November 2015

Energy-Efficient Content Delivery Networks

Vimal Mathew

University of Massachusetts - Amherst

ENERGY-EFFICIENT CONTENT DELIVERY NETWORKS

A Dissertation Presented

by

VIMAL MATHEW

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2015

College of Information and Computer Sciences

ENERGY-EFFICIENT CONTENT DELIVERY NETWORKS

A Dissertation Presented

by

VIMAL MATHEW

Approved as to style and content by:

Ramesh Sitaraman, Co-chair

Prashant Shenoy, Co-chair

James Kurose, Member

David Irwin, Member

James Allan, Chair
College of Information and Computer Sciences

To my parents.

ACKNOWLEDGMENTS

I would like to thank Ramesh Sitaraman and Prashant Shenoy, my thesis advisors, without whom I would not have received a doctorate. I am fortunate to have come in contact with many great teachers both inside and outside the department, making my time at Amherst an unforgettable experience. The town of Amherst itself has been a revelation. I have spent many an hour walking along winding trails past gurgling brooks where I have found solace when it was needed most. The winters though, I could do without.

The time spent at UMass has been enriched by my fellow students and friends. I have been fortunate to meet Sameer Singh, Gaurav Chandalia, Aditya Nemmaluri, Manikandan Somasundaram, Nitai Giri, Chang Wang, Bruno Castro da Silva, Armita Kaboli, Bo Liu, JP Mahalik, and Navin Sharma among many others. Shouts out to Sir Iqbal, wherever he may be.

ABSTRACT

ENERGY-EFFICIENT CONTENT DELIVERY NETWORKS

SEPTEMBER 2015

VIMAL MATHEW

B.Tech., COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY INDIA

M.S., INDIAN INSTITUTE OF TECHNOLOGY MADRAS INDIA

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Ramesh Sitaraman and Professor Prashant Shenoy

Internet-scale distributed systems such as content delivery networks (CDNs) operate hundreds of thousands of servers deployed in thousands of data center locations around the globe. Since the energy costs of operating such a large IT infrastructure are a significant fraction of the total operating costs, we argue for redesigning them to incorporate energy optimization as a first-order principle. We focus on CDNs and demonstrate techniques to save energy while meeting client-perceived service level agreements (SLAs) and minimizing impact on hardware reliability.

Servers deployed at individual data centers can be switched off at low load to save energy. We show that it is possible to save energy while providing client-perceived *availability* and limited impact on hardware *reliability*. We propose an optimal offline algorithm and an online algorithm to extract energy savings and evaluate them on real production workload traces. Our results show that it is possible to reduce the

energy consumption of a CDN by 51% while ensuring a high level of availability and incurring an average of one on-off transition per server per day.

We propose a novel technique called *cluster shutdown* that switches off an entire cluster of servers, thus saving on both server and cooling power. We present an algorithm for cluster shutdown that is based on realistic power models for servers and cooling equipment and can be implemented as a part of the global load balancer of a CDN. We argue that cluster shutdown has intrinsic architectural advantages over server shutdown techniques in the CDN context, and show that it outperforms *server shutdown* in a wide range of operating regimes.

To reduce energy costs, we propose a *demand-response* technique that responds to pricing signals from a smart grid by deferring elastic load. We propose an optimal offline algorithm for demand response and evaluate it on production workloads from a commercial CDN using realistic electricity pricing models. We show that energy cost savings can be achieved with no increase in the bandwidth cost.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

Large Internet-scale distributed systems deploy hundreds of thousands of servers in thousands of data centers around the world. Such systems currently provide the core distributed infrastructure for many popular Internet applications that drive business, e-commerce, entertainment, news, and social networking. The energy cost of operating an Internet-scale system is already a significant fraction of the total cost of ownership (TCO) [9]. The environmental implications are equally important. A large distributed platform with 100,000 servers will expend roughly 190,000 MWH per year, enough energy to sustain more than 10,000 households. In 2005, the total data center power consumption was already 1% of the total US power consumption while causing as much emissions as a mid-sized nation such as Argentina. Further, with the deployment of new services and the rapid growth of the Internet, the energy consumption of data centers is expected to grow at a rapid pace of more than 15% per year in the foreseeable future [20]. These factors necessitate rearchitecting Internet-scale systems to include energy optimization as a first-order principle.

1.2 CDN architecture

An important Internet-scale distributed system to have emerged in the past decade is the *content delivery network* (CDN, for short) that delivers web content, web and IP-based applications, downloads, and streaming media to end-users (i.e., *clients*) around the world. A large CDN, such as that of a commercial provider like Akamai,

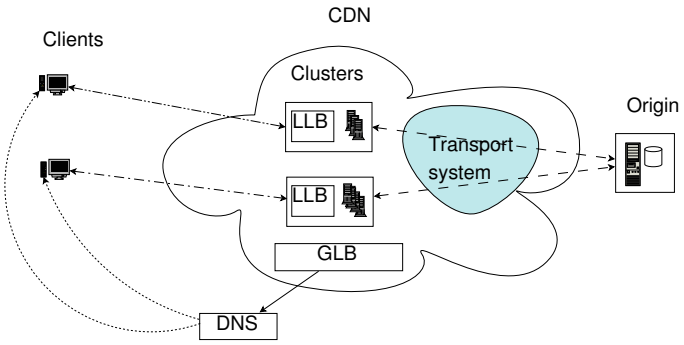


Figure 1.1: System components of a CDN

consists of hundreds of thousands of servers located in over a thousand data centers around the world and account for a significant fraction of the world's enterprise-quality web and streaming media traffic today [35]. The servers of a CDN are deployed in *clusters* where each cluster consists of servers in a particular data center in a specific geographic location. The clusters are typically widely deployed on the "edges" of the Internet in most major geographies and ISPs around the world so as to be proximal to clients. Clusters can vary in size from tens of servers in a small Tier-3 ISP to thousands of servers in a large Tier-1 ISP.

