the 10th International Symposium on High Performance Computer Architecture*.

[32] Govindan, S., J. Liu, A. Kansal, and A. Sivasubramaniam (2011) "Cuanta: Quantifying Effects of Shared On-chip Resource Interference for Consolidated Vitual Machines," in *Proceedings of the 2011 ACM Symposium on Cloud Computing*, SOCC '11.

[33] Bitirgen, R., E. Ipek, and J. F. Martinez (2008) "Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach," in *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*.

[34] Urgaonkar, B., G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi (2005) "An analytical model for multi-tier internet services and its applications," in *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*.

[35] Urgaonkar, B., P. Shenoy, and T. Roscoe (2002) "Resource overbooking and application profiling in shared hosting platforms," in *Proceedings of the 5th symposium on Operating systems design and implementation*, OSDI '02, ACM, New York, NY, USA, pp. 239–254.
URL `http://doi.acm.org/10.1145/1060289.1060312`

[36] Lim, S.-H., B. Sharma, B. C. Tak, and C. R. Das (2011) "A dynamic energy management scheme for multi-tier data centers," *Performance Analysis of Systems and Software (ISPASS), IEEE International Symposium on*.

[37] Shmoys, D., J. Wein, and D. Williamson (1991) "Scheduling parallel machines on-line," *Annual IEEE Symposium on Foundations of Computer Science*.

[38] Denning, P. J. (1968) "The working set model for program behavior," *Communications of the ACM*, **11**(5), pp. 323–333.

[39] Padala, P., K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant (2009) "Automated control of multiple virtualized resources," in *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*.

[40] Nathuji, R., A. Kansal, and A. Ghaffarkhah (2010) "Q-clouds: managing performance interference effects for QoS-aware clouds," in *Eurosys*.

[41] The Apache Software Foundation (2003), "The Apache HTTP Server Project," Http://httpd.apache.org.

[42] ObjectWeb Consortuim (2006), "JOnAS," `http://jonas.objectweb.org`.

[43] Cecchet, E., J. Marguerite, and W. Zwaenepoel (2002) "Performance and scalability of EJB applications," *SIGPLAN Not.*, **37**(11), pp. 246–261.

[44] ObjectWeb Consortuim (2006), "C-JDBC," `http://c-jdbc.objectweb.org`.

[45] MySQL (2006), "MySQL," http://www.mysql.com.

[46] Amza, C., E. Cecchet, A. Chanda, A. L. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel "bottleneck characterization of dynamic web site benchmakrs," in *Third IBM CAS Conference, 2002*.

[47] "NYU-TPC-W," `http://www.cs.nyu.edu/pdsg/`.

[48] "TPC-W: Benchmarking An E-Commerce Solution," `http://www.tpc.org/information/other/techarticles.asp`.

[49] FinanceTech (2009), "FinanceTechnology Network," Available from `http://www.financetech.com`.

[50] Marwah, M., R. Sharma, R. Shih, C. Patel, V. Bhatia, M. Mekanapurath, R. Velumani, and S. Velayudhan (2009) "Data analysis, visualization and knowledge discovery in sustainable data centers," in *COMPUTE '09: Proceedings of the 2nd Bangalore Annual Compute Conference*.

[51] Liu, J., D. Poff, and B. Abali (2009) "Evaluating high performance communication: a power perspective," in *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, ACM, New York, NY, USA, pp. 326–337.
URL `http://doi.acm.org/10.1145/1542275.1542322`

[52] Feng, W. B., C. P. Baron, L. Bhuyan, and D. Panda "Performance characterization of a 10-Gigabit Ethernet TOE," in *High Performance Interconnects, 2005. Proceedings of 13th Symposium on*.

[53] Abdelzaher, T. F., K. G. Shin, and N. Bhatti (2002) "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 13.

[54] Villela, D., P. Pradhan, and D. Rubenstein (2007) "Provisioning servers in the application tier for e-commerce systems," *ACM Trans. Interet Technol.*, **7**(1), p. 7.

