MANAGING ENERGY USAGE AND COST THROUGH LOAD DISTRIBUTION IN MULTI-DATA-CENTER SERVICES

BY KIEN TRUNG LE

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Ricardo Bianchini and Thu D. Nguyen

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

January, 2013

ABSTRACT OF THE DISSERTATION

Managing Energy Usage and Cost Through Load Distribution in Multi-Data-center Services

by Kien Trung Le

Dissertation Directors: Ricardo Bianchini and Thu D. Nguyen

Multi-data-center services will soon be common place. These services raise many questions about how exactly to distribute the offered load across the data centers. In fact, different load distributions may produce wildly different monetary costs, performance, and/or energy consumption. Moreover, these large services consume large amounts of energy produced via carbon-dioxide-intensive means. We refer to this as "brown energy", in contrast to renewable or "green" energy. This large energy consumption represents significant and fast-growing financial and environmental costs. Increasingly, services are exploring dynamic methods to manage energy and energy-related costs while respecting their service-level agreements (SLAs). Furthermore, it will soon be important for these services to manage their usage of brown or green energy.

Based on these observations, this dissertation explores load distribution policies that minimize cost or energy consumption, while respecting the services' performance requirements. First, we design, implement, and evaluate software support for multi-data-center Internet service providers to take advantage of temporal and spatial variability of electricity prices and on-site generation of green energy. Second, we extend this framework with support for capping brown energy consumption without excessively increasing costs or degrading the performance of Internet services. Third, we design,

implement, and evaluate cost-aware policies for placing and migrating virtual machines in high performance cloud computing services.

Our results show that our framework is very effective at allowing services to trade off brown energy consumption and cost. For example, using our policies, the service can reduce brown energy consumption by 24% for only a 10% increase in cost, while still abiding by SLAs. Our virtual machine migration framework demonstrates that (1) our policies can provide large cost savings, (2) load migration enables savings in many scenarios, and (3) all electricity-related costs must be considered at the same time for higher and consistent cost savings. Overall, our work demonstrates the value of load distribution across data centers, formal optimization techniques, workload prediction, and electricity price prediction for energy management.

Acknowledgements

First, I would like to thank my advisors Prof. Ricardo Bianchini and Prof. Thu Nguyen for their uttermost support during my very long graduate student career. Thank you for your enormous amount of encouragement when things did not turn out the way we wanted; your patience when I got stuck; and, not lastly, your gentle pushes when I slowed down. During these years, you two continuously challenged me to achieve my best. So once again, thank you Ricardo and Thu for everything.

Next, I would like to thank my wife, Thuy Pham who has supported, encouraged, and believed in me during this long and rough road that leads to this dissertation.

I also thank:

- my parents for their encouragement, support and love throughout my life.
- My sister, Havi, her husband Andrew and my brother-in-law, Binh for their support and love,
- my collaborators Prof. Margaret Martonosi, Prof. Yogesh Jaluria, Özlem Bilgir, Jingru Zhang, Jiandong Meng for their works that made this dissertation possible,
- my committee members Prof. Badri Nath, Prof. Prashant Shenoy, and Prof. Martonosi for reading and commenting on this dissertation,
- all the members of Panic-Lab and Dark-Lab, past and present; Matias Cuenca Acuna, Christopher Peery, Kiran Nagaraja, Xiaoyan Li, Wei Wang, Tuan Phan, Md. Haque, Bill Katsak, Daniela Vianna, Fabio Oliveira, Rekha Bachwani, Wei Zheng, Luiz Ramos, Ana Paula Centeno, Íñigo Goiri for many fun discussions and memorable activities,
- my friends around this area for the many social activities.

Dedication

To my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this dissertation, we address the problem of managing energy and energy-related costs for service providers that operate multiple distributed geographically data centers. In particular, we focus on services such as those provided by Google, Amazon, Microsoft and others. In this chapter, we first motivate our work, then give an overview of the dissertation and document our contributions.

1.1 Motivation

Data centers are major energy consumers [6, 86, 90]. Figure 1.1 shows that in 2005, the data centers in the US consumed 56 Billion KWhs, an amount of energy that is almost equivalent to that consumed by the entire transportation manufacturing industry (the industry that makes automobiles, airplanes, ships, trucks, and other means of transportation). In 2010, this number had increased by 53% to 85.6 Billion KWhs, representing about 2.2% of total US electricity use. As the world's main economies start to recover from a deep recession, the growth in data center electricity consumption is expected to accelerate further.

Data centers are operated by organizations from a wide range of industries (from financial to IT) with most of the large data centers run by large service providers such as Google, Microsoft and Amazon. With the advance of cloud computing (outsourcing of computing resources to service providers), the number of data centers employed by these providers will likely to increase. As the number of data centers and demand for these services increase, the issues of managing energy and energy costs become more important. Figure 1.2 and other reports [22, 27, 91] indicate that infrastructure

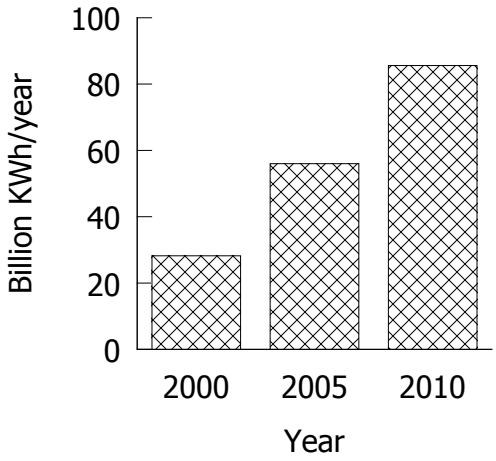


Figure 1.1: Electricity consumption of US data centers [90].

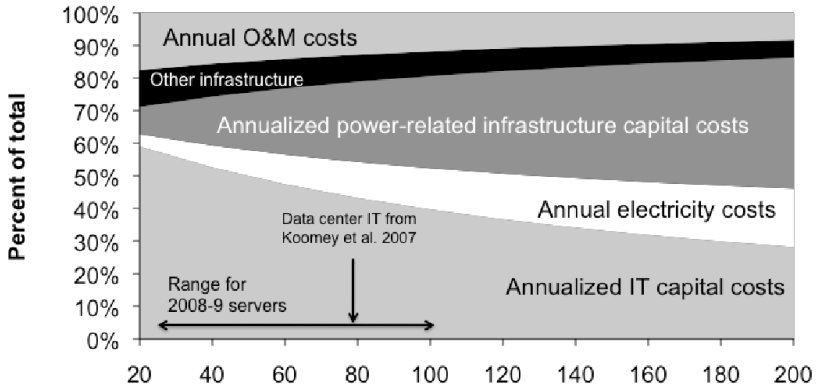


Figure 1.2: IT power (W)/Inflation adjusted IT costs [91]. X axis is the IT power use per thousand dollars of server cost, and Y axis is different data center cost components.

equipment related to power and cooling may be responsible for about half of total annualized costs in typical data center facilities and that this fraction is growing over time as IT equipment acquisition costs decline and IT equipment energy use increases [91].

This large energy consumption of data centers has both financial and environmental costs. Table 1.1 summarizes these costs in the context of some of the most popular Internet service providers which shows that these organizations consume a huge amount of electricity, and consequently have significant carbon footprint. In fact, a typical enterprise-class data center has a peak power of 50 MW, equivalent to 1/10 of the peak

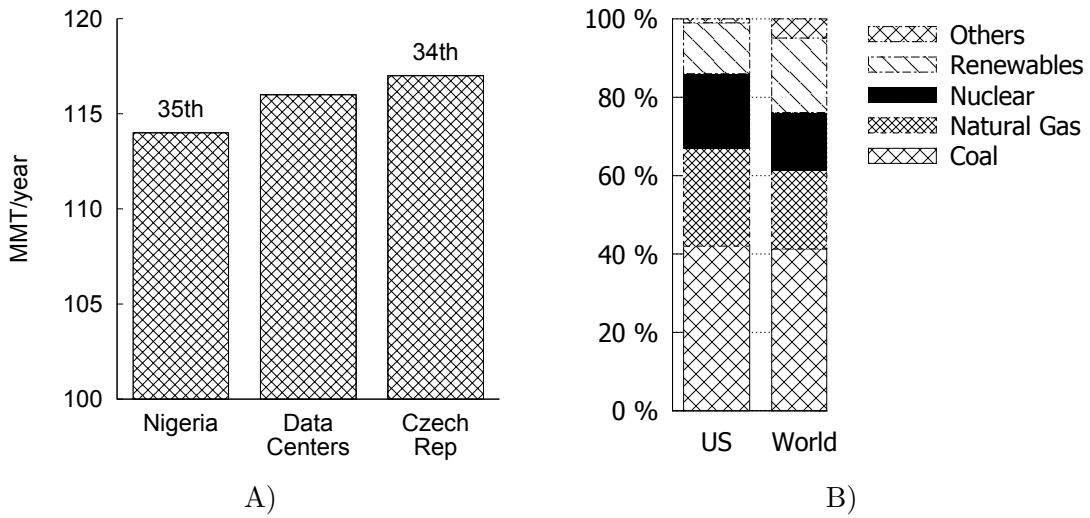


Figure 1.3: A) Carbon emissions of world-wide data centers comparing to that of Nigeria ranks 34^{th} and Czech Republic ranks 35^{th} on the world [105]. B) Electricity generation sources [132, 168].

power of a base load plant, and consumes as much energy as a city with 50,000 homes.

Figure 1.3A, based on a survey from 2008 [105], illustrates that world-wide data center carbon emission is somewhere between that of Nigeria, the most populous country in Africa, and Czech Republic, one of the most developed countries in the former Eastern European Bloc. The reason behind this is that most of the electricity produced worldwide (including the US) comes from burning coal or natural gas, a carbon-intensive approach to energy production. Figure 1.3B shows that this percentage is about 67% in the US and 60% world-wide. In fact, a single server can have roughly the same footprint as a Sports Utility Vehicle (SUV) [62].

There are many ways these organizations can reduce their impacts on environment. For example, they can invest in renewables by financing new plants and pumping the produced energy into the electrical grid. In fact, Google is taking this approach. On the other hand, they can directly reduce their brown energy consumption and use more green energy instead. The current trend is that some power utilities are starting to allow large electricity consumers to select a mix of brown and green power. Based on the power mix, the utility promises to pump the proper amount of green energy into the electricity grid. However, the power-mix approach often leads to substantially higher

Company	Servers	Electricity	Cost	CO_2(Tons)
eBay	16K	0.6×10^5 MWh	\$3.7M	0.43×10^7
Akamai	40K	1.7×10^5 MWh	\$10M	1.2×10^7
Rackspace	50K	2×10^5 MWh	\$12M	1.4×10^7
Microsoft	>200K	$> 6 \times 10^5$ MWh	>\$36M	4.3×10^7
Google	>500K	$> 6.3 \times 10^5$ MWh	>\$38M	4.5×10^7

Table 1.1: Financial and environmental Costs of Internet Services [135].

energy costs, as most utilities charge a premium for green energy. For data centers, a better approach may be to either generate their own green energy (self-generation) or draw power directly from a nearby green energy plant (co-location). This approach is becoming quite popular, as demonstrated by the many data centers (partially or completely) powered by solar or wind energy that are popping up all over the globe (see a partial list at `http://www.ecobusinesslinks.com/green_web_hosting.htm`). For example, Apple is constructing a 20MW solar array at its North Carolina data center [43]. McGraw-Hill has just already built a 14MW solar array for its data center [44]. Two small cloud service providers, Green House Data and AISO, run entirely on renewable energy [4, 70].

1.2 Our Goal

To address the energy consumption problem, there have been a lot of works (see chapter 3) in the computer science community. However, most of these techniques address the energy consumption of a single data center. This dissertation complements these efforts as our goal is to create software frameworks to help *multi-data-center* services to manage their energy consumption and energy costs.

Today, many large organizations operate multiple data centers. The reasons for this include natural business distribution, the need for high availability and disaster tolerance, the sheer size of their computational infrastructure, and/or the desire to provide uniform access times to the infrastructure from widely distributed client sites.

Interestingly, the geographical distribution of the data centers often exposes many opportunities for optimizing energy consumption and costs by intelligently distributing

the computational workload. In this dissertation, we seek to exploit such opportunities. First, we seek to exploit data centers that pay different and perhaps variable electricity prices. In fact, many power utilities now allow consumers to choose hourly [12] and peak/off-peak pricing instead of paying a fixed price. Second, different locations are exposed to different outside temperature which directly impacts the amount of energy spent on cooling. Third, we seek to exploit data centers that are located in different time zones, which adds an extra component to price and outside temperature variability. For example, one data center may be under peak-demand prices while others are under off-peak-demand prices. Finally, we seek to exploit data centers powered by green energy to reduce brown energy consumption. Such data centers can have on-site renewable energy generation or get the power as a mix of green and brown energy from the utility.

To make our investigation of these degrees of freedom more concrete, we consider a multi-data-center architecture where data centers are placed behind a set of front-end devices. The front-ends are responsible for inspecting each client request and forwarding it to one of the data centers that can serve it, according to a *request distribution or job placement policy*. Despite their wide-area distribution of requests, services must strive not to violate their service-level agreements (SLAs). This architecture is very popular among Internet services, such as Google or iTunes, and major cloud service providers including Microsoft, Salesforce, Skytap, HP, IBM, Amazon and Google.

1.3 Dissertation Structure

This dissertation proposes and evaluates several frameworks (using optimization and heuristics) to distribute load across geographically distributed data centers. The frameworks enable services to manage their energy consumption and costs, while respecting their SLAs. There are many issues that must be confronted in building load distribution frameworks for service providers that offer different types of services. Some of these are shared by all services; for example, all of them use load distribution as a mechanism to exploit variation in electricity prices, time zones and outside temperature, and manage the SLA. Some of these are specific to types of services; for example, Internet service

requests have short response time and very predictable diurnal patterns whereas high-performance computing (HPC) applications tend to have much longer run-times as well as unpredictable arrivals and resource usage. This affects the techniques that we can use to solve the problem. This dissertation is, therefore, structured based on the type of service for which we build a framework. Specifically, we concentrate on two types of services: Internet services and HPC cloud services. Note that we have also built a framework for Web services, but as it does not consider energy issues (only cost and performance), we include it in this dissertation only as an appendix. Next, we overview our frameworks for these services.

Framework for Internet Services. As mentioned, large Internet services are supported by multiple data centers for high capacity and availability, and low response times. The data centers sit behind front-end devices that inspect each client request and forward it to one of the data centers that can serve it, according to a *request distribution policy*.

In chapter 4, we propose a framework that exploits the geographical distribution of data centers powered by green energy. In fact, there are many such data centers popping up all over the globe [4,72,83,84,158,162]. Our goal is to design and implement request-distribution approaches for reducing the overall cost and energy consumption of multi-data-center Internet services.

Specifically, we propose and evaluate a software framework for optimization-based request distribution. The framework enables services to manage their energy consumption and costs, while respecting their SLAs. At the same time, the framework considers the existing requirements for high throughput and availability. Furthermore, the framework allows services to exploit data centers that pay different (and perhaps variable) electricity prices, data centers located in different time zones, and data centers that have on-site green energy generation.

Based on the framework, we propose request distribution policies for two scenarios: EPrice and GreenDC. In scenario EPrice, our framework optimizes for electricity cost based on time zone effects and price variability during the day. In scenario GreenDC, our framework considers green energy produced on-site at each data center location in

addition to timezone effects and temporal variability of the electricity price. We propose an optimization-based policy that defines the fraction of the clients' requests that should be directed to each data center. The front-ends periodically (e.g., once per hour) solve the optimization problem defined by the policy, using mathematical optimization algorithms, time series analysis for load prediction, and statistical performance data from data centers. After fractions are computed, the front-ends abide by them until they are recomputed. For comparison, we also propose a simpler heuristic policy that designed with the same goals and constraints as the other policies. This policy is greedy and operates quite differently. During each hour, it first exploits the data centers with the best power efficiency, and then starts exploiting the data centers with the cheapest electricity.

In chapter 5, we extend the framework from chapter 4 for scenarios in which caps are imposed on the brown energy that services can consume. Such caps would promote the conservation of brown energy, while encouraging services to power some of their data centers with renewable energy sources. We argue that placing caps on the brown energy consumption of data centers can help businesses, power utilities, and society deal with the large energy consumption and carbon footprint of their data centers.

Regardless of the capping scheme, the research question is how to create the software support for capping brown energy consumption[1] without excessively increasing costs or degrading performance. Given current world-wide efforts to mitigate climate change, this question is extremely timely. Moreover, this question has not been addressed before, since we are the first to propose and evaluate approaches to cap brown energy consumption.

Results from chapter 4 and chapter 5 show how our framework is very effective in allowing a service to abide by caps and SLAs, while trading off energy consumption and cost. Furthermore, in scenarios aimed at minimizing energy costs without caps, our results demonstrate our framework's ability to exploit variations in electricity prices and green energy both at and across data centers.

---

[1]The energy caps we study should not be confused with power caps, which are used to limit the "instantaneous" power draw of data centers.

Framework for HPC cloud services. In chapter 6, we build a load distribution framework for HPC cloud computing services. In particular, we consider services such as Amazon's EC2, which implement "infrastructure as a service" (IaaS). In IaaS services, users submit virtual machines (including the user's entire software stack) to be executed on the physical machines of the provider. Users are responsible for the management of their virtual machines, whereas the provider can potentially decide where the virtual machines should execute. IaaS services can host different user applications, ranging from interactive services to large high-performance computations. In this chapter, we focus on IaaS services that support HPC workloads (e.g., [11, 38]).

In this context, our study proposes policies for virtual machine placement and migration across data centers. The policies are executed by front-end devices, which select the destination for each job (possibly a collection of virtual machines) when it is first submitted. Later, a (possibly different) front-end may decide to migrate the (entire) job to another data center. We assume that users provide an estimate of the running time of each job. Using this estimate, the front-end can estimate the cost of executing the job at each data center. Specifically, our policies model the energy costs, peak power costs, the impact of outside temperature on cooling energy consumption, and the overhead of migration. For comparison, we also consider policies that are cost-unaware, such as Round Robin, and a policy that is cost-aware but static.

An important aspect of our load placement approach is that a change in electricity price at a data center (e.g., when the data center transitions from on-peak to off-peak prices) could cause a significant increase in its load in just a short time. As cloud service providers operate their data centers closer to the overheating point (when the servers' inlet air reaches an unsafe temperature) to reduce cost, such load increases can compromise the ability of the data centers to adjust their cooling before servers start to turn themselves off. This problem is exacerbated if providers turn off the water chillers (when not needed) to further reduce cost. Chillers typically incur significant delays to become fully effective for cooling when they are turned on. We study this effect and its implications for data center cooling and our policies. In particular, our policies predict changes in the amount of load offered to the data centers, and start "pre-cooling" them

if necessary to prevent overheating.

Our study of the load placement policies relies on event-driven simulations of a network of data centers using real workload traces. The results of this study show that our cost-aware policies achieve substantially lower costs than their simpler counterparts. Migration provides non-trivial cost benefits, as long as the amount of data to migrate is not excessive (in which case the policies decide not to migrate). Moreover, our results show that it is critical for our policies to consider their three cost components: server energy, peak power, and cooling energy. Finally, our results explore the impact of different values for our key simulation parameters, including different electricity prices and outside temperatures.

Framework for composing web services. Large-scale, value-added Internet services composed of independent cooperating or competing services will soon become common place. Several groups have addressed the performance, communication, discovery, and description aspects of these services. However, little work has been done on effectively composing paid services and the quality-of-service (QoS) guarantees that they provide. In Appendix A, we address these issues in the context of distributed file storage. In particular, we propose, implement, and evaluate a cost-effective, QoS-aware distributed file service comprising a front-end file service and back-end (third-party) storage services. Our front-end service uses mathematical modeling and optimization to provide performance and availability guarantees at low cost by carefully orchestrating the accesses to the back-end services. Experimental results from our prototype implementation validate our modeling and optimization. We conclude that our approach for providing QoS at low cost should be useful to future composite Internet services.

1.4 Contributions

This dissertation makes the following contributions:

- We observe that multiple data centers that are geographically distributed give many opportunities for cost and/or energy savings. Such opportunities stem from differences in electricity prices, time zones, climates, and access to green energy.

- We propose frameworks for Internet services and HPC cloud providers that operate wide area distributed computing platforms backed by multiple geographically distributed data centers to manage energy and energy-related costs.

- Based on these frameworks, we propose and evaluate a set of cost-aware job placement and migration policies that are designed to exploit the opportunities mentioned above to manage energy and energy cost while abiding by SLAs. Specifically, we propose optimization-based as well as simple and greedy heuristic-based load distribution policies. These policies rely on prediction of incoming load, outside temperature, electricity price, and statistical performance measurements of back-end data centers.

- We demonstrate the transient effect of large and rapid increases in load on the cooling of a data center.

- We evaluate the behavioral, energy, and cost implications of frameworks and policies extensively through simulation and real experimentation using real traces of workload, electricity prices, carbon prices, and outside temperature.

- We show how to apply load distribution to minimize costs in the context of composing independent paid third-party web services.

Chapter 2

Background

In this chapter, we give a brief overview about data centers, electricity markets, renewable energy, and the carbon market.

2.1 Data Centers

A data center is a facility used to house an enterprise's 1) IT equipment such as servers, telecommunication and storage systems, and 2) supporting infrastructures such as power delivery and cooling systems. Data centers range widely in size, from closet-sized rooms to warehouse-sized custom buildings. At the small side of the range, the data center typically involves only a few computers without any special infrastructure for cooling or power delivery. One may find these small data centers at universities or small enterprises. In the middle of the range, the data center typically comprises server, communication, and storage hardware mounted in racks, which are organized into rows as in Figure 2.1. The rows form aisles for either hot or cold air. These data centers involve significant cooling and power delivery infrastructures. One may find these medium data centers at larger enterprises or co-location facilities (also called "colos"). Colos host the software and possibly hardware of other enterprises. Finally, at the large end of the spectrum, a data center consists of many thousands of servers with extensive communication and storage infrastructures. They also include massive cooling and power delivery infrastructures. These data centers typically run large Internet services, such as Google, Microsoft, Yahoo, EBay, or Amazon. Because of their enormous scale, these data centers tend to use relatively homogeneous hardware and software. In this dissertation, we focus on medium and large-scale data centers.

Figure 2.1: CoreSite data center in Virginia.

According to [6] two thirds of US servers are housed in data centers smaller than 5,000 square feet and with less than 1 MW of critical power (power used for IT equipment). Most large data centers are "colos" and can support a critical load of 10–20 MW. Very few data centers today exceed 30 MW of critical capacity.

From the perspective of availability, data centers are classified into tiers [81, 166]. Tier 1 data centers have a single non-redundant power and cooling distribution path serving the IT equipment. This results in a typical availability level of 99.671%. Tier 2 specification requires the data center to have one extra redundancy for every component which improves the availability to approximately 99.741%. Tier 3 data centers have multiple independent power and cooling distribution paths (but only one is active) to the IT equipment, and N+2 component redundancy which provides redundancy even while components are in maintenance. This boosts the availability to around 99.982%. Tier 4 requires that, in addition to the IT equipment, all cooling equipment is powered by multiple independent paths. Fault-tolerant site infrastructure is also required for electrical power storage and distribution facilities. This results in a typical 99.995% availability. The boundaries between the tiers are blurry. Most commercial data centers

fall somewhere between tiers III and IV, trading off construction costs and reliability. Also, data center reliability strongly depends on the quality of the organization running the data center.

2.1.1 Data Center Efficiency

The most common metric used to determine the energy efficiency of a data center is power usage effectiveness, or PUE. This metric is defined as the ratio of the total power entering the data center divided by the power used by the IT equipment.

$$\text{PUE} = \frac{\text{Total Facility Power}}{\text{IT Equipment Power}} \tag{2.1}$$

The PUE of the average data center is notoriously high; according to a 2006 study [99], 85% of current data centers were estimated to have a PUE of greater than 3.0 and only 5% have a PUE of 2.0. Other studies [71, 165] are more promising, showing an average PUE of approximately 2.0 for a set of 22 data centers surveyed. Multiple sources of overhead contribute to high PUE values as shown in Figure 2.2. In a typical raised-floor data center, chillers contribute the largest fraction of overhead, typically 30–50% of IT power [19]. Next are computer room air conditioning (CRAC) units, consuming 10–30% of IT power (mostly in fans), followed by the UPS system, consuming 7–12% of IT power through AC–DC–AC conversion losses. Other facility elements, such as humidifiers, power distribution units (PDUs), and lighting, further add to higher PUE levels. Much of this poor efficiency is not because of limitations imposed by physics but rather by the lack of attention to efficiency. According to recent studies [6, 153], improving to the "state-of-the-art" practice can reduce the value of PUE to the range of 1.2–1.4.

Because the cooling system is a significant source of energy consumption in a data center, the next section is devoted to describing the cooling system in more details.

2.1.2 Data Center Cooling

The previous section has shown that the cooling system can consume as much as 45% of the energy used by a typical raised-floor data center [19]. For a 10 MW facility, this

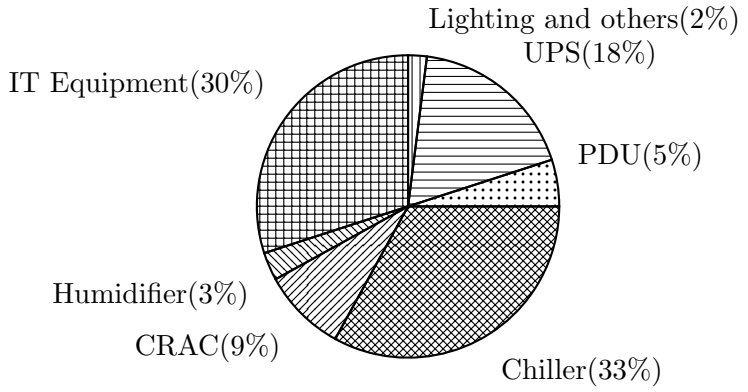


Figure 2.2: Energy entering the data center is broken into consumption areas such as server load and computing operations, cooling equipment, and power conversion and distribution [19].

would represent a cost of up to $3.9M annually, assuming an electricity price of $0.1 per kWh.

Figure 2.3 shows a simplified architecture of the cooling system including its main components: 1) computer room air conditioning (CRAC) units, 2) the chiller and 3) the cooling tower. As shown in the figure 2.2, the chiller power consumption dominates overall cooling system power usage.

The CRAC units supply cold air to server inlets by pumping chilled air into the under-floor plenum. This cold air is routed to the front of server racks through perforated tiles, then flows through the servers, and comes out as warm air in the back of servers. As mentioned, racks are arranged in alternate hot and cold aisles to avoid mixing hot and cold air because mixing causes inefficiency. The warm air from the back of the servers recirculates back to the intakes of the CRACs which consist of coils through which a liquid coolant (water/glycol) is pumped; fans push the warm air through these coils, thus cooling it. CRAC power consumption is dominated by fan power consumption, which grows cubically with the fan speed [126].

The chiller supplies chilled water or glycol by removing heat from the warm water return from CRACs. Using a compressor, this heat is transferred to a second water loop, where it is pumped to an external cooling tower where heat is released to the outside atmosphere. Chiller power consumption depends on the amount of heat extracted from

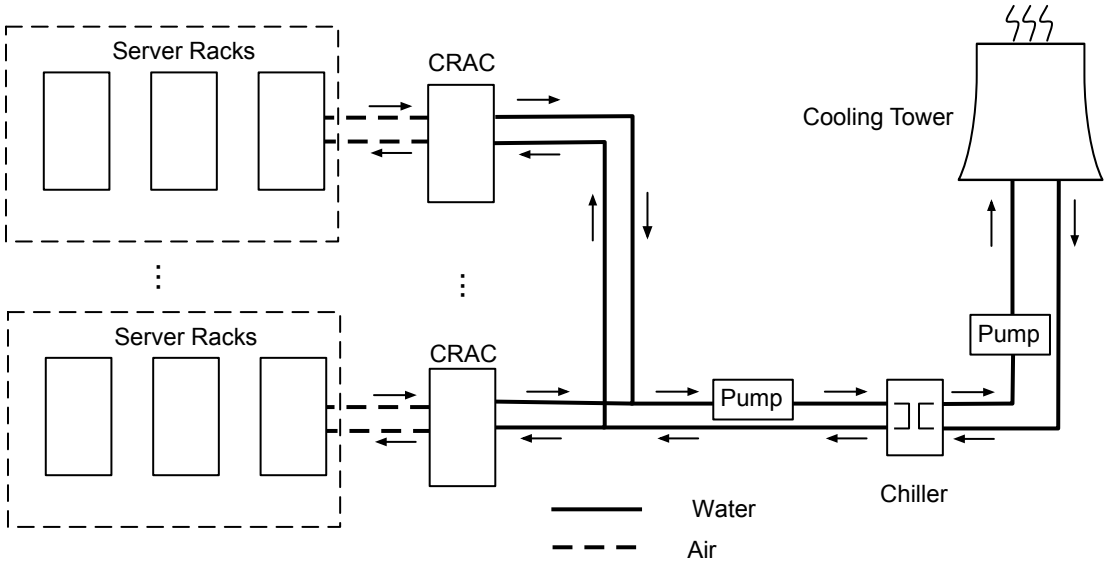


Figure 2.3: Data center with cooling infrastructure [126].

the chilled water return, the preset temperature of the chilled water supply to the CRACs, the water flow rate, the outside temperature, and the outside humidity. In current data centers, the water flow rate and chilled water supply temperature are held constant during operation.

2.1.2.1 Free Cooling

Free cooling is an economical way of leveraging low external temperatures to avoid the usage of a chiller in the cooling system. For example, under low ambient temperature, bypassing the chiller can give energy savings of up to 75% [131], without compromising cooling requirements. Free cooling can be either a fluid-side or air-side optimization [95,131,155]. An air-side system is a control mechanism that regulates the use of outside air to directly cool the data center room (outlet of CRACs). It utilizes a system of sensors, filters, ducts and dampers to control the volume and condition the quality of outside air to satisfy server inlet air standards. In contrast, fluid-side systems use the cold outside air to cool the water/glycol loop taking over the role of the chiller. This keeps the outside air out of the space and eliminates the need to condition that air.

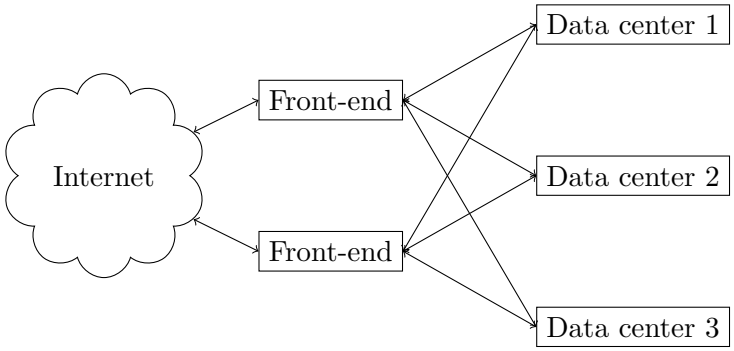


Figure 2.4: A typical multi-data-center architecture. Data centers 1,2, and 3 are the back-ends. The front-end devices are typically spread across multiple data centers.

2.2 Load Distribution

Large service providers typically operate multiple data centers. A typical architecture has front-ends and back-ends as shown in Figure 2.4. Front-ends inspect incoming clients' requests or jobs and redirect them to the appropriate back-end that can serve them. While there is a logical separation between front-end and back-end, physically front- and back-end can be in the same data center. The content of back-end data centers is usually mirrored, so that the service can achieve high performance and tolerate data center or network failures. The number mirrors is typically 2 or 3, because further replicating content would increase state-coherence traffic without a commensurate benefit in availability or performance. When the computation can be served within a short time interval, typically sub-second interval, like those of Internet services, we call it a *request*. This is the workload assumed in Chapter 4, 5 and Appendix A. When the load involves long running computations, typically in the order of minutes, hours or even days, we call it a *job*. The job placement policy is the topic of Chapter 6.

Current load distribution policies. Current load distribution policies over the wide area typically attempt to balance the load across mirror data centers, minimize the response time seen by clients, and/or guarantee high availability, e.g. [31, 141, 170]. For example, round-robin DNS [31] attempts to balance the load by returning the address of a different front-end for each DNS translation request. More interestingly, Ranjan *et al.* [141] redirect dynamic-content requests from an overloaded data center to a less

loaded one, if doing so is likely to produce a lower response time. In contrast, request distribution over the local area (i.e., within a data center) may promote load balancing, cache locality, high throughput and availability, and/or enable energy or temperature management, e.g. [33, 78, 79, 122, 129, 178].

2.3 Electricity Cost Components and Their Dynamics

2.3.1 Cost of Energy

Utilities typically offer their customers three types of contracts with price expressed in dollar per KWh: 1) fixed, 2) time of use (TOU), and 3) dynamic pricing. Under fixed pricing, a customer pays a fixed price throughout the year. This is typical to most home users. Under TOU, price varies with time of day. For example, there might be two prices, one is from 8 am to 8 pm of weekdays (this is called on-peak prices), and one for all other times (called off-peak prices). This scheme is based on the observation that the electricity demand during the day on workdays is typically higher than during other time periods. However, TOU fails to match the price of electricity with the actual the demand and supply for a given time in the day. The dynamic pricing schemes address this problem by setting prices based on supply and demand for electricity. There are three types of dynamic pricing schemes [135] depending on how far ahead the price is set: 1) day-ahead pricing, 2) hour-ahead (or 30 minutes in the UK); and 3) real-time pricing (prices are set only 5 minutes ahead).

In the dynamic schemes, the electricity price can vary significantly over the course of a day. For example, during the summer, one may see a 10-fold difference in the price of electricity from its lowest point in the early morning hours to its peak point in the middle of the afternoon [161]. Similarly, electricity demand and prices also vary significantly over larger time granularities, e.g. monthly and seasonally. As illustrated in [133], winter electricity prices show morning and evening peaks rather than summer's more pronounced afternoon peak.

Because of these substantial price variations, large electricity consumers (typically large businesses) often use dynamic pricing, instead of paying a fixed price. For example,

businesses using day-ahead pricing receive a schedule from the power utility listing the next day's expected prices. Although the actual day-of-consumption prices usually differ from the predicted prices, the predicted trends are typically accurate enough for businesses to schedule their electricity consumption intelligently and save money. In fact, they can also place their data centers in distant time zones so as to create greater opportunities for such savings.

The fact that electricity prices vary substantially during the day (because of dynamic pricing) and across geographic regions (because of different in demand and supply and/or time zone effect) means that intelligently routing computation to where electricity is cheap can help businesses significantly reduce their electricity costs.

2.3.2 Peak Power Costs

Peak power demand cost is the cost paid for the peak power drawn at any particular time. Even though it has frequently been overlooked, peak power cost can be significant because the peak power draw impacts the required generation capacity as well as the power distribution grid.