The primary goal of a CDN is to serve content such as web pages, videos, and applications with high availability and performance to end users. The key component that ensures availability and performance is the CDN's *load balancing* system that assigns each incoming request to a server that can serve that request. To this end, a CDN's load balancing system routes each user's request to a server that is *live* and *not overloaded*. Further, to enhance performance, a CDN ensures that each user request is routed to a server that is *proximal* to that user. The proximity (in a network sense) ensures that the network path between the user's device and the CDN's server has low latency and loss. The process of routing user requests to servers is a two stage process. A *global load balancer* (called GLB) assigns the user to a cluster of servers based on the availability of server resources in the cluster, performance, and

bandwidth costs. A *local load balancer* (called LLB) assigns the user to a specific server that is capable of serving the requested content within the chosen cluster. The choice of server is dictated by server liveness, content footprint, and current server loads with respect to their capacities. A comprehensive discussion of the rationale and system architecture of CDNs is available in [35].

1.3 Prior work

Energy management in data centers has been an active area of research in recent years [13]. Techniques that have been developed in this area include, use of DVFS to reduce energy, use of very low-power servers [7], the use of renewable energy [46, 18], routing requests to locations with the cheapest energy [42] and dynamically activating and deactivating nodes as demand rises and falls [12, 50, 21]. A key difference between much of this prior work with ours is the focus on CDNs, with a particular emphasis on the interplay between energy management and the local/global load balancing algorithms in the CDN. We also examine the impact of our energy saving techniques on client SLAs, hardware reliability and operating costs.

1.4 Contributions

CDNs typically run at low utilization due to the high business cost of failing to meet customer SLAs. Our focus in this thesis is reducing energy costs of CDNs with minimal impact on SLAs and operating costs.

- We propose an optimal offline algorithm and an online algorithm for saving energy that can be incorporated into the local load balancer within a cluster of servers. Our results show that it is possible to reduce the energy consumption of a CDN by 51% while ensuring a high level of availability and incurring an average of one on-off transition per server per day. We also show the online

algorithm can handle load spikes caused by flash crowds, but at a cost of lower energy savings.

- We propose and explore a novel technique called *cluster shutdown* that turns off an entire cluster of servers of a CDN. We present an algorithm for cluster shutdown that is based on realistic power models for servers and cooling equipment and can be implemented as a part of the global load balancer of a CDN. We argue that cluster shutdown has intrinsic architectural advantages over server shutdown techniques in the CDN context, and show that it outperforms server shutdown over a range of operating regimes.

- We propose a demand-response technique where the system temporarily reduces its energy usage in response to pricing signals from a smart grid. Our proposed demand-response technique involves deferring the load from elastic requests to later time periods in order to reduce the server demand and the current energy usage, and hence, energy costs. We propose an optimal offline algorithm for demand response and evaluate it on production workloads from a commercial content delivery network using realistic electricity pricing models. For a hybrid pricing model that combines time-of-use and demand charges, we show that 32% energy cost savings can be achieved when only 40% of the load is elastic and the service delay is at most 6 hours. Further, we show that almost all the savings can be achieved with no increase in bandwidth cost.

1.5 Roadmap

The remainder of this thesis is as follows. Chapter 2 provides some background and a description of related work. Chapter 3 describes our local load balancing algorithm with server shutdown. In Chapter 4 we present our energy-aware global load balancer

for *cluster shutdown*. Chapter 5 presents our demand-response technique to reduce energy costs. Chapter 6 concludes with future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter provides a brief overview of energy efficient techniques for data centers and large distributed systems. While a general overview is provided here, the chapters that follow include a more detailed discussion on related work.

2.1 Motivation

The ideal for any energy efficient system is to attain *energy-proportionality*, where the energy consumption scales linearly with the workload [9]. To achieve this ideal requires the individual components of the system to be energy-proportional. Alternately, the system could be redesigned to consolidate workload and dynamically switch off idling resources to use less energy [10]. The challenge is to gain energy efficiency without a loss in performance.

2.2 Energy efficient data centers

The trend towards energy efficiency of server hardware has led to lower energy usage in data centers. Processors running in low-power modes can consume less than one-third of their peak power while still executing instructions [9]. The availability of such low-power active modes allow Dynamic Voltage and Frequency Scaling (DVFS) techniques to save considerable energy with limited performance impact [37]. Dynamic Component Deactivation (DCD) techniques, where idling components such as processor cores or storage subsystems are adaptively switched off, have also increased the dynamic power range of server hardware leading to greater efficiency [37, 10, 16].

At a data center level, energy-aware scheduling algorithms can consolidate workload among fewer servers to reduce energy consumption while minimizing any loss in performance [40, 12, 47]. In cases when individual applications deployed on the data center are not energy-aware, virtualization techniques can be leveraged to provide energy management. Computing resources can be split into a number of Virtual Machines (VMs), each of which provides a user-level view of a dedicated machine. VMs can then be migrated between physical hosts or hibernated when not in use. These capabilities facilitate consolidation and load-balancing even when the applications and corresponding workload within the VMs are not energy aware. A number of frameworks that provide energy-efficient resource management in virtualized data centers have been studied in the past [34, 43, 22, 49, 52].