[55] Chen, J., G. Soundararajan, and C.Amza (2006) "Autonomic Provisioning of Backend Databases in Dynamic Content Web Servers," *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pp. 231–242.

[56] Stewart, C. and K. Shen (2005) "performance modeling and system management for multi-component online service," in *NSDI '05: 2nd Symposium on Networked Systems Design & Implementation*.

[57] Balaji, P., S. Narravula, K. Vaidyanathan, S. Krishnamoorthy, J. Wu, and D. K. Panda (2004) "Sockets Direct Protocol over InfiniBand in clusters: is it beneficial?" in *ISPASS '04: Proceedings of the 2004 IEEE International Symposium on Performance Analysis of Systems and Software*.

[58] Sun Microsystems, Inc. (2003), "J2EE: Java 2 Platform Enterprise Edition," Available from `http://java.sun.com/j2ee/`.

[59] Gurumurthi, S., A. Sivasubramaniam, M. J. Irwin, N. Vijaykrishnan, M. Kandemir, T. Li, and L. K. John (2002) "Using Complete Machine Simulation for Software Power Estimation: The SoftWatt Approach," in *HPCA '02: Proceedings of the 8th International Symposium on High-Performance Computer Architecture*.

[60] Wu, W., L. Jin, J. Yang, P. Liu, and S. X.-D. Tan (2007) "Efficient power modeling and software thermal sensing for runtime temperature monitoring," *ACM Trans. Des. Autom. Electron. Syst.*, **12**(3), pp. 1–29.

[61] Hsu, C.-H. and W.-C. Feng (2005) "A Power-Aware Run-Time System for High-Performance Computing," in *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*.

[62] Rusu, C., A. Ferreira, C. Scordino, and A. Watson (2006) "Energy-Efficient Real-Time Heterogeneous Server Clusters," in *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, IEEE Computer Society, Washington, DC, USA, pp. 418–428.

[63] Elnozahy, M., M. Kistler, and R. Rajamony (2003) "Energy conservation policies for web servers," in *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, USENIX Association, Berkeley, CA, USA, pp. 8–8.

[64] Bianchini, R. and R. Rajamony (2004) "Power and energy management for server systems," *Computer*, **37**(11), pp. 68–76.

[65] Mesquite Software, "CSIM," `http://www.mesquite.com`.

[66] InfiniBand Trade Association (2004), "InfiniBand Architecture Specification, Volume 1 & 2, Release 1.2," http://www.infinibandta.org.

[67] Garcia, D. F. and J. Garcia (2003) "TPC-W E-Commerce Benchmark Evaluation," *Computer*, **36**(2), pp. 42–48.

[68] Ersoz, D., M. S. Yousif, and C. R. Das (2007) "Characterizing Network Traffic in a Cluster-based, Multi-tier Data Center," in *ICDCS '07: Proceedings of the 27th International Conference on Distributed Computing Systems*.

[69] Urgaonkar, B., P. Shenoy, A. Chandra, and P. Goyal (2005) "Dynamic Provisioning of Multi-tier Internet Applications," in *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*.

[70] Wang, Z., C. McCarthy, X. Zhu, P. Ranganathan, and V. Talwar (2008) "Feedback Control Algorithms for Power Management of Servers," in *3rd Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID'08)*.

[71] Yokogawa Electric Corporation (1998), "WT210 Digital Power Meter User's Manual," .

[72] U.S. Environmental Protection Agency, E. P. (2007), "EPA Report to congress on Server and Data Center Energy Efficiency," .

[73] Reiser, M. and S. S. Lavenberg (1980) "Mean-Value Analysis of Closed Multichain Queuing Networks," *Journal of the ACM*, **27**(2).

[74] consortium, O., "RUBiS:Rice University Bidding System," `http://rubis.ow2.org`.