The charge for peak power drawn is typically expressed as dollar per KW for the maximum power demand within some *accounting period* (e.g., 1 month). Many utilities track peak power by tracking average power draw over 15-minute intervals. Thus, the 15-minute interval with the highest average power drawn determines the peak power charge for the accounting period.

Govindan *et al.* estimate that this component can grow to as high as 40% of the electricity cost of a data center [69]. This high cost suggests that businesses can reduce their electricity costs substantially by carefully managing their peak power draw.

2.4 Current and Future Renewable Electricity Costs

To avoid consuming brown electricity, businesses can use renewable or green energy such as solar, wind, geothermal, and biomass. In this dissertation, we primarily focus on wind and solar power, which have great potentials and whose use is currently limited

by availability and/or cost. However, our framework is not intrinsically restricted to these sources.

World-wide electricity-generating capacity from wind and solar power has been increasing exponentially over the last 20-30 years [29]. With larger production and competition, green electricity costs are coming down fast. For example, wind-generated electricity cost $0.38 per KWh in the 1980s. Today, it has dropped to $0.04 per KWh or lower at some sites [29].

Nevertheless, the overall costs from these sources are still higher than those of fossil fuels, because of the capital outlay for initial construction or installation. For example, the cost to build a wind farm is approximately $1.3 million to $1.7 million per MW, compared to a cost of $800,000 per MW for gas-fired plant [57]. As power utilities that produce and distribute green energy start to appear, these costs will be amortized over a large user base. Even before that happens, most US states now have a wide variety of economic incentives for customers to purchase/build their own installations. These incentives can cut capital costs in half in California, for example. Looking ahead, it is clear that these newer sources of energy will become even more appealing, especially as their efficiencies and costs continue to improve. In fact, as governments become more aggressive at seeking to curb carbon emissions (perhaps via carbon taxes or brown energy caps), we envision a rich mix of green and brown electricity options being affordable and viable in the next several years.

One can have access to renewable energy in a few different ways. First, one can build a solar farm or wind mills on-site. In the extreme case, data centers would be powered only by these green sources [7]. The drawback of this approach is that when green energy is not available, the data center cannot function. Therefore, data centers will likely be supplemented with grid power in addition to on-site green production. This approach is assumed in Chapter 4. Another way to access the green energy is by requesting a mix of energy from the utility that combines both green and brown energy. We use this approach in Chapter 5.

2.5 Carbon Market

Carbon cap-and-trade is a market-based approach used to control carbon emission by providing economic incentives for achieving reductions. A central authority (usually a governmental body) sets or allocates a limit on the amount of carbon that can be emitted in the form of emissions permits. These represent the right to emit a specific volume of carbon. Enterprises are required to hold a number of permits equivalent to their emissions. If an enterprise needs to increase its emission, it must buy permits from those whose permits exceed their emission. In effect, the buyer is paying a charge for polluting, while the seller is being rewarded for having reduced emissions. The largest of such programs is the European Union Emission Trading Scheme.

In cap-and-trade scenarios, our policies make decisions based in part on the market price of carbon offsets. To illustrate how this market behaves and the feasibility of basing decisions on it, Figures 2.5–2.7 show the price (in Euros) of 1 ton of carbon in the futures market for December 2008 at different time-granularities. Figure 2.5 depicts how the December 2008 futures price fluctuated on a single day: April 3, 2008. Figure 2.6 depicts the December 2008 futures price as collected during the entire week of March 31, 2008. Figure 2.7 depicts the December 2008 futures price as collected during the month of December 2007. The data plotted in the figures was obtained from [130].

Although there can be periods of variability, these figures show that the overall increasing or decreasing price trends remain clear, even at relatively short time scales. In addition, although the graphs span relatively short periods, they show exploitable price variations—on the order of 10%. Finally, we can see that it typically takes at least a few hours for prices to change appreciably.

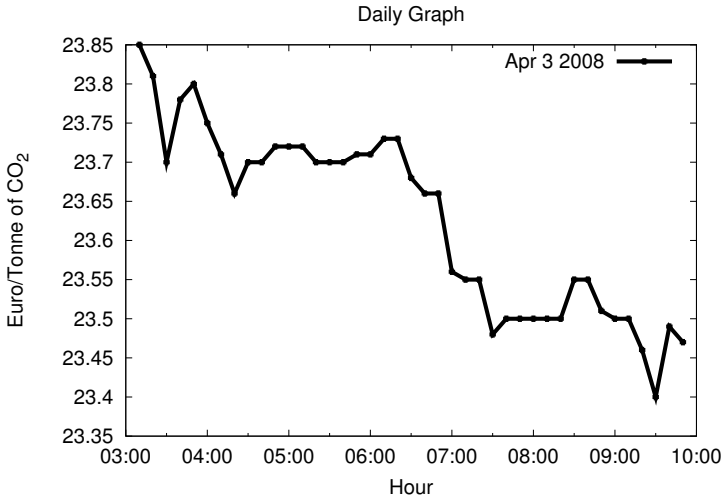


Figure 2.5: Prices on April 3, 2008. Decreasing trend.

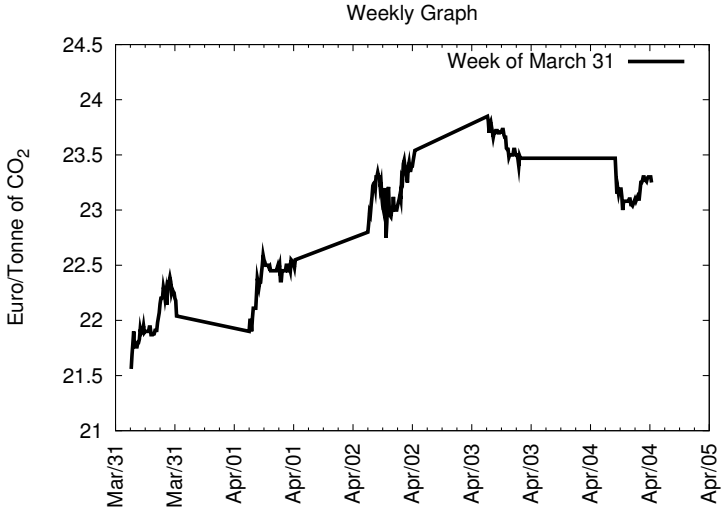


Figure 2.6: Prices on week of March 31, 2008. Increasing trend.

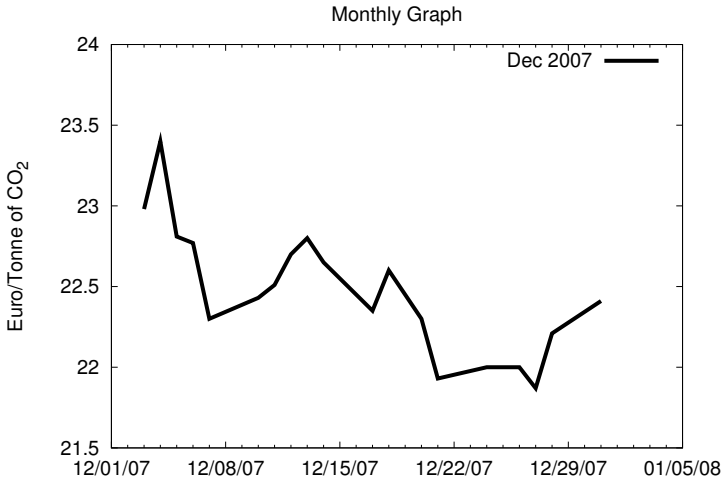


Figure 2.7: Prices in Dec. 2007. Decreasing trend.

These observations suggest that the market has enough variability to make dynamic decisions worthwhile. Our framework and policies enable services to take full advantage of price variations. Moreover, the market's stability suggests that market-based decisions will be meaningful; excessive variability or the absence of clear trends could render our decisions inadequate in just a short time. Nevertheless, if a period of instability is ever detected, our policies can be tuned to mitigate the impact of the instability on the request distribution.

Chapter 3

Related Work

In this chapter, we discuss relevant works in the areas of data center energy, power, and thermal management, including attempts to use renewable energy sources. After that, we discuss the related works on geographical (multi-data-center) load balancing, the main mechanism we use to manage energy and energy-related costs.

3.1 Conserving Energy in Data Centers

A large body of prior work has focused on conserving energy in servers and data centers. Many of these works focused on creating and/or leveraging components' low-power states, such as those for the CPU [51, 53, 87, 110, 138, 154], memory [45–47, 49, 53], disk [32, 74, 127], and network interface [1, 80]. Other works focused on creating and/or leveraging entire-server low-power states [5, 13, 108].

Despite all of these works, commercial servers are still not energy-proportional [20], i.e. even when completely idle, they consume a large fraction of their peak power. Even worse, these servers do not yet include many of the ideas proposed in the prior papers.

For these reasons, a different research direction has been to couple within-data-center load distribution with widely available low-power server states, such as completely off or ACPI's S3 state [2]. The idea is to concentrate the load (e.g., interative requests, long-running jobs, virtual machines) onto a subset of the servers and transition others to the low-power state. This idea was originally called Load Concentration (LC) [128, 129], but has since been called "server consolidation", among other less-common names. Many works have considered LC or variations/extensions of it [34–36, 51, 79, 82, 100, 108, 109, 137, 145, 147, 163].

To avoid requiring LC, some recent works have also considered using low-power, laptop-style servers in data centers [14, 85, 97, 99].

Our techniques are orthogonal to all of these works, as we consider multi-data-center mirrored services and their load distribution, rather than single data center environments. Moreover, this dissertation differs from previous works in that 1) none of them considered the goal of capping energy consumption; and 2) none of them considered green energy or carbon market interactions. Nevertheless, we utilize some of these techniques to manage energy consumption at each data center in our multi-data-center setups. For example, we assume LC is in effect within each data center.

3.2 Capping Power Consumption in Data Centers

Many previous works [54, 55, 69, 140, 171] considered power capping of data centers. [55, 140] used DVFS at each server and a cluster-wide controller to limit the power consumption of the entire system. More recently, Fan et al. [54] showed that there is a large difference between the actual power usage and the theoretical peak at real data centers. Govidan et al. [69] proposed using UPS batteries to shave peak power. Our power capping technique is orthogonal to these works because we leverage the geographical distribution of data centers and migrate workload between data centers to reduce peak power. Nevertheless, we can utilize some of these techniques to manage power consumption at each data center in our multi-data-center setups.

Importantly, note that our problem of capping energy (more specifically capping brown energy) is qualitatively different than capping power. In particular, capping power enables cheaper cooling and packaging, and adding more hardware to a system without harming it. However, it may not conserve energy, if obeying the cap causes a significant increase in running time [49]. In contrast, capping brown energy can conserve brown energy and promote green energy.

3.3 Data Center Thermal Management

Prior work has considered two classes of thermal management policies for data centers: those that manage temperatures under normal operation [112, 113, 115, 151] and those that manage thermal emergencies [37, 56, 78, 139, 173].

The works that focused on normal operation target reducing cooling costs. However, they make the handling of sudden large increases in load even more difficult. The reason is that these works enable operation with higher inlet temperatures at the servers; leading to less slack in the available cooling when the load increase comes.

In contrast, the main goal of policies for managing thermal emergencies has been to control temperatures while avoiding unnecessary performance degradation. A large increase in load that causes temperatures to rise quickly can be considered a thermal emergency. However, none of these previous techniques can tackle major change in load caused by migration without serious performance degradation.

Other works have looked at various ways to improve the cooling efficiency at different parts of the cooling system. At the scheduling level, jobs can be placed to avoid hot-spots [112, 113] or to improve airflow [18]. Parolini et al. [124, 125] presents a unified approach thermal management approach based on a modeling framework involving server temperatures and CRAC speed settings at different workload utilization level. Others have looked at using free cooling [95,131] to reduce chiller energy consumption or using in-row cooling [21] to reduce hot recirculation and enable variable speed CRACs.

Our work differs from these works in many aspects: 1) none of them considered multi-data-center environments and migration computation to where the outside temperature is low to reduce cooling energy; and 2) none of them considered the cooling transient effect as a result of load migration.

3.4 Exploiting Renewable Energy in Data Centers

There have been some works that addressed the use of green energy in data centers to lower brown energy consumption, monetary costs, and environmental impact [7, 8, 16, 63, 64, 66, 67, 93, 94, 98, 101, 102, 104, 144, 150].

Most of these efforts [8, 16, 63, 64, 66, 67, 93, 94, 98, 102, 144, 150] targeted single data center environments and tried to use as much green energy as possible. Ren et al. [144] proposed an optimization framework to evaluate various ways to incorporate renewable energy: self-generation, off-site generation feeding into the grid, power purchase agreements feeding into the grid, and renewable energy certificates. They found that both self-generation and off-site generation can lower costs and carbon footprints. In addition, they found self-generation to be the best approach when carbon reduction targets are moderate (up to 30%), especially due to its ability to reduce peak grid power cost. Goiri et al. [66] discuss the space requirements and capital costs of self-generation with wind and solar, now and in the future. Liu et al. [102] present a framework to integrate IT workload, energy supply, and cooling management in the context of a single data center. The key component of the approach is "demand shifting", which schedules non-critical IT workloads and allocates IT resources within a data center according to the availability of green power and the efficiency of cooling. Blink [150] proposed to adjust the duty cycle of the servers according to the availability of green energy. In contrast with these higher level approaches, SolarCore [98] is a multi-core power management scheme designed to exploit PV solar energy. SolarCore focuses on a single server, so it is closer to the works that leverage green energy in embedded systems.

Other works have considered multi-data-center setups [7, 58, 101, 103, 104, 156, 179]. In a short position paper, Stewart and Shen [156] discussed how to maximize green energy use in data centers [156] and briefly touched on multi-data-center scenarios. Liu et al. [101, 104] studied a global scheduler that could maximize the green energy consumption given dynamic electricity pricing. [103] explored how to power an existing network of data centers (almost) entirely with renewables and UPS batteries. Akoush et al. [7] focused on load migration between data centers that are at remote locations and powered only by renewable energy. Forte [58] addresses the problem of minimizing carbon emission using request distribution. This paper approached the carbon emission problem from a different angle than our work by addressing the temporal and spatial aspect of carbon emission. More specifically, they argued that 1) electricity producing

at different places contains different amount of carbon, and 2) during peak hours, peak-generators are turned on to meet the demand, thus increase carbon emission temporally. GreenWare [179] tried maximize the percentage of renewable energy used to power a network of distributed data centers, subject to the desired cost budgets of Internet service operators.

Our work on exploiting green energy differs from these efforts in two key ways. First, we focus on multi-data-center settings, whereas most other efforts focused on single data center scenarios. Second, all but one of the multi-data-center papers [156] came after our work, and extended it in a few ways. Specifically, Liu et al. [103, 104] went deep into the modeling, algorithms, and optimality of the load distribution with and without green energy. [7, 103] explored options of data centers powered only with green energy and battery. Forte [58] studied the temporal and spatial variation of carbon emission. GreenWare [179] solved a slightly different problem by having the cost function as a constraint rather than part of the objective function.

3.5 Load Distribution Policies

The goal of load distribution is to map clients' requests to a data center that can serve it. This can be done by a third party entity or by the service itself. The simplest algorithm is DNS-based where third-party authoritative DNS servers resolve name requests from client according to an appropriate statistical model [28]. This model is adopted by Akamai and most CDNs. Donar [174], a decentralized third-party replica-selection service, provides an API for services to specify a wide range of policies such as closest replica, load-balancing, optimizing network cost and shifting traffic to under-utilized replicas. Alternatively, the mapping can be done at services' ingress proxies which is adopted by large Internet services such as Google, Yahoo and Microsoft. The advantage of this approach is that the load distribution entity has all the knowledge of the peer data centers since they belong to the same organization, which could lead to much more sophisticated load distribution policies. Although some existing services do exploit aspects of geographic distribution to a limited extent (i.e., they route requests to the data center closest to the user, and route requests away from slow or failed data centers),

our approach is unique in 1) its focus on electricity cost reduction within performance constraints, 2) its basis on formal optimization techniques, and 3) its aggressive distribution policies leveraging electricity price variations, green energy, brown energy caps, and carbon markets.

Recently, there have been a few works on the topic of load distribution in Internet services [52, 123, 134, 135, 142, 149] to address different aspects of geographical distribution such as electricity prices. Unfortunately, Rao [142] and Shah [149] did not consider network latencies, realistic SLAs, or time-varying workloads. Moreover, these studies did not simulate or implement their proposed request distribution schemes or any competing heuristics. Also, none of these works considered brown energy caps, market interactions, request types, or power mixes. Qureshi's work is closest to our own. He first [134] considered variable electricity prices for each data center (the price varies on an hourly basis) and proposed to shut down entire data centers when their electricity costs are relatively high. More recently, Qureshi et al. [135] also studied dynamic request distribution based on hourly electricity prices. Our work differs from Qureshi's work in many ways: (1) we built a real distributed implementation for experimentation and simulator validation, whereas Qureshi relied solely on simulations; (2) we introduce brown energy caps (and carbon market interaction), whereas Qureshi did not consider caps or attempt to conserve energy; (3) we minimize costs based on caps, green energy, electricity prices, outside temperature and carbon markets, whereas Qureshi focused solely on hourly electricity prices; (4) we use statistical response time information from data centers to abide by SLAs, whereas Qureshi used a simple radius metric to determine the set of data centers to send requests to; and (5) we demonstrate that optimization easily outperforms heuristics such as those used by Qureshi. Contributions (2), (3), and (4) mandate significant changes in the load distribution approach and also amplify the benefits of optimization-based techniques over heuristics.

Recently, there have been some works [52, 123] that considered the use of UPS batteries at the data centers to reduce electricity-related costs. However, they did not consider green energy, brown energy caps, or carbon market interactions.

Finally, besides Internet services, our work addresses job (virtual machine) placement and migration in multi-data-center cloud environments as well. There have been a few works on cost-aware load placement in grid and federated cloud scenarios [30,59,65]. Unfortunately, these works did not consider energy prices, peak power costs, or any cooling issues. There have also been some works on job migration in grid environments [96,111]. These efforts focused mainly on migrating a job to a site with available resources or to a site where execution can be shorter. They did not consider costs of any kind, energy consumption, or cooling. More importantly, these previous works did not consider the transient effect of the large and rapid increases in load they may cause for certain data centers.

In [26, 77, 176], techniques for migrating virtual machines over the wide area were outlined. These works focused mostly on the mechanics of virtual machine migration. We leverage their techniques for migrating jobs in our policies

Chapter 4

Managing Energy and Costs for Internet Services

This chapter proposes and evaluates a framework for optimization-based request distribution that enables multi-data-center Internet service providers to manage their energy consumption and costs, while respecting their SLAs. It also allows services to take full advantage of the geographical locations of distributed data centers. Specifically, our framework exploits data centers that pay different electricity prices (the pricing scheme can be either fixed, TOU or dynamic as explained in Chapter 2), data centers located in different time zones' and data centers that are powered by renewable energy sources. At the same time, the framework considers the existing requirements for high throughput and availability. Based on the framework, we propose request distribution policies for two scenarios: EPrice (which leverages temporal electricity price variability) and GreenDC (which leverages on-site green electricity generation). We propose an optimization-based policy which uses mathematical optimization algorithms, time series analysis for load prediction, and statistical performance data from data centers. We also propose a greedy heuristic designed with the same goals and constraints as the optimization-based policy.

We evaluate these policies using real electricity prices, network latencies and a day-long trace from a commercial service. Our results show that the optimization-based policies can accrue substantial cost reductions by intelligently leveraging time zones and hourly electricity prices. The results also show that we can exploit green energy to achieve significant reductions in brown energy consumption for small increases in cost.

Symbol	Meaning
$f_i(t)$	% requests to be forwarded to center i
Overall Cost	Total energy cost ($)
$Cost_i(t)$, $c_i(t)$	Avg. cost ($) of a request at center i
$c_i^{green}(t)$	Avg. cost ($) of a request at center i using green energy
$BCost_i(offered_i, t)$, $b_i(offered_i, t)$, $b_i^{green}(offered_i, t)$	Base energy costs ($) of center i under $offered_i$ load
GE_i	Amount of green energy that green center i can consume
$GEC_i(t)$	Total amount of green energy that green center i has consumed up to time t
LC_i	Load capacity (reqs/sec) of center i
LR(t)	Expected peak service rate (reqs/sec)
LT(t)	Expected total service load (#reqs)
$offered_i$	LR(t) times $f_i(t)$ (reqs/sec)
$CDF_i(L, offered_i)$	Expected % requests that complete within L time, given $offered_i$ load

Table 4.1: Framework parameters. (t) represents time.

4.1 Framework

Recall from Chapter 1 that we assume that a front-end is chosen to first handle a client request via round-robin DNS or some other high-level policy. The front-ends execute one of our policies and forward each request to a data center that can serve it. The reply is sent to the original front-end, which in turn forwards it to the client.

4.1.1 Principles and Guidelines

For our policies to be practical, it is not enough to minimize energy costs; we must also guarantee high performance and availability. Our policies respect these requirements by having the front-ends: (1) prevent data center overloads; and (2) monitor the response time of the data centers, and adjust the request distribution to correct any performance or availability problems.

We assume that the service has a single SLA with its customers, which is enforced on a daily basis, the "accounting period". The SLA is specified as (L, P), meaning that at least P% of the requests must complete in less than L time, *as observed by the front-end devices*. The SLA guarantee provided by our policies and framework can be

Minimize:

$$Overall\ Cost = (\sum_t \sum_i f_i(t) \times LT(t) \times Cost_i(t)) + (\sum_t \sum_i BCost_i(offered_i, t)) \tag{4.1}$$

Policy EPrice:

$$\text{Cost}_i(t) = c_i(t) \tag{4.2}$$

$$\text{BCost}_i(offered_i, t) = b_i(offered_i, t) \tag{4.3}$$

Policy GreenDC:

$$\text{Cost}_i(t) = \begin{cases} c_i^{green}(t) & \text{GEC}_i(t) \leq \text{GE}_i \\ c_i(t) & otherwise \end{cases} \tag{4.4}$$

$$\text{BCost}_i(offered_i, t) = \begin{cases} b_i^{green}(offered_i, t) & GEC_i(t) \leq GE_i \\ b_i(offered_i, t) & otherwise \end{cases} \tag{4.5}$$

Constraints:

$$\begin{array}{lcl} 1.\ \forall t \forall i\ f_i(t) \geq 0 & \Rightarrow & \textit{i.e., each fraction cannot be negative.} \\ 2.\ \forall t \sum_i\ f_i(t) = 1 & \Rightarrow & \textit{i.e., the fractions for each request type need to add up to 1.} \\ 3.\ \forall t \forall i\ (f_i(t) \times LR(t)) \leq LC_i & \Rightarrow & \textit{i.e., the offered load to a data center should not overload it.} \\ 4.\ \frac{\sum_t \sum_i (f_i(t) \times LT(t) \times CDF_i(L, offered_i))}{\sum_t LT(t)} \geq P & \Rightarrow & \textit{i.e., the SLA must be satisfied.} \end{array} \tag{4.6}$$

combined with Internet QoS approaches to achieve end-to-end guarantees [180].

Note that the SLA definition implies that *the service does not need to select a front-end device and data center that are closest to each client for the lowest response time possible*; all it needs is to have respected the SLA at the end of each accounting period.

In addition, note that our framework does not address short-term variations in green energy availability. Instead, we assume that each data center has a daily supply of green energy (the amount on each day may be different) produced locally and stored either in batteries or in the electrical grid.

Finally, we assume that each data center reconfigures itself by leaving only as many servers active as necessary to service the expected load for the next hour (plus an additional 20% slack for unexpected increases in load); other servers can be turned off, as in [34–36, 128].

4.1.2 Optimization-Based Distribution

Our framework comprises the parameters listed in Table 4.1. Using these parameters, we can formulate optimization problems defining the behavior of our request distribution policies. The optimization seeks to find the fraction $f_i(t)$ of requests that should be sent to each mirror data center i, during "epoch" t. (An epoch is defined as a period of fixed fractions. There can be many epochs during a single accounting period.) The next subsection describes two specific optimization problems (policies). Section 4.1.2.2 describes the instantiation of the parameters. Section 4.1.2.3 discusses how to solve the problems.

4.1.2.1 Problem Formulations

Policy EPrice: Leveraging time zones and variable electricity prices. The $f_i(t)$ fractions should minimize the overall energy cost, *Overall Cost*. Equation 4.1 defines *Overall Cost* with two additive components. The first represents the energy cost of processing the client requests that are offered to the service. The second represents the "base" energy cost, i.e. the cost of the energy that is spent when the active servers are idle. In the EPrice policy, the per-request $Cost_i$ and the base energy cost $BCost_i$ have trivial definitions (Equation 4.2 and 4.3). *Overall Cost* should be minimized under the constraints that follow the equations.

Policy GreenDC: Leveraging data centers powered by green energy. The formulation above does not distinguish data centers based on their energy source. However, we expect that data centers will increasingly often be located near sources of green energy, such as wind and solar farms. In this scenario, the same service could have some data centers that are powered by brown energy (brown data centers), and others that are powered by green energy (green data centers). Because the supply of green energy may not be enough to power a data center throughout the entire period, green data centers must also be connected to the regular electrical grid.

To formalize this scenario, we can redefine $Cost_i$ and $BCost_i$ for the green data centers as in the GreenDC policy (Equation 4.4 and 4.5), where GE_i is the amount

of green energy that green center i can consume during the accounting period. The definitions of $Cost_i$ and $BCost_i$ for brown data centers stay the same as before.

Other options. We have not yet explored services with session state, i.e. soft state that only lasts a user's session with the service. In such services, the distribution is constrained since all requests of a session must be sent to the same data center. Nevertheless, it is easy to extend our work to handle sessions by (1) estimating the average number of requests per session; and (2) computing fractions that guide the distribution of the first request of a session. It is also fairly easy to handle (1) requests that involve writes to persistent state and (2) multiple request types, instead of averaging across all types like we do now. We will explore these issues in our future work.

4.1.2.2 Instantiating Parameters

To instantiate the parameters of our formulations exactly, the front-ends would have to communicate and coordinate their decisions. To avoid these overheads, we explore a simpler approach in which the optimization problem is solved independently by each of the front-ends. If the front-ends guarantee that the constraints are satisfied from their independent points of view, the constraints will be satisfied globally.

In this approach, $LT(t)$ and $LR(t)$ (and consequently $offered_i$) are defined for each front-end. In addition, the load capacity of each data center is divided by the number of front-ends. To instantiate CDF_i, each front-end collects the recent history of response times of center i when the front-end directs $offered_i$ load to it. For this purpose, each front-end has a table of these <offered load, percentage> entries for each data center that is filled over time. Similarly, we create a table of <offered load, base energy cost> entries to instantiate $BCost_i$.

4.1.2.3 Solving the Optimization Problem

The solution for an entire accounting period provides the best energy cost. However, such a solution is only possible when we can predict future load intensities, electricity prices, and data center response time distributions (CDF_i). Electricity price predictions are trivial when the price is constant or when there are two prices. When prices vary

Characteristic	Optimization (EPrice & GreenDC)	CA-Heuristic
Accounting period	1 day	1 day
Epoch length	4 hours	1 hour
Load predictions	Per front-end for entire day	Per front-end for next hour
Energy price predictions	Entire day	Next hour
Recomputation decision	Epoch boundary	Epoch boundary
Communication with DCs	Yes	Yes

Table 4.2: Main characteristics of distribution approaches.

hourly, we can use the day-ahead prediction that is provided by the utility for each day [12]. Typically, these day-ahead prices are reasonably good predictions of actual prices. For predicting load intensities, we consider Auto-Regressive Moving Average (ARMA) modeling [25]. We do not attempt to predict CDF_i. Instead, we assume the current CDF_i tables as predictions.

We cannot use fast Linear Programming (LP) solvers, because solving for an entire day at once involves a few non-linear functions (e.g., $BCost_i$ and CDF_i). Instead of LP, we use Simulated Annealing [88] and divide the day into six 4-hour epochs, i.e. $t = 1..6$. We will consider running an LP solver every hour in our future work.

Because the assumptions/predictions that we make/use when computing a solution may become invalid/inaccurate over time, we must check for deviations. If there is any significant deviation at an epoch boundary, we recompute the solution. We must also recompute if a data center becomes unavailable (or, in our second formulation, the green energy expires at a data center). In practice, recomputations occur at the granularity of multiple hours.

After a recomputation and every hour, the front-ends inform the data centers about their predicted loads for the next hour, so that they can reconfigure. The "Optimization" column of Table 4.2 summarizes our approach.

4.1.3 Heuristics-Based Request Distribution

We also propose a heuristic policy (CA-Heuristic) that is still cost-aware, but is simpler and less computationally intensive than the optimization-based approach. The heuristic

is greedy and uses 1-hour epochs. At each epoch boundary, each front-end computes $R = P \times E$ (the number of requests that must have lower latency than L), where E is the number of requests the front-end expects in the next epoch. E can be predicted using ARMA for each front-end. Each front-end also orders the data centers that have $CDF_i(L, LC_i) \geq P$ according to the ratio $Cost_i(t)/CDF_i(L, LC_i)$, from lowest to highest ratio. The remaining data centers are ordered by the same ratio. A final list, called *MainOrder*, is created by concatenating the two lists of data centers.

Requests are forwarded to the first data center in MainOrder until its capacity is met. At that point, new requests are forwarded to the next data center on the list and so on. After the front-end has served R requests in less than L time, it can disregard MainOrder and start forwarding requests to the cheapest data center (lowest $Cost_i(t)$) until its capacity is met. At that point, the next cheapest data center can be exercised and so on.

If the prediction of the number of requests to be received in an epoch consistently underestimates the offered load, serving R requests within L time may not be enough to satisfy the SLA. To prevent this situation, whenever the prediction is inaccurate, the heuristic adjusts the R value for the next epoch to compensate.

At each epoch boundary, the front-ends inform the centers about their predicted loads for the next epoch. The last column of Table 4.2 overviews our heuristic.

4.2 Evaluation

4.2.1 Methodology

We implemented a simulator of a large Internet service. For simplicity, we simulate a single front-end located on the East Coast of the US. The front-end distributes requests to 3 data centers, each of them located on the West Coast, on the East Coast, and in Central Europe.

Request trace. We use a *real request trace from a commercial search engine*, Ask.com. The trace includes roughly 17 million requests, a representative fraction of the requests the engine processes during a day. Figure 4.1 shows the 90th percentile of the actual and

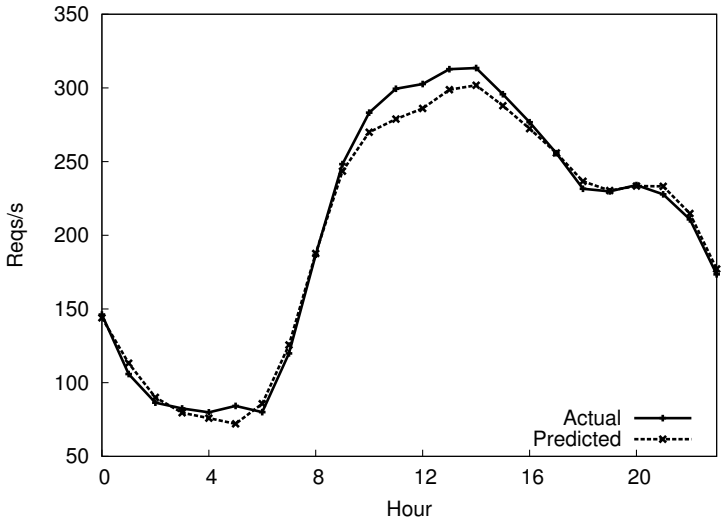


Figure 4.1: Actual and predicted load intensities.

Data Center	Brown energy (cents/KWh)	Green energy (cents/KWh)	Capacity (reqs/s)
DC 1 (West US)	11.1	15.0 (solar)	125
DC 2 (East US)	11.7	—	215
DC 3 (Europe)	9.7	8.0 (wind)	125

Table 4.3: Default simulation parameters. Capacities have been scaled down to match our request trace.

ARIMA-predicted request rates during the day. Our ARIMA modeling uses 24 auto-regressive parameters (corresponding to a full day), 24 moving average parameters, and 1 level of integration. The figure shows that the ARIMA predictions are accurate and only deviate noticeably from the actual rates during the busiest part of the trace.

To generate a realistic distribution of data center response times, we installed a simple service on 3 PlanetLab machines. When a request is received, the service replies with a local, randomly chosen block of 8 KBytes. The requests were made from a machine at Rutgers, according to a Poisson process with mean inter-request time of 1 second. For each machine (data center), we then have a *pool of response times*. The response times mimic those that would be seen by a front-end device placed at Rutgers. To each response time, we add a request processing time (discussed below).

However, this initial pool of response times for each data center is not enough. It

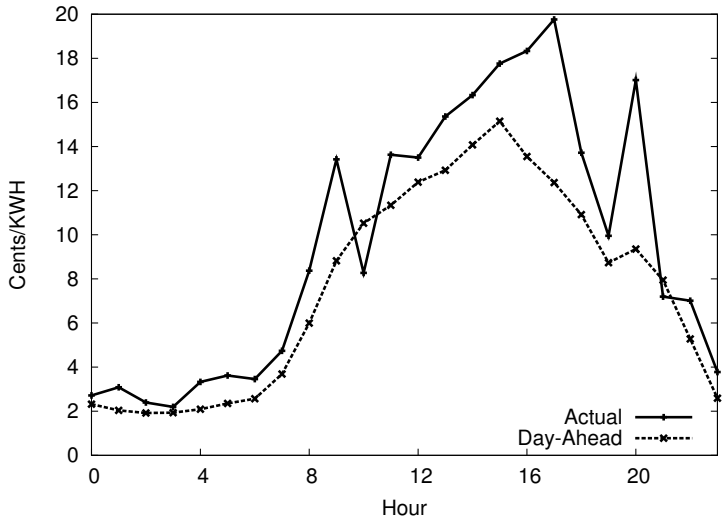


Figure 4.2: Actual and day-ahead brown electricity prices.

represents the response times that the data center would exhibit under a very light offered load. Thus, we generated three more pools for each data center, each representing the response times under a different range of offered load. The ranges split the maximum throughput of a data center into four equal parts. The initial pool corresponds to the lowest load range, whereas the fourth pool corresponds to the highest. To generate the response times of each pool, we simply added additional latency to the response times of the preceding range. Specifically, we increase the latency by 5% from one range to the next, to reflect increased network contention around the data center.

When simulating the request trace, the simulated front-end decides which data center should receive each request (according to one of our policies) and, based on the current average load the data center has been recently receiving, randomly selects a response time from the corresponding pool. By default, the brown data center is closest to the front-end in terms of network latencies; it is assumed to be in the East Coast of the US with the front-end. The wind data center is furthest away from it, somewhere in Central Europe (6 hours later). The solar data center is assumed to be in the West Coast of the US (3 hours earlier).