Cooling can also contribute significantly to the energy costs in a data center. The ratio of total energy to IT energy is a standard metric called PUE (Power Usage Effectiveness) that has a typical value of about 2 implying cooling energy is roughly equal to IT energy in typical data center deployments. But in more recent energy-efficient designs, PUE is smaller but cooling energy is still a significant fraction of the IT energy. Further, cooling energy consumption is not power-proportional since cooling still takes a significant amount of energy even when the servers have low utilization and are not producing much heat. Recent trends in data centers such as self-contained, modular[45], or containerized[41] deployments have allowed fine-grained control of cooling resources and greater energy efficiency. Cooling-aware workload management techniques have been studied in the past [24]. Thermal-aware workload placement techniques that place load on cool portions of the data center have been studied in [33, 51].

Data centers handling elastic loads have an opportunity to reduce *energy costs* when electricity is priced differently at different times of the day.Electricity grids incentivize their customers, through differential pricing, to reduce usage during peak

periods. Data centers can adapt to these pricing patterns through a technique called *demand-response* where they reduce energy consumption during peak periods by shifting load to times when electricity is priced cheaper [26, 27, 32, 53, 56] .

2.3 Large scale distributed systems

Large scale distributed systems deployed over geographically distributed data centers across the world have complex dependencies and requirements that need to be met. Any change to the global load balancer could potentially map a user to a farther off server, potentially increasing latency and impacting the user-level experience. Consolidating load in specific data centers could increase bandwidth costs, reduce reliability and impact the operating cost of the distributed system. Therefore redesigning such systems for energy efficiency can be a challenging task. For instance, CDNs impose their own set of SLA requirements and require an appropriate approach [30, 31]. While increasing the energy efficiency of individual data centers is possible, another approach focuses on reducing *energy costs* by leveraging differences in pricing across geographically separated data centers. Recent research has shown the potential for significant savings [42, 44, 54] while meeting performance requirements.

CHAPTER 3

ENERGY EFFICIENCY THROUGH SERVER SHUTDOWN

Recent work in server energy management has suggested the technique of utilizing deep-sleep power-saving modes or even completely turning off servers during periods of low load, thereby saving the energy expended by idle servers [14, 23]. We explore the potential applicability of this technique in the CDN context where it is important to understand the interplay of the three objectives below.

- *Maximize energy reduction.* Idle servers often consume more than 50% of the power of a fully-loaded one [9]. This provides the opportunity to save energy by "rebalancing" (i.e., redirecting) the request traffic onto fewer servers and turning the remaining servers off.

- *Satisfy customer SLAs.* Content providers who are the CDN's customers would like their content and applications to be served with a high level of availability and performance to their clients. Availability can be measured as the fraction of client requests that are successfully served. A typical SLA would require at least "four nines" of *end-to-end* availability (i.e., 99.99%). To achieve this end-to-end SLA goal, we estimate that any acceptable technique for powering off servers should cause no more than a loss of 0.1 basis points of availability in the data center, leading us to target 99.999% *server* availability with our techniques. In addition to the availability SLA, the content providers also require good performance. For instance, clients downloading http content should experience small download times and clients watching media should receive high quality streams with high bandwidth

and few freezes. Since turning off servers to save energy reduces the live server capacity used for serving the incoming request load, it is important that any energy saving technique minimizes the impact of the decreased capacity on availability and performance.

- *Minimize server transitions.* Studies have shown that frequently turning an electronic device on and off can impact its overall lifetime and reliability. Consequently, CDN operators are often concerned about the wear and tear caused by excessive on-off server transitions that could potentially decrease the lifetime of the servers. Additionally, when a server is turned off, its state has to be migrated or replicated to a different live server. Mechanisms for replicating content footprint and migrating long-standing TCP connections exist in the CDNs today [35] as well as in other types of Internet-scale services [6, 14]. However, a small degree of client-visible performance degradation due to server transitions is inevitable. Consequently an energy saving technique should limit on-off server transitions in order to reduce wear and tear and the impact on client-visible performance.

The three objectives above are often in conflict. For instance, turning off too many servers to maximize energy reduction can decrease the available live capacity of the CDN. Since it takes time to turn on a server and bring it back into service, an unexpected spike in the load can lead to dropped requests and SLA violations. Likewise, turning servers on and off frequently in response to load variations could enhance energy reduction but incur too many server transitions. Our goal is to design energy-aware techniques for CDNs that incorporate all three objectives and to understand how much energy reduction is realistically achievable in a CDN. Since CDNs are yet to be aggressively optimized for energy usage today, our work hopes to guide the future architectural evolution that must inevitably incorporate energy as a primary design objective.

While we focus on CDNs, our work also applies to other CDN-like distributed systems that replicate services within and across server clusters and employ some form of load balancing to dynamically route requests to servers. On a different dimension, it is also important to note that our focus is energy *usage* reduction rather than energy *cost* reduction. Note that energy cost reduction can be achieved by dynamically shifting the server load to locations with lower energy prices without necessarily decreasing the total energy usage [42].