[75] Rajamani, K. and C. Lefurgy (2003) "On evaluating request-distribution schemes for saving energy in server clusters," in *ISPASS '03: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*.

[76] Gandhi, A., M. Harchol-Balter, R. Das, and C. Lefurgy (2009) "Optimal power allocation in server farms," in *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*.

[77] Horvath, T. and K. Skadron (2008) "Multi-mode energy management for multi-tier server clusters," in *PACT '08: Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, ACM, New York, NY, USA, pp. 270–279.

[78] Little, J. (1961) "A proof of the queueing formula:L=λW," *Operations Research*, **9**, pp. 383–387.

[79] Irani, S., S. Shukla, and R. Gupta (2007) "Algorithms for power savings," *ACM Trans. Algorithms*, **3**(4), p. 41.

[80] Menasce, D. (2002) "Two-level iterative queuing modeling of software contention," in *10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems.*

[81] Hoste, K., A. Phansalkar, L. Eeckout, A. Georges, L. K. John, and K. De Bosschere (2006) "Performance prediction based on inherent program similarity," in *PACT*.

[82] Koh, Y., R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu (2007) "An Analysis of Performance Interference Effects in Virtual Environments," in *ISPASS*.

[83] Mars, J., L. Tang, R. Hundt, K. Skadron, and M. L. Soffa (2011) "Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*.

[84] Wachs, M., L. Xu, A. Kanevsky, and G. R. Ganger (2011) "Exertion-based Billing for Cloud Storage Access," in *HotCloud*.

[85] Sharma, B., V. Chudnovsky, J. H. Rasekh Rifaat, and C. R. Das (2011) "Modeling and Synthesizing task Placement Constraints in Google Computer Clusters," in *Proceedings of the second ACM Symposium on Cloud Computing*.

[86] "FileBench," `http://www.solarisinternals.com/wiki/index.php/FileBench`.

[87] "Apache Hadoop," `http://hadoop.apache.org`.

[88] Shriver, E., A. Merchant, and J. Wilkes (1998) "An analytic behavior model for disk drives with readahead caches and request reordering," in *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*.

[89] Sanders, P., N. Sivadasan, and M. Skutella (2009) "Online Scheduling with Bounded Migration," *Mathematics of Operations Research*, **34**(2), pp. 481–498.

[90] Singh, A., M. Korupolu, and D. Mohapatra (2008) "Server-storage virtualization: integration and load balancing in data centers," in *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*.

[91] Meng, X., V. Pappas, and L. Zhang (2010) "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM'10: Proceedings of the 29th conference on Information communications*.

[92] Shmoys, D. B., C. Stein, and J. Wein (1991) "Improved approximation algorithms for shop scheduling problems," in *Proceedings of the second annual ACM-SIAM symposium on Discrete algorithms*.

[93] Fleischer, R. and M. Wahl (2000) "Online Scheduling Revisited," in *Algorithms - ESA 2000*.

[94] Chen, B., A. van Vliet, and G. J. Woeginger (1994) "A lower bound for randomized on-line scheduling algorithms," *Information Processing Letters*, **51**(5), pp. 219 – 222.

[95] Rudin, I., J. F., and R. Chandrasekaran (2003) "Improved Bounds for the Online Scheduling Problem," *SIAM J. Comput.*, **32**, pp. 717–735.

[96] Coffman, E. G., Jr., M. R. Garey, and D. S. Johnson (1997) *Approximation algorithms for bin packing: a survey*, PWS Publishing Co., Boston, MA, USA.

Vita

Seung-Hwan Lim

Education

Pennsylvania State University, University Park, PA
PhD in Computer Science and Engineering, Spring 2012.

Seoul National University, Seoul, Korea
MS in Computer Engineering, Fall 2000

Seoul National University, Seoul Korea
BS in Computer Engineering, Fall 1998

Professional experience

Samsung Electronics, Suwon, Korea
Software Engineer, 2000 – 2005