Electricity prices, sources, and time zones. We simulate schemes with one electricity price, two prices (on/off peak), and hourly prices. For the on/off-peak scheme

(On/Off), off-peak hours are from 9pm to 7am. For the hourly scheme (Dynamic), Figure 4.2 shows the day-ahead and actual brown electricity prices we use [12]. The day-ahead prices predict trends fairly accurately, but not absolute prices. To mimic different brown electricity prices for each data center, we simply shift our default prices 3 hours earlier or 6 hours later. To make all pricing schemes comparable, the prices for the constant and on/off-peak schemes are computed based on the real prices in Figure 4.2. When we consider green data centers, their electricity price is always assumed constant. We also assume that the amount of green energy available daily at each green site is enough to process 25% of the requests in the trace. The constant brown and green prices are listed in Table 4.3.

Other parameters. We assume that a request consumes 60 J of dynamic energy to process by 2 machines, including cooling, conversion, and delivery overheads. This is equivalent to consuming 150 W of dynamic power per machine during request processing. By default, we study machines that are fully energy-proportional [20], i.e. they consume no base energy. In addition, we study the impact of this assumption. The SLA requires 90% of the requests to complete in 700 ms (processing time plus 500 ms) or less. *The SLA was satisfied at the end of the accounting period (one day) in all our simulations.* Table 4.3 lists the data center capacities.

Cost-unaware distribution. As the simplest basis for comparison, we use a cost-unaware policy (CU-Heuristic) that is similar to CA-Heuristic. It orders data centers according to performance, i.e. $CDF_i(L, LC_i)$, from highest to lowest. Requests are forwarded to the first data center on the list until its capacity is met. At that point, new requests are forwarded to the next data center on the list and so on. Data center reconfiguration happens as in CA-Heuristic.

4.2.2 Results

Effect of cost-awareness and pricing scheme. Figure 4.3 depicts the energy cost of the EPrice, CA-Heuristic, and CU-Heuristic policies under the three (brown) electricity pricing schemes. The figure shows many important results: (1) As expected, both cost-aware policies reduce costs compared to CU-Heuristic, even under constant pricing;

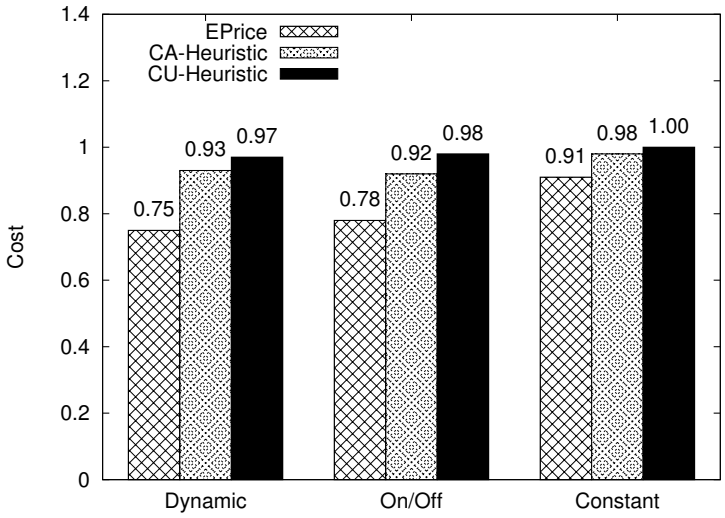


Figure 4.3: Pricing and cost-awareness.

(2) the On/Off and Dynamic schemes enable significant cost reductions compared to constant pricing, especially under EPrice; and (3) EPrice always achieves lower cost than CA-Heuristic. In fact, combining cost-awareness and dynamic pricing enables EPrice to reduce cost by 25%. In general, EPrice behaves better than CA-Heuristic because it often uses the cheapest but worst-performing data center (Europe), instead of the expensive but best-performing data center (US East Coast). The reason is that EPrice predicts that it can compensate for the poor performance of the European data center during future periods of low load.

Effect of time zones. Figure 4.3 assumes that each data center is in a different time zone. When this is not the case, serving a request costs the same at any data center. For this reason, all policies achieve the same cost, regardless of pricing scheme. This cost is slightly higher than that of CU-Heuristic in Figure 4.3, suggesting that multiple time zones are critical to enable cost savings.

Effect of green data centers. Figure 4.4 depicts the cost and brown energy consumption of the GreenDC policy, CA-Heuristic, and CU-Heuristic, under dynamic pricing. The results are normalized against the default results for EPrice with dynamic pricing ("All-Brown"), i.e. the leftmost bar in Figure 4.3. Figure 4.4 shows that GreenDC can decrease brown energy consumption by 35% by leveraging the green data centers at

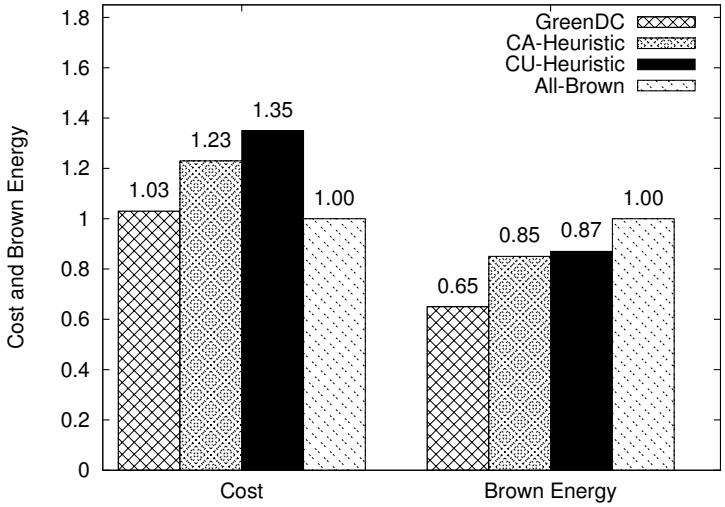


Figure 4.4: Green data centers.

only a 3% cost increase. The heuristic policies save substantially less brown energy at much higher costs than GreenDC. Again, the reason is that the heuristic policies often use the East Coast data center, instead of the wind-based European data center.

Effect of base energy. The results above all assume that servers do not consume any power when idle. Figure 4.5 quantifies the effect of the base energy by comparing the default results for EPrice, CA-Heuristic, and CU-Heuristic to those when a server consumes 75W and 150W when idle. We assume that no data center consumes green energy. This figure shows that increasing the base energy reduces the cost savings achievable by our optimization approach. The gains are smallest (but still non-trivial) for Base = 150W. This result shows that the benefits of our approach will increase with time, as servers become more energy-proportional.

4.3 Summary

In this chapter, we proposed a framework for optimization-based request distribution in multi-data-center Internet services. We also proposed two policies for managing these services' energy consumption and cost, while respecting their SLAs. The policies take advantage of time zones, variable electricity prices, and green energy. Finally, we proposed a simple heuristic for achieving the same goals. Our evaluation showed

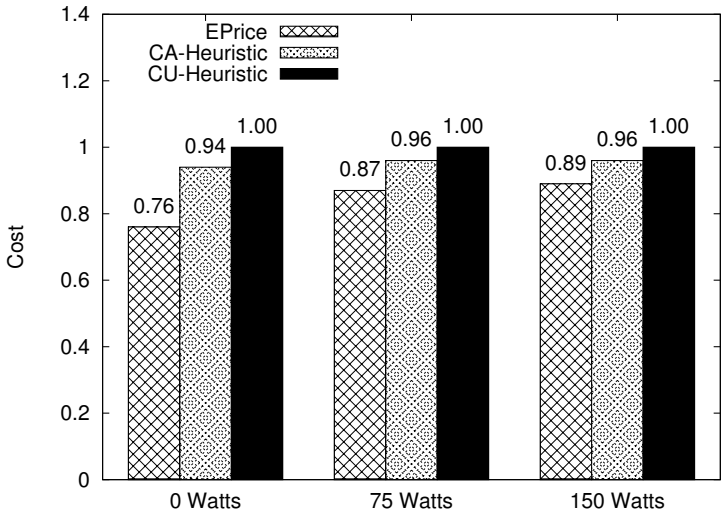


Figure 4.5: Base energy.

positive results. Overall, our work demonstrates the value of load distribution across data centers, formal optimization techniques, workload prediction, and electricity price prediction for energy management.

Chapter 5

Brown Energy Capping of Internet Services

In the last chapter, we showed that intelligent distribution of computation load is a very effective tool to manage energy and cost for multi-data-center Internet services. In this chapter, we show how to extend the framework to manage cost and retaining the same level of performance when Internet service providers are bounded by a brown energy cap.

The brown energy caps we consider are inspired by recent developments around the world. Specifically, we have seen governments imposing Kyoto-style *cap-and-trade* schemes on large brown energy consumers. For example, the UK government started a mandatory cap-and-trade scheme for businesses consuming more than 6 GWh per year in April 2010 [167], which means that a business with even a relatively small 700-KW data center will have to participate. Under this scheme, if a business' brown cap is exhausted, it will have to purchase offsets from the market to countervail its subsequent brown energy consumption. In the US, Congress had discussed a federal cap-and-trade scheme, while regional schemes have already been created [143].

Utilities may also impose caps on large electricity consumers to encourage energy conservation or manage their own costs. In this scenario, consumers that exceed the cap could pay higher brown electricity prices, in a scheme we call *cap-and-pay*.

Finally, we also now see individuals and businesses aggressively conserving energy and voluntarily offsetting their emissions in many ways [68, 75]. They may also voluntarily set brown energy caps for themselves in an effort to manage their energy-related costs; in this scenario, caps translate into explicit targets for energy conservation. When carbon-neutrality can be used as a marketing tool, businesses may also use caps to

predict their expenditures to achieve neutrality. We refer to these voluntary caps as *cap-as-target*.

With these capping schemes in mind, we propose and evaluate a software framework for optimization-based request distribution. The framework enables services to manage their energy consumption and costs, while respecting their SLAs. For example, the framework considers the energy cost of processing requests before and after the brown energy cap is exhausted. Under cap-and-trade, this involves tracking the market price of carbon offsets and interacting with the market after cap exhaustion. At the same time, the framework considers the existing requirements for high throughput and availability. Furthermore, the framework allows services to exploit data centers that pay different TOU electricity prices, data centers located in different time zones, and data centers that can consume green energy (by power utilities allow them to select a mix of brown and green energy, as many utilities do today). Importantly, the framework is general enough to enable energy management in the absence of brown energy caps, different electricity prices, or green energy.

Based on the framework, we propose request distribution policies for cap-and-trade and cap-and-pay scenarios. Similar to Chapter 4, we propose optimization-based and greedy heuristic request distribution policies. In addition, we propose two more optimization-based policies that leverage a fast linear programming solver, but rely on a shorter prediction horizon. The techniques we use to solve these policies have also been adapted to the fact that the accounting period is one year instead of one day.

Using simulation, a real system distributed over four universities, a request trace from a commercial service, and real network latency, electricity price, and carbon market traces, we evaluate our optimization and heuristic-based request distributions in terms of energy costs and brown energy consumptions. We also investigate the impact of the size of the cap, the performance of the data centers, and the servers' energy proportionality.

5.1 Request Distribution Policies

Our ultimate goal is to design request-distribution policies that minimize the energy cost of a multi-data-center Internet service, while meeting its SLA. Next, we discuss the principles and guidelines behind our policies, and then present each policy in turn.

5.1.1 Principles and Guidelines

For our policies to be practical, it is not enough to minimize energy costs; we must also guarantee high performance and availability, and do it all dynamically. Our policies respect these requirements by having the front-ends (1) prevent data center overloads; and (2) monitor their response times, and adjust the request distribution to correct any performance or availability problems.

When data centers can use green energy, we assume that the power mix for each center is contracted with the corresponding power utility for an entire year. The information used to determine the mixes comes from the previous year. The brown energy cap is associated with the entire service (i.e., all of its data centers) and also corresponds to a year, the "energy accounting period". Any leftover brown energy can be used in the following year. When a service exceeds the cap, it must either purchase carbon offsets corresponding to its excess consumption (cap-and-trade) or start paying higher electricity prices (cap-and-pay). The service has a single SLA with customers, which is enforced on a weekly basis, the "performance accounting period". The SLA is specified as (L, P), meaning that at least $P\%$ of the requests must complete in less than L time, *as observed by the front-ends*. This definition means that any latency added by the front-ends in distributing requests to distant data centers is taken into account. Our SLA can be combined with Internet QoS approaches to extend the guarantees all the way to the users' sites [180].

Our SLA implies that *the service does not need to select a front-end device and data center that are closest to each client for the lowest response time possible*; all it needs is to have respected the SLA at the end of each week. However, some services may not be able to tolerate increases in network latency. For those services, set L low

and P high. Other services are more flexible. For example, many services have heavy processing requirements at the data centers themselves; additional network latency would represent a small fraction of the overall response time. For those services, L can be increased enough to allow more flexibility in the request distribution and lower energy costs. In Section 6.3, we investigate the tradeoff between the response time requirements (defined by the SLA) and our ability to minimize costs.

We assume that each data center reconfigures itself by leaving only as many servers active as necessary to service the expected load for the next hour plus an additional 20% slack. (The extra servers can deal with unexpected increases in load and compensate for any inaccuracy of our load prediction. In fact, as we shall demonstrate in Section 6.3, our predictions in a few cases underestimate the load by roughly 10%. The 20% slack includes an additional safety margin of 10% on top of this small prediction inaccuracy.) The other servers can be turned off to conserve energy, as in [34–36, 79, 129]. Turning servers off does not affect the service's data set, as we assume that servers rely on network-attached storage (and local Flash), like others have done as well, e.g. [34]. For such servers, the turn-on delay is small and can be hidden by the slack. In addition, their turn-on/off energy is negligible compared to the overall energy consumed by the service (transitions occur infrequently, as shall also be seen in Section 6.3). The reason is that turning a server on takes on the order of one minute [36, 129] (turning it off is substantially faster), whereas the minimum reconfiguration interval in our systems is one hour. For this reason, we do not model the transition energy explicitly throughout the paper.

5.1.2 Optimization-Based Distribution

Our framework comprises the parameters listed in Table 5.1. Using these parameters, we can formulate optimization problems defining the behavior of our request distribution policies. The optimization seeks to define the power mixes m_i^{brown},m_i^{green} for each data center i, once per year. At a finer granularity, the optimization also seeks to define the fraction $f_i^y(t)$ of requests of type y that should be sent by the front-end devices to

Symbol	Meaning
$f_i^y(t)$	Percentage of requests of type y to be forwarded to center i in epoch t
m_i^{brown}	Percentage of brown electricity in the power mix of center i
m_i^{green}	Percentage of green electricity in the power mix of center i
Overall Cost	Total energy cost (in $) of the service
BEC	Brown energy cap (in KWh) of all centers combined
$BC(t)$	Brown energy consumed (in KWh) of all centers combined up to time t
$p^y(t)$	Percentage of requests of type y in the workload in epoch t
M_i^y	Equals 1 if data center i can serve requests of type y. Equals 0 otherwise
$ReqCost_i^y(t)$	Avg. energy cost (in $) incl. cap-violation charges of serving a request of type y at center i in epoch t
$BaseCost_i(offered_i, t)$	Base energy cost (in $) incl. cap-violation charges of center i in epoch t,under $offered_i$ load
$b_i^y(t)$	Avg. energy cost (in $) of serving a request of type y using only brown electricity at center i
$g_i^y(t)$	Avg. energy cost (in $) of serving a request of type y using only green electricity at center i
$b_i^{base}(offered_i, t)$	Base energy cost (in $) of center i in epoch t, under $offered_i$ load, using only brown electricity
$g_i^{base}(offered_i, t)$	Base energy cost (in $) of center i in epoch t, under $offered_i$ load, using only green electricity
$market_i^y(t)$	Avg. carbon market cost (in $) to offset a request of type y in center i in epoch t
$marketBase_i(offered_i, t)$	Market cost (in $) of offsetting the base energy in center i in epoch t, under $offered_i$ load
$fee_i^y(t)$	Avg. fee for cap violation (in $) for a request of type y in center i in epoch t
$feeBase_i(offered_i, t)$	Fee for cap violation (in $) for the base energy in center i in epoch t, under $offered_i$ load
$LR(t)$	Expected peak service load rate (in reqs/sec) for epoch t
$LT(t)$	Expected total service load (in number of reqs) for epoch t
LC_i	Load capacity (in reqs/sec) of center i
$CDF_i(L, offered_i)$	Expected percentage of requests sent to center i that complete within L time, given $offered_i$ load
$offered_i$	$\sum_y f_i^y(t) \times M_i^y \times p^y(t) \times LR(t)$ (in reqs/sec)

Table 5.1: Framework parameters. Note that we define the "peak" service load rate, $LR(t)$, as the 90th percentile of the actual load rate to eliminate outlier load rates. Also, note that $b_i^y(t)$, $g_i^y(t)$, $market_i^y(t)$, and $fee_i^y(t)$ exclude the cost of the "base" energy.

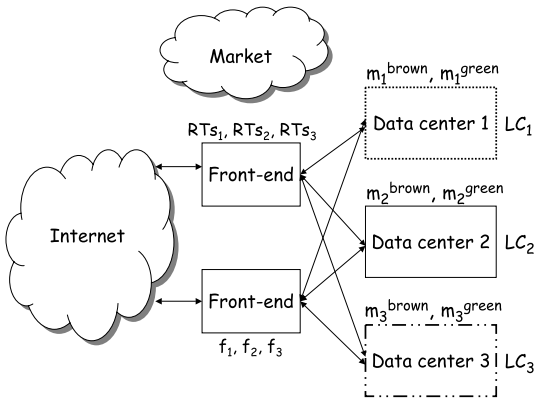


Figure 5.1: Each data center i consumes m_i^{brown}% brown energy and m_i^{green}% green energy, and has a certain processing capacity (LC) and a history of response times (RTs). The f_i's are the percentages of requests that should be sent to the data centers.

each data center i, during "epoch" t. During each epoch, the fractions are fixed. Depending on the solution approach, a set of fractions is computed for one or more epochs at a time. *The computation is performed off the critical path of request distribution.* The front-ends must recompute the fractions, if the load intensity, the electricity price, the data center response times, or the market prices change significantly since the last computation. A solution is typically computed no more than twice per hour.

After each recomputation and/or every hour, the front-ends inform the data centers about their expected loads for the next hour. With this information, the data centers can reconfigure by leaving only as many servers active as necessary to service the expected load (plus the 20% slack). Figure 5.1 depicts an example service and the main characteristics of its data centers.

The next subsection describes two specific optimization problems (policies), one for cap-and-trade and one for cap-and-pay. Subsection 5.1.2.2 describes the instantiation of the parameters. Subsection 5.1.2.3 discusses how to solve the problems.

5.1.2.1 Problem Formulations

Cap-and-trade policy. Our first optimization problem seeks to optimize the overall energy cost, *Overall Cost*, in a cap-and-trade scenario. Equation 5.1 of Figure 5.2

$$\begin{aligned} \textit{Overall Cost} &= (\textstyle\sum_t \sum_i \sum_y f_i^y(t) \times M_i^y \times p^y(t) \times LT(t) \times ReqCost_i^y(t)) \\ &\quad +(\textstyle\sum_t \sum_i BaseCost_i(\textit{offered}_i, t)) \end{aligned} \tag{5.1}$$

$$ReqCost_i^y(t) = \begin{cases} m_i^{brown} \times b_i^y(t) + m_i^{green} \times g_i^y(t) & \text{if BC(t)} \leq \text{BEC} \\ market_i^y(t) + m_i^{brown} \times b_i^y(t) + m_i^{green} \times g_i^y(t) & otherwise \end{cases} \tag{5.2}$$

$$BaseCost_i(\textit{offered}_i, t) = \begin{cases} m_i^{brown} \times b_i^{base}(\textit{offered}_i, t) + m_i^{green} \times & \\ g_i^{base}(\textit{offered}_i, t) & \text{if BC(t)} \leq \text{BEC} \\ marketBase_i(\textit{offered}_i, t) + m_i^{brown} \times & \\ b_i^{base}(\textit{offered}_i, t) \quad + \quad m_i^{green} \quad \times & otherwise \\ g_i^{base}(\textit{offered}_i, t) & \end{cases} \tag{5.3}$$

1. $\forall i\ m_i^{brown}$ *and* $m_i^{green} \geq 0 \Rightarrow$ *i.e., each part of the mix cannot be negative.*
2. $\forall i\ m_i^{brown} + m_i^{green} = 1 \Rightarrow$ *i.e., the mixes need to add up to 1.*
3. $\forall t \forall i \forall y\ f_i^y(t) \geq 0 \Rightarrow$ *i.e., each fraction cannot be negative.*
4. $\forall t \forall y \sum_i\ f_i^y(t) \times M_i^y = 1 \Rightarrow$ *i.e., the fractions for each request type need to add up to 1.*
5. $\forall t \forall i\ \textit{offered}_i \leq LC_i \Rightarrow$ *i.e., the offered load to a data center should not overload it.*
6. $\sum_t \sum_i \sum_y (f_i^y(t) \times M_i^y \times p^y(t) \times LT(t) \times CDF_i(L, \textit{offered}_i)) \,/\, \sum_t LT(t) \geq P$
   $\Rightarrow$ *i.e., the SLA must be satisfied.*

(5.4)

Figure 5.2: Cap-and-trade formulation. We solve our formulations to find $f_i^y(t)$, m_i^{brown}, and m_i^{green}. The goal is to minimize the energy cost (*Overall Cost*), under the constraints that no data center should be overloaded and the SLA should be met. The energy cost is the sum of dynamic request-processing (*ReqCost*) and static (*BaseCost*) costs, including any carbon market costs due to violating the brown energy cap.

defines *Overall Cost*, where $p^y(t)$ is the percentage of requests of type y in the service's workload during epoch t, $LT(t)$ is the expected total number of requests for the service during epoch t, $ReqCost_i^y(t)$ is the average energy cost (including cap-violation charges) of processing a request of type y at data center i during epoch t, and $BaseCost_i(\textit{offered}_i, t)$ is the "base" energy cost (including any cap-violation charges) of data center i during epoch t, under a given $\textit{offered}_i$ load. Base energy is the energy that is spent when the active servers are idle; it is zero for a perfectly energy-proportional system [20], and non-zero for systems with idle power dissipation. We define $\textit{offered}_i$ as $\sum_y f_i^y(t) \times M_i^y \times p^y(t) \times LR(t)$, where M_i^y is set when center i can serve requests of

type y and $LR(t)$ is the peak request rate during epoch t.

The per-request cost, $ReqCost_i^y(t)$, is defined in Equation 5.2, where $g_i^y(t)$ is the average cost of serving a request of type y at center i using only green electricity in epoch t, $b_i^y(t)$ is the same cost when only brown electricity is used, BEC is the brown energy cap for the service, and $market_i^y(t)$ is the average cost of carbon offsets equivalent to the brown energy consumed by center i on a request of type y in epoch t.

Essentially, this cost-per-request model says that, before the brown energy cap is exhausted, the cost of executing an average request of a certain type is the average (weighted by the power mix) of what it would cost using completely green or brown electricity. Beyond the cap exhaustion point, the service needs to absorb the additional cost of purchasing carbon offsets on the market. This cost model also means that *the optimization problem is non-linear when the power mixes have not yet been computed* (since the mixes are multiplied by the fractions in *Overall Cost*).

The base energy cost $BaseCost_i(\mathit{offered}_i, t)$ is defined similarly in Equation 5.3. In this equation, $b_i^{base}(\mathit{offered}_i, t)$ is the base energy cost of center i in epoch t under brown electricity prices, $g_i^{base}(\mathit{offered}_i, t)$ is the same cost but under green electricity prices, and $marketBase_i(\mathit{offered}_i, t)$ is the market cost of offsetting the base energy of center i in epoch t.

Overall Cost should be minimized under the constraints that follow the equations above, where LC_i is the processing capacity of center i, and $CDF_i(L, \mathit{offered}_i)$ is the percentage of requests that were served by center i within L time (as observed by the front-ends) when it most recently received $\mathit{offered}_i$ load. Figure 5.2 includes summary descriptions of the constraints. *Note that we could have easily added a constraint to limit the distance between a front-end and the centers to which it can forward requests. This would provide stricter limits on response time than our SLA.*

After the optimization problem is solved once to compute the power mixes for the year and the first set of fractions, the power mix variables can be made constants and the power mix-related constraints can be disregarded. Under these conditions, the problem can be made linear by solving it for one epoch at a time.

Cap-and-pay policy. The formulation just presented assumes that services may purchase carbon offsets on an open trading market to compensate for having exceeded their brown energy caps. If carbon offsets happen to be cheap, the service does not have a significant incentive to conserve as much brown energy as possible. If governments or power utilities decide that they want to promote brown energy conservation (and green energy production/consumption), an alternative is to levy high fees on the service when brown energy caps are exceeded. We also study this scenario. The only modification required to the formulation above is to replace $market_i^y(t)$ and $marketBase_i(\mathit{offered}_i, t)$ with $fee_i^y(t)$ and $feeBase_i(\mathit{offered}_i, t)$, respectively.

Cap-as-target. A cap-as-target formulation would be similar to those above. The only difference would be the penalty for violating the self-imposed cap, which would account for the additional costs of achieving carbon neutrality (e.g., the cost of planting trees). As cap-and-trade and cap-and-pay are enough to explore the benefits of our framework and policies, we do not consider cap-as-target further in this dissertation.

Applying our formulations to today's services. It is important to mention that *the formulations mentioned above are useful even for current Internet services, i.e. those without brown energy caps or green energy consumption.* The brown data centers that support these services are spread around the country (and perhaps even around the world) and are exposed to different brown electricity prices. In this scenario, our framework and policies can optimize costs while abiding by SLAs, by leveraging the different time zones and electricity prices. To model these services, all we need to do is specify "infinite" energy caps and 100%–0% power mixes for all data centers.

Complete framework and policies. For clarity, the description above did not address services with session state (i.e., soft state that only lasts the user's session with the service) and on-line writes to persistent state (i.e., writes are assumed to occur out-of-band). Sessions need not be explicitly considered in the framework or policy formulations, since services ultimately execute the requests that form the (logical) sessions. (Obviously, sessions must be considered when actually distributing requests, since multiple requests belonging to the same session should be executed at the same data center.) Dealing with on-line writes to persistent state mostly requires extensions

to account for the energy consumed in data coherence activities. We present the extended framework and a cap-and-trade formulation including writes in Appendix A. In addition, we briefly evaluate the effect of session length and percentage of write requests in Section 6.3.

5.1.2.2 Instantiating Parameters

Before we can solve our optimization problems, we must determine input values for their parameters. However, doing so is not straightforward in the presence of multiple front-end devices. For example, as aforementioned, the response time of the data centers must be measured at each front-end device. Moreover, no front-end sees the entire load offered to the service. Thus, to select the parameters exactly, the front-ends would have to communicate and coordinate their decisions. To avoid these overheads, we explore a simpler approach in which the optimization problem is solved independently by each of the front-ends. If the front-ends guarantee that the constraints are satisfied from their independent points of view, the constraints will be satisfied globally.

In this approach, $LT(t)$ and $LR(t)$ (and consequently $\mathit{offered}_i$) are defined for each front-end. In addition, the load capacity of each data center is divided by the number of front-ends. To instantiate CDF_i, each front-end collects the recent history of response times of data center i when the front-end directs $\mathit{offered}_i$ load to it. For this purpose, each front-end has a table of these <offered load, percentage of requests served within L time> entries for each data center that is filled over time. Each table has 4 entries corresponding to different levels of utilization (up to 25%, between 25% and 50%, between 50% and 75%, and between 75% and 100%). Similarly, we create a table of <offered load, base energy consumption> entries for each data center. The entries are filled by computing the ratio of the peak load and the load capacity of the data center, and assuming that the same ratio of the total set of servers (plus the 20% slack) is left active. This table only needs to be re-instantiated when servers are upgraded or added to/removed from the data center.

Because the CDF_i and base-energy tables include entries covering the entire range of $\mathit{offered}_i$ and consequently $f_i^y(t)$, a solution to our optimization problem will actually

consider the effect of any $f_i^y(t)$ we may select on the performance and energy cost of data center i. This aspect of our formulation contributes heavily to the stability of our request distribution approach.

Whenever a solution to the problem needs to be computed, the only runtime information that the front-ends need from the data centers is the amount of energy that they have already consumed during the current energy accounting period. Other information, such as load capacities and electricity costs, can be readily available to all front-ends. In our future work, we will study the tradeoff between the overhead of direct front-end communication in solving the problem and the quality of the solutions.

5.1.2.3 Solution Approaches

We study three solution approaches that differ in the extent to which they use load-intensity prediction and linear programming (LP) techniques. Next, we discuss each of the approaches in turn. The discussion assumes the carbon-curbing formulation but extends trivially to the other formulations as well.

Simulated Annealing (SA): Solving for mixes and fractions using week-long predictions. The solution of the optimization problem for an entire energy accounting period (one year) provides the best energy cost. However, such a solution is only possible when we can predict future offered load intensities, electricity prices, carbon market prices, and data center response times.

Electricity price predictions are trivial when the price is constant or when there are only two prices (on-peak and off-peak prices). Other predictions are harder to make far into the future. Instead, we predict detailed behaviors for the near future (the next week, matching the performance accounting period) and use aggregate data for the rest of the year. Specifically, our approach divides the brown energy cap into 52 chunks, i.e. one chunk per week. The energy associated with each chunk is weighted by the aggregate amount of service load predicted for the corresponding week. The intuition is that the amount of brown energy required is proportional to the offered load. Based on the chunk defined for the next week, we solve the optimization problem. Thus, we only need detailed predictions for the next week.

For predicting load intensities during this week, we consider Auto-Regressive Integrated Moving Average (ARIMA) modeling [25]. We do not attempt to predict carbon market prices or CDF_i. Instead, we assume the current market price and the current CDF_i tables as predictions. (As explained below, we recompute the request distribution, i.e. the fractions f_i^y, when these assumed values have become inaccurate.)

Unfortunately, in this solution approach we cannot use LP solvers, which are very fast, even when the power mixes have already been computed. The reason is that, when we compute results for many epochs at the same time, we are still left with a few non-linear functions (i.e., $ReqCost_i^y$, the load intensities, and consequently, $BaseCost_i$ and CDF_i). Instead of LP, we use Simulated Annealing [88] and divide the week into 42 4-hour epochs, i.e. $t = 1..42$. (We could have used a finer granularity at the cost of longer processing time.) For each epoch, the load intensity is assumed to be the predicted "peak" load intensity (actually, the 90th-percentile of the load intensity to exclude outlier intensities) during the epoch.

We use this approach to determine the power mixes once per year. After this first solution, we may actually recompute the request distribution in case our predictions become inaccurate over the course of each week. Specifically, at the end of an epoch, we recompute if any electricity price has changed significantly (this may only occur when real-time, hourly electricity pricing is used [134,135]), or the actual peak load intensity was either significantly higher or lower (by more than 10% in our experiments) than the prediction. We do the same for parameters that we do not predict (market price and CDF_i); at the end of an epoch, we recompute if any of them has changed significantly with respect to its assumed value. We also recompute at the end of an epoch if the predicted load for the next epoch is significantly higher or lower than the current load. In contrast, we recompute immediately whenever the brown cap is exhausted or a data center becomes unavailable, since these are events that lead to substantially different distributions. Given these conditions, recomputations occur at the granularity of many hours in the worst case.

Obviously, we must adjust the brown energy cap every time the problem is solved to account for the energy that has already been consumed. Similarly, we must adjust the

```
Solve the problem to define mixes for the year and
  fractions for every 4-hour epoch of the first week;
Request mixes from utilities;
Every week do
  If !first week then
    Solve the problem to define fractions for every
      4-hour epoch of the week;
  Start distributing requests according to fractions;
  Every epoch of the week do
    Recompute fractions at the end of an epoch if
      any electricity price has changed significantly
      predictions were inaccurate for the epoch
      the predicted load for the next epoch is
        significantly different than the current load;
    Recompute fractions immediately if
      the brown energy cap expires
      a data center becomes unavailable;
    After each recomputation and every hour
      Install new fractions (if they were recomputed);
      Inform data centers about their predicted loads
        for the next hour;
  End do;
End do;
```

Figure 5.3: Overview of the SA solution approach.

SLA percentage P, according to the requests that have already been serviced within L time. Each recomputation produces new results only for the rest of the current week. Any leftover energy at the end of a week is added to the share of the next week.

Finally, after a recomputation occurs and every hour, the front-ends inform the data centers about their predicted loads for the next hour (i.e., the predicted load intensity at each front-end times the fraction of requests to be directed to each data center). Each data center collects the predictions from all front-ends and reconfigures its set of active servers accordingly. Figure 5.3 summarizes the SA approach to solving our optimization problems.

LP1: Solving for fractions using one-hour predictions. As we mentioned above, using week-long predictions means that fast LP solvers cannot be applied. However, if we assume shorter epochs of only 1 hour and focus solely on the next hour ($t = 1$), we can convert the optimization problem into an LP problem and solve it to compute the fractions f_i^y.

The conversion works by transforming non-linear functions into constant values. Specifically, we transform $ReqCost_i^y$ into the expected per-request costs for the next epoch, since we know whether the energy cap has been exceeded. For the load intensity,

Characteristic	SA	LP1	LP0	CA-Heuristic
Energy accounting period	1 year	1 year	1 year	1 year
Power mix computation	SA once per year	SA once per year	SA once per year	SA once per year
Performance accounting period	1 week	1 week	1 week	1 week
Epoch length	4 hours	1 hour	1/2 hour	1 hour
Load predictions	Per front-end for next week	Per front-end for next epoch	None	Per front-end for next epoch
Recomputation/ reordering decision	Epoch boundary	Epoch boundary	Epoch boundary	Epoch boundary
Communication with DCs	Yes	Yes	Yes	Yes

Table 5.2: Main characteristics of policies and solution approaches.

we predict it to be the expected peak load (as represented by the 90th percentile) rate for the next hour, again using ARIMA modeling. With the expected load, we transform $BaseCost_i$ into the expected base energy costs for the next epoch. Like we did above, we do not predict carbon market prices or CDF_i, instead using their current values as a prediction for the next hour.

We recompute a solution under the same conditions as SA: (1) at the end of an epoch for inaccurate load predictions or assumed values; (2) at the end of an epoch, when a prediction for the next epoch is significantly different than that for the current epoch; and (3) immediately when the brown energy cap expires or a data center becomes unavailable. We adjust the cap as before. Given these conditions, recomputations (which are much faster than those of SA) occur more frequently than under SA, but typically no more frequently than every hour.

After a recomputation, the front-ends inform the data centers about their predicted loads for the next hour. The data centers reconfigure accordingly. Figure 5.4 summarizes the LP1 approach to solving our optimization problems.