3.1 Our Contributions

Our work is the first to propose energy-aware mechanisms for load balancing in CDNs with a quantification of the key practical tradeoffs between energy reduction, hardware wear-and-tear due to server transitions, and service availability that impacts customer SLAs. The load balancing system of a CDN operates at two levels [35]. The *global load balancing* component determines a good cluster of the CDN for each request, while the *local load balancing* component chooses the right server for the request within the assigned cluster. We design mechanisms for energy savings, both from the local and global load-balancing standpoint. Further, we evaluate our mechanisms using real production workload traces collected over 25 days from 22 geographically distributed clusters across the US from a large commercial CDN. Our specific key contributions are as follows.

- In the offline context when the complete load sequence for a cluster is known ahead of time, we derive optimal algorithms that minimize energy usage by varying the number of live servers required to serve the incoming load.

- On production CDN workloads, our offline algorithm achieves a significant system-wide energy reduction of 59.5%. Further, even if the average transitions is restricted to be below 1 transition per server per day, an energy reduction of 58.66% can be

achieved, i.e., 98.6% of the maximum energy reduction can be achieved with minimal server wear-and-tear.

- We propose a load balancing algorithm called Hibernate that works in an online fashion that makes decisions based on past and current load but not future load, much like a real-life load balancing system. Hibernate achieves an energy reduction of 56%, i.e., within 94% of the offline optimal.

- By holding an extra 10% of the servers as live spares, Hibernate achieves the sweet spot with respect to all three metrics. Specifically, the algorithm achieves a system-wide energy reduction of 51% and a service availability of at least five nine's (99.999%), while incurring an average of at most 1 transition per server per day. The modest decrease in energy reduction due to the extra pool of live servers is well worth the enhanced service availability for the CDN.

- In a global flash crowd scenario when the load spikes suddenly across all clusters of the CDN, Hibernate is still able to provide five nine's of service availability and maintain customer SLAs as long as the rate at which load increases is commensurate with the percentage of server capacity that the algorithm keeps as live spares.

- Energy-aware global load balancing can redistribute traffic across clusters but had only a limited impact on energy reduction. Since load can only be redistributed between proximal clusters for reasons of client performance, these clusters had load patterns that are similar enough to not entail a large energy benefit from load redistribution. However, a 10% to 25% reduction in server transitions can be achieved by redistributing load across proximal clusters. But, perhaps the key benefit of global load balancing is significantly increased service availability. In our simulations, global load balancing enhanced service availability to almost 100%. In situations where an unpredictable increase in load would have exceeded the live capacity

of a cluster causing service disruption, our global load balancing spread the load increase to other clusters with available live capacity.

In summary, our results show that significant energy reduction is possible in CDNs if they are rearchitected with energy awareness as a first-order principle. Further, our work also allays the two primary fears in the mind of CDN operators regarding turning off servers for energy savings: the ability to maintain service availability, especially in the presence of a flash crowd, and the impact of server transitions on the hardware lifetimes and ultimately the capital expenditures associated with operating the CDN.

3.2 Roadmap

After formulating our models and methodology (Section 5.1), we study local load balancing (Section 3.4) in an offline setting with the assumption that the entire traffic load pattern is known in advance (Section 3.4.1), and then extend it to the more realistic online situation where future traffic is unknown (Section 3.4.2). Then, we explore the gains to be had by moving traffic between clusters via global load balancing (Section 3.5). Finally, we discuss related work (Section 5.4) and offer conclusions (Section 5.5).

3.3 Model Formulation and Methodology

CDN Model. Our work assumes a global content delivery network (CDN) that comprises a very large number of servers that are grouped into thousands of clusters. Each cluster is deployed in a single data center and its size can vary from tens to many thousands of servers. We assume that incoming requests are forwarded to a particular server in a particular cluster by the CDN's load balancing algorithm. Load balancing in a CDN is performed at two levels: global load balancing, where a user request is sent to an "optimum" cluster, and local load balancing, where a user

request is assigned a specific server within the chosen cluster. Load balancing can be implemented using many mechanisms such as IP Anycast, load balancing switches, or most commonly, the DNS lookup mechanism [35]. We do not assume any particular mechanism, but we do assume that those mechanisms allow load to be arbitrarily re-divided and re-distributed among servers, both within a cluster (local) and across clusters (global). This is a good assumption for typical web workloads that form a significant portion of a CDN's traffic.

Energy Model. Since our goal is to minimize energy usage, we model how servers consume energy as a function of load. Based on our own testing of typical off-the-shelf server configurations used by CDNs, we use the standard linear model [9] where the power (in Watts) consumed by a server serving load λ is

$$power(\lambda) \triangleq P_{idle} + (P_{peak} - P_{idle})\lambda, \tag{3.1}$$

where the load $0 \leq \lambda \leq 1$ is the ratio of the actual load to the peak load, P_{idle} is the power consumed by an idle server, and P_{peak} is the power consumed by the server under peak load. We use typical values of 92 Watts and 63 Watts for P_{peak} and P_{idle} respectively. Though we use the linear energy model above in all our simulations, our algorithmic results hold for any power function that is convex.

In addition to the energy consumed by live servers that are serving traffic, we also capture the energy consumed by servers that are in transition, i.e., either being turned off or tuned on. Servers in transition cannot serve load but consume energy; this energy consumption is due to a number of steps that the CDN must perform during shutdown or startup. When a server is turned off, the load balancing system first stops sending any new traffic to the server. Further, the CDN must wait until existing traffic either dies down or is migrated off the server. Additionally, the control responsibilities of the server would need to be migrated out by performing leader election and other relevant processes. Once the server has been completely isolated

from the rest of the CDN, it can be powered down. When a server is turned on, these same steps are executed in the reverse. In both cases, a server transition takes several minutes and can be done automatically by the CDN software. To capture the energy spent during a transition, we model a fixed amount of energy usage of α Joules for each server transition, where α typically corresponds to 38 kilo Joules.

Workload Model. The workload entering the load balancing system is modeled as a discrete sequence λ_t, $1 \leq t \leq n$, where λ_t is the average load in the t^{th} time slot. We always express load in the normalized unit of actual load divided by peak server capacity.[1] Further, we assume that each time slot is δ seconds long and is large enough for the decisions made by the load balancing algorithm to take effect. Specifically, in our experiments, we choose a typical δ value of 300 seconds.

Algorithmic Model for Load Balancing. While a real-life load balancing system is complex [35], we model only those aspects of such a system that are critical to energy usage. For simplicity, our load balancing algorithms redistribute the incoming load rather than explicitly route incoming requests from clients to servers. The major determinant of energy usage is the number of servers that need to remain live (i.e., turned on) at each time slot to effectively serve the incoming load. The exact manner in which load is distributed to those live servers is less important from an energy standpoint. In fact, in the linear energy model described in Equation 3.1, the precise manner in which load is distributed to the live servers makes no difference to energy consumption.[2] In reality, the precise manner in which the load is distributed to the live servers does matter greatly from the perspective of managing footprint and other server state. However, we view this a complementary problem to our own and methods exist in the research literature [6, 14] to tackle some of these issues.

---

[1]For simplicity, we assume that the servers in the CDN are homogeneous with identical capacities, though our algorithms and results can be easily extended to the heterogeneous case.

[2]In the more general model where the power function is convex, distributing the load evenly among the live servers minimizes energy consumption.

The local load balancing algorithm of a CDN balances load between live servers of a given cluster. In each time interval t, the algorithm distributes the load λ_t that is incoming to that cluster. Let m_t denote the number of live servers in the cluster. Servers are typically not loaded to capacity. But rather a *target load threshold* Λ, $0 < \Lambda \leq 1$, is set such that the load balancing algorithm attempts to keep the load on each server of the CDN to no more than the fraction Λ of its capacity. Mathematically, if $l_{i,t}$ is the load assigned to live server i at time t, then $\sum_{i=1}^{i=m_t} l_{i,t} = \lambda_t$ and $l_{i,t} \leq \Lambda$, for $1 \leq i \leq m_t$. In addition to serving the current load, the load balancing algorithm also decides how many additional servers need to be turned on or off. The changes in the live server count made in time slot t is reflected in m_{t+1} in the next time slot.

The global load balancing algorithm works in an analogous fashion and distributes the global incoming load to the various server clusters. Specifically, the global incoming load is partitioned between the server clusters such that no cluster receives more than a fraction Λ of its capacity. Further, clients are mapped to proximal clusters to ensure good performance.

Online versus Offline. The load balancing algorithms work in an *online* fashion where decisions are made at time t without any knowledge of the future load $\lambda_{t'}$, $t' > t$. However, our work also considers the offline scenario where the load balancing algorithm knows the entire load sequence $\lambda_t, 1 \leq t \leq n$ ahead of time and can use that knowledge to make decisions. The offline algorithms provide the theoretically best possible scenario by making future traffic completely predictable. Thus, our provably-optimal offline algorithms provide a key baseline to which realistic online algorithms can be compared.

Metric Definitions. We are interested in the interplay of three metrics: energy reduction, service availability as it relates to customer SLA's, and server transitions. The energy reduction achieved by an algorithm that can turn servers on or off equals the percentage energy saved in comparison to a baseline where all servers remain

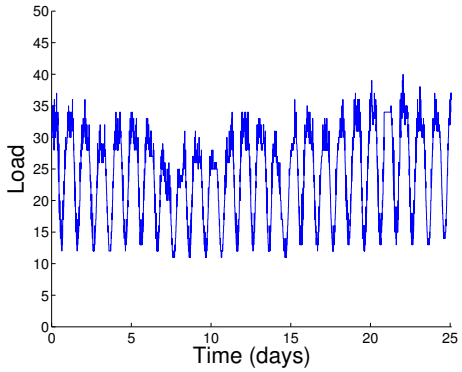


Figure 3.1: Average load per server measured every 5 minutes across 22 Akamai clusters in the US over 25 days. Note load variations due to day, night, weekday, weekend, and holidays (such as low load on day no. 8, which was Christmas).

turned on for the entire period. Since most CDNs today are not aggressively optimized for energy, the baseline is representative of the actual energy consumption of such systems. A server cluster that receives more load than the total capacity of its *live* servers cannot serve that excess load which must be dropped. The client requests that correspond to the dropped load experience a denial of service. The service availability over a time period is computed as $100 * (\textit{total served load})/(\textit{total input load})$. Finally, the server transitions are expressed either as total amount over the time period, or as an average amount expressed as the number of transitions per server per day.

Empirical Data from the Akamai Network. To validate our algorithms and to quantify their benefits in a realistic manner, we used extensive load traces collected over 25 days from a large set of Akamai clusters (data centers) in the US. The 22 clusters captured in our traces are distributed widely within the US and had 15439 servers in total, i.e., a representative sampling of Akamai's US deployments. Our load traces account for a peak traffic of 800K requests/second and an aggregate of 950 million requests delivered to clients.

The traces consist of a snapshot of total load served by each cluster collected every 5-minute interval from Dec 19th 2008 to January 12th 2009, a time period that includes the busy holiday shopping season for e-commerce traffic (Figure 3.1).