LP0: Solving for fractions without using predictions. Finally, we can solve our optimization problem without using any predictions at all. We do so using 30-minute epochs and focusing solely on the next epoch ($t = 1$) with an LP solver. Every time the problem is solved, we use the current values for all problem parameters. For example,

```
Solve the problem using SA to define mixes for the year;
Request mixes from utilities;
Solve the problem using LP to define fractions for the
  first 1-hour epoch;
Start distributing requests according to fractions;
Every epoch do
  Recompute fractions at the end of an epoch if
    any electricity price has changed
      significantly or
    predictions were inaccurate for the epoch or
    the predicted load for next epoch is significantly
      different than the current load;
  Recompute fractions immediately if
    the brown energy cap expires or
    a data center becomes unavailable;
  After each recomputation
    Install new fractions;
    Inform data centers about their new predicted loads;
End do;
```

Figure 5.4: Overview of the LP1 solution approach.

the "current" load offered to a data center is its average offered load in the last 5 minutes. As in SA and LP1, we check for deviations at the end of each epoch and recompute a solution if any significant deviations occur. We also recompute immediately, if the cap is exhausted or a center becomes unavailable. Due to its lack of predictions, we expect this approach to cause recomputations nearly every 30 minutes.

After a recomputation, the front-ends inform the data centers about their expected loads for the next half hour (the current load offered to each front-end times the fraction of requests to be directed to each center). The data centers reconfigure accordingly. Figure 5.5 summarizes the LP0 approach to solving our optimization problems.

5.1.3 Heuristics-Based Request Distribution

We also propose a cost-aware heuristic policy (CA-Heuristic) that is simpler and less computationally intensive than the optimization-based approaches described above. The policy deals only with dynamic request distribution; the power mixes are still computed using the optimization-based solution with week-long predictions (SA).

CA-Heuristic is greedy and uses 1-hour epochs. It tries to forward each request to the best data center that can serve it (based on a metric described below), without violating two constraints: (1) the load capacity of each data center; and (2) the SLA requirement that $P\%$ of the requests complete in less than L time, as seen by the

```
Solve the problem using SA to define mixes for the year;
Request mixes from utilities;
Solve the problem using LP to define fractions for the
  first 30-minute epoch;
Start distributing requests according to fractions;
Every epoch do
  Recompute fractions at the end of an epoch if
    the current values for the formulation parameters
      are significantly different than when the
      problem was last solved;
  Recompute fractions immediately if
    the brown energy cap expires or
    a data center becomes unavailable;
  After each recomputation
    Install new fractions;
    Inform data centers about their new expected loads;
End do;
```

Figure 5.5: Overview of the LP0 solution approach.

front-end devices. To avoid the need for coordination between front-ends, they divide the load capacity of each data center by the number of front-ends. In addition, each front-end verifies that the SLA is satisfied from its point of view.

The heuristic works as follows. At each epoch boundary, each front-end computes $R = P \times E$ (the number of requests that must have lower latency than L), where E is the number of requests the front-end expects in the next epoch. E can be predicted using ARIMA. Each front-end also orders the data centers that have $CDF_i(L, LC_i) \geq P$ according to the ratio $Cost_i(t)/CDF_i(L, LC_i)$, from lowest to highest ratio, where $Cost_i(t)$ is the average cost of processing a request (weighted across all types) at data center i during epoch t. The remaining data centers are ordered by the same ratio. A final list, called *MainOrder*, is created by concatenating the two lists.

Requests are forwarded to the first data center in MainOrder until its capacity is met. At that point, new requests are forwarded to the next data center on the list and so on. After the front-end has served R requests in less than L time, it can disregard MainOrder and start forwarding requests to the cheapest data center (lowest $Cost_i(t)$) until its capacity is met. At that point, the next cheapest data center can be exercised and so on.

If the prediction of the number of requests to be received in an epoch consistently underestimates the offered load, serving R requests within L time may not be enough to satisfy the SLA. To prevent this situation, whenever the prediction is inaccurate, the

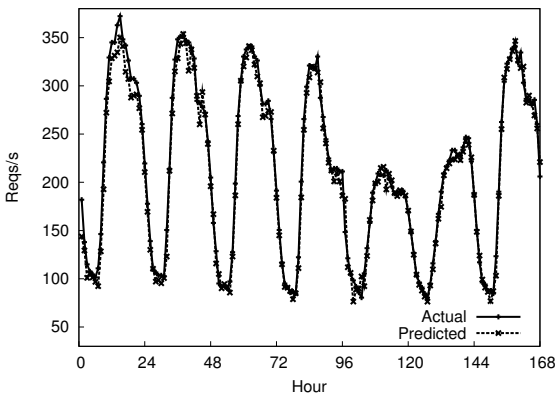


Figure 5.6: Actual and predicted load intensities.

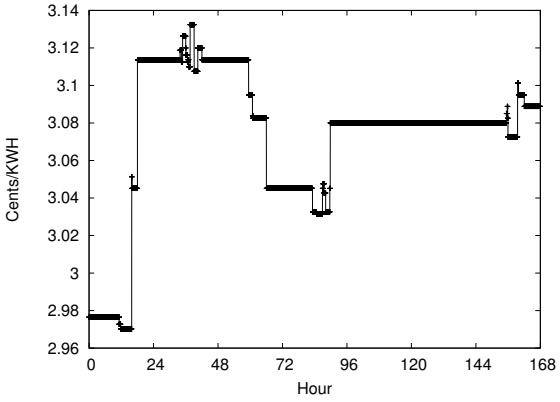


Figure 5.7: Carbon market prices converted to cents/KWh.

heuristic adjusts the R value for the next epoch to compensate.

At each epoch boundary, the front-ends inform the centers about their predicted loads for the next epoch. These predictions are based on the per-front-end load predictions and their MainOrder lists (the lists may change because of changes to $Cost_i$).

Table 5.2 overviews our policies and solution approaches.

5.2 Evaluation

5.2.1 Methodology

To evaluate our framework and policies, we use both simulation and real-system experimentation. Our simulator of a multi-data-center Internet service takes as input a request trace, an electricity price trace, and a carbon market trace or a fee trace. Using these traces, it simulates a request distribution policy, and the data center response times and energy consumptions. Our evaluations are based on year-long traces, as well as sensitivity studies varying our main simulation parameters.

For simplicity, we simulate a single front-end device located on the East Coast of the US. The front-end distributes requests to 3 data centers, each of them located on the West Coast, on the East Coast, and in Europe. *The simulator has been validated against a real prototype implementation, running on servers at four universities located in these same regions* (University of Washington, Rutgers University, Princeton University, and EPFL). We present our validation results in subsection 5.2.2.

Request trace, time zones, and response times. Our request trace is built from a 1-month-long real trace from a commercial search engine, Ask.com. The trace corresponds to a fraction of the requests Ask.com received during April 2008. Due to commercial and privacy concerns, the trace only lists the number of requests for each second. (Even though search engines are sensitive to increases in response time, this trace is representative of the traffic patterns of many real services, including those that can tolerate such increases. In our work, the request traffic, and our ability to predict it and intelligently distribute it are much more important than the actual service being provided.)

To extrapolate the trace to an entire year, we use the search volume statistics for Ask.com from Nielsen-online.com. Specifically, we normalize the total number of requests for other months in 2008 to those of April. For example, to generate the trace for May 2008, we multiply the load intensity of every second in April by the normalized load factor for May.

Figure 5.6 shows the 90th percentile of the actual and ARIMA-predicted request rates during one week of our trace. Our ARIMA modeling combines seasonal and non-seasonal components additively. The non-seasonal component involves 3 auto-regressive parameters and 3 moving average parameters (corresponding to the past three hours). The seasonal component involves 1 auto-regressive parameter and 1 moving average parameter (corresponding to the same hour of the previous week). The figure shows that the ARIMA predictions are very accurate.

For simplicity, we assume that all requests are of the same type and can be sent to any of the 3 data centers. A request takes 400 ms to process on average and consumes 60 J of dynamic energy (i.e., beyond the base energy), including cooling, conversion, and delivery overheads. This is equivalent to consuming 150 W of dynamic power during request processing. By default, we study servers that are perfectly energy-proportional [20], i.e. they consume no base energy ($BaseCost = 0$, no need to turn machines off). Servers today are not energy-proportional, but base energy is expected to decrease considerably in the next few years (both industry and academia are seeking energy-proportionality). Nevertheless, we also study systems with different amounts of

Data Center	Brown energy cost	Green energy cost
West Coast	5.23 cents/KWh	7.0 cents/KWh
East Coast	12.42 cents/KWh	18.0 cents/KWh
Europe	11.0 cents/KWh	18.0 cents/KWh

Table 5.3: Default electricity prices [3, 39, 48, 106, 116].

base energy ($BaseCost \neq 0$, some machines are turned off).

The default SLA we simulate requires 90% of the requests to complete in 500 ms (i.e., the processing time plus 100 ms) or less. *The SLA was satisfied at the end of the performance accounting period (one week) in all our simulations.* We study other SLAs as well.

To generate a realistic distribution of data center response times, we performed real experiments with servers located in the 3 regions mentioned above. The requests were issued from a client machine on the East Coast. Each request was made to last 400 ms on average at a remote server, according to a Poisson distribution. The client exercised the remote servers at 4 utilization levels in turn to instantiate the CDF_i tables. We leave 20% slack in utilizations, so the utilizations that delimit the ranges are really, 20%, 40%, 60%, and 80%. The time between consecutive requests issued by the client also followed a Poisson distribution with the appropriate average for the utilization level. The results of these experiments showed that higher utilizations have only a small impact on the servers' response time. Overall, we find that the servers exhibit average response times of 412 ms (East Coast), 485 ms (West Coast), and 521 ms (Europe) as measured at the client. With respect to our SLA, only the East Coast server can produce more than 90% of its replies in 500 ms or less. The other servers can only reply within 500 ms 76% (West Coast) and 16% (Europe) of the time.

The response times collected experimentally for each data center and utilization level form independent pools for our simulations. Every time a request is sent to a simulated data center, the simulator estimates the current offered load to the center and randomly selects a response time from the corresponding pool.

Electricity prices, carbon prices, and fees. We simulate two electricity prices at each data center, one for "on-peak" hours (weekdays from 8am to 8pm) and another for

"off-peak" hours (weekdays from 8pm to 8am and weekends) [39]. The on-peak prices are listed in Table 5.3; by default, the off-peak prices are 1/3 of those on-peak. Note that the West Coast center is located in a state with relatively cheap electricity.

We simulate both cap-and-trade (default) and cap-and-pay. For cap-and-trade, the carbon market trace was collected from PointCarbon [130] for one month (August 2008). Figure 5.7 shows the carbon prices during a week, converted from Euros/ton of carbon to cents/KWh. To extend the trace to an entire year, we used the same normalization approach described above, again using data from [130]. Under cap-and-pay, the default fee for violating the brown energy cap is set at the price of brown electricity, meaning that, above the cap, the price of brown energy doubles. We also study smaller fees.

Other parameters. The default brown energy cap is equivalent to 75% of the dynamic energy required to process the trace, but we study the effect of this parameter as well. We assume that green energy can be at most 30% of the energy used at each data center. The load capacity of all data centers was assumed to be 250 requests/sec. We have scaled down the load capacity to match the intensity of our request trace.

Cost-unaware distribution. As the simplest basis for comparison, we use a cost-unaware policy (CU-Heuristic) that is similar to CA-Heuristic but disregards electricity prices and cap-violation penalties. It orders data centers according to performance, i.e. $CDF_i(L, LC_i)$, from highest to lowest. Requests are forwarded to the best-performing data center on the list until its capacity is met. At that point, new requests are forwarded to the next data center on the list and so on. Data center reconfiguration happens as in CA-Heuristic.

5.2.2 Real Implementation and Validation

We also implemented real prototypes of our request distribution approaches. To do so, we extended a publicly available HTTP request distribution software (called HAProxy [76]) to implement our optimization infrastructure and policies. The optimization code was taken verbatim from the simulator. The software was also modified to obey the optimized fractions in distributing the requests to the data centers. Overall, we added roughly 3K lines of new code to the roughly 18K lines of HAProxy.

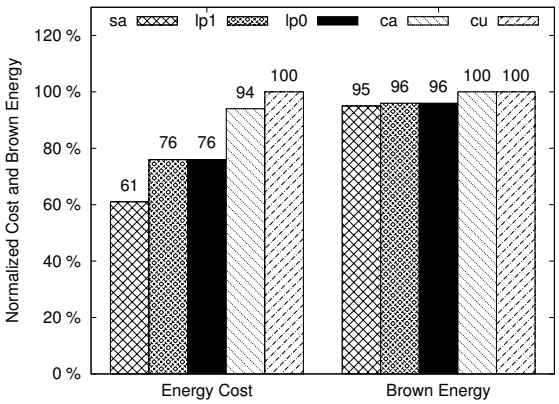


Figure 5.8: Comparing policies and solution approaches.

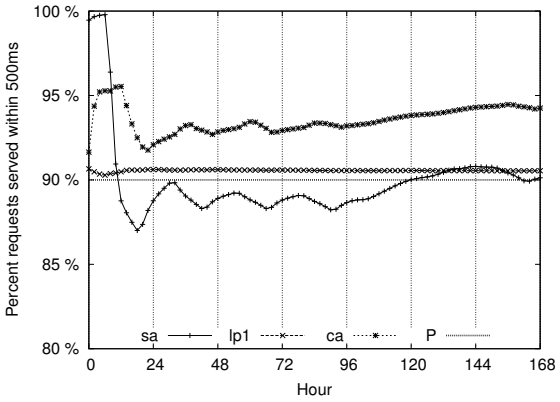


Figure 5.9: Detailed behavior toward meeting the SLA.

Unfortunately, running the implementation in real time is impractical; computing the system's full-year results would require one full year per parameter setting. Thus, the results in Section 5.2.3 are simulated, but we run the real implementation for 40 hours to validate the simulator under the same assumptions.

For our validation experiments, we ran our front-end software on a machine located on the East Coast. This front-end machine was the same as our client in the response time experiments above. The data centers were also represented by the same remote servers we used in those experiments. The requests were issued to the front-end from another machine on the East Coast. This latter machine replayed a scaled-down version of two 4-hour periods during the first weekday of the Ask.com trace: a low-load period (from midnight to 4am, which includes the lowest load rate of the day) and a high-load period (from noon to 4pm, which includes the highest load rate for the day). We run each of our solution approaches and heuristics with each of the 4-hour traces. The latencies observed in these runs were fed to the simulator for a proper comparison. The experiments assumed the same default parameters as the simulator (including the same power mixes).

Our validation results are very positive. Table 5.4 summarizes the differences between the real executions and the simulations in terms of energy cost, brown energy consumption, and percentage of requests completed within 500 ms. The simulator and

	Energy Cost	Brown Energy	SLA
Low Load			
SA	0.03%	0.07%	0.1%
LP1	0.01%	0.12%	0.39%
LP0	0.03%	0.04%	0.27%
CA	0.07%	0.09%	0.24%
CU	0.06%	0.06%	0.14%
High Load			
SA	0.81%	0.03%	0.31%
LP1	1.99%	0.42%	1.06%
LP0	2.34%	0.54%	0.96%
CA	4.14%	1.68%	0.25%
CU	5.99%	1.52%	1.7 %

Table 5.4: Differences between simulation and prototype.

the prototype produce results that are within 2% of each other. The only exceptions are the energy costs of the two heuristics when they are exposed to a high request load. However, even in those cases, the difference is at most 6% only. The reason for the larger difference is that the simulator and the real system use different approaches for specifying the capacity of the data centers. Our simulator specifies them in requests per second, whereas the prototype uses a number of concurrent connections. Under low loads, the request traffic is not high enough for this difference in approach to noticeably alter the energy costs.

5.2.3 Results

5.2.3.1 Comparing Policies and Solution Approaches

We first compare our optimization-based policy for cap-and-trade (under different solution approaches), our cost-aware heuristic policy (CA-Heuristic), and the cost-unaware heuristic policy (CU-Heuristic). All policies and approaches use the same power mixes for the data centers. The mixes were computed by SA to be 80/20 (East Coast), 70/30 (West Coast), and 81/19 (Europe), where the first number in each case is the percentage of brown electricity in the mix. With these mixes, the on-peak electricity prices per KWh become 13.54 cents (East Coast), 5.76 cents (West Coast), and 12.33 cents (Europe).

Figure 5.8 plots the energy cost (bars on the left) and the brown energy consumption (bars on the right) for the optimization-based policy with SA, LP1, and LP0 solution approaches, CA-Heuristic (labeled "CA"), and CU-Heuristic ("CU"). The cost and consumption bars are normalized against the CU-Heuristic results.

The figure shows that both cost-aware policies produce lower costs than CU-Heuristic, as one would expect. CU-Heuristic sends the vast majority of requests to the most expensive but also best performing data center (East Coast). The figure also shows that the optimization-based policy achieves substantially (up to 35%) lower costs than CA-Heuristic, regardless of the solution approach. The reason is that CA-Heuristic tries to satisfy the SLA every hour, greedily sending requests to the most expensive data center until that happens. Comparing the solution approaches to the optimization problem, we can see that solving it for an entire week at a time as in SA leads to the lowest cost. In fact, SA achieves 39% lower costs than CU-Heuristic. Interestingly, the comparison between LP0 and LP1 shows that accurately predicting the load of the next hour is essentially useless, when solving the problem for only one hour at a time. Nevertheless, LP0 and LP1 still produce 24% lower costs than CU-Heuristic. The reason for SA's lowest cost is that it exploits the cheapest data center (West Coast) more extensively than the other approaches, as it deduces when it can compensate later for that center's lower performance (by sending requests to the best-performing data center during its off-peak times) and still meet the SLA.

To illustrate these effects, Figure 5.9 plots the cumulative percentage of requests serviced within 500 ms by the policies and solution approaches (LP0 almost completely overlaps with LP1, so we do not show it), as they progress towards meeting the SLA ($L = 500$ms, $P = 90\%$) during the first week of the trace. SA's ability to predict future behaviors allows it to exploit low electricity prices on the West Coast (during on-peak times on the East Coast), serve slightly less than 90% of the requests within 500 ms for most of the week, and still satisfy the SLA at the end of the week. Due to their inability to predict behaviors for more than 1 hour, the other solution approaches have to be conservative about performance all the time.

Returning to Figure 5.8, the optimization-based policy led to only slightly lower

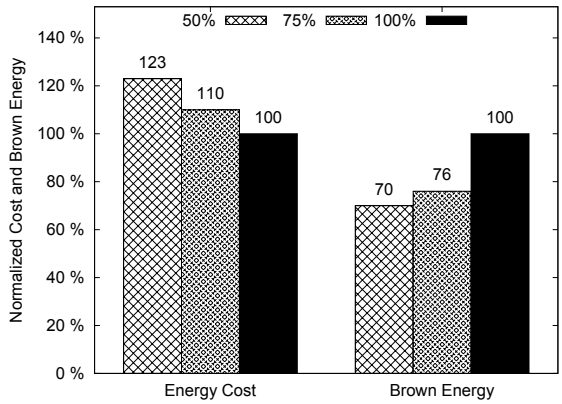


Figure 5.10: Effect of cap on brown energy consumption.

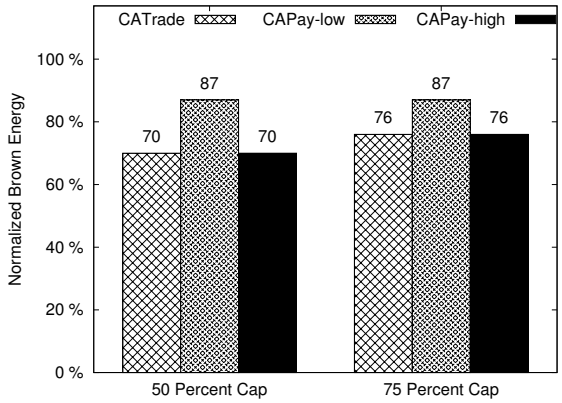


Figure 5.11: Comparing cap-and-trade and cap-and-pay.

brown energy consumptions than CA-Heuristic and CU-Heuristic. This result is not surprising, since the brown energy consumption is determined for the most part by the choice of power mix at each data center. Recall that this choice is the same for all policies and solution approaches. As we show in the next subsection, the way to conserve brown energy is to lower the brown energy cap.

Regarding the frequency of recomputations, we find that SA has to recompute a solution once every 7.6 days on average, whereas LP1 and LP0 do so roughly every 2 hours and every 1.5 hour on average, respectively. The average times to solve the optimization problem once are 392 s for SA and 1.5 ms for the LP approaches on a 3.0-GHz oct-core machine. These frequencies and times represent negligible energy overheads for the service.

These results suggest that the optimization-based policy with SA is the most cost-effective approach. The advantage of SA over LP1 and LP0 decreases as we shorten the performance accounting period. Nevertheless, SA still behaves substantially better than its linear counterparts for daily periods. Most services do not require finer granularity enforcement than a day. However, *for the few services that require tight hourly SLA guarantees, LP1 and LPO are the best choice*; although SA could be configured to produce the same results, it would do so with higher overhead.

Qureshi's heuristic. Qureshi *et al.* [135] proposed a greedy heuristic to distribute requests across data centers based on electricity prices that vary hourly. The heuristic sends requests to the cheapest data center at each point in time, up to its processing capacity, before choosing the next cheapest data center and so on. The heuristic has a very coarse control of response times by which a latency-based radius is defined around each front-end; data centers beyond the radius are not chosen as targets. We implemented their heuristic for comparison and found that it either leads to very high costs or is unable to meet our SLAs, depending on the size of the radius around our East Coast front-end. When only the East Coast data center can be reached, the heuristic behaves exactly like CU-Heuristic, meeting the SLA but at a very high cost. When the West Coast data center can be reached as well, the SLA is violated because that data center is cheaper most of the time but does not meet the SLA. In this case, only 80% of the requests can be serviced within 500ms. When any data center can be reached, the situation becomes even worse, since the European data center is sometimes the cheapest but violates the SLA by a greater amount. In this latter case, the heuristic misses the SLA by almost 40%. For these reasons, we do not consider their heuristic further.

5.2.3.2 Conserving Brown Energy

The results we have discussed so far assume that the brown energy cap is large enough to process 75% of the requests in our trace. To understand the impact of the cap, Figure 5.10 displays the energy cost and brown energy consumption of SA under brown energy caps of 100% (effectively no cap), 75%, and 50% of the energy required to process the

requests in the trace. The results are normalized to the no-cap scenario.

The figure shows that lowering the brown energy cap from 100% to 75% enables a savings of 24% in brown energy consumption at only a 10% increase in cost. The cost increase comes from having to pick power mixes that use more green energy; consuming brown energy beyond the cap and going to the carbon market is actually more expensive under our simulation parameters. Interestingly, decreasing the cap further to 50% increases the brown energy savings to 30% but at a much higher cost increase (24%). The reason is that there is not enough green energy to compensate for the 50% cap (the maximum amount of green energy in the power mixes is 30%). Thus, the service ends up exceeding the cap and paying the higher market costs.

These results suggest that services can significantly reduce their brown energy consumption at modest cost increases, as long as the caps are carefully picked.

5.2.3.3 Comparing Cap-and-Trade and Cap-and-Pay

The results we have discussed so far assume the cap-and-trade policy. Because of the relatively high cost of carbon offsets in our trace, services avoid exceeding the brown energy cap to the full extent permitted by the amount of available green energy. To understand the impact of the penalties for exceeding the cap, we now compare the cap-and-trade and cap-and-pay policies under two different fee settings for the latter policy.

Figure 5.11 plots the brown energy consumption under cap-and-trade (labeled "CATrade"), cap-and-pay with a low fee ("CAPay-low"), and cap-and-pay with a high fee ("CAPay-high"), as a function of the cap size. The low fee is lower than the cost of green energy, whereas the high fee is higher than that cost. All policies use SA as the solution approach.

The figure shows that CATrade and CAPay-high behave the same regardless of cap size. The reason is that they choose to consume as much green energy as possible, as discussed above. However, these policies behave quite differently than CAPay-low. Under CAPay-low, there is no incentive to use green energy, since the charge for violating the cap is actually lower than the cost of green energy. As a result, CAPay-low

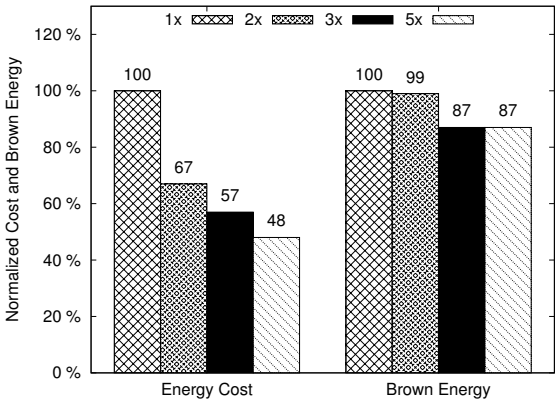


Figure 5.12: Effect of ratio of electricity prices.

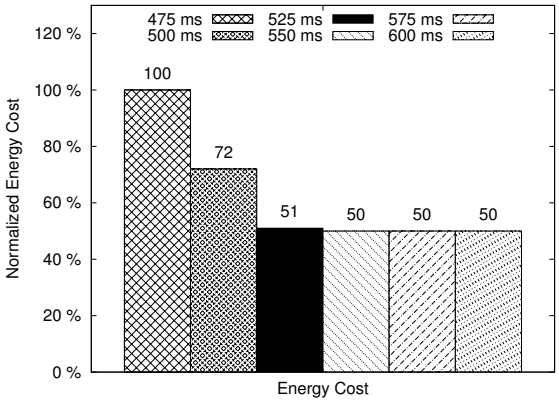


Figure 5.13: Effect of latency requirement (L).

consumes substantially more brown energy than CATrade and CAPay-high.

These results illustrate that, if the goal is to force services to conserve brown energy, the penalties must be selected so that there is a strong incentive not to exceed the cap. Furthermore, given that carbon market prices are difficult to control, a cap-and-pay scheme may be a more effective approach to encourage brown energy conservation.

5.2.3.4 Sensitivity Analysis

So far, we have studied the impact of the distribution policy and solution approach, the cap size, and the capping scheme. In this subsection, we first qualitatively discuss the impact of different classes of parameters on our results. After that, we present a series of results quantifying this impact.

Qualitative discussion. After experimenting extensively with our infrastructure, we have found that four classes of parameters (besides those we evaluated above) have a strong impact on our results: (1) the relative electricity prices at the data centers over time; (2) the response time requirements imposed by the SLA; (3) the energy consumed by servers when they are idle; and (4) the percentage of writes in the workload.

The first class of parameters is important in that the electricity price differentials are the very source of possible cost savings. In addition, the availability of cheap green energy at the data centers has a direct impact on the power mixes and on our ability to conserve brown energy at low cost. Furthermore, when prices are different at each

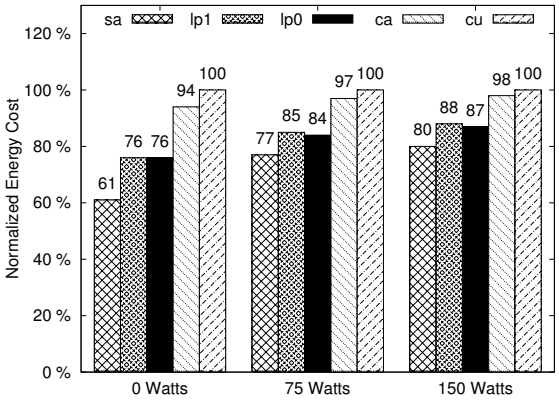


Figure 5.14: Effect of base energy.

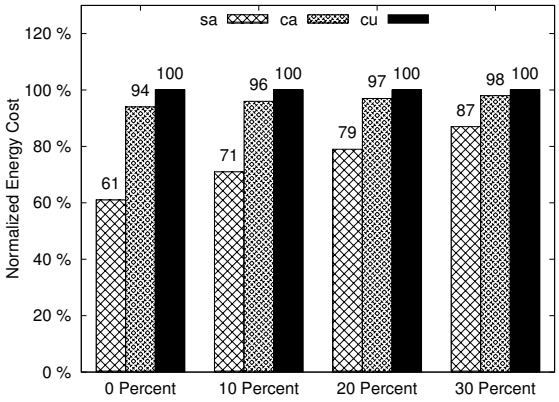


Figure 5.15: Effect of write percentage.

data center but constant over time, time zones are not exploited and the cost savings are limited. When on- and off-peak prices are different at each center, the ordering of data centers with respect to price changes over time. In fact, it is this ordering that really matters in defining the request distribution, not the absolute prices.

The second class is important because it determines how flexible the request distribution can be. Although this class involves many parameters (e.g., data center response times), it can be studied by varying a single knob, the SLA. Services with strict response time requirements may not be able to utilize cheap data centers that are far away from the front-end. In contrast, a front-end placed near the cheapest data center would enable all approaches (be they optimization-based or not) to easily meet the SLA at low cost; they would all use this data center most of the time. (Note however that, in practice, there would most likely be front-ends near all data centers, i.e. some front-ends would be far away from the cheapest data center as in our case study.)

The third and fourth classes determine how much of the overall energy cost can be managed through sophisticated request distribution. The third class also determines how much energy can be saved by turning servers off.

Relative electricity prices. Figure 5.12 plots the energy cost and brown energy consumption of the cap-and-trade policy under the SA approach, in scenarios where on-peak prices are 1x, 2x, 3x, and 5x the off-peak prices. The results are normalized against the 1x case (constant pricing). Recall that the default we have used thus far is

a 3x ratio.

As one would expect, the figure shows that larger ratios reduce costs. However, the reductions are far from linear, despite the fact that there is substantially more off-peak than on-peak time during the year. The reason is that most requests (60%) are actually processed during on-peak times at the East and West Coast data centers. With respect to brown energy consumption, we see distinct behaviors at low and high ratios. At low ratios, power mixes tend to involve less green energy, since exceeding the cap is relatively inexpensive. In contrast, at high ratios, green energy becomes inexpensive a large percentage of the time.

These results suggest that services should take advantage of differentiated electricity prices to the extent that they can negotiate them with power companies.

Response time requirements. Here, we compare different response time requirements (SLAs) and the cost savings that can be achieved when they allow flexibility in request distribution. Figure 5.13 shows the energy costs of SA for $P = 90\%$, as we vary L from 475 to 600ms. Recall that our results so far have assumed $L = 500$ms. The response times have a negligible impact on the brown energy consumption.

The figure demonstrates that having less strict SLAs enables significant cost savings, e.g. 28% and 49% for $L = 500$ms and 525ms, respectively. The reason is that the West Coast data center can be used more frequently in those cases. As the latency requirement is relaxed further, no more cost savings can be accrued since at that point all data centers can meet the SLA independently. We also conducted an experiment where a very strict $P = 99\%$ is enforced with $L = 550$ms (this is the first response time for which $P = 99\%$ can actually be satisfied). We observed that the cost compared to $P = 90\%$ increases by 16%.

These results illustrate the fundamental tradeoff between SLA requirements and our ability to lower costs. When SLAs are strict, we can meet them but at a high cost. When these requirements can be relaxed, significant cost savings can be achieved.

Base energy. Figure 5.14 plots the energy costs of the policies and solution approaches, as a function of the amount of power servers consume when idle. (Since we assume a dynamic power range of 150W, a base power of 150W roughly represents today's

servers.) The costs are normalized to those of CU-Heuristic. Recall that all results we discussed thus far assumed no base energy.

The figure shows exactly the same trends across base powers. In fact, even under the most pessimistic assumptions, SA can still produce 20% energy cost reductions.

This result suggests that the benefits of our optimization-based framework and policies will increase with time, as servers become more energy-proportional.

Request types and on-line writes to persistent state. To assess the impact of these workload characteristics, we modified the Ask.com trace to include 3 different request types, such that one type involves writes. Our systems handle each write request at a data center by propagating coherence messages to its mirror data centers. All simulation parameters were kept at their default values, but writes were assumed to consume 60 J at each data center because of coherence activity.

Figure 5.15 depicts the energy costs of SA, CA-Heuristic, and CU-Heuristic, as a function of the percentage of write requests. The figure shows that SA produces costs that are 19% and 21% lower than those of CA-Heuristic and CU-Heuristic, respectively, when 20% of the requests are writes. These cost savings are smaller than those from Section 5.2.3.1, but are still quite significant. Write percentages that are substantially higher than 20% are not as likely in practice. Nevertheless, the SA cost savings are still 11-13% when as many as 30% of the requests are writes.

Session length. Our results so far have assumed that each session contained a single request. For completeness, we also study the impact of longer sessions. To evaluate this impact, we created two variations of the Ask.com trace that assigned each request to a specific user session. In the first variation, we created sessions with an average of 3 requests, whereas the second variation has sessions with an average of 10 requests. Recall that our systems distribute entire sessions (rather than individual requests), such that all requests belonging to a session are forwarded to the same center as the first request of the session.

Our results show that the session length has only a negligible ($\leq$ 1%) impact on energy cost and brown energy consumption. The reason is that, for any sufficiently large number of sessions, the percentage of requests of each type forwarded to a particular

data center is roughly the same across all session lengths.

5.3 Summary

In this chapter, we proposed a framework for enabling multi-data-center Internet services to manage their brown energy consumption and leverage green energy, while respecting their SLAs and minimizing costs. Moreover, our framework enables services to exploit different electricity prices and data centers located at different time zones to minimize costs, even in the absence of brown energy caps, different electricity prices, or green energy.

We proposed three optimization-based techniques: 1) LP0, 2)LP1, and 3) SA and a simple heuristic for achieving the same goals. We evaluated our proposals extensively, using simulations, real experiments involving sites from US and Europe, and real traces. Using simulation, a request trace from a commercial service, and real network latency, electricity price, and carbon market traces, we evaluated our optimization and heuristic-based request distributions in terms of energy costs and brown energy consumptions. We also investigated the impact of the capping scheme, the size of the cap, the performance of the data centers, and server energy proportionality. Using a real system distributed over four universities, we validated the simulation with real experiments.

We made several interesting observations from our results. First, our optimization-based distributions achieved lower energy costs than our simpler heuristic. Our best optimization-based distribution achieves 35% lower costs (and similar brown energy consumption) while meeting the same SLA. Second, we demonstrated that predicting load intensities many hours into the future reduces costs significantly. Third, we showed that brown energy caps and data centers that exploit green energy can be used to limit brown energy consumption without significantly increasing energy costs or violating the SLA. For example, we can reduce the brown energy consumption by 24% for a 10% increase in cost. Fourth, we found that our optimization-based distribution can achieve cost reductions of 24% by exploiting different electricity prices at widely distributed data centers, even in the absence of brown energy caps and green energy.

Chapter 6

Virtual Machine Placement for High Performance Computing Clouds

In the last two chapters, we have shown that we can use load distribution as a mechanism to manage energy usage and cost across the data centers of a multi-data-center Internet service. In this chapter, we develop a similar framework for multi-data-center cloud services that support high performance computing. Recall from Chapter 1 that while HPC cloud services do share some characteristics with Internet services, their workloads differ significantly, necessitating different frameworks. For example, Internet services deal with short-running requests, rather than longer running jobs that may comprise multiple virtual machines. This difference is relevant in that techniques such as load migration do not make sense for short-running requests. Thus, we assume that an HPC cloud service supports client jobs as a set of virtual machines that should be run concurrently. The load distribution takes the form of a set of policies for placing virtual machines (when a job enters the system) and for migrating virtual machines between data centers when it is profitable to do so.

Specifically, this chapter proposes and then evaluates a set of cost-aware virtual machine placement and migration policies targeted specifically for geographically distributed computing platforms. These policies model in details the following main components of electricity costs: 1) the cost of energy consumed by IT equipments, cooling as well as power delivery system, and 2) a peak power charge at each data centers. Compared to the previous two chapters, we introduce two new important cost components: the cost of operating the cooling infrastructure and the peak power charges.

Similar to Internet services, clients submit jobs to one of the front-end servers. These jobs have a soft deadline that the service should try not to violate. Meanwhile,

the service has the freedom of moving the jobs between its data centers in order to minimize its operating cost. The polices we evaluate include traditional cost-unaware heuristics such as round robin, best fit, worst fit as well as shortest distance-based. We also propose two cost-aware policies, one involves migration of jobs while the other does not.

One interesting management aspect that is introduced with job migration is that our policies may suddenly move a large load from one data center to another. For example, suppose that at a particular time, it becomes much cheaper to run computations at a data center B than A (because of the start of off-peak electricity pricing). The policy with migration may decide to move the entire workload of A, which may currently be heavily loaded, to B. When this occurs, if nothing else is done, then data center B may overheat because its cooling system cannot adapt fast enough to the arriving load. Thus, our policy must consider transient temperature effects, and pre-cool a data center if it expects to shift a significant amount of load to that data center in the near future.

In this chapter, we first describe the cooling model developed by our colleagues from Department of Mechanical Engineering, which uses Computation Fluid Dynamics simulations of temperatures in realistic data centers. This cooling model also details the transient cooling effects when sudden load is placed on a data center. The model shows that under such scenarios, large and rapid changes in load can indeed produce overheating. It also shows that certain cooling strategies are substantially worse than others, and require longer periods of pre-cooling to prevent overheating. The transient effect will become more prevalent, as cloud service providers are increasingly running their data centers at higher temperatures (i.e., closer to the overheating point) to reduce cooling energy consumption [89]. We also use this model to study how cooling varies with outside temperature to show that it is potentially profitable to use different data centers depending on which has lower outside temperatures at a given time.

We then present our framework and load distribution policies. Finally, we present our simulation-based evaluation, which uses real HPC workload traces, outside temperature data and electricity/peak power demand prices. Our evaluation studies the ability of different cooling strategies to handle load spikes, compares the behaviors of

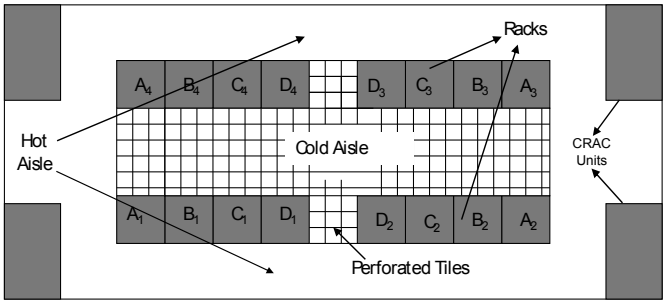


Figure 6.1: Data center with cooling. The water chiller is not shown.

our dynamic cost-aware policies to cost-unaware and static policies, and explores the effects of many parameter settings. Among other interesting results, we demonstrate that (1) our policies can provide large cost savings, (2) load migration enables savings in many scenarios, and (3) all electricity-related costs must be considered at the same time for higher and consistent cost savings.

6.1 Cooling Model

While many efforts have been expended to decrease this overhead as an aggregate, less attention has been paid to the dynamic behavior of cooling systems and how to leverage it to reduce cost and/or energy consumption. To explore this dynamic behavior, we use the cooling model developed by our colleagues from the Department of Mechanical Engineering for the data center shown in Figure 6.1. The data center contains 480 servers mounted in 16 racks in a 7m×8m×3m room.[1] Each server consumes a base power of 200W (power drawn when idle) and 300W when fully utilized.

The modeled cooling system is typical of today's data centers, with CRAC units that take in hot air produced by servers, and blow cold air into the under-floor plenum. The cold air comes up through the perforated tiles on the floor and into the servers' air inlets to cool the servers. (Our modeled data center has four CRAC units.) If the outside temperature is sufficiently low, the CRACs discharge the hot air to and take cold air from outside. If the outside temperature is too high, the CRACs circulate the

[1]The model simulates a relatively small data center to keep simulation times reasonable. When scaling to larger data centers, the efficiency of the cooling system can be improved in various ways, possibly leading to decreased energy consumption on the order of 30% [119].

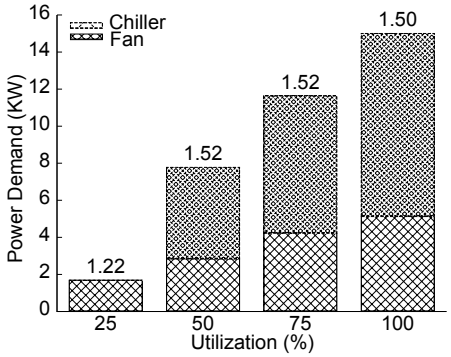


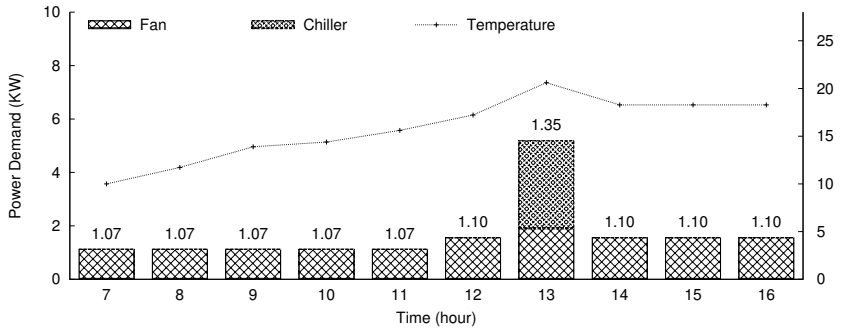
Figure 6.2: Impact of load on cooling power, assuming perfect LC; e.g., only 25% of the servers are on at 25% utilization. The numbers on top of the bars are the PUEs when counting only server and cooling energy. Outside temperature is 21°C and the maximum allowable temperature in the data center is 30°C.

hot air through a water chiller for cooling before sending it back into the data center as cool air.

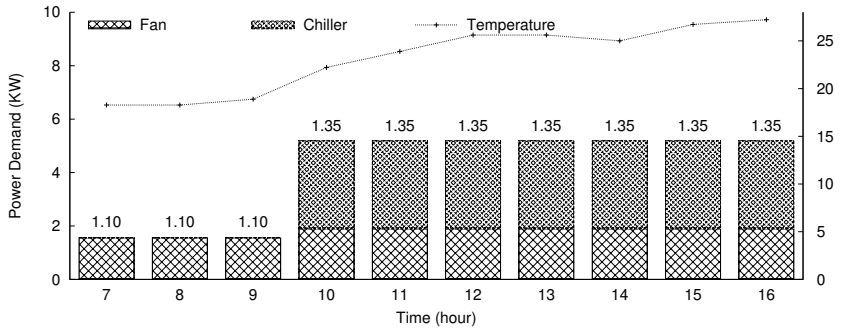
When the load is fixed, the total cooling energy consumed by the data center is the sum of the work performed by the CRACs and the chiller [148]. There are two main settings which affect the cooling system's energy consumption: (1) the CRAC fan speed, which determines the air flow rate through the data center; and (2) whether the chiller is turned on. Given an outside temperature and a data center utilization level, the cooling system can be designed to adjust the CRAC fan speed and chiller on/off setting to ensure reliable operation, as well as operate the data center in the most energy-efficient manner.

Our colleagues simulated the 3D model of the data center that includes all aspects of the air flow in the room and through the servers. Different utilization levels are modeled as racks being turned off in alternating sides. For example, we model a utilization of 50% with 4 fully on racks and 4 fully off racks on each side of the data center. We apply ANSYS Fluent 12.0 turbulent k-epsilon with standard wall functions to obtain the results presented next.

Figure 6.2 shows the power demand of the cooling system (using the most efficient setting), as a function of load when the outside temperature is 21°C. The number on top of each bar is the PUE, i.e., the total energy consumed by the data center divided



(a) Northern California



(b) Georgia

Figure 6.3: Impact of outside temperature on cooling power demand. The load at each data center is 50%. The maximum allowable temperature in the data center is 30°C.

by the energy consumed by the servers and networking equipment. This figure shows that, for our scenario, cooling power demand can change by a factor of more than 7 as the load changes from 25% to 100%. The power demand of the chiller dominates when it has to be turned on. However, the power demand of both the fans and chiller increases with increasing load.

Figure 6.3 shows the outside temperature and cooling power demand as a function of time for two different locations during the same day. One can easily see that cooling energy consumption can change significantly throughout the day as the temperature changes.

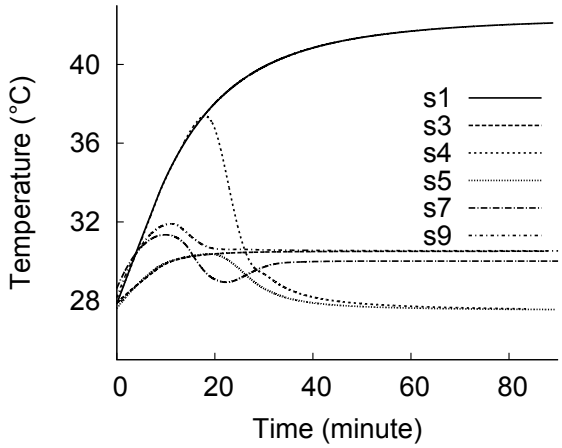
The significant variability of cooling energy consumption vs. outside temperature and load presents an interesting opportunity for dynamic load distribution to minimize energy consumption and/or cost. For example, if a service is replicated across two data centers, one in Northern California and one in Georgia, during the hottest and coolest

Strategy	**Time (Minute)**	**Flow Rate (CFM)**	**Chiller**	**Through Chiller**
1	$t > 0$	3350	OFF	NO
3	$t > 0$	10200	OFF	NO
4	$0 < t < 20$ $t > 20$	3350 10200	ON	NO YES
5	$0 < t < 20$ $t > 20$	10200	ON	NO YES
7	$0 < t < 5$ $5 < t < 10$ $10 < t < 15$ $t > 15$	3350 5650 10200 15200	OFF	NO
9	$0 < t < 5$ $5 < t < 10$ $10 < t < 15$ $15 < t < 35$ $t > 35$	3350 5650 10200 10200 10200	OFF OFF OFF ON ON	NO NO NO NO Yes

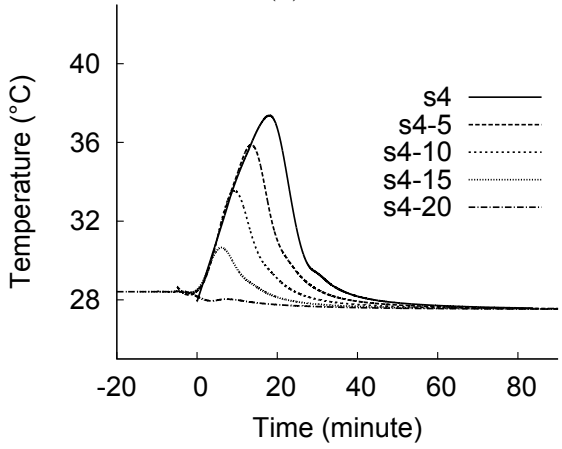
Table 6.1: Potential responses of a cooling system to sudden large load increase. Each scenario is a sequence of actions taken at different times. Actions include increasing flow rate (fan speed), turning on chiller, and circulating air through chiller.

periods of the day, it may make sense to direct load to Georgia because the cooling energy consumed is similar but the energy price may be lower in Georgia. On the other hand, when it is hot in Georgia but cool in Northern California, it may be beneficial to direct more load to Northern California because the cooling energy consumption is much smaller.

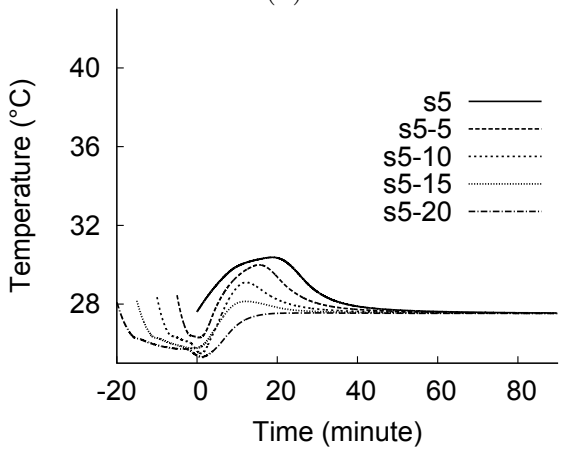
While it makes sense to explore dynamic load distribution (because of variable electricity prices as well as cooling), one has to carefully account for transient cooling effects. As service providers seek to reduce cost by operating data centers at higher temperatures, cooling systems have less time to react to load changes. This can lead to overheating if water chillers are dynamically turned off (when not needed) to further reduce cost. Specifically, while fan speeds can be changed almost instantaneously, there is a significant delay between when the chiller is turned on and when it becomes fully effective for cooling. Figure 6.4 shows what happens when the data center is operating at maximum inlet air temperature of 28°C and the load rises rapidly from 25% to 75%. Such a large change in load may happen because a dynamic load distribution policy



(a)



(b)



(c)

Figure 6.4: Temperature in the data center vs. different cooling responses when utilization suddenly changes from 25% to 75%. Each line in (a) represents a cooling strategy listed in Table 6.1. (b) Strategy 4 with pre-cooling (taking each cooling action earlier) by 5 to 20 minutes. (c) Strategy 5 with pre-cooling by 5 to 20 minutes.

decides to migrate a large amount of load to the data center (because the electricity prices just dropped) or because a very large job (or batch of jobs) just arrived.

The figure plots the maximum inlet air temperature inside the data center against time, for a number of cooling responses (strategies) to the sudden load change occurring at time 0. Table 6.1 describes the cooling strategies in terms of flow rate and chiller activity. As their numbering suggests, these strategies are a subset of those we considered. We study a large number of different response strategies because different cooling systems may respond differently to load changes. (There has been little need to understand these response strategies until the recent trend of operating data centers at temperatures much closer to the overheating point; e.g., operating at 28°C when the maximum allowed temperature is 30°C.) The outside temperature is assumed to be 23°C. The ultimate goal is to keep maximum inlet temperatures under 30°C [17].

The figure shows that not adjusting the cooling settings (strategy 1) is not an option; the inlet temperatures increase significantly with the higher load. The other strategies lead to higher temperatures initially, but later bring them down. For example, strategy 3, which aggressively increases the flow rate but does not activate the chiller, stabilizes at above 30°C. Strategy 7, which ultimately uses a larger flow rate than strategy 3, but increases the flow rate more conservatively is even worse, allowing the temperature to rise to over 32°C before stabilizing just slightly above 30°C.

The only responses that eventually lead to a stable temperature under 30°C are those that use the chiller. Further, it is not sufficient to turn on the chiller when the load change is observed. In fact, the only strategies that do not lead to any overheating are the ones where the chiller was turned on early for pre-cooling (Figures 6.4(b) and (c)). How early pre-cooling must take place depends on the cooling strategy. Strategy 5, which immediately increases the flow rate when the chiller is turned on, only needs to pre-cool 5 minutes before the load change. On the other hand, strategy 4, which waits until the chiller is ready before increasing the flow rate, requires 20 minutes of pre-cooling.

As a result of the above transient effects, our cost-aware load distribution policy with migration has to predict future migrations, so that it can pre-cool the target data

center if necessary (Section 6.2). To be safe, the policy looks ahead 20 minutes to ensure that the chiller will be fully effective before any load is migrated. We do not attempt to predict the arrival of very large jobs or large bursts of jobs. Thus, rather than pre-cool, arriving jobs may have to be delayed until they can be started without causing overheating.

6.2 Our Framework

We now consider policies for distributing client load across multiple data centers to minimize energy cost. We begin by discussing the assumptions and guidelines behind our policies. We then present our framework for computing the energy cost and the estimation that our dynamic cost-aware policies will use. Finally, we present the dynamic cost-aware load distribution policies, which explicitly attempt to minimize cost, and several cost-unaware and static policies that will serve as comparison baselines.

6.2.1 Assumptions and Guidelines

Recall that jobs arrive to a front-end device, which may be located at one of the data centers used to serve the HPC workload or a separate location.[2] The front-end uses a load distribution policy to direct each job to one of the data centers for execution. Over time, if the policy includes job migration, the front-end may direct jobs to be migrated from one data center to another to reduce cost.

In addition to minimizing cost, the load distribution policies should also meet SLAs, and ensure that data centers are properly cooled and not loaded past their capacities. Our policies delay jobs to avoid overheating or overloading data centers. However, delaying jobs may lead to missed SLAs and possibly monetary penalties. Thus, the policies do not delay jobs past their SLAs except to preserve cooling and capacity constraints.

Each arriving job includes a specification of the amount of resources that it needs

[2]While we only consider one front-end in this chapter for simplicity, it is possible to have multiple front-ends in general. The front-ends can either collaborate or statically divide the resources of the data centers among them for load distribution.

and an estimate of its run-time. It is quite easy to specify the amount of needed resources; this is typically under the direct control of the client (e.g., run this job on n VMs). It is more challenging to specify accurate run-times. However, providing run-time estimates is common practice in HPC systems [114, 164], where clients often run the same applications many times and so can predict run-time based on experience. Further, to use a cloud service, clients must at least have approximations of their jobs' run-times to ensure timely completion of the jobs and minimize cost. Incentives for accurately estimating jobs' run-times may include: (1) lowering the priority or even suspending unfinished jobs that have exceeded their estimated run-times significantly, and (2) charging clients based on their resource use *and* the difference between their estimated and actual run times.

HPC jobs often need to access persistent data. Client persistent data is typically replicated across (a small number of) geographically distributed data centers to guard against catastrophic unavailability or loss. This replication allows flexibility in but also constrains job placement. Specifically, if the data is stored at only a subset of the service provider's data centers, then each client's jobs may only be placed at or moved to a data center that contains the client's persistent data. Updates to persistent data are streamed to replicas in the background when the corresponding jobs terminate. Furthermore, as in [176], the service will track all writes to the persistent store, enabling the transfer of only diffs when migrating VMs. Results are either small and easily transferable to clients, or are written to persistent storage and so accessible from any replica.

Each job must be placed in a single data center to avoid requiring intra-job communication to cross data centers. (When migrating, the entire job has to be moved.) Services are typically over-provisioned such that there is always enough capacity to accommodate the offered workload (except for very rare events such as flash crowds). However, fragmentation of available resources across data centers may make it impossible to place an incoming job (that requires more resources than are currently available at any single data center). In this case, the job will be queued at the front-end until a data center becomes available.

The service's SLA with its customers involves each job completing within its specified run-time plus a slack that is a function of the resource requirement and run-time. Our policies attempt to place each arriving job at the data center that would lead to lowest cost. However, a low-cost data center may not be ready to accept a job because of the lag time to reach the needed level of cooling or because needed servers have been turned off (see below). In this case, all of our policies will place the job at a more expensive data center if waiting for the low-cost data center will lead to the job missing its SLA. If it is impossible to meet a job's SLA, then the job will be placed at the data center causing the smallest SLA violation.

To reduce energy consumption and cost, each data center only keeps as many servers active as necessary to service the current workload plus maintain a threshold percentage slack. The remaining servers are turned off to conserve energy [34–36,79,129] after they have been idle for a threshold amount of time. Server turn-on and turn-off both require time during which the server is consuming energy. A server cannot be used until turn-on has been completed. The slack allows arriving jobs requiring less than the threshold percentage of a data center's capacity to be started immediately, without waiting for servers to be turned on. Of course, when there is a burst of arrivals, even small jobs may have to wait for servers to be turned back on. Within each data center, we assume that data is stored on network-attached storage (and local Flash) so that turning servers off does not affect data availability; others have made the same assumption, e.g., [34].

The cooling system is always set to the lowest setting necessary to maintain a target maximum inlet temperature. As explained in Section 6.1, changing the fan speed does not incur significant delay. Thus, fan speed can be immediately adjusted as desired for job arrivals and exits. Turning on the chiller, however, does incur a significant delay. To avoid this delay, our policies only turn off the chiller after it has not been needed for a threshold amount of time.

Our policies currently do not dynamically consolidate jobs within a data center. Instead, the policies try to fully pack each server when jobs arrive so that additional consolidation makes very little difference.

Finally, we assume that data centers have homogeneous hardware; i.e., servers are

identical. We make this simplifying assumption because we are most interested in load distribution across multiple data centers in this chapter. To handle heterogeneous hardware, we envision a data center-level resource manager in addition to the front-end (which is a global resource manager). Each data center resource manager is responsible for managing VMs assigned to that data center, including assigning them to physical machines and consolidating them, thus hiding the details of the actual hardware from the front-end. When the front-end needs to decide where to place a batch of jobs, it would contact the data center manager of each potential hosting data center to get the estimated cost for running that batch at that data center.

6.2.2 Cost Computing Framework

Our framework for computing the energy cost of operating the service comprises the parameters listed in Table 6.2. Using these parameters, we formulate the following cost function:

The total energy cost of the service (Equation 6.1) has two components, the cost for energy consumed by ($Cost_d^E$) and the cost for the peak power demand of ($Cost_d^P$) the service. $Cost_d^E$ is computed across time, where the time horizon is divided into discrete time epochs (Equation 6.2). Each epoch ends (and the next one starts) when any event affecting the energy consumption or cost of the system occurs. These events include changes in energy prices, changes in peak power prices, job arrivals and completions, server turn on and off, and change in cooling settings. The events are chosen to guarantee that prices and the number of jobs running at each data center (hence the number of active machines and their states) are constant within any single epoch.

For each time epoch, the cost of energy consumed is the product of the power demand ($P_{d,t}$), length of the time epoch ($|t|$), and energy price ($Price_{d,t}^E$). In turn, the power demand ($P_{d,t}$) is the sum (Equation 6.3) of the base power demand of active machines ($P_{d,t}^B$), the dynamic power needed to support the jobs currently running at the data center ($P_{d,t}^D$), and the power needed by the cooling system ($P_{d,t}^C$). Note that the power demand of an idle machine, i.e., one that is turned on but not running any jobs, is P_m^B, while the power demand of a fully loaded machine, i.e., one running a VM

Symbol	Meaning
$Cost$	Total energy cost ($) of the service
DC	The set of data centers
$Cost_d^E$	Cost for energy consumed at data center d
$Cost_d^P$	Cost for peak power at data center d
$P_{d,t}$	Avg. power demand (KW) at data center d in epoch t
$P_{d,t}^B$	Base power demand at data center d in epoch t
$P_{d,t}^D$	Avg. dynamic power demand at data center d in epoch t
$P_{d,t}^C$	Avg. cooling power demand at data center d in epoch t
P_m^B	Base power demand of one idle server
P_m^D	Avg. dynamic power demand of one loaded server
$Price_{d,t}^E$	Energy price ($/KWh) at data center d in epoch t
$PPeriods_d$	On-peak and off-peak time periods at data center d; each period is a set of time epochs
$PPower_{d,o}$	Peak power demand at data center d, o is *on-peak* or *off-peak*
$Price_{d,o}^P$	Peak power price ($/KW) at data center d, o is *on-peak* or *off-peak*
$S_{d,t}^a$	Set of *active* servers at data center d in epoch t
$U_{s,t}$	Avg. utilization (%) of server s in epoch t
$Temp_{d,t}$	Avg. outside temperature at data center d in epoch t
$Cost_d^j$	Estimated cost of placing a job j at data center d
T_j	Run time of job j
M_j	The number of VMs comprising job j

Table 6.2: Parameters of energy cost computing framework.

$$Cost = \sum_{d \in DC} (Cost_d^E + Cost_d^P) \tag{6.1}$$

$$Cost_d^E = \sum_t P_{d,t}|t|Price_{d,t}^E \tag{6.2}$$

$$P_{d,t} = P_{d,t}^B + P_{d,t}^D + P_{d,t}^C \tag{6.3}$$

$$P_{d,t}^B = |S_{d,t}^a|P_m^B \tag{6.4}$$

$$P_{d,t}^D = \sum_{s \in S_{d,t}^a} U_{s,t}P_m^D \tag{6.5}$$

$$P_{d,t}^C = f(P_{d,t}^B + P_{d,t}^D, Temp_{d,t}) \tag{6.6}$$

$$Cost_d^P = \sum_{o \in PPeriods_d} PPower_{d,o}Price_{d,o}^P \tag{6.7}$$

$$PPower_{d,o} = \max_{t_p \in o} \frac{\sum_{t \in t_p} P_{d,t}|t|}{|t_p|} \tag{6.8}$$

per each processor that it has, is $P_m^B + P_m^D$. We assume that the dynamic power demand of a machine is proportional to its load (e.g., $P_m^B + 0.25P_m^D$ when running 1 VM on a 4-processor machine). Thus equations 6.4 and 6.5 sum the power that is being drawn by all machines currently turned on in the data center.

We assume two peak power charges (see Chapter 3), one for on-peak hours and one for off-peak hours, each with a potentially distinct peak power demand price (Equation 6.7). For each of the on-peak and off-peak periods, the cost is computed as the maximum power across all measurement intervals (e.g., 15 minutes each) in the period. Because each measurement interval may include many epochs, we compute the power of each interval as the average power over all the epochs within the interval (Equation 6.8).

6.2.3 Dynamic Cost-Aware Job Placement

Given the above framework for computing energy cost, we are now faced with the problem of distributing jobs to minimize the overall cost (Equation 6.1) while still meeting their SLAs. In this section, we introduce two dynamic cost-aware, greedy distribution policies that attempt to place an arriving job j with minimal cost using the following estimates:

$$Cost_d^j = \delta(Cost_d^E) + \delta(Cost_d^P) \tag{6.9}$$

$$\delta(Cost_d^E) = \sum_{t \in T_j} \delta(P_{d,t})|t|Price_{d,t}^E$$

$$\delta(Cost_d^P) = \sum_{p \in PPeriods} \delta(PPower_{d,p})Price_{d,p}^P$$

where $\delta(Cost_d^E)$ is the change in cost due to additional energy consumption, i.e., additional base and dynamic server energy used to run j at data center d and any additional cooling required, and $\delta(Cost_d^P)$ is the change in cost due to increased peak power usage. Note that $\delta(Cost_d^P)$ may be 0 if running j at d does not increase the power demand above the peaks already reached in the past. On the other hand, if j crosses peak pricing periods, then it may increase both peaks, which contributes to increasing the cost of the peak power demand. The energy consumed for cooling is computed using predicted outside temperatures.

The above estimates account for changes in energy price, peak power charge, and temperature; this is one reason why there may be multiple time epochs in T_j. We do not attempt to predict arriving load, however, so the estimates are only based on jobs currently in the system and j. The estimates can be computed either by accounting for jobs finishing and exiting the system (more accurate during low load periods) or by assuming that the workload at d will stay the same for all of T_j (more accurate during high load periods).

We do not show equations for $\delta(PPower_{d,p})$ and $\delta(P_{d,t})$ for brevity, since they are relatively straightforward variations of Equations 6.3–6.6 and 6.8.

Cost-aware Distribution (CA). Consider a job j arriving at the start of an epoch t. Let D' be the subset of data centers that has sufficient unused capacity to accept j. If D' is empty, then j is queued as explained above. Otherwise, CA orders the data centers in D' from least to most additional cost to run j using Equation 6.9. Then, CA places j at the first data center that can meet j's SLA. (Recall that a data center may not be able to meet a job's SLA because of delays necessary for ramping up cooling and/or turning on servers.) If no data center can meet j's SLA, then CA places j at the data center that will violate j's SLA by the smallest amount.

Cost-aware Distribution with Migration (CAM). This policy places arriving jobs in the same way as CA. However, at the end of each epoch, it searches for job migrations that would lower the overall cost to complete the set of running jobs. We use the following simple search heuristic:

for d_c from cheapest to most expensive data center:
  for d_e from most expensive data center down to d_c:
    move jobs from d_e to d_c if have idle
    capacity at d_c and move would reduce cost

The cheapest to most expensive ordering of data centers can be computed using several metrics. Currently, we use an efficiency metric expressed as the energy cost (energy consumption of the servers and cooling system) divided by the number of active CPUs.

At the more expensive data center (d_e), jobs are ordered from highest to lowest using their remaining energy consumption. Consider jobs in this ordering and migrate a job j to the cheaper data center (d_c) if $C^j_{d_e} - C^j_{d_c} - C^j_M > 0$, where $C^j_{d_e}$ and $C^j_{d_c}$ are computed for the remaining run-time of j and C^j_M is the cost of migrating j from d_e to d_c. C^j_M comprises the cost of keeping the servers that j was running on at d_e and the servers that j will be running on at d_c active during the transfer time T. T is a function of the bandwidth between two data centers and the amount of data that needs to be transferred, which includes VM memories and any new persistent data generated since the job started running.

If single jobs cannot be moved profitably and there is still idle capacity at the cheaper

data center, then we consider migrating a batch of jobs because batch migration may improve the amortization of cooling cost (e.g., expense of turning on the chiller is amortized across many jobs instead of just one), making the batch migration cost-effective whereas migration of single jobs would not be. The batching algorithm is very simple. With the same job ordering as above, consider a batch of 2 up to a threshold batch size from the beginning of the ordering.

As described thus far, migration may increase the workload at a cheaper data center significantly. For example, consider the case when the electricity prices for a data center change from on-peak to off-peak. During on-peak hours, the data center may be mostly idle. At the start of off-peak pricing, however, it may be desirable to fully load the data center to leverage the lower prices. Recall from Section 6.1 that such a large shifting of load can result in overheating. Thus, CAM actually looks into the future for events that can lead to large-scale migration and pre-cools as necessary, so that migration can be done immediately when profitable. This prediction is relatively easy to implement because the look-ahead period is short (20 minutes delay for chiller turn-on) and only a few event types (currently only changes in energy and peak power demand prices) can result in large-scale migration.

6.2.4 Baseline Policies for Comparison

Round-Robin (RR). Arriving jobs are distributed in a round-robin fashion among the data centers for load balancing. This policy is completely oblivious to costs.

Worst Fit (WF). This policy selects the data center with the most available resources that will fit the arriving job. The policy is similar to RR but achieves better load balancing, because it explicitly considers the current load at each data center as well as the demand of the arriving job.

Static Cost-Aware Ordering (SCA). This policy accounts for differing costs in a gross, static manner. Consider the cost (including all three components: server energy consumption, peak power demand, and cooling) for supporting an average load (e.g., 50%) over a relatively long period of time (e.g., month or year) at each data center. Order the data centers from cheapest to most expensive. Direct each arriving job to the

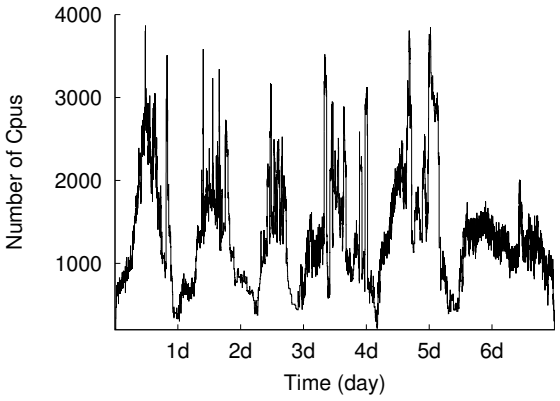


Figure 6.5: Number of CPUs demanded by active jobs.

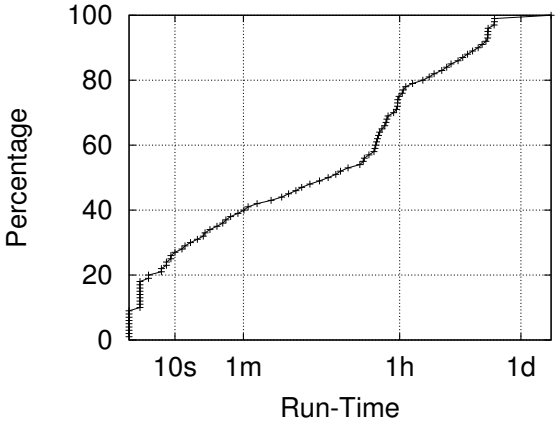


Figure 6.6: CDF of job run times.

cheapest data center that has sufficient capacity to accept and run the job immediately. If no data center can run the job immediately, then choose the data center with the best response time.

6.3 Evaluation

6.3.1 Methodology

We use simulation to evaluate our framework and policies. Our multi-data center simulator takes as input an HPC workload trace, energy price traces, peak power charges, and temperature traces. Using these inputs, it simulates a load distribution policy and the resulting job completion times and data center energy consumptions. Our simulator accounts for the base energy of servers that are on (turning-on and turning-off), dynamic energy consumption, cooling levels needed to support specific data center loads vs. outside temperatures (cooling energy), delays in turning chillers on, and the resource usage (cores, memory, power demand, and migration footprints) of every job at a one-second granularity. Our evaluations are based on week-long traces, as well as sensitivity studies varying our main simulation parameters.

Workload. We use a trace of jobs submitted to an HPC cluster available from the Parallel Workloads Archive [41]. This trace contains approximately four months (December 1999 through April 2000) of data derived from the accounting records of the

LSF software running on a 2048-node Origin 2000 cluster (Nirvana) at Los Alamos National Lab. The trace, LANL-O2K, contains a record for each job serviced by the cluster, with each record containing a job number, submitted time, actual run-time, and maximum number of cores and amount of memory used.

We extracted a random week from this trace (January 24-31, 2000) and use it as the workload for all experiments discussed below. This week-long trace contains 6680 jobs, with a peak processing demand of 3867 cores. The maximum memory requirement of any one job is 4GB. Figure 6.5 shows the number of CPUs demanded by active jobs and Figure 6.6 shows the CDF of job run-times for our workload trace. Job arrival shows a clear diurnal pattern that might be leveraged in the future for load prediction. There is a significant number of short running jobs (almost 30% run for less than 10 seconds) but there are also many long running jobs (>20% run for 1 hour or longer).

To map the above workload to our environment, we assume that each job can be split into VMs, one single-core VM per each core used by the job. The memory used by a job is equally divided among its VMs. When migrating a job, we assume that only the memory images of the VMs and a small amount of persistent data, expressed as the diffs of the data set [176], need to be transferred to the target data center.

As previously discussed, each arriving job specifies the number of VMs needed and an estimated run time. The service SLA is to complete each job within 105% of the job's total processing time (run-time×number of processors) plus 30 seconds. The latter part of the slack is to avoid missing the SLA for short running jobs when machines need to be turned on to accommodate them.