3.4 Local Load Balancing

We explore energy-aware algorithms for local load balancing in a CDN. First, we derive optimal offline algorithms that provably provide the maximum energy reduction that is theoretically possible (Section 3.4.1). Then, we derive practical online algorithms and evaluate them on realistic load traces from a CDN (Section 3.4.2), paying particular attention to how well they do in comparison to the theoretical baselines provided by the offline algorithms.

3.4.1 An Optimal Offline Algorithm

Given the entire input load sequence, $\lambda_t, 1 \leq t \leq n$, for a cluster of M servers and a load threshold Λ, an offline algorithm produces a sequence m_t, $1 \leq t \leq n$, where m_t is the number of servers that need to be live at time slot t. Note that given the output schedule, it is straightforward to create an on-off schedule for the servers in the cluster to achieve the number of live servers required at each time step. The global load balancing algorithm ensures that the input load sequence can be feasibly served by the cluster if all M servers are live, i.e., $\lambda_t \leq \Lambda M$ for all $1 \leq t \leq n$. In turn, an energy-aware local load balancing algorithm orchestrates the number of live servers m_t such that load λ_t can be served by m_t servers without exceeding the target load threshold Λ, i.e., $\lambda_t \leq \Lambda m_t$, for all $1 \leq t \leq n$. Assuming that load λ_t is evenly distributed among the m_t live servers, the energy expended in the cluster for serving the input load sequence equals

$$\delta \sum_{t=1}^{t=n} m_t \cdot power(\lambda_t/m_t) + \alpha \sum_{t=1}^{t=n} |m_t - m_{t-1}|,$$

where the first term is the energy consumption of the live servers and the second is the total energy for server transitions.

We develop an optimal offline local load balancing algorithm OPT using dynamic programming. Algorithm OPT produces a schedule $m_t, 1 \leq t \leq n$, that can serve

the input load with the smallest energy usage. We construct a two-dimensional table $E(t, m)$ that denotes the minimum energy required to serve the load sequence $\lambda_1, \lambda_2, \cdots, \lambda_t$ while ending with m live servers at time t. We assume that the algorithm begins at time zero with all M servers in live state. That is, $E(0, m) = 0$, if $m = M$, and $E(0, m) = +\infty$, if $m \neq M$. We inductively compute the table entries as follows:

$$\begin{aligned} E(t,m) &= \min_{0 \leq m' \leq M} \{E(t-1,m') + \delta m \cdot power(\lambda_t/m) \\ &+ \alpha \cdot |m - m'|\},\ if\ \lambda_t \leq \Lambda m \\ &= +\infty,\ otherwise \end{aligned} \tag{3.2}$$

Specifically, if it is feasible to serve the current load λ_t with m servers, we extend the optimal solution for the first $t-1$ steps to the t^{th} step using Equation 3.2. The first term in Equation 3.2 is the cost of a previously computed optimal solution for the first $t-1$ steps, the second term denotes the energy consumed by the live servers in time slot t, and the third term denotes the energy consumed in transitioning servers at time slot t. If it is infeasible to serve the current load with m servers, we set the optimal cost $E(t, m)$ to infinity. Once the table is filled, the optimal solution corresponds to entry $E(n, m)$ such that $E(n, m) = \min_{0 \leq s \leq M} E(n, s)$. The theorem below follows.

Theorem 1. *Algorithm* OPT *produces an optimal load balancing solution with the smallest energy consumption in time $O(nM^2)$ and space $O(nM)$, where n is number of time slots and M is the number of servers in the cluster.*

Since we are also interested in knowing how much energy reduction is possible if we are only allowed a small bounded number of server transitions, we develop algorithm OPT(k) that minimizes energy while maintaining the total number of server transitions to be at most k. To this end, we use a three-dimensional table $E(t, m, k)$,

$0 \leq t \leq n$, $0 \leq m \leq M$, and $0 \leq k \leq K$. (For simplicity, we assume that all entries of $E(t, m, k)$ with arguments outside the allowable range equal $+\infty$.) $E(t, m, k)$ is the optimum energy required to serve the input load sequence $\lambda_1, \lambda_2, \cdots, \lambda_t$ while ending with m live servers at time t and incurring no more than k transitions in total. Since we start with all servers live at time zero, $E(0, m, k) = 0$, for all $0 \leq k \leq K$, provided $m = M$. And, $E(0, m, k) = +\infty$, for all $0 \leq k \leq K$, if $m \neq M$. The table is filled inductively using the following formula:

$$\begin{aligned} E(t, m, k) &= \min_{m-k \leq m' \leq m+k} \{E(t-1, m', k - |m - m'|) \\ &+ \delta m \cdot power(\lambda_t / m) \\ &+ \alpha \cdot |m - m'|\}, \ if \ \lambda_t \leq \Lambda m \\ &= +\infty, \ otherwise \end{aligned} \tag{3.3}$$

For each $0 \leq k \leq K$, the optimal energy attainable with at most k transitions is simply $E(n, m, k)$ such that $E(n, m, k) = \min_{0 \leq s \leq M} E(n, s, k)$. The theorem follows.