Data centers. We simulate a single front-end located on the East Coast of the US in most cases. The front-end distributes requests to three equal-size data centers located in the US West Coast (Northern California), US East Coast (Georgia), and Europe (Switzerland), respectively. We assume that the clients' persistent data are replicated across the three data centers, so that each job can be placed at any data center that has sufficient idle capacity to host the entire job. Similar to [176], we assume an inter-data-center bandwidth of 464 Mbps.

Data Center	Energy	Peak Demand
West Coast (SFO)	10.8 ¢/KWh	12 $/KWh
East Coast (ATL)	10.7 ¢/KWh	12.83 $/KWh
Europe (GVA)	11 ¢/KWh	13.3 $/KWh

Table 6.3: Electricity prices [3, 39, 48, 50, 60, 121].

Provisioning each data center to be consistent with our modeled data center in Section 6.1 gives us a total of 5760 cores (1440 servers) across the three data centers. This corresponds to approximately 49% spare capacity at the peak load. This provisioning is consistent with current practices and allows the service to support the workload even if one of the data center fails.

Within a data center, each server has 4 cores and 4GB of memory. In our workload trace, no single VM requires more than 1GB of memory. Thus, job placement is essentially a core selection problem. That is, a job can be placed at any data center that has sufficient numbers of idle cores. Within a data center, each VM of a job can be placed on any idle core. In our experiments, jobs are placed within a data center to minimize fragmentation, i.e., minimize the number of active servers.

Each server's base power demand is 200W, with a peak power demand of 300W (corresponding to 100% utilization). Turning a machine on and off both require 30 seconds [34–36, 79, 129]. Each data center turns off machines, if it has over 20% idle capacity. If not needed to maintain spare capacity, machines are turned off after they have been idle for 60 seconds.

Cooling. We use the cooling model developed in Section 6.1 to compute the energy consumed by the cooling system at each data center. Each data center adjusts its cooling system so that temperatures inside the data center never exceed 30°C. Jobs are never run before the cooling system can accommodate the increased heat. Specifically, if a chiller has to be turned on, then arriving jobs are delayed until the chiller is ready (20 minutes). The front-end can ask data centers to pre-cool (by turning on the chiller early) in preparation to receive load; this is used in our cost-aware with migration policy. The chiller is turned off after it has not been used for 40 minutes.

Electricity prices. We simulate two sets of electricity prices at each data center, one

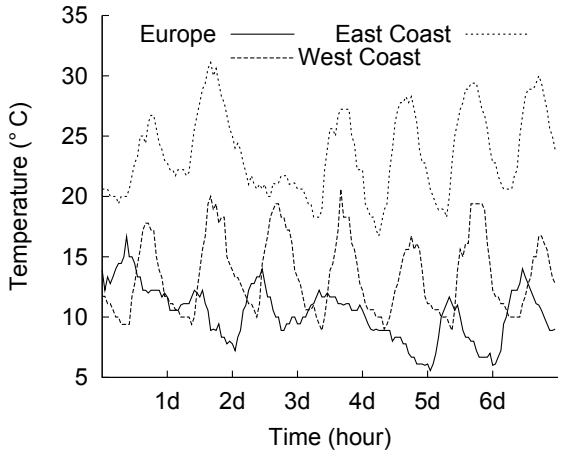


Figure 6.7: Outside temperatures.

for "on-peak" hours (weekdays from 8am to 8pm) and another for "off-peak" hours (weekdays from 8pm to 8am and weekends) [3, 39, 48]. The on-peak prices, listed in Table 6.3, are obtained either from previous publications [69] or from information listed on power utility providers' Web sites [50,60,121]; by default, the off-peak prices are 1/3 of the on-peak prices.

Outside temperatures. We collected historical weather data from the Weather Underground Website [172] for the week of May 1-7, 2010. This period is interesting because not all locations are hot enough to require the chiller to be on all the time, and not all of them are cold enough to depend on just free cooling. Figure 6.7 shows the outside temperatures for our simulated period. Note that the East Coast location is quite a bit hotter than the other two locations but has slightly lower energy prices.

6.3.2 Performance

We begin our evaluation by comparing the energy cost of the service when using our dynamic cost-aware load distribution policies against the baseline policies. Figure 6.8 plots the total energy used by the service (left group of bars) and total cost (right group of bars). The energy usage is divided into three categories: server base energy, server dynamic energy, and cooling. The cost is divided into four categories: server base energy, server dynamic energy, cooling energy, and peak power. Both energy usage and

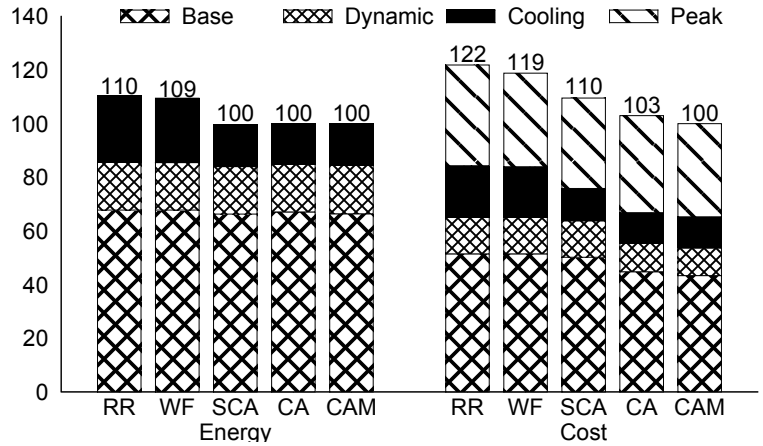


Figure 6.8: Energy used and total cost under different distribution policies.

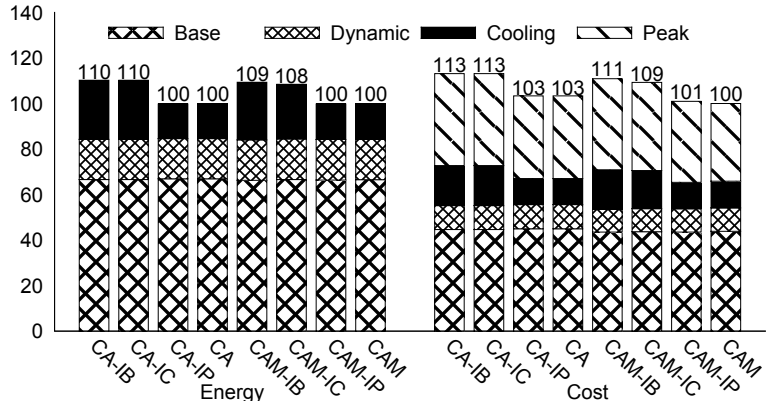


Figure 6.9: Impact of ignoring cost of peak power charge and/or cooling. IP = ignore peak power charge, IC = ignore cooling cost, IB = ignore both.

cost are normalized against the results for CAM. SCA used a static data center ordering of Europe (most preferred), West Coast, and East Coast (least preferred). All SLAs were met by all policies.

We observe from these results that *dynamic* cost-aware load distribution can reduce cost significantly. Specifically, CAM outperforms the two load balancing policies, WF and RR, by 19% and 22%, respectively. CAM also outperforms SCA by 10%, showing that it is possible to leverage short-term differences in electricity prices and outside temperatures to reduce cost.

The ability to migrate jobs leads to a modest cost reduction; CAM outperforms CA by 3%. This is because CA makes placement decisions one-job at a time and never

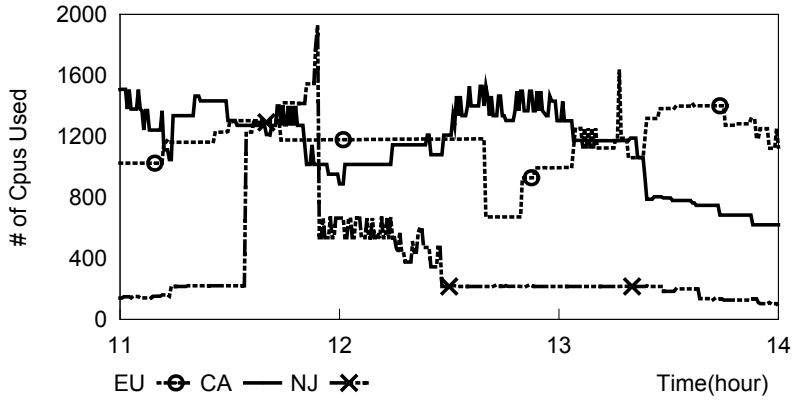


Figure 6.10: Load distribution under CA.

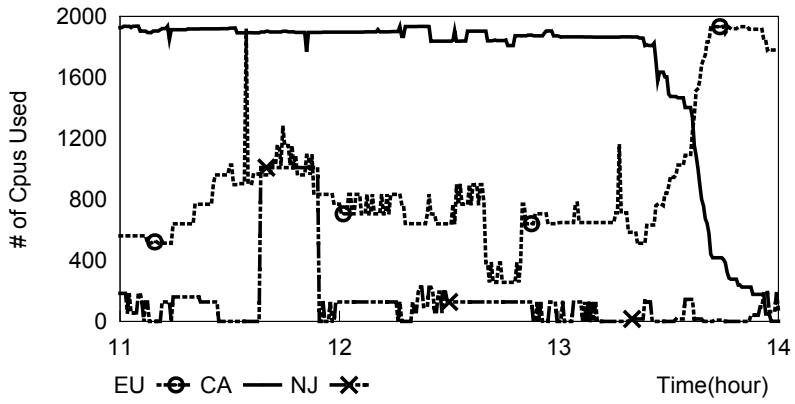


Figure 6.11: Load distribution under CAM.

has a chance to correct its placement. In particular, it is difficult to predict the cost of cooling on a job-per-job basis. Consider a set of jobs that arrive close to each other. When CA considers the placement of each job, placement to a particular data center d might seem expensive because the chiller would need to be turned on. CA would then place each job elsewhere, possibly at higher expense for electricity consumption or peak power, even though placing the entire set of jobs at d would have allowed the cost of running the chiller to be amortized across the set, making d the lowest costing data center. CAM corrects this problem because it can consider the effect of migrating multiple jobs together.

The above differences can be observed in Figures 6.10 and 6.11, which show the reactions of CA and CAM, respectively, to changes in electricity prices and temperatures during a 4-hour period. CA uses the West Coast data center the most before hour 11

because of low off-peak electricity prices. However, it prefers to direct some load to the Europe data center rather than loading the West Coast data center to maximum capacity to avoid turning on the chiller. After hour 13, it gradually prefers Europe because of decreasing temperature (and increasing temperature in the West Coast). In contrast, CAM loads the West Coast data center to maximum capacity until hour 13; this difference in behavior is due exactly to CAM's ability to migrate batches of jobs as explained above. During the same time period, SCA uses the Europe data center almost exclusively, because it is the preferred data center.

Overall, CAM outperforms CA by trading-off slightly higher cooling cost for larger reductions in peak power and IT costs. Both CAM and CA lead to higher peak power cost than SCA because they may heavily load different data centers at different times. For example, Figure 6.11 shows CAM loading the West Coast data center to maximum capacity before hour 13, while loading the Europe data center to maximum capacity after hour 13. This causes both data centers to incur high peak power charges. In contrast, SCA always loads the same data center to maximum capacity before using the next. Thus, the second and third data centers may incur low peak power charges (because they are never highly utilized). CAM and CA both outperform SCA because the savings in cooling and IT energy consumption more than makes up for the higher peak power charges.

6.3.3 Peak Power and Cooling-Awareness

Next, we consider what happens if the dynamic cost-aware policies do not account for the cost of cooling and/or peak power demand. That is, we consider versions of the CA and CAM policies that use Equations 6.1 and 6.9 without the $Cost_d^P$ component and/or Equation 6.3 without the $P_{d,t}^C$ component. Figure 6.9 shows that such incomplete cost consideration can be expensive both in terms of cost and energy usage. Specifically, not accounting for cooling cost leads to 13% increased energy usage by CAM and 10% by CA because the policies place jobs on the East Coast even during the hottest periods to leverage slightly lower electricity prices. This leads to 8% increased cost for CAM and 13% for CA. On the other hand, not considering peak power demand has relatively little

impact in our scenario because variations in peak power charges coincide with variations in electricity prices: on-peak and off-peak periods are the same for electricity prices and peak power charges. Moreover, the ratios of peak power charges and electricity prices are close to each other across locations.

6.3.4 Sensitivity

In this section, we explore changes in relative performance between CAM, CA, and SCA when various parameters are changed.

Run-time predictions. In the above experiments, we assumed that jobs' estimated run-times are exact. We also simulated scenarios where estimated run-times differ from actual run-times according to a Poisson distribution with 10% average. This leads to inaccuracies ranging from 0 to 33% (0 to ∼7.9 hours). We found no significant changes to the results.

Migration time. The cost advantage gained by CAM depends on the amount of data that has to be transferred per migration. For the results presented above, each job migration required an average transfer of 50MB. We also considered scenarios where the transfer size is increased by 1-10 GB per migration. As expected, the cost advantage of CAM decreases with increasing transfer size. However, CAM still outperformed SCA by approximately 8% when over 10GB must be transferred per migrated job. Further, the energy consumed under CAM only increased by 1%.

Outside temperature. To gauge the impact of outside temperatures, we moved the data center in Northern California to Southern California, where the temperature is typically higher. This scenario favors CAM as it can start jobs at this data center when profitable to do so (e.g., off-peak electricity price) or when necessary because of capacity constraints (e.g., less expensive data centers are full), and then move the jobs to other data centers. This leads to an additional 1% performance advantage for CAM over both CA and SCA.

Electricity prices. Figure 6.12 shows what happens when the energy price at the hottest data center is much cheaper: 5.23 cents/KWh on-peak. This setting represents an actual location on the East Coast (North Carolina) but with similar peak power

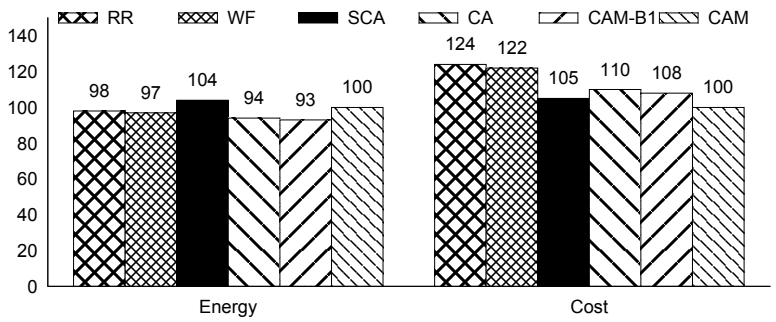


Figure 6.12: Energy used and total cost under different distribution policies when the East Coast data center is in North Caroline. CAM-B1 denotes CAM with batch size of 1.

charges and outside temperatures. In this case, the East Coast data center becomes the 2nd best for SCA, reducing the difference between SCA and CAM. In fact, the key difference between CAM and SCA is the ability of CAM to leverage variable electricity prices (e.g., off-peak on the West Coast is cheaper than on-peak on the East Coast) to reduce cost.

Interestingly, in this scenario, SCA slightly outperforms CA. This results from CA's mis-estimation of the per-job cost of cooling as discussed previously. Figure 6.12 also shows the advantage of using job batching for migration. CAM without batching (CAM-B1) performs 3% worse than SCA and 8% worse than CAM with batches of up to 10.

Relative data center sizes. Next, we considered what happens when the relative sizes of the data centers are changed. When we change the East Coast data center to twice as large as the other two, the cost advantage of CAM increases relative to all other policies. This is because the East Coast data center is, on average, the most expensive data center. CAM benefits from its ability move jobs away from this data center when they were placed there because of capacity constraints. When we reduced the East Coast data center to 1/2 the size of the other two, the cost advantage of CAM decreases relative to the other policies because it is easier for them to avoid this expensive data center with more capacity elsewhere.

Capacity over-provisioning. Finally, we investigate the effect of service capacity over-provisioning. Recall that the default parameter in our experiments was $\sim$1.5 times

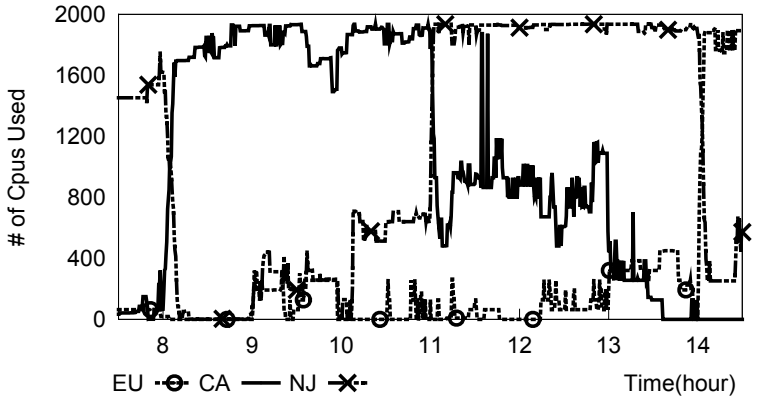


Figure 6.13: Large migrations by CAM at hours 8,11 and 14.

the peak load. If we decrease the over-provisioning factor, the dynamic policies' performance improves relative to SCA. Recall that CAM and CA reduce overall cost relative to SCA by trading off higher peak power charges for lower energy consumption cost. With less over-provisioning, SCA is forced to utilize the less preferred data centers more, leading to increased peak power charges. CAM's performance also improves relative to CA. This is because CA and CAM are also forced to distribute jobs to more expensive data centers because of capacity constraints. However, CAM can move jobs back to the less expensive data centers when resources there are freed by exiting jobs, while CA cannot.

6.3.5 Migration and Pre-Cooling

Figure 6.13 shows three large migrations by CAM. At hour 8, electricity prices on the East Coast (North Carolina) went from off-peak to on-peak. This triggered a large migration from the East Coast to the West Coast. At time 11, electricity prices on the West Coast went from off-peak to on-peak. In response, CAM migrated a large number of jobs from the West Coast back to the East Coast. At time 14, electricity prices changed from on-peak to off-peak at the Europe data center. In response, CAM migrated a large number of jobs from the East Coast to Europe. In these instances, it was critical for the front-end to pre-cool the target data center by turning on the chiller 20 minutes before the migrations took place.

6.4 Summary

In this chapter, we have developed techniques to lower energy costs for HPC cloud service providers that operate multiple geographically distributed data centers. Specifically, we designed heuristics that intelligently place and migrate load across the data centers to take advantage of time-based differences in electricity prices and outside temperatures. Our policies account for three important electricity-related costs: energy price, peak power price, and the energy consumed by the cooling system.

To support our study, we used the detailed model of data center cooling for a realistic data center and cooling system developed by our colleagues from the Department of Mechanical Engineering. We simulated the model to obtain the cooling power demand as a function of data center load and outside temperature. This allowed us to simulate in detail the impact of cooling on total energy cost, and explore whether intelligent consideration of this impact during load distribution can lead to cost savings. We also simulated the model to study transient cooling effects after abrupt, large changes in data center loads. The results led to the conclusion that pre-cooling is necessary to prevent overheating in these scenarios. Our policies incorporate this necessary pre-cooling.

We evaluated our framework using real workload traces, real electricity prices, and real outside temperatures. Our simulation results showed that dynamic cost-aware load distribution can lead to significant cost savings. The actual savings depend on many parameters that determine how much dynamic cost diversity exists, and the flexibility with which policies can affect load distribution (e.g., if there is little spare capacity then there is almost no flexibility for load distribution). Intuitively, awareness of cooling is most important when electricity price and/or peak power charge are only slightly lower at a hot location than at a cool location. In this case, not considering the cooling cost can lead to large cost and energy usage penalties when load distribution policies greedily choose the cheapest electricity price. Similarly, awareness of peak power charges is most important when the ratios of peak power charges to energy cost are widely different across time and/or data centers. However, to correctly account for all possible different scenarios, it is critical that the dynamic load distribution policy considers all three cost

components.

We conclude that cost-aware load placement policies can lower operational costs significantly for cloud service providers. However, these policies must properly account for the significant changes in load that they may cause and how those changes may affect the provider's cooling infrastructure. Our policies are a strong first step in these directions.

Chapter 7

Conclusion

In this dissertation, we explored several novel load distribution techniques to aid service providers that operate multiple data centers to manage their energy consumption and reduce energy-related costs. In particular, our techniques took advantage of differences arising from geographical distribution of data centers: 1) differences in electricity prices, 2) time zone effects, 3) differences in outside temperature to help cooling and 4) differences in availability of renewable energy. We showed that our techniques are very effective in reducing energy consumption and energy-related cost while satisfying the desired SLAs.

In chapter 4, we introduced a framework for optimization-based request distribution in multi-data-center Internet services. We proposed two policies for managing these services' energy consumption and cost, while respecting their SLAs. These two policies correspond to two scenarios: 1) variable electricity price and 2) on-site generation of renewable energy such as wind and solar. We also proposed a simple heuristic for achieving the same goals. Our evaluation showed that optimization-based techniques can leverage advantages offered by data centers' geographical differences much better than the simple heuristic, leading to significant cost savings. We also demonstrated that our policy is very effective in maximizing the usage of available green energy.

In chapter 5, we extended the optimization-based framework with two more policies (LP0 and LP1) which are based on formal optimization and make use of linear programming. We also targeted a very different scenario for promoting the use of green energy instead of brown energy. Specifically, we assumed that a percentage of energy provided by utilities can be contracted to be generated from green sources. Further, we assumed that service providers are constrained by carbon emission caps, and exceeding

this cap has a monetary cost. Our framework again enables services to exploit different electricity prices and data centers located at different time zones to minimize costs, even in the absence of brown energy caps, different electricity prices, or green energy. We evaluated our proposals extensively, using simulations, real experiments, real workload traces, real electricity prices and real carbon market traces.

In chapter 6, we have studied the possibility of lowering energy costs for HPC cloud service providers that operate multiple geographically distributed data centers. Specifically, we designed policies that intelligently place and migrate load across the data centers to take advantage of time-based differences in electricity prices and temperatures. Our policies account for three important electricity-related costs: energy price, peak power price, and the energy consumed by the cooling system. Using a detailed model of data center cooling for a realistic data center and cooling system developed by our colleagues, we showed the impact of cooling on total energy cost, and explored whether intelligent consideration of this impact during load distribution can lead to cost savings. We also studied transient cooling effects after abrupt, large changes in data center loads. The results led to the conclusion that pre-cooling is necessary to prevent overheating in these scenarios. We have showed that intelligent placement and migration of HPC load can indeed have to significant cost savings. Further, all electricity-related costs must be considered to maximize and ensure consistent cost savings.

Overall, this dissertation has demonstrated the value of load distribution across data centers, formal optimization techniques, workload prediction, and electricity price prediction for energy management. Given current high energy costs, likely future caps on carbon emissions and/or brown energy consumption, rapid expansion of renewable energy, and emergence of cloud computing platforms, frameworks such as ours should become extremely useful in practice. We further show that load distribution can be used to reduce costs in the context of Web services in Appendix A.

Appendix A

A Cost-Effective Distributed File Service with QoS Guarantees

A.1 Introduction

Large-scale, value-added Internet services composed of independent cooperating or competing services will soon become common place. We refer to these services as *composite services*. Two technology trends suggest this new class of services: the progress toward ubiquitous Internet connectivity even from devices with limited resources, and the increasing adoption of service communication, discovery, and description standards, such as the Simple Object Access Protocol (SOAP), the Universal Description, Discovery and Integration Service (UDDI), and the Web Service Definition Language (WSDL). Together, these trends are forcing functionality and data into the network infrastructure in the form of remotely accessible services.

Composite services promise anytime, anywhere access to powerful services and vast data sets. A composite service may use constituent services that provide complementary functionality or data. For example, a composite stock service might use a service that provides stock quotes in some currency and a service that translates an amount of money (e.g., a stock quote) in one currency into another. In contrast, a composite service may use services that provide the same functionality or data. For example, a composite job-scheduler service might use multiple job-execution services. Regardless of type, we expect that composite services and their constituent services will provide service-level agreements (SLAs) for a monetary charge.

In terms of structure, composite services are organized into a front-end service and multiple independent back-end services. The front-end service monitors and aggregates

the back-end services, whereas the back-end services communicate with the front-end service but not with each other. In the above examples, the stock and job-scheduler services are called front-end services, whereas the stock-quote, currency-exchange, and job-execution services are called the back-end services.

For several years, researchers have been studying composite services in one form or another in the CORBA, Grid, and Web Service communities. These works have mostly focused on the performance, communication protocols, discovery mechanisms, and description of these composite services. Little work has been done on effectively composing paid services and the quality-of-service (QoS) guarantees that they provide.

In this appendix, we address these issues in the context of distributed file storage. In particular, we propose, implement, and evaluate a cost-effective, QoS-aware composite file service comprising a front-end file service and back-end (third-party) storage services. The composite file service is intended to support soft real-time applications that involve large data files, such as the visualization of large-scale scientific data (e.g., [152]). For these applications, it is important to guarantee that data files will be available a large fraction of the time, and that a large percentage of file accesses will be served within a certain amount of time.

The composite service provides "soft" availability and performance guarantees, i.e. in extreme scenarios, such as a network partition separating front-end and back-end services, the guarantees may be violated. When these violations occur, the service compensates users for the violations.

Our front-end service allows users to choose the performance and availability guarantees that they desire on a per-file basis. Based on the chosen availability guarantee, the front-end service replicates the file across the back-end services. Based on both chosen guarantees, the back-end services' behaviors, and their SLAs, the front-end service intelligently distributes the requests across the back-end services to provide the chosen guarantees at low cost.

The front-end service uses mathematical modeling and optimization to carefully orchestrate the accesses to the back-end services. More specifically, the front-end service combines two algorithms: Base and OptWait. Base is reminiscent of traditional job

scheduling. It sends each request to one of the back-end services that replicate the corresponding file, according to a ratio determined by the mathematical machinery to meet the file's performance guarantees while minimizing access cost. In contrast, OptWait is more sophisticated. It may actually send each request to multiple back-end services in turn (starting with the cheaper ones) until the request is satisfied. The amount of time it waits for each service to respond is determined mathematically and depends on the probability that the service will return a reply during that time and on the file's performance guarantee. Because we can mathematically decide on the best algorithm, our composite service picks the best algorithm for each file.

Because our initial focus (and the focus of this appendix) is on the request-distribution aspect of our work, we have implemented a prototype of our composite service with a single front-end file server. The server implements the NFS protocol and executes our mathematical machinery. It communicates with client machines using a standard NFS protocol over UDP, whereas it communicates with back-end services using XML over HTTP. Several Internet storage services, e.g. Amazon.com's S3 [10], could implement the back-end services. However, for greater control of our experiments, we implemented our own back-end services, which provide data blocks named by absolute number.

Experimental results from our prototype implementation validate our modeling and optimization approach. Our analysis of the algorithms studies several different parameters, including the performance and availability guarantees, and the characteristics and behavior of the back-end services. Our most important results show that our composite service is successful at providing the guarantees that it promises. The results also show that, independently, Base and OptWait provide the lowest cost in different parts of the parameter space, whereas our combined system always produces the lowest cost.

A.2 Related Work

Our work builds upon previous research on service composition, QoS-aware resource management, and distributed file and storage systems.

Service composition. This has been an important research topic in the Web Services

community, e.g. [73, 177]. These works typically consider the QoS-aware composition of services from constituent services that provide complementary computational functionality. For this reason, they do not consider request-distribution policies across the services. Our work differs from these efforts as we study request-distribution policies that are both QoS- and cost-aware, across functionally-equivalent constituent services.

QoS-aware resource management. A large body of work has been done on this topic, especially in the context of networks, server clusters, and grid environments, e.g. [34,92,157]. These works consider resource allocation, provisioning, reservation, and negotiation, as well as admission-control policies in guaranteeing QoS (and sometimes optimizing costs) for the systems' users.

The extent of the performance guarantees provided by our composite service is limited to the front-end and back-end services' behaviors, as well as the communication between front-end and back-end services; the composite service cannot provide performance guarantees about the communication between clients and the front-end service. All other works on server-side QoS guarantees have this same limitation. We envision combining our QoS guarantees with those of future networks to completely eliminate this limitation. Nevertheless, an easy approach to tackle this problem with current network technology is to place front-end servers on the same local-area network as clients. In this approach, the front-end server could be an appliance, like today's load balancing or storage appliances.

Although we can benefit from previous QoS works in managing the resources of our front-end service and by leveraging network QoS, this appendix focuses on request distribution across the black-box back-end services, which allow us no control over their resource allocation. In fact, the back-end services can themselves be distributed. The only information about them that we rely upon is their SLAs.

Distributed file and storage systems. Most of the research in distributed file and storage systems has been focused on cluster or local-area network environments, in which resources are dedicated to the system and owned by the same administrative entity, e.g. [61, 136, 160]. Due to their low communication latencies, these systems are amenable to small data and meta-data transfers. In contrast, peer-to-peer file and

storage systems have also become prominent in recent years, e.g. [24, 42, 146]. These works have typically concentrated on achieving extreme performance scalability and availability in the presence of high churn in the online membership of constituent nodes.

Although our composite file service can be seen as a peer-to-peer system in the strictest sense, it lacks a few defining characteristics of previous systems, such as peers that often become unavailable. Further, we are interested in pushing the boundaries of traditional distributed file systems, such as NFS, by using them across the wide area. Two papers have addressed the effect of high latencies on file system traffic [107, 117], but neither of them considered QoS or costs. We expect Internet block-storage services to become widespread in the future, as protocols such as iSCSI become more popular.

Summary of contributions. As far as we know, this work is unique in a few respects. First, our work seems to be the first to focus on cost- and QoS-aware request distribution across third-party services. Second, our OptWait request-distribution algorithm departs from traditional scheduling policies by potentially assigning a request to multiple back-end services in turn. Finally, our approach of considering the entire set of recent response times from each back-end service, rather than using a single metric such as the recent average response time or the maximum recent response time, in mathematically determining request distributions is also novel.

A.3 Our Composite File Service

In this section, we discuss the basic principles behind our composite file service, our request-distribution algorithms, and our current implementation.

A.3.1 Basic Principles

Overview. As already mentioned, our composite file service comprises a front-end file service and a number of back-end block-storage services. The front-end service translates the file system API, e.g. create, read, write, and unlink, into block accesses that are forwarded to one or more back-end services. The front-end service composes the user-requested guarantees from the back-end services at low cost. In fact, even if a

single storage service could provide the required guarantees directly to the user (who could use a local file system and iSCSI, for example, bypassing the front-end service), the composite file service could still provide them for a lower cost, e.g. by forwarding some of the requests to a back-end service with lower cost per access whenever possible.

In our design, the front-end service is implemented by a number of distributed servers for both performance and availability. Each user mounts the file system through one of the front-end servers, which is chosen using a separate Web interface listing all available front-end servers and their geographical locations. The same file system can be mounted concurrently at different front-end servers. However, the front-end service provides no consistency guarantees when read-write and write-write file sharing is not done on the same front-end server. When the same front-end server is used, strong consistency is guaranteed. To guarantee high availability and fault tolerance, all data and meta-data are replicated across several back-end services. Furthermore, the front-end servers only store soft state, such as a disk cache of meta-data, and keep write-ahead logs of updates in the back-end. All files are accessible from an inode-map stored at a few specific back-end services (and cached on the disks of the front-end servers). Thus, if a front-end server fails, the user can mount the file system through another front-end server, which can take over for the failed server using its write-ahead log.

The back-end block-storage services may be provided by different service providers. Although our front-end service treats the back-end services as "black boxes", we do assume that each back-end service is bounded by an SLA with the front-end file service. In particular, each back-end service i promises to meet an availability guarantee of A_i and a performance guarantee of $(P_i,\ L_i)$ at a cost of $(c_i^r,\ c_i^w,\ c_i^s)$. The two guarantees specify that service i will be servicing access requests $A_i\%$ of the time and, when it is available, $P_i\%$ of the accesses will complete within time L_i. The SLAs are defined over a long period of time, say one month, so that short-lived performance anomalies do not cause SLA violations. The cost tuple $(c_i^r,\ c_i^w,\ c_i^s)$ specifies that each read access costs c_i^r, each write access costs c_i^w, and each unit of storage per unit of time costs c_i^s. Table A.1 summarizes the notation used in our modeling.

In computing request distributions, the front-end service uses the availability and

Notation	Definition
A_{front}	Availability of the front-end service
A_i	Availability guarantee provided by back-end service i
(P_i, L_i)	Performance guarantee provided by back-end service i: When service is available, P_i% of requests should be served in L_i time
(c_i^r, c_i^w, c_i^s)	Read, write, and storage costs of back-end service i
A_f	Availability requested by the creator of file f
(P_f, L_f)	Performance requested by the creator of file f: When service is available, P_f% of requests should be served in L_f time
H_f	Set of back-end services that store file f
S_f	Size of file f
r_f, w_f	Expected percentage of reads and writes to file f
R_f, W_f	Actual percentage of reads and writes to file f
P_f^r, P_f^w	Percentage of reads and writes to file f that complete in L_f time
$CDF_i(L)$	Percentage of requests served by back-end service i in L time
p_i	Probability of sending a request to back-end service i (optimized by Base)
(l_i, p_i)	Length of wait at back-end service i and expected percentage of requests served by i during the wait (optimized by OptWait)
$Cost(f)$	Expected monetary cost of serving file f
$AccessCost_t(f)$	Actual monetary cost of serving file f during interval t
$TotalCost(f)$	Actual monetary cost of serving file f over all intervals

Table A.1: Notation and definitions.

cost information from the SLAs with the back-end services. Instead of relying on the performance guarantees provided by the back-end services in computing distributions, we use the latency of requests as observed at the front-end service to encompass the latency of the wide-area network. Specifically, the front-end service monitors the latency of block accesses to each back-end service over two periods of 12 hours per day. The request distributions computed during a period of 12 hours are based on the cumulative distribution function (CDF) of the latencies observed during the same period of the day before. For example, the request distributions computed during the afternoon on Wednesday are based on the latencies observed during the afternoon on Tuesday. This approach is motivated by the cyclical workloads of many Internet services [34]. We plan to investigate more sophisticated approaches for considering block access latencies as future work.

File creation and access. When a file f is first created, the user can specify a desired availability guarantee of A_f and a performance guarantee of (P_f, L_f). (Files for which

the user requests no guarantees are stored at a single back-end service and served on a best-effort basis.) These desired characteristics, if accepted by the front-end service, determine that it must be able to serve access requests to f $A_f\%$ of the time and that $P_f\%$ of the requests must complete within time L_f, when the service is available. If a file access request involves $n > 1$ blocks, the target latency for the request becomes nL_f. Again, these guarantees are defined over a long period of time, e.g. one month.

Obviously, we can only meet the requested availability if the front-end service itself is more available than A_f. If that is the case, it will choose a set of back-end services H_f to host f that meets (or exceeds) A_f. The front-end service randomly selects back-end services from three classes – inexpensive, medium, and expensive – one at a time in round-robin fashion. These classes are likely to correspond to services with generally high, medium, and low response times, respectively, although that is not a requirement. Assuming that failures are independent, the front-end service will select a set of back-end services that satisfies the following inequality:

$$A_{front} \times (1 - \prod_{i \in H_f} (1 - A_i)) \geq A_f \tag{A.1}$$

where A_{front} is the availability of the front-end service. This formulation assumes that the back-end services are always reachable from the front-end service across the network. However, it can be easily replaced by more sophisticated formulations without affecting the rest of the system.

The front-end will choose a minimal set H_f in the sense that, if any back-end service is removed from H_f, the remaining set would no longer be able to meet A_f. Once H_f has been chosen, the front-end service will solve a cost-optimization problem for the two algorithms and choose the one that produces the lowest cost for f.

At this point, file f can be accessed by clients. On a read to f, the front-end service will forward a request to a subset of H_f for each needed block according to the chosen algorithm. On a write, the front-end will forward the request to all back-end services in H_f to maintain the target data availability, while concurrently writing to the write-ahead log if necessary. The front-end service only waits for the possible write ahead and one back-end service to process the write before responding to the client. In the

background, the front-end service will ensure that the write is processed by the other back-end services in H_f as well. When write sharing is done through the same front-end server, this approach to processing writes favors lower latency without compromising strong consistency; the pending writes can be checked before a subsequent read is forwarded to the back-end.

Optimizing costs. Our request-distribution algorithms, Base and OptWait, are run by the front-end service to minimize the cost of accessing the back-end services in H_f. As mentioned above, their respective optimization problems are solved at first during file creation, but they may need to be solved again multiple times over the file's lifetime. In particular, whenever the file is opened, a new distribution is computed but only if the current distribution is stale, i.e. it was not computed based on the same period of the day before. After the back-end services are selected and the request distribution is computed, the front-end service can inform the client about the cost of each byte of storage and the (initial) average cost of each block access, given the requested guarantees. Note that the cost of accessing the write-ahead logs is not included in the cost computations; this cost is covered by our service fees (discussed below).

Because we select the H_f back-end services randomly from three classes of services, our cost optimization produces a "locally" optimal cost; it is possible that this cost will not be the lowest possible cost (i.e., the "globally" optimal cost) for a system with a large number of back-end services. Attempting to produce the lowest possible cost would involve searching an exponentially large space of back-end service groupings, which could take hours/days of compute time to explore meaningfully, even if a heuristic algorithm were to be used. We plan to explore this issue in our future work.

The front-end accumulates the access costs accrued during the periods of stable request distribution, i.e. in between consecutive changes to the request distribution. The overall cost of the composite service is then the sum of the costs for each stable period. Periodically, say every month, the front-end service charges each of its users based on how many accesses and how much storage the front-end service required of its

back-end services on behalf of the user. Formally, the total cost to be charged is:

$$TotalCost(f) = \sum_{\forall_t} AccessCost_t(f) + S_f \sum_{i \in H_f} c_i^s \tag{A.2}$$

where $AccessCost_t(f)$ is the access cost of each period t of stable request distributions since the last calculation of $TotalCost(f)$ and S_f is the maximum size of the file since the last calculation of $TotalCost(f)$. We define $AccessCost_t(f)$ exactly below.

Service fees and compensation. Finally, note that the costs incurred by the front-end service are actually higher than the sum of $TotalCost(f)$ for all files. As mentioned above, the cost of accessing the write-ahead logs is not included in $TotalCost(f)$. In addition, when the client load is low, the front-end service may need to send additional accesses to the back-end services to properly assess their current performance (and availability). These extra accesses increase costs for the front-end service; the extra cost can be amortized across the set of users as a "service fee".

Further, there may be situations in which the guarantees provided by the front-end service are violated. For example, the network between the front-end service and some of the back-end services may become unusually slow or back-end services may start violating their SLAs. As mentioned above, the front-end service responds to these situations by recomputing its request distributions accordingly, but the recomputations may not occur early enough. Nevertheless, in case of back-end SLA violations, the front-end service will be compensated for them and the compensations can be passed on to its users. In case of network problems, the front-end service can use its service fees to compensate users.

A.3.2 Base

In Base, a read request to a file f is forwarded to a single back-end service $i \in H_f$ with probability p_i. (Writes are sent to all back-end services in H_f.) Base computes these probabilities so as to minimize the cost of servicing accesses to f while respecting the performance guarantees requested for the file. Formally, Base needs to minimize:

$$Cost(f) = r_f \sum_{i \in H_f} p_i c_i^r + w_f \sum_{i \in H_f} c_i^w \tag{A.3}$$

subject to the following two constraints:

$$1.\ \forall i \in H_f, p_i \geq 0 \text{ and } \sum p_i = 1 \qquad 2.\ r_f P_f^r + w_f P_f^w \geq P_f$$

where r_f is the fraction of read block accesses to f, w_f is the fraction of write block accesses to f, P_f^r is the percentage of read accesses that complete within L_f, and P_f^w is the percentage of write accesses that complete within L_f.

Equation A.3 computes the average cost of reads and writes, reflecting the read-to-write ratio ($r_f : w_f$), and the fact that each read incurs the cost of only 1 back-end access according to the probabilities p_i (hence $p_i c_i^r$), while each write incurs the cost of accessing all back-end services. Constraint 1 states that the probabilities of accessing each back-end service in H_f have to be non-negative and add up to 1. Constraint 2 requires that the percentage of reads and writes that complete within L_f time must be at least P_f to meet the guarantees requested by the user.

We then define P_f^r and P_f^w as:

$$P_f^r = \sum_{i \in H_f} p_i CDF_i(L_f) \qquad P_f^w = \max_{i \in H_f}(CDF_i(L_f)) \tag{A.4}$$

where the $CDF_i(L)$ operator produces the percentage of requests satisfied within L time by back-end service i, as observed at the front-end service. P_f^w is determined by the best performing back-end service because the front-end forwards each write in parallel to all back-end services and replies to the client when the first one completes.

Equations A.3 and A.4 together with the two constraints completely define Base's optimization problem, except for how to determine r_f and w_f. The user can optionally estimate r_f and w_f and pass them as parameters at file creation time. If the user does not provide this information, we split constraint 2 above into two parts, $P_f^r \geq P_f$ and $P_f^w \geq P_f$, and instantiate Equation A.3 with the assumption that $r_f = 1$ and $w_f = 0$. This approach correctly but conservatively ensures that the solution to the optimization problem provides the required guarantees for f.

After each period t of stable request distributions computed by Base, we compute the cost of accessing the H_f back-end services during the period as:

$$AccessCost_t(f) = R_f \sum_{i \in H_f} p_i c_i^r + W_f \sum_{i \in H_f} c_i^w \tag{A.5}$$

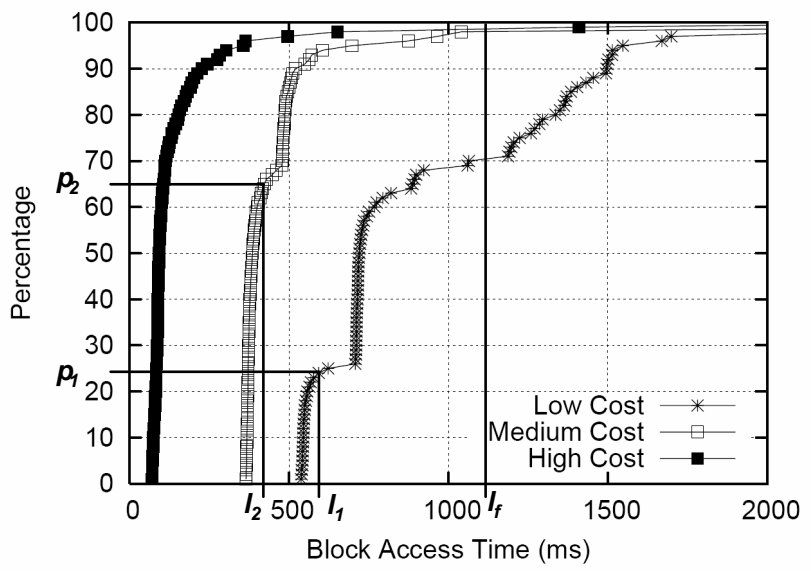


Figure A.1: Performance CDFs for three services. An OptWait distribution might specify that a request should be forwarded to multiple back-end services in turn.

where R_f is the number of read requests and W_f is the number of write requests serviced during period t.

Finally, note that a malicious client is not able to lower its access costs by providing fake values for r_f and w_f, since these costs are computed based on the actual requests made by the client during each period of time.

A.3.3 OptWait

In OptWait, the front-end service takes the different approach of possibly forwarding a read request to more than one back-end service. In particular, the front-end service forwards each read request to the back-end services in sequence, from least to most expensive, waiting for a bounded amount of time for each service to respond before trying the next service.

The basic idea behind OptWait is illustrated in Figure A.1, which shows three performance CDFs for three back-end services. Let us assume that the left-most curve represents the most expensive service, whereas the right-most curve represents the least expensive service. OptWait would first forward a request to the least expensive service,

waiting for an amount of time l_1. This would allow OptWait to take advantage of the percentage of requests (p_1) that complete fairly quickly. If the request did not complete within l_1 time, OptWait would then forward the request to the medium-cost service and wait for some wait time l_2. Again, the goal would be to leverage the steep part of the medium-cost service's CDF. If, after $l_1 + l_2$ time, the request still had not completed at either back-end service, OptWait would then forward the request to the most expensive service and wait for the request to complete at any of the three back-end services.

The key to OptWait is setting appropriate l_i times. Like in Base, we do so by optimizing the access cost under the performance constraints imposed by the guarantees requested by the user. Assuming H_f with 3 back-end services, our problem is to minimize the following equation:

$$\begin{aligned} Cost(f) = r_f[&p_1 C_1 \\ &+((1 - CDF_1(l_1 + l_2))p_2 + CDF_1(l_1 + l_2) - p_1)(C_1 + C_2) \\ &+(1 - (1 - CDF_1(l_1 + l_2))p_2 - CDF_1(l_1 + l_2))(C_1 + C_2 + C_3)] \\ &+w_f \textstyle\sum_{i \in H_f} c_i^w \end{aligned} \tag{A.6}$$

where $p_i = CDF_i(l_i)$, $CDF_i(l) = 0$ when service i is not being used for reads (i.e., $l_i = 0$), $C_i = 0$ when service i is not being used for reads and $C_i = c_i^r$ when it is, and $l_i = \infty$ when i is the last service being used for reads.

Equation A.6 computes the cost of writes in the same manner as the Base cost function (Equation A.3), as the two algorithms treat writes in the same way. More interestingly, it computes the cost of reads by summing up the multiplication of the probability that each back-end service will need to be accessed by the cost of doing so. For example, if services 1 and 2 are used for reads, the first two lines of the equation compute the cost, whereas the third line becomes 0. The first line multiplies the probability that service 1 replies within l_1 time (p_1) by the cost of accessing service 1. For the requests that are not serviced by service 1 within l_1, service 2 would be activated. Thus, the second line of the equation sums up the probability that service 1 does not reply within $l_1 + l_2$ time but service 2 does reply within l_2 time ((1 −

$CDF_1(l_1 + l_2))p_2)$, and the probability that service 1 replies after l_1 but before $l_1 + l_2$ time ($CDF_1(l_1 + l_2) - p_1$). The second part of the cost is obtained by multiplying this probability by the cost of making one access to service 1 and one access to service 2.

Equation A.6 should be minimized subject to the following constraints:

$$1.\ \forall i \in H_f, l_i \geq 0 \qquad\qquad 2.\ r_f P_f^r + w_f P_f^w \geq P_f$$

where constraint 1 simply states that times have to be non-negative and constraint 2 is the same as that for Base. (Just as for Base, the front-end service can break constraint 2 into two parts and compute costs for $r_f = 1$ and $w_f = 0$, if the user does not provide information about r_f and w_f as a parameter.) We define P_f^w just the same as for Base, since the two algorithms handle writes in the same way. In contrast, P_f^r is defined as:

$$\begin{aligned} P_f^r = {} & CDF_1(L_f) \\ & +(1 - CDF_1(L_f))CDF_2(L_f - l_1) \\ & +(1 - CDF_1(L_f))(1 - CDF_2(L_f - l_1))CDF_3(L_f - l_1 - l_2) \end{aligned} \tag{A.7}$$

where again $CDF_i(l) = 0$ when service i is not being used for reads.

In plain English, the first additive component of Equation A.7 represents the probability that the least-expensive service will reply in a timely manner (within L_f time) if it is used, the second component is the probability that service 2, if used, will reply in a timely manner (given that a request is only forwarded to it after l_1 time) but not service 1, and so on.

After each period t of stable request distributions computed by OptWait, we compute the cost of accessing the H_f back-end services during the period by replacing r_f and w_f in Equation A.6 by R_f and W_f, respectively.

A.4 Implementation

We have implemented a prototype front-end file service called Figurehead to explore our request-distribution algorithms in real systems with real workloads. Although Figurehead should be supported by multiple geographically distributed servers in practice, it is currently based on a single node as a proof-of-concept implementation.

Figurehead consists of four components: an NFS version 2 facade that allows the file service to be accessed through standard NFS clients, a file system that supports the NFS facade and uses remote back-end block services for storage, an optimization module that computes the best request distribution strategy, and a module that constantly monitors the performance of the back-end services. All components were written in Java and run in user space. Relevant details about these four components are as follows.

NFS facade. The multi-threaded NFS facade accepts NFS remote procedure calls via UDP. It implements the NFS version 2 protocol almost completely; the only calls that have not been implemented are those dealing with symbolic links.

The one complication that the NFS protocol poses for Figurehead is that opens and closes are not sent through to the server. Thus, whenever the NFS facade receives a create or the first access to an unopened file, it opens the file and caches the opened-file object returned by the file system. A cached opened-file object is closed and discarded after it has not been accessed for 5 minutes.

File system. The file system behind our NFS facade uses the same meta-data scheme to represent a file as the Linux ext2 file system. The inode was changed to include information about the availability and performance guarantees requested by the creator of a file. An inode-map maps each inode to the set of back-end services that is hosting the file. All data and meta-data except for the inode-map are stored at the back-end services in 8-KByte blocks. The file system communicates with the back-end services over a Web Service interface, namely the RPC implementation from Apache Axis [15].

When a file is first created, the file system chooses a set of back-end services to host the file as described in Section A.3.1. It then allocates an inode, saves the availability and performance guarantees for the file in the inode (along with other traditional file-system information, such as owner and time of creation), enters the mapping of $\textit{inode-number} \rightarrow H_f$ into its inode-map, and writes the inode to the appropriate back-end services. The file system also opens the file.

When a file is opened, the file system extracts the set of back-end services that is hosting the file (H_f) from the inode-map, obtains their access time CDFs from the monitoring module, reads the inode to obtain the performance guarantees, and asks

the request distribution module to compute the best request distribution strategy for the file. This last step is not necessary when the file is being re-opened and the current request distribution was computed based on the same period of the day before. To determine whether to recompute a request distribution, Figurehead maintains information about when each distribution is computed. When a previous request distribution exists but a new computation is required, the computation is performed in the background and adopted when completed. When client requests arrive, the file system uses the file meta-data to identify the corresponding blocks and forwards the appropriate block operations to the back-end services. Reads are handled according to the current request distribution, whereas writes are forwarded to all back-end services in H_f.

The file system maintains a write buffer to ensure that each write to a file f eventually reaches all of the nodes in H_f. When a write request arrives, the file system assigns a thread per back-end service in H_f the task of ensuring that the write eventually reaches a particular back-end. Each write is then discarded from the write buffer once it has propagated to all back-ends in H_f. We assume that the back-end services can handle small "overwrites;" that is, a write that only partially overwrites a previously written block can be sent directly to the back-end services without having to read the old data and compose a new complete-block write. This avoids making small overwrites more expensive than a complete-block write because of the need to read the block.

The file system implements two levels of meta-data caching. First, all meta-data is currently cached on a local disk (and is never evicted) using a Berkeley database [120]. This cache reduces the number of accesses to the back-end services by eliminating repeated remote meta-data accesses. In fact, the cache makes the meta-data accesses to the back-end services relatively infrequent for the large-file applications we target (dominated by reads and/or overwrites), so these accesses are not currently reflected in our mathematical machinery. Second, file-specific meta-data, i.e. inodes and indirect blocks, are cached in memory for open files as the meta-data is accessed. This avoids repeatedly accessing the cache on disk for a stream of accesses to the same file. Meta-data of an open file that is cached in memory is evicted when the file is closed. Our

policy of holding a file opened in the NFS facade for 5 minutes beyond its last access implies that meta-data for an open file is also cached in memory by the file system for the same amount of time.

Finally, since the NFS clients cache data themselves, our file system (in fact, the entire front-end service) does not cache data at all.

Request-distribution module. This module solves the optimization problems posed by Base and OptWait, and chooses the algorithm that produces the lowest cost. The Base optimization problem is solved using the linear programming solver lp_solve [23] and produces the p_i probabilities with a precision of a few decimal places. Unfortunately, minimizing cost in OptWait is not a linear programming problem. To solve it, we consider all feasible combinations of the probabilities p_i's (in steps of 1% in our current implementation) for the back-end services in H_f to compute the best l_i's wait times. Even though this is essentially a brute force approach, it does not take long to compute as the size of H_f is small (typically two or three), even for high P_f requirements. We report running times for this module in Section A.5.

Monitoring module. This module is responsible for monitoring each back-end service in terms of its performance as seen at the front-end service. Specifically, this module probes each back-end service periodically with regular block accesses (every 5 seconds in our current implementation). With the access times measured from these accesses, this module constructs the performance CDF for the service.

Figurehead limitations. Currently, Figurehead has three limitations. First, as we mentioned above, it is implemented by a single server, rather than a collection of geographically distributed servers. Second, we have not yet implemented the write-ahead log for crash recovery. Third, the monitoring module currently does not use information from regular accesses to the back-end services, always issuing additional block accesses to assess their performance (and availability). These extra accesses increase costs and would not be required when the regular load on the back-end services is high enough. We are currently addressing these limitations.

A.5 Evaluation

In this section, we first explore and compare the two request distribution algorithms over the space of different costs and back-end service behaviors. We then study the impact of using past access time data to predict current behaviors of the back-end servers. Finally, we evaluate our prototype Figurehead implementation, and validate that it provides the performance guarantees computed by the mathematical machinery.

Ideally, we would like to study our system using actual back-end services on the Internet. However, at this point, there are not enough of them to provide a large range of data. Thus, we have collected access times over a period of close to one month from 50 PlanetLab machines to support our evaluation. These data were collected by running a simple block-storage service on each machine, populating each service with 5120 blocks, and randomly accessing a block according to a Poisson process with mean inter-access time of 1 second from a client machine located at our site.

A.5.1 Base vs. OptWait

We first compare Base and OptWait mathematically assuming fixed access time CDFs for the back-end services. In particular, we chose data from three PlanetLab nodes, `planetlab2.cs.umass.edu`, `planetlab1.cs.unibo.it`, and `planet-lab.iki.rssi.ru`, whose CDFs are shown in Figure A.1. We study a set of three nodes because they provide a sufficiently rich space to understand the behaviors of the two algorithms, yet is not overly complicated to explain.

Overall results. Figure A.2 plots the average cost ($Cost(f)$) achieved by Base and OptWait for a read-only workload as a function of the per-file guaranteed latency (L_f), with a per-file percentage guarantee (P_f) of 95%. (The results are similar for other P_f values.) Each of the curves represents a different combination of algorithm and per-access cost for each back-end service. For example, the curve labeled OptWait [5,10,15] represents the cost computed by OptWait when $c_1^r = 5, c_2^r = 10$, and $c_3^r = 15$ fractions of dollar per access (what fraction exactly is irrelevant to our study). Table A.2 lists the optimized costs and request distributions for Base and OptWait for costs [5,10,15].

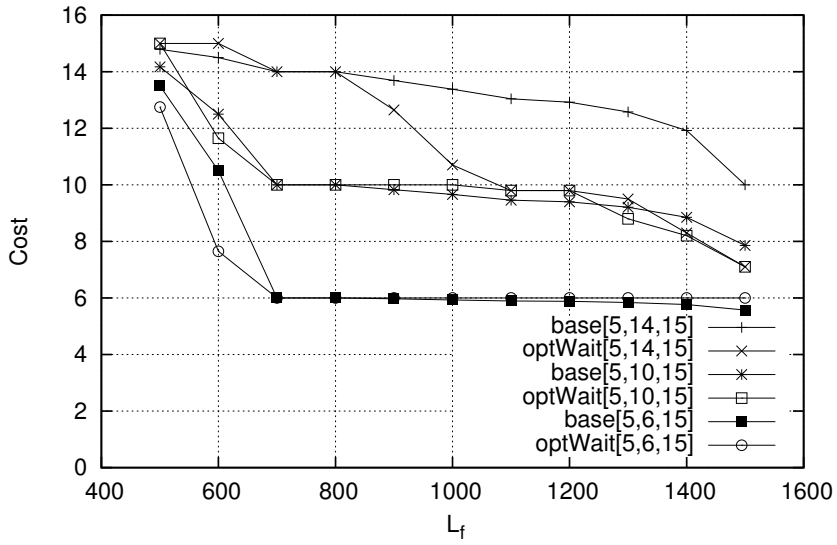


Figure A.2: Costs achieved by Base and OptWait vs. L_f, assuming a read-only workload and $P_f = 95\%$.

From these figures, we can see that neither Base nor OptWait is always better than the other. At the extremes, i.e. at very low or very high latency guarantees, the two algorithms behave the same because there is no room for optimization. For very low latency guarantees, the only choice is to use the most expensive service all the time (if it is possible to meet the guarantee at all). For very high latency guarantees, the obvious best choice is to use the cheapest service all the time.

In between these extremes, the relative behavior of Base and OptWait depends on the shapes of the access time CDFs of the back-end services, as well as their costs. For example, consider the costs achieved by Base and OptWait for cost [5, 10, 15] at latency guarantees of 500ms and 600ms. At 500ms, Base achieves lower cost than OptWait because it is able to use the medium-cost service 17% of the time, whereas OptWait cannot yet use the medium-cost service (see Table A.2). In this case, for p_2 in OptWait to be greater than 0, l_2 would have to be at least 365ms, leaving insufficient time for accessing the high-cost service should the request fail to complete at the medium-cost service within l_2. At 600ms, OptWait does better than Base because its greater use of the medium-cost service, 89% vs 50%, more than offsets the 11% of the time that it has to use both the medium-cost and high-cost service.

L_f (ms)	Base Cost	Base Dist	OptWait Cost	OptWait Dist
500	14.17	[0,17,83]	15.00	[(0,0),(0,0),(∞,100)]
600	12.50	[0,50,50]	11.65	[(0,0),(511,89),(∞,100)]
700	10.00	[0,100,0]	10.00	[(0,0),(∞,100),(0,0)]
800	10.00	[0,100,0]	10.00	[(0,0),(∞,100),(0,0)]
900	9.83	[3,97,0]	10.00	[(0,0),(∞,100),(0,0)]
1000	9.66	[7,93,0]	10.00	[(0,0),(∞,100),(0,0)]
1100	9.46	[11,89,0]	9.80	[(923,68),(0,0),(∞,100)]
1200	9.40	[12,88,0]	9.80	[(923,68),(0,0),(∞,100)]
1300	9.21	[16,84,0]	8.80	[(794,62),(∞,100),(0,0)]
1400	8.85	[23,77,0]	8.20	[(923,68),(∞,100),(0,0)]
1500	7.86	[43,57,0]	7.10	[(1404,86),(0,0),(∞,100)]
1600	5.00	[100,0,0]	5.00	[(∞,100),(0,0),(0,0)]

Table A.2: Costs and distributions with back-end service costs = [5,10,15] and P_f = 95%. The Base distributions are listed as [p_1, p_2, p_3], whereas the OptWait distributions are listed as [(l_1, p_1), (l_2, p_2), (l_3, p_3)]. l_1, l_2, l_3 are given in ms.

In general, we observe that Base can typically start using a lower-cost back-end service before OptWait as the guaranteed response time increases. This is because Base never resends requests. However, eventually, OptWait can use the lower-cost service more aggressively because it can avoid the tail of the CDF by re-sending requests to the more expensive services as needed.

Impact of the back-end service costs. Observe that Base's distribution of requests is independent of the ratio between the costs of the three back-end services. That is, as long as $c_3^r > c_2^r > c_1^r$, Base will choose the same set of distribution probabilities (p_1, p_2, p_3) regardless of the ratios c_1:c_2:c_3. OptWait, on the other hand, may alter its distribution strategy based on the cost ratios. For example, consider in Table A.3 the distributions computed for L_f within the interval [1200ms, 1400ms] for costs [5, 6, 15] vs. [5, 10, 15]. For [5, 10, 15], OptWait chooses to use either the low- and medium-cost or low- and high-cost services. For [5, 6, 15], OptWait only chooses to use the medium-cost service. This is because the medium-cost service is only slightly more expensive than the low-cost service; immediately choosing it is less costly than potentially having to forward the request to two services.

Impact of the shape of the CDFs. Base and OptWait also behave differently with

L_f (ms)	Back-End Costs	Base Cost	Base Dist	OptWait Cost	OptWait Distribution
1200	[5,10,15]	9.40	[12,88,0]	9.80	[(923,68),(0,0),(∞,100)]
1200	[5,6,15]	5.88	[12,88,0]	6.00	[(0,0),(∞,100),(0,0)]
1200	[5,14,15]	12.92	[12,88,0]	9.80	[(923,68),(0,0),(∞,100)]
1300	[5,10,15]	9.21	[15.79,84.21,0]	8.80	[(794,62),(∞,100),(0,0)]
1300	[5,6,15]	5.84	[15.79,84.21,0]	6.00	[(0,0),(∞,100),(0,0)]
1300	[5,14,15]	12.58	[15.79,84.21,0]	9.50	[(1064,70),(0,0),(∞,100)]
1400	[5,10,15]	8.85	[23.08,76.92,0]	8.20	[(923,68),(∞,100),(0,0)]
1400	[5,6,15]	5.77	[23.08,76.92,0]	6.00	[(0,0),(∞,100),(0,0)]
1400	[5,14,15]	11.92	[23.08,76.92,0]	8.30	[(1285,78),(0,0),(∞,100)]

Table A.3: Costs and distributions with $P_f = 95\%$, as a function of L_f and back-end service costs. The Base distributions are listed as [p_1, p_2, p_3], whereas the OptWait distributions are listed as [(l_1, p_1), (l_2, p_2), (l_3, p_3)]. l_1, l_2, l_3 are given in ms.

respect to the shapes of the CDFs. In general, Base's behavior depends on the three key points $CDF_1(L_f)$, $CDF_2(L_f)$, and $CDF_3(L_f)$, whereas OptWait's behavior depends on the shape of all CDFs between 0% and $CDF_i(L_f)$. These dependencies can be seen clearly in Figures A.3 and A.4. Figure A.3 shows the CDFs for 4 back-end services from which we derived two sets of three services {low-cost-1, medium-cost, high-cost} and {low-cost-2, medium-cost, high-cost}.

Figure A.4 shows that OptWait behaves significantly better when using low-cost-2 in the interval [600ms, 1600ms] because low-cost-2 is substantially "steeper" than low-cost-1. Base is also able to leverage low-cost-2's better behavior to improve its cost, but less so than OptWait. The reason is that Base only leverages the fact that low-cost-2 gives a better $CDF_1(L_f)$ than low-cost-1, rather than the fact that low-cost-2 gives an additional 30% of requests completing under 700ms over low-cost-1 in this interval.

A.5.2 Validating the Mathematical Machinery

We now validate our mathematical approach when servicing actual file system workloads. We also validate that the prediction of back-end service behaviors using past access time data do not significantly degrade our QoS guarantees. First, we use simulation to analyze the mathematical approach independent of the details of an actual implementation. Next, we evaluate our prototype implementation.

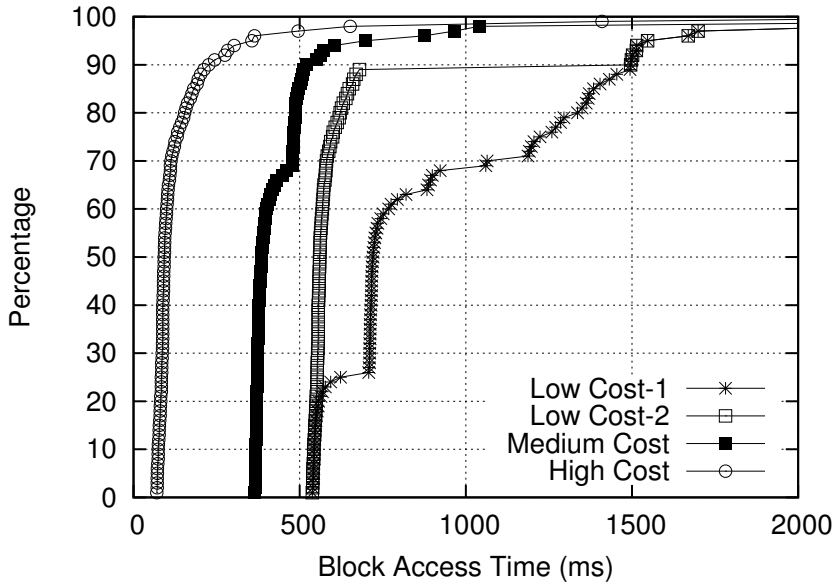


Figure A.3: (a) CDFs for 4 back-end services.

Workloads. We use two realistic workloads. The first models an interactive visualization application, where the user is navigating through a large amount of data–for example, a large rendering model or large scientific data set. This application is exactly the type of soft real-time application that Figurehead is designed to support.

This workload is constructed based on publications on visualization systems [9, 40, 175], and has the following attributes: a random Poisson read access stream with a mean interarrival time of 50ms on a large data file. It currently does not make a difference to Figurehead whether a read stream is random or sequential, since Figurehead does not currently do any prefetching or caching. We assume a random read access stream because these accesses are dependent on the user's interactive navigation.

The second workload models a scientific application running on a grid environment. Although this is not a classical soft real-time application, it still constitutes an interesting workload because predictability of data access can significantly reduce the burden of resource management and coordination of the stages of a multi-stage application such as the one described in [159].

This workload is constructed based on data extracted from [118, 152, 159, 169], and has the following attributes: a sequential read access stream from a single large file

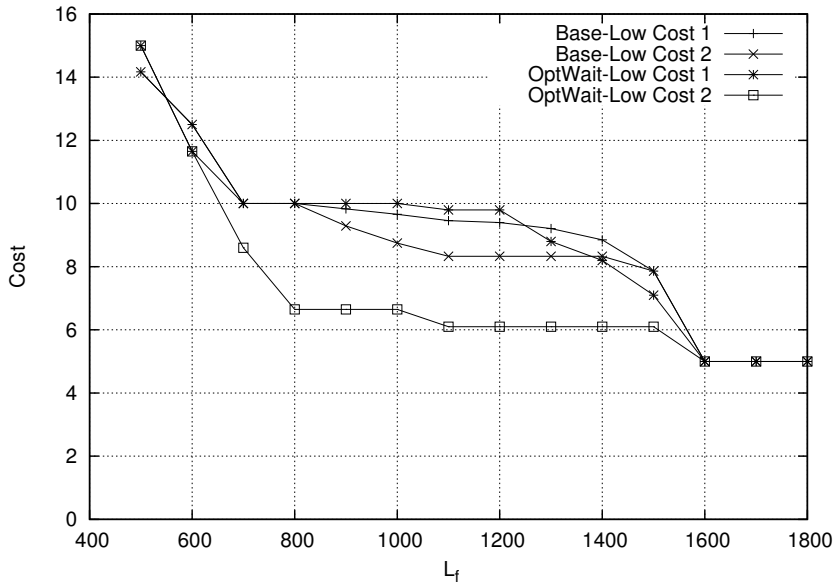


Figure A.4: Access cost achieved by Base and OptWait when using two different sets of three back-end services {low-cost-1, medium-cost, high-cost} and {low-cost-2, medium-cost, high-cost}. Both with cost [5,10,15] and $P_f = 95\%$.

followed by a sequential write access stream to the same file. This read/write access stream represents a multi-phase application with an initial read phase to load input data and a final write phase that saves the computed results. We assume that intermediate results generated between the initial and final phases are stored on local storage rather than a file system such as Figurehead. We further assume that the initial input data and the final results have the same size; thus, the read-to-write ratio is 1:1. Finally, both the read and write access streams are Poisson processes with mean interarrival times of 50ms.

Because the WAN latencies we consider are larger than 50ms, we assume that the access streams of both applications are generated by a number of concurrent threads.

Simulation using *a priori* knowledge of back-end service behaviors. Our first experiment is as follows. Take a trace of the three machines whose overall behaviors are shown in Figure A.1 over a period of 9 days. Construct a CDF for each back-end service for each 12-hour period of the 9 days. For each 12-hour period, use the corresponding CDF to compute the distribution using Base and OptWait for $P_f = 95\%$, $L_f = 600ms$, costs [5,10,15], and $c^r = c^w$ for all back-end services. Then, simulate Figurehead's

		Min	Max	Avg
V-B	**Expected**	95	95	95
	Simulated	95.06	95.89	95.48
S-B	**Expected**	95	95	95
	Simulated	95.28	97.28	96.07
V-O	**Expected**	95.2	97.36	96.22
	Simulated	95.61	97.89	96.57
S-O	**Expected**	95	97.33	95.7
	Simulated	95.56	98.56	96.78

Table A.4: Simulation results with $(P_f, L_f) = (95\%, 600\text{ms})$ and costs [5,10,15]. **V** denotes the visualization workload, **S** the scientific workload, **B** the Base algorithm, and **O** the OptWait algorithm. **Expected** is the percentage of requests expected to complete before L_f as computed by the algorithm. **Simulated** is the actual percentage of requests that completed before L_f in a simulation run. **Min**, **Max**, **Avg** are the minimum, maximum, and average values across 18 runs using 18 half-day traces from the PlanetLab machines.

response time for 18000 accesses for each workload using the 12-hour traces that were used to construct the CDFs. This corresponds to statistical oracular knowledge of the behaviors of the back-end services.

Table A.4 shows the results for 18 runs of each application/distribution algorithm pair, where each run was performed using a distinct half-day period of the 9-day trace. For both workloads under Base and OptWait, the simulation always leads to exceeding the QoS guarantee. This is because we construct and use the CDFs in a conservative manner. In particular, each CDF is represented by a set of 100 discrete points, representing the latency corresponding to each percentage point on the CDF. Now suppose that the mathematical engine needs a percentage value corresponding to the latency 1000ms. If our CDF has the points (999ms, 95%) and (1001ms, 96%), then we would return 95%, rather than an interpolated value between 95% and 96%. We choose this conservative approach because an interpolated value would be optimistic sometimes but pessimistic other times, making the mathematical machinery less predictable.