Theorem 2. *Algorithm* OPT(k) *produces the optimal solution with the least energy and no more than k total server transitions. $OPT(k)$ can be computed for all $0 \leq k \leq K$ in time $O(nM^2K)$ and space $O(nMK)$.*

Empirical Results. We ran algorithm OPT with a typical value of the load threshold ($\Lambda = 75\%$) on our CDN load traces from 22 geographically distributed clusters of a large CDN over a span of 25 days. Figure 3.2 shows the $y\%$ of clusters that achieved at least $x\%$ energy reduction, for $0 \leq x \leq 100$. For each of the 22 clusters, OPT achieved energy reduction in the range 52% to 87%. Further, viewing all the clusters of the CDN as a single system, the system-wide energy reduction of OPT across all the clusters was 59.5%. Thus, significant gains are possible in the offline scenario by optimally orchestrating the number of live servers in each cluster.

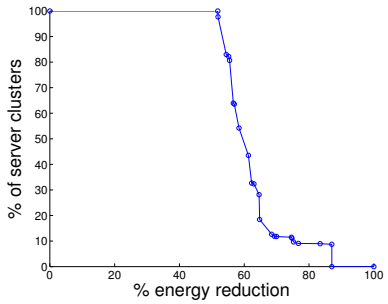


Figure 3.2: Optimal Offline Energy Reduction. The median cluster achieved a 58% reduction.

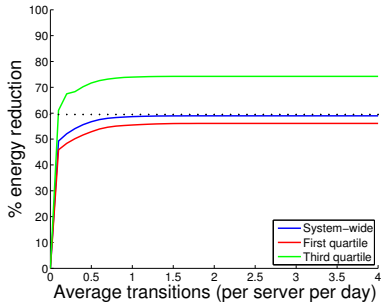
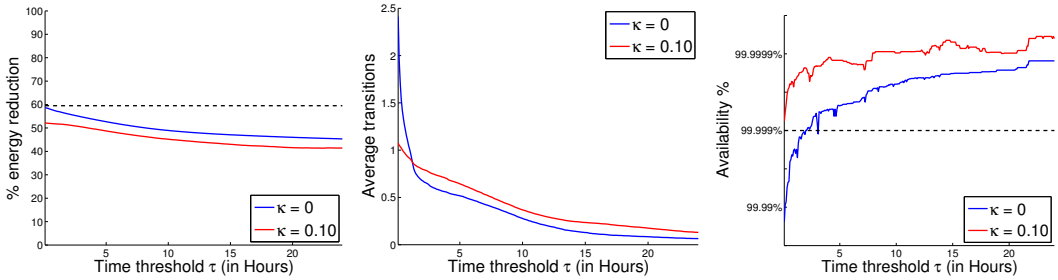


Figure 3.3: Energy reduction attainable with bounded server transitions. About 98.6% of the optimal reduction can be achieved with just 1 transition per server per day. The dotted-line represents the optimal reduction with unbounded transitions.

Next, we study how much energy reduction is possible if the server transitions are bounded and are required to be infrequent. Figure 3.3 shows the optimal system-wide energy reduction for each value of the average transitions that is allowable. These numbers were obtained by running algorithm OPT(k) for all clusters for a range of values of k. As more transitions are allowed, more energy reduction is possible since there is a greater opportunity to turn servers on and off in response to load variations. As the transition bound become large the energy reduction asymptotically reaches the maximum reduction possible for the unbounded case of 59.5%. The key observation however is that even with a small number of transitions, say 1 transition per server per day, one can achieve at least 58.66% system-wide energy reduction in the offline setting. In other words, with an average of just 1 transition per server per day one can obtain 98.6% of the energy reduction benefit possible with unbounded



(a) Energy reduction decreases as τ increases as spare servers are kept live longer.

(b) Average Transitions (per server per day) is acceptably small at about $\tau = 2$.

(c) Availability is significantly enhanced by holding a 10% pool of live spare servers.

Figure 3.4: The three key metrics for algorithm Hibernate on typical CDN load traces.

transitions. Besides system-wide energy reduction, Figure 3.3 also shows the variation in the energy reduction across clusters by plotting the first and third quartile values for each transition bound.

Note that algorithms OPT and OPT(k) never drop any load and achieve an SLA of 100% availability, since they are offline algorithms with complete knowledge of the entire load sequence. After computing the entire sequence of live servers, $m_t, 1 \leq t \leq n$, an offline algorithm ensures that m_t live servers are available at time t by transitioning $|m_t - m_{t-1}|$ servers at time $t-1$.

3.4.2 Online Algorithms

In contrast to offline algorithms, an online algorithm knows only the past and current load but has no knowledge of the future load. This accurately models any real-life load balancing system. At time t, an online algorithm does not know load λ_{t+1} and must estimate the number of servers to transition at the current time step t so that they are available to serve the load at $t+1$. Achieving a balance between the three metrics of energy reduction, transitions, and service availability that impacts customer SLAs is challenging. If the algorithm keeps a larger number of live servers to serve future load than is necessary, then the energy consumption is increased. In

contrast, if the algorithm keeps too few live servers, then some load might have to be dropped leading to decreased availability and potential customer SLA violations. Our key contribution in this section is algorithm Hibernate that achieves the "sweet spot" with respect to all three metrics, both for typical CDN traffic and flash crowds. While Hibernate only uses the past and current load to make decisions, it is also possible to use workload forecasting techniques to predict the future workload and use these predictions to enhance the efficacy of Hibernate. The design of such a predictive Hibernate is future work.

Algorithm Hibernate takes two parameters as input, a spare capacity threshold $0 \leq \kappa \leq 1$ and a time threshold $\tau \geq 0$. A key aspect of the algorithm is that it manages a pool of live servers that are considered "spare" in the sense that they are in excess of what is necessary to serve the current traffic. Intuitively, spare servers are kept as a buffer to help absorb unpredictable traffic surges in the future. For simplicity, assume that the servers in the cluster are numbered from 1 to M. Further, assume that the first m_t servers are live at time t, while the rest of the servers are turned off. At each time t, the algorithm does the following.

- Serve the current load λ_t using the current set of m_t live servers. If $\lambda_t > m_t$, the live capacity of the cluster is insufficient to serve the input load. In this case, a load amount of $\lambda_t - m_t$ is dropped and the rest of the load is served.

- The number of live servers deemed necessary to serve load λ_t is $\lceil \lambda_t/\Lambda \rceil$, where Λ is the target load threshold of the CDN. If $m_t > \lceil \lambda_t/\Lambda \rceil$, then the live servers numbered $\lceil \lambda_t/\Lambda \rceil + 1$ to m_t are marked as "spare".

- The spares are managed according to two rules:

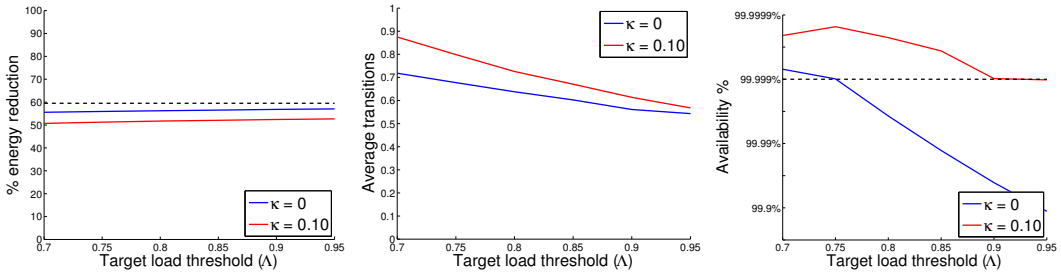
  - **Spare Capacity Rule:** Target at least $\lceil \kappa M \rceil$ servers to be kept as spare, where $0 \leq \kappa \leq 1$. Specifically, if the number of spares $m_t - \lceil \lambda_t/\Lambda \rceil$ is smaller than $\lceil \kappa M \rceil$, then turn on $\min\{\lceil \kappa M \rceil + \lceil \lambda_t/\Lambda \rceil, M\} - m_t$ servers. (The servers turned on in

the current time step t will be live and available to serve load only in the next time step $t+1$.)

- **Hibernate Rule:** If a server was considered spare in each of the last τ time slots it is a candidate for being turned off, similar to how a laptop hibernates after a specified period of idleness. However, the hibernate rule is applied only to servers in excess of the spare capacity threshold. Specifically, if the number of spares $m_t - \lceil \lambda_t/\Lambda \rceil$ is more than $\lceil \kappa M \rceil$, then examine servers numbered $\lceil \lambda_t/\Lambda \rceil + \lceil \kappa M \rceil + 1$ to m_t and turn off any server that was marked as spare in *all* of the last τ time steps.

Empirical Results. We ran algorithm Hibernate on typical CDN load traces collected over 25 days and across 22 clusters for multiple values of τ and two values of κ with the results summarized in Figure 3.4. Note that as the time threshold τ increases, energy reduction and transitions generally decrease and availability generally increases. The reason is that as τ increases, live servers that are spare are turned off after a longer time period, resulting in fewer transitions. However, since more servers are left in an live state, the energy reduction is smaller, but availability is larger as the additional live servers help absorb more of the unexpected load spikes. The trade-off between requiring no spare capacity ($\kappa = 0$) and requiring a 10% spare capacity ($\kappa = 0.1$) is also particularly interesting. If we fix a typical value of $\tau = 2$ hours, Hibernate provides an acceptable number of transitions (< 1 transition per server per day) with or without spare capacity. Requiring 10% spare capacity decreases the energy reduction by roughly 10%, since a pool of spare servers must be kept live at all times (Figure 3.4a). However, the modest decrease in energy reduction may well be worth it for most CDNs, since availability is much higher (five nine's or more) with 10% spare capacity than with no spare capacity requirement (Figure 3.4c).

Handling typical workload fluctuations: A key decision for a CDN operator is the target utilization Λ that the system should be run at in order to handle typical



(a) % Energy reduction increases if the servers run hotter. The dotted line represents the offline optimal energy reduction.

(b) Average transitions (per server per day) decreases with Λ as there is more effective capacity in each server.

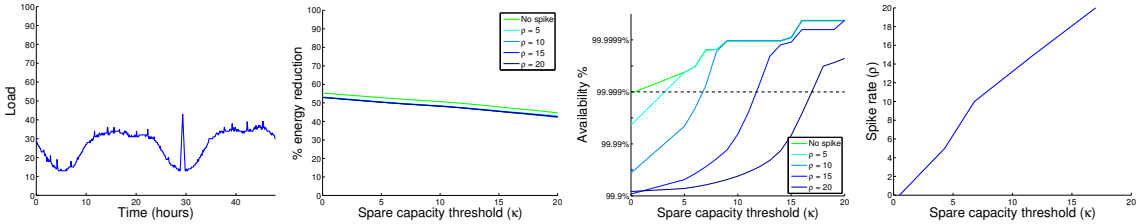
(c) Availability decreases as the system is run hotter but is enhanced by the addition of a 10% pool of live spare servers.

Figure 3.5: Variation of the three metrics with the target load threshold Λ

workload variations. The value of Λ is typically kept "sufficiently