An additional interesting observation to make is that mathematically, Base always achieves a distribution that should theoretically give the exact P_f required (in this case, 95%). OptWait, on the other hand, because of our discrete approach for computing the best distribution, typically overachieves compared to the required P_f. (Note that,

		Min	Max	Avg	Failures
V-B	**Expected**	95	95	95	0
	Simulated	92.72	97.72	95.35	5
S-B	**Expected**	95	95	95	0
	Simulated	94.28	98.33	96.11	6
V-O	**Expected**	95.2	97.36	96.24	0
	Simulated	93.83	98.83	96.42	5
S-O	**Expected**	95	97.33	95.76	0
	Simulated	95.67	98.61	96.8	0

Table A.5: Simulated results for $(P_f, L_f) = (95\%, 600\text{ms})$ and costs [5,10,15] when using data access times from 12 hours ago to predict the current behaviors of back-end services. The notation is the same as in Table A.4. **Failures** is the number of 12-hour simulation runs that did not meet the QoS guarantee.

for $L_f = 600ms$, OptWait achieves lower cost than Base despite this overachievement.) As shall be seen, this overachievement makes OptWait more robust when the CDF is computed based on past data.

Impact of using past access times to predict current back-end service behaviors. We now consider the impact of not having *a priori* information on the expected behaviors of the back-end services. In particular, as mentioned in Section A.3.1, we run the same experiments as above but use a CDF constructed from the response times observed in the same 12-hour period 1 day ago to predict each back-end service's behavior in the current 12-hour period (e.g., 8am-8pm from Tuesday to predict behavior for 8am-8pm Wednesday). Table A.5 shows the results for 16 12-hour runs (we could not use the first two half-day periods because they did not have any past history for prediction).

As expected, past data is not a perfect predictor of current behavior. This leads to a number of 12-hour simulation runs where Figurehead would not be able to achieve the QoS guarantee. In fact, approximately 35% of the runs missed the QoS guarantee under Base. OptWait has a comparable failure rate for the Visualization workload but was perfect for the Scientific workload. As already mentioned, OptWait is somewhat more resilient to the imperfect predictor because it typically overachieves compared to the required P_f. On the other hand, the imperfect predictor can also lead the 12-hour runs to achieve more than the QoS requirement, i.e. more than P_f of the requests

complete within L_f time. In fact, the **Max** values for both Base and OptWait are larger in Table A.5 than in Table A.4.

However, the most important observation here is that *both request-distribution algorithms provide the performance guarantees that they promise* when the entire 8 day period is considered (see the simulated **Avg** entries). (Recall that QoS guarantees are defined over long periods of time, such as one month.) The reason for this result is that the QoS requirement is exceeded during the majority of the 12-hour periods, which more than compensates for the many fewer periods when the requirement is not met.

A.5.3 Prototype Behavior

We now validate that our prototype, Figurehead, actually provides the performance guarantees computed by the mathematical machinery. All results reported below were obtained by running on 5 PCs connected by a Gb/s Ethernet switch. Each PC is configured with 1 hyper-threading Intel Xeon 2.8 GHz processor, 2 GBytes of main memory, and 25 GBytes of disk space. Three of the machines were used as back-end block servers and one as the client. The other machine ran Figurehead. We always assume that the three back-end services are needed to meet the client's specified availability requirement. Again, all the experiments assume $P_f = 95\%$, $L_f = 600ms$, costs [5,10,15], and $c^r = c^w$ for all back-end services. To mimic a wide-area network, we inserted delays to the completion times of accesses to the back-end services. We use the same 9-day trace as in the last subsection; the delays were randomly chosen from the appropriate half-day period. (We used the traces instead of running the back-end services themselves on PlanetLab nodes for repeatability.)

Microbenchmarks. We first present results from microbenchmarks to illustrate the performance of Figurehead. For these microbenchmarks, we did not inject any network delays so that performance reflects what is achievable over a LAN. We also assume that r_f and w_f are known ahead of time; i.e., r_f is 1 when measuring read performance and 0 when measuring write performance. We measured write performance for appends (rather than overwrites) to a file.

Using these microbenchmarks, we find that the times required to read and write

1 byte of data are approximately 30ms and 66ms, respectively. Appends are more expensive than reads because they require writing meta-data. Overall, Figurehead reads and writes are about one order of magnitude slower than on a local disk. The higher access latency of Figurehead arises mainly from using a Berkeley database as disk cache and the Web Services interface to access the back-end block servers. These inefficiencies can be easily eliminated in a production-grade implementation. However, the fairer comparison is between accessing a back-end service through Figurehead and accessing it directly, both on a WAN. Because network trips dominate in this scenario, Figurehead would impose a much lower overhead. For example, the lowest average latency we measured for the PlanetLab nodes is 165ms. Given this latency, Figurehead would impose roughly a 30% degradation when all accesses are appends.

Another important issue is the overhead of computing request distributions. The time to solve a Base and OptWait optimization problem is approximately 710us and 14ms, respectively. We found that, while the time to solve OptWait does increase with L_f, it does so quite modestly. The reason for the slight time increase is that a higher L_f tends to generate a larger search space in OptWait. Finally, these optimization times do not change significantly with changing P_f and so we do not show those results here.

Macrobenchmarks. Finally, we ran the two workloads described in the last section concurrently against a running instance of our Figurehead prototype. We ran each workload/distribution algorithm pair 4 times, each time for a distinct half-day period from the 9-day trace (the first 4 half-day periods). Overheads from the system (e.g, computing time inside the Figurehead front-end) led to a degradation in meeting the QoS requirement P_f by almost nothing to at most 1%. Detailed measurements show that the main sources of overheads were synchronization delays, inaccuracies in the sleep function used to emulate WAN latencies, and accessing the Berkeley DB. Despite these overheads, *the prototype consistently provides the proper guarantees when all the periods are considered.*

A.6 Summary

In this appendix, we addressed the issue of composing functionally-equivalent, third-party services into higher level, value-added services by developing a distributed file service. In this context, we proposed two request-distribution algorithms that optimize costs at the same time as providing performance and availability guarantees. To achieve this goal, both algorithms rely on information about the behavior of the third-party services to mathematically determine the request distributions. While one algorithm is reminiscent of traditional scheduling policies, the other departs significantly from these policies, as it may schedule the same request at multiple third-party services in turn.

We found that both algorithms provide the guarantees that they promise. Comparing the algorithms, we found that neither is consistently the best. Nevertheless, using our mathematical modeling, the system can actually select the best algorithm for each file a priori. Experimental results from our prototype implementation characterized its performance and the optimized access costs under the two algorithms.

Composite services such as the one we studied are in the horizon. Based on our experience and results, we believe that these services can benefit from our modeling and optimization approach for guaranteeing quality-of-service at low cost.

References

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy proportional datacenter networks. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[2] acpi.info. Advanced configuration and power interface specification, Retrieved in November 2012. `http://www.acpi.info/spec50.htm`.

[3] E. I. Administration. Average Retail Price of Electricity to Ultimate Customers by End-Use Sector, by State. `http://www.eia.doe.gov/cneaf/electricity/epm/table5_6_b.html`.

[4] Affordable Internet Services Online, Retrieved in March 2008. `http://www.aiso.net/`.

[5] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[6] U. E. P. Agency. EPA Report on Server and Data Center Energy Efficiency, 2007.

[7] S. Akoush, R. Sohan, A. Rice, A. Moore, and A. Hopper. Free lunch: Exploiting renewable energy for computing. In *Workshop on Hot Topics in Operating Systems (HotOS)*, 2011.

[8] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. In *Workshop on Power-Aware Computing and Systems (HotPower)*, 2011.

[9] J. M. Almeida, J. Krueger, D. L. Eager, and M. K. Vernon. Analysis of Educational Media Server Workloads. In *NOSSDAV*, 2001.

[10] Amazon. Amazon Simple Storage Service. urlhttp://aws.amazon.com/s3.

[11] Amazon. High Performance Computing Using Amazon EC2. `http://aws.amazon.com/ec2/hpc-applications/`.

[12] Ameren. Real-time Prices. `https://www2.ameren.com/RetailEnergy/realtimeprices.aspx`.

[13] V. Anagnostopoulou, S. Biswas, H. Saadeldeen, A. Savage, R. Bianchini, T. Yang, D. Franklin, and F. T. Chong. Barely-alive memory servers: Keeping data active in a low-power state. *ACM Emerging Technologies in Computing Systems*, 2012.

[14] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: A fast array of wimpy nodes. In *Symposium on Operating Systems Principles (SOSP)*, 2009.

[15] Apache. Apache Axis. `http://ws.apache.org/axis/`.

[16] M. Arlitt, C. Bash, S. Blagodurov, Y. Chen, T. Christian, D. Gmach, C. Hyser, N. Kumari, Z. Liu, M. Marwah, A. McReynolds, C. Patel, A. Shah, Z. Wang, and R. Zhou. Towards the Design and Operation of Net-Zero Energy Data Centers. In *Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*, 2012.

[17] ASHRAE. Environmental Guidelines for Datacom Equipment, 2008. American Society of Heating, Refrigeration, and Air-Conditioning Engineers.

[18] A. Banerjee, T. Mukherjee, G. Varsamopoulos, and S. K. S. Gupta. Cooling-aware and thermal-aware workload placement for green hpc data centers. In *International Green Computing Conference (IGCC)*, 2010.

[19] L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis Lectures on Computer Architecture*, 4(1), 2009.

[20] L. A. Barroso and U. Hölzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 40(12), December 2007.

[21] J. Bean and K. Dunlap. Close-coupled rows save energy, 2007.

[22] C. Belady. In the data center, power and cooling costs more than the it equipment it supports. *Electronics cooling*, 13(1):24, 2007.

[23] M. Berkelaar. LP_Solve. `ftp://ftp.es.ele.tue.nl/pub/lp_solve/`.

[24] R. Bhagwan, K. Tati, Y.-C. Cheng, S. Savage, and G. M. Voelker. Total Recall: System Support for Automated Availability Management. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[25] G. Box and G. Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1990.

[26] R. Bradford, E. Kotsovinos, A. Feldmann, and H. Schiöberg. Live wide-area migration of virtual machines including local persistent state. In *International Conference on Virtual Execution Environments (VEE)*, 2007.

[27] K. Brill. Data center energy efficiency and productivity. *White Paper,(The Uptime Institute)*, 2007.

[28] T. Brisco. DNS Support for Load Balancing. RFC 1794 (Informational), 1995.

[29] L. Brown. *Stabilizing Climate*. W. W. Norton and Company, 2006.

[30] R. Buyya, M. Murshed, D. Abramson, and S. Venugopal. Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost-time optimization algorithm. *Software–Practice and Experience*, 35(5), 2005.

[31] V. Cardellini, M. Colajanni, and P. Yu. Dynamic Load Balancing on Web-Server Systems. *IEEE Internet Computing*, 3(3), 1999.

[32] E. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2003.

[33] E. V. Carrera and R. Bianchini. Efficiency vs. Portability in Cluster-Based Network Servers. In *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2001.

[34] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing Energy and Server Resources in Hosting Centers. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[35] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2008.

[36] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing Server Energy and Operational Costs in Hosting Centers. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2005.

[37] J. Choi, Y. Kim, A. Sivasubramaniam, J. Srebric, Q. Wang, and J. Lee. Modeling and Managing Thermal Profiles of Rack-mounted Servers with ThermoStat. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2007.

[38] S. Computing and N. Services. High Performance Compute Cloud. `http://www.sara.nl/index_eng.html`.

[39] K. Coughlin, R. White, C. Bolduc, D. Fisher, and G. Rosenquist. The Tariff Analysis Project: A Database and Analysis Platform for Electricity Tariffs. `http://repositories.cdlib.org/lbnl/LBNL-55680`.

[40] P. Crandall, R. Aydt, A. Chien, and D. Reed. Input/Output Characteristics of Scalable Parallel Applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 1995.

[41] D. Feitelson. Parallel Workloads Archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`.

[42] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-Area Cooperative Storage with CFS. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[43] Data Center Knowledge. Apple Plans 20MW of Solar Power for iDataCenter, 2012. `http://www.datacenterknowledge.com/archives/2012/02/20/apple-plans-20mw-of-solar-power-for-idatacenter/`.

[44] Data Center Knowledge. Data Centers Scale Up Their Solar Power, 2012. `http://www.datacenterknowledge.com/archives/2012/05/14/data-centers-scale-up-their-solar-power/`.

[45] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *International Conference on Autonomic Computing (ICAC)*, 2011.

[46] V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin. Hardware and software techniques for controlling dram power modes. *IEEE Transactions on Computers*, 50(11), 2001.

[47] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[48] Department of Energy and Climate Change. Quarterly Energy Prices. `http://stats.berr.gov.uk/uksa/energy/sa20090625b.htm`.

[49] B. Diniz, D. Guedes, W. Meira, Jr., and R. Bianchini. Limiting the power consumption of main memory. In *International Symposium on Computer Architecture (ISCA)*, 2007.

[50] Duke Energy. North Carolina Electric Rates. `http://www.duke-energy.com/north-carolina.asp`.

[51] E. N. Elnozahy, M. Kistler, and R. Rajamony. Energy Conservation Policies for Web Servers. In *USENIX Symposium on Internet Technologies and Systems (USITS)*, 2003.

[52] M. Etinski, M. Martonosi, K. Le, R. Bianchini, and T. Nguyen. Optimizing the use of request distribution and stored energy for cost reduction in multi-site internet services. In *Conference on Sustainable Internet and ICT for Sustainability (SustainIT)*, 2012.

[53] X. Fan, C. Ellis, and A. Lebeck. The synergy between power-aware memory systems and processor voltage scaling. *Power-Aware Computer Systems*, 2005.

[54] X. Fan, W.-D. Weber, and L. A. Barroso. Power Provisioning for a Warehouse-sized Computer. In *International Symposium on Computer Architecture (ISCA)*, 2007.

[55] M. Femal and V. Freeh. Boosting Data Center Performance Through Non-Uniform Power Allocation. In *International Conference on Autonomic Computing (ICAC)*, 2005.

[56] A. Ferreira, D. Mosse, and J. Oh. Thermal Faults Modeling using a RC model with an Application to Web Farms. In *ECRTS*, 2007.

[57] FPL Energy. Responsible Solutions – Reliable Energy, Retrieved in April 2008. `http://www.fplenergy.com/index.shtml`.

[58] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav. Its not easy being green. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2012.

[59] S. Garg, R. Buyya, and H. Siegel. Scheduling Parallel Applications on Utility Grids: Time and Cost Trade-off Management. In *Australian Conference on Computer Science*, 2009.

[60] Georgia Power. Business Pricing - Georgia Power. `http://www.georgiapower.com/`.

[61] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg, and J. Zelenka. A Cost-Effective, High-Bandwidth Storage Architecture. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998.

[62] Global Action Plan. An Inefficient Truth, Retrieved in December 2007. `http://www.globalactionplan.org.uk/upload/resource/Full-report.pdf`.

[63] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, A. Shah, R. Sharma, and Z. Wang. Capacity planning and power management to exploit sustainable energy. In *International Conference on Network and Service Management (CNSM)*, 2010.

[64] I. Goiri, R. Beauchea, K. Le, T. D. Nguyen, M. E. Haque, J. Guitart, J. Torres, and R. Bianchini. GreenSlot: Scheduling Energy Consumption in Green Datacenters. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2011.

[65] I. Goiri, J. Guitart, and J. Torres. Characterizing Cloud Federation for Enhancing Providers' Profit. In *International Conference on Cloud Computing (CLOUD)*, 2010.

[66] I. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[67] I. Goiri, K. Le, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks. In *European Conference on Computer Systems (EuroSys)*, 2012.

[68] Google. Green Initiatives, Retrieved in December 2010. `http://www.google.com/intl/en/corporate/green/index.html`.

[69] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and Limitations of Tapping into Stored Energy for Datacenters. In *International Symposium on Computer Architecture (ISCA)*, 2011.

[70] Green House Data. An Economically Responsible Data Center, 2012. `http://www.greenhousedata.com/`.

[71] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt. Best practices for data centers: Lessons learned from benchmarking 22 data centers. In *Summer Study on Energy Efficiency in Buildings*, 2006.

[72] Greenest Host, Retrieved in March 2008. `http://www.greenesthost.com/`.

[73] X. Gu and K. Nahrstedt. Distributed Multimedia Service Composition with Statistical QoS Assurances. *IEEE Transactions on Multimedia*, 2005.

[74] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Drpm: dynamic speed control for power management in server class disks. In *International Symposium on Computer Architecture (ISCA)*, 2003.

[75] K. Hamilton, M. Sjardin, A. Shapiro, and T. Marcello. Fortifying the Foundation: State of the Voluntary Carbon Markets 2009, 2009. `http://ecosystemmarketplace.com/documents/cms_documents/StateOfTheVoluntaryCarbonMarkets_2009.pdf`.

[76] HAProxy. The Reliable, High Performance TCP/HTTP Load Balancer. `http://haproxy.1wt.eu/`.

[77] E. Harney, S. Goasguen, J. Martin, M. Murphy, and M. Westall. The efficacy of live virtual machine migrations over the internet. In *Workshop on Virtualization Technologies in Distributed Computing (VTDC)*, 2007.

[78] T. Heath, A. Centeno, P. George, L. Ramos, Y. Jaluria, and R. Bianchini. Mercury and Freon: Temperature Emulation and Management in Server Systems. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[79] T. Heath, B. Diniz, E. V. Carrera, W. Meira, Jr., and Bianchi. Energy Conservation in Heterogeneous Server Clusters. In *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2005.

[80] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: saving energy in data center networks. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[81] U. Hölzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition, 2009.

[82] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu. Dynamic Voltage Scaling in Multi-tier Web Servers with End-to-End Delay Control. *IEEE Transactions on Computers*, 2007.

[83] Ilisys, Retrieved in March 2008. `http://www.ilisys.com.au/`.

[84] Iron Mountain Hosting, Retrieved in March 2008. `http://www.imountain.com/`.

[85] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *International Symposium on Computer Architecture (ISCA)*, 2010.

[86] J. M. Kaplan, W. Forrest, and N. Kindler. Revolutionizing Data Center Energy Efficiency, 2008.

[87] S. Kaxiras and M. Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Morgan and Claypool Publishers, 1st edition, 2008.

[88] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598), 1983.

[89] D. C. Knowledge. Google: Raise Your Data Center Temperature. `http://www.datacenterknowledge.com/archives/2008/10/14/google-raise-your-data-center-temperature/`.

[90] J. Koomey. Growth in Data Center Electricity Use 2005 to 2010, 2011. Analytic Press.

[91] J. Koomey, C. Belady, M. Patterson, A. Santos, and K. Lange. Assessing trends over time in performance, costs, and energy use for servers. *Microsoft, Intel and Hewlett-Packard Corporation, Tech. Rep., Aug*, 17, 2009. `www.intel.com/assets/pdf/.../servertrendsreleasecomplete-v25.pdf`.

[92] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software–Practice and Experience*, 2002.

[93] A. Krioukov, S. Alspaugh, P. Mohan, S. Dawson-Haggerty, D. Culler, and R. Katz. Design and Evaluation of an Energy Agile Computing Cluster. Technical Report EECS-2012-13, University of California at Berkeley, 2012.

[94] A. Krioukov, C. Goebel, S. Alspaugh, Y. Chen, D. Culler, and R. Katz. Integrating Renewable Energy Using Data Analytics Systems: Challenges and Opportunities. *Bulletin of the IEEE Computer Society Technical Committee*, 2011.

[95] A. Kumar. Use of air side economizer for data center thermal management. Master's thesis, Georgia Institute of Technology, 2008.

[96] K. Kurowski, B. Ludwiczak, J. Nabrzyski, A. Oleksiak, and J. Pukacki. Dynamic grid scheduling with job migration and rescheduling in the gridlab resource management system. *Scientific Programming*, 12(4), 2004.

[97] W. Lang, J. Patel, and S. Shankar. Wimpy node clusters: what about non-wimpy workloads? In *Workshop on Data Management on New Hardware (DaMoN)*, 2010.

[98] C. Li, W. Zhang, C.-B. Cho, and T. Li. Solarcore: Solar energy driven multi-core architecture power management. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2011.

[99] K. Lim, P. Ranganathan, J. Chang, C. Patel, T. Mudge, and S. Reinhardt. Understanding and designing new server architectures for emerging warehouse-computing environments. In *International Symposium on Computer Architecture (ISCA)*, 2008.

[100] M. Lim, V. Freeh, and D. Lowenthal. Adaptive, Transparent Frequency and Voltage Scaling of Communication Phases in MPI Programs. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2006.

[101] M. Lin, Z. Liu, A. Wierman, and L. Andrew. Online algorithms for geographical load balancing. In *International Green Computing Conference (IGCC)*, 2012.

[102] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2012.

[103] Z. Liu, M. Lin, A. Wierman, S. Low, and L. Andrew. Geographical load balancing with renewables. In *Workshop on Sustainable ICT, ICT for Sustainability, and Smart Grid (GreenMetrics)*, 2011.

[104] Z. Liu, M. Lin, A. Wierman, S. H. Low, and L. L. Andrew. Greening Geographical Load Balancing. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2011.

[105] J. Mankoff, R. Kravets, and E. Blevis. Some computer science issues in creating a sustainable world. *Computer*, 41, 2008.

[106] G. E. Marketplace. Green Electricity Marketplace. `http://www.greenelectricity.org/`.

[107] R. Martin and D. Culler. NFS Sensitivity to High Performance Networks. In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 1999.

[108] D. Meisner, B. T. Gold, and T. F. Wenisch. PowerNap: Eliminating Server Idle Power. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[109] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch. Power management of online data-intensive services. In *International Symposium on Computer Architecture (ISCA)*, 2011.

[110] D. Meisner and T. F. Wenisch. Dreamweaver: architectural support for deep sleep. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[111] R. S. Montero, E. Huedo, and I. M. Llorente. Grid resource selection for opportunistic job migration. *Book Series Lecture Notes in Computer Science*, 2004.

[112] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling cool: Temperature-aware workload placement in data centers. In *USENIX Annual Technical Conference (USENIX)*, 2005.

[113] J. Moore, J. S. Chase, and P. Ranganathan. Weatherman: Automated, Online and Predictive Thermal Mapping and Management for Data Centers. In *International Conference on Autonomic Computing (ICAC)*, 2006.

[114] A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. *IEEE Transactions on Parallel Distributed Systems*, 12, 2001.

[115] T. Mukherjee, Q. Tang, C. Ziesman, S. Gupta, and P. Cayton. Software Architecture for Dynamic Thermal Management in Datacenters. In *International ICST Conference on Communication System Software and Middleware (COMSWARE)*, 2007.

[116] T. G. P. Network. Buy Green Power. `http://apps3.eere.energy.gov/greenpower/buying/buying_power.shtml`.

[117] W. T. Ng, H. Sun, B. Hillyer, E. Shriver, E. Gabber, and B. Ozden. Obtaining High Performance for Storage Outsourcing. In *USENIX Conference on File and Storage Technologies (FAST)*, 2002.

[118] N. Nieuwejaar and D. Kotz. The Galley Parallel File System. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 1996.

[119] NREL. Best Practices Guide for Energy-Efficient Data Center Design, 2010. National Renewable Energy Laboratory (NREL), U.S. Department of Energy.

[120] M. A. Olson, K. Bostic, and M. I. Seltzer. Berkeley DB. In *USENIX Annual Technical Conference (USENIX)*, 1999.

[121] Pacific Gas and Electric. ELECTRIC SCHEDULES. `http://www.pge.com/`.

[122] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1998.

[123] D. S. Palasamudram, R. K. Sitaraman, B. Urgaonkar, and R. Urgaonkar. Using batteries to reduce the power costs of internet-scale distributed networks. In *Symposium on Cloud Computing (SOCC)*, 2012.

[124] L. Parolini, E. Garone, B. Sinopoli, and B. Krogh. A hierarchical approach to energy management in data centers. In *Conference on Decision and Control (CDC)*, 2011.

[125] L. Parolini, B. Sinopoli, and B. Krogh. Reducing data center energy consumption via coordinated cooling and load management. In *Workshop on Power-Aware Computing and Systems (HotPower)*, 2008.

[126] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder. Understanding and abstracting total data center power. In *Workshop on Energy-Efficient Design (WEED)*, 2009.

[127] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2004.

[128] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems. In *Workshop on Compilers and Operating Systems for Low Power (COLP)*, 2001.

[129] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Dynamic Cluster Reconfiguration for Power and Performance. In *Compilers and Operating Systems for Low Power*, 2003. Earlier version published in COLP, 2001.

[130] PointCarbon, Retrieved in March 2008. `http://www.pointcarbon.com`.

[131] G. Posladek. An investigation into using free cooling and community heating to reduce data centre energy consumption. Master's thesis, Strathclyde University, 2008.

[132] Power Scorecard. Electricity from Coal, Retrieved in April 2008. `http://www.powerscorecard.org/tech_detail.cfm?resource_id=2`.

[133] Power Smart Pricing. Power Smart Pricing. `http://www.powersmartpricing.org/program-guide.php`.

[134] A. Qureshi. Plugging Into Energy Market Diversity. In *Workshop on Hot Topics in Networks (Hotnets)*, 2008.

[135] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs. Cutting the Electric Bill for Internet-Scale Systems. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2009.

[136] P. Radkov, L. Yin, P. Goyal, P. Sarkar, and P. Shenoy. A Performance Comparison of NFS and iSCSI for IP-Networked Storage. In *USENIX Conference on File and Storage Technologies (FAST)*, 2004.

[137] K. Rajamani and C. Lefurgy. On Evaluating Request-Distribution Schemes for Saving Energy in Server Clusters. In *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2003.

[138] K. Rajamani, C. Lefurgy, S. Ghiasi, J. Rubio, H. Hanson, and T. Keller. Power management solutions for computer systems and datacenters. In *International Symposium on Low Power Electronics and Design (ISLPED)*, 2008.

[139] L. Ramos and R. Bianchini. C-Oracle: Predictive Thermal Management for Data Centers. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2008.

[140] P. Ranganathan, P. Leech, D. Irwin, and J. Chase. Ensemble-Level Power Management for Dense Blade Servers. In *International Symposium on Computer Architecture (ISCA)*, 2006.

[141] S. Ranjan, R. Karrer, and E. Knightly. Wide Area Redirection of Dynamic Content by Internet Data Centers. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2004.

[142] L. Rao, X. Liu, L. Xie, and W. Liu. Minimizing Electricity Cost: Optimization of Distributed Internet Data Centers in a Multi-Electricity-Market Environment. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2010.

[143] Regional Greenhouse Gas Initiative. Regional Greenhouse Gas Initiative: An Initiative of the Northeast and Mid-Atlantic States of the US, Retrieved in April 2008. `http://www.rggi.org/`.

[144] C. Ren, D. Wang, B. Urgaonkar, and A. Sivasubramaniam. Carbon-Aware Energy Capacity Planning for Datacenters. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2012.

[145] B. Rountree, D. K. Lownenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch. Adagio: Making DVS Practical for Complex HPC Applications. In *International Conference on Supercomputing (ICS)*, 2009.

[146] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Symposium on Operating Systems Principles (SOSP)*, 2001.

[147] C. Rusu, A. Ferreira, C. Scordino, and A. Watson. Energy-Efficient Real-Time Heterogeneous Server Clusters. In *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006.

[148] E. Samadiani, Y. Joshi, J. Allen, and F. Mistree. Adaptable Robust Design of Multi-Scale Convective Systems Applied to Energy Efficient Data Centers. *Numerical Heat Transfer, Part A: Applications*, 57(2), 2010.

[149] A. J. Shah and N. Krishnan. Optimization of Global Data Center Thermal Management Workload for Minimal Environmental and Economic Burden. *IEEE Transactions on Components and Packaging Technologies*, 31(1), 2008.

[150] N. Sharma, S. Barker, D. Irwin, and P. Shenoy. Blink: managing server clusters on intermittent power. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2011.

[151] R. Sharma, C. Bash, C. Patel, R. Friedrich, and J. Chase. Balance of Power: Dynamic Thermal Management for Internet Data Centers. Technical Report HPL-2003-5, HP Labs, 2003.

[152] S. G. Shasharina, N. Wang, and J. R. Cary. Grid Service for Visualization and Analysis of Remote Fusion Data. In *Workshop on Challenges of Large Applications in Distributed Environments (CLADE)*, 2004.

[153] Silicon Valley Leadership Group. Data center energy forecast, 2008. `https://microsite.accenture.com/svlgreport/Documents/pdf/SVLG_Report.pdf`.

[154] D. Snowdon, S. Ruocco, and G. Heiser. Power management and dynamic voltage scaling: Myths and facts. In *Workshop on Power Aware Real-time Computing (PARC)*, 2005.

[155] R. Spangler. Smart Approaches to Free-Cooling in Data Centers, 2008.

[156] C. Stewart and K. Shen. Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter. In *Workshop on Power-Aware Computing and Systems (HotPower)*, 2009.

[157] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverQoS: An Overlay Based Architecture for Enhancing Internet QoS. In *Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[158] Sustainable Hosting, Retrieved in March 2008. `http://www.sustainablehosting.com/`.

[159] D. Thain, J. Bent, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and M. Livny. Pipeline and Batch Sharing in Grid Workloads. In *International Symposium on High Performance Distributed Computing (HPDC)*, 2003.

[160] C. A. Thekkath, T. P. Mann, and E. K. Lee. Frangipani: A Scalable Distributed File System. In *Symposium on Operating Systems Principles (SOSP)*, 1997.

[161] TheWattSpot. TheWattSpot. `http://www.thewattspot.com/`.

[162] ThinkHost, Retrieved in March 2008. `http://www.thinkhost.com/`.

[163] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. Delivering energy proportionality with non energy-proportional systems: optimizing the ensemble. In *Workshop on Power-Aware Computing and Systems (HotPower)*, 2008.

[164] D. Tsafrir, Y. Etsion, and D. G. Feitelson. Modeling user runtime estimates. In *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*, 2005.

[165] W. Tschudi, T. Xu, D. Sartor, and J. Stein. High-performance data centers: A research roadmap. Technical Report LBNL-53483, Lawrence Berkeley National Laboratory, 2004. `http://hightech.lbl.gov/documents/DataCenters_Roadmap_Final.pdf`.

[166] W. P. Turner IV, J. H. Seader, V. Renaud, and K. J. Brill. Tier classifications define site infrastructure performance. Uptime Institute White Paper, 2009. `http://www.iswest.com/colocation/TierClassification.pdf`.

[167] UK Government. Carbon Reduction Commitment, Retrieved in July 2009. `http://www.carbonreductioncommitment.info/`.

[168] U.S. Energy Information Administration. FAQ: What is U.S. electricity generation by energy source?, Retrieved in November 2012. `http://www.eia.gov/tools/faqs/faq.cfm?id=427&t=3`.

[169] F. Wang, Q. Xin, B. Hong, S. Brandt, E. Miller, D. Long, and T. McLarty. File System Workload Analysis for Large-Scale Scientific Computing Applications. In *NASA Goddard Conference on Mass Storage Systems and Technologies*, 2004.

[170] L. Wang, V. Pai, and L. Peterson. The Effectiveness of Request Redirection on CDN Robustness. In *Symposium on Operating Systems Design and Implementation (OSDI)*, 2002.

[171] X. Wang and M. Chen. Cluster-level Feedback Power Control for Performance Optimization. In *International Symposium on High-Performance Computer Architecture (HPCA)*, 2008.

[172] Weather Underground. Weather Underground. `http://www.wunderground.com/`.

[173] A. Weissel and F. Bellosa. Dynamic thermal management in distributed systems. In *Workshop on Temperature-Aware Computer Systems (TACS)*, 2004.

[174] P. Wendell, J. W. Jiang, M. J. Freedman, and J. Rexford. Donar: decentralized server selection for cloud services. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM)*, 2010.

[175] W. R. Wong and R. R. Muntz. Providing Guaranteed Quality of Service for Interactive Visualization Applications (poster). In *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, 2000.

[176] T. Wood, K. K. Ramakrishnan, P. Shenoy, and J. van der Merwe. Cloudnet: dynamic pooling of cloud resources by live wan migration of virtual machines. In *International Conference on Virtual Execution Environments (VEE)*, 2011.

[177] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, 30(5), 2004.

[178] W. Zhang. Linux Virtual Server for Scalable Network Services. In *Linux Symposium*, 2000.

[179] Y. Zhang, Y. Wang, and X. Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *International Conference on Middleware (Middleware)*, 2011.

[180] W. Zhao, D. Olshefski, and H. Schulzrinne. Internet Quality of Service: An Overview. Technical Report CUCS-003-00, Department of Computer Science, Columbia University, 2000.

Vita

Kien T. Le

2003 **B.S. in Computer Science,**
Rutgers University, New Brunswick, New Jersey, USA

2007 **M.S. in Computer Science,**
Rutgers University, New Brunswick, New Jersey, USA

2013 **Ph.D. in Computer Science,**
Rutgers University, New Brunswick, New Jersey, USA

Selected Publications

2007 "A Cost-Effective Distributed File Service with QoS Guarantees". Kien Le, Ricardo Bianchini and Thu D. Nguyen. In Proceedings of the 8th International Middleware Conference (Middleware 2007), November 2007.

2009 "Cost- and Energy-Aware Load Distribution Across Data Centers". Kien Le, Ricardo Bianchini, Margaret Martonosi and Thu D. Nguyen. In Proceedings of the Workshop on Power Aware Computing and Systems (Hotpower '09) , October 2009.

2010 "Capping the Brown Energy Consumption of Internet Services at Low Cost". Kien Le, Ozlem Bilgir, Ricardo Bianchini, Margaret Martonosi and Thu D. Nguyen. In Proceedings of the 1^{st} International Green Computing Conference (IGCC), August 2010. **Winner of the Best Paper Award**

2011 "Intelligent Placement of Datacenters for Internet Services". Íñigo Goiri, Kien Le, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. In Proceedings of the 31st International Conference on Distributed Computing Systems (ICDCS 2011), June 2011.

2011 "GreenSlot: Scheduling Energy Consumption in Green Datacenters". Íñigo Goiri, Kien Le, Md. E. Haque, Ryan Beauchea, Thu D. Nguyen, Jordi Guitart, Jordi Torres and Ricardo Bianchini. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC11), November 2011.

2011 "Cost-Aware Virtual Machine Placement in Cloud Computing Systems". Kien Le, Jingru Zhang, Jiandong Meng, Ricardo Bianchini,

Thu D. Nguyen, and Yogesh Jaluria. In Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC11), November 2011.

2012 "GreenHadoop: Leveraging Green Energy in Data-Processing Frameworks". Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres and Ricardo Bianchini. In Proceedings of EuroSys 2012, April 2012.

2012 "Optimizing the Use of Request Distribution and Stored Energy for Cost Reduction in Multi-Site Internet Services". Maja Etinski, Margaret Martonosi, Kien Le, Ricardo Bianchini, and Thu Nguyen. In Proceedings of SustainIT, October 2012.

2013 "Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy". Íñigo, William Katsak, Kien Le, Thu D. Nguyen and Ricardo Bianchini. To appear In Proceedings of ASPLOS 2013, March 2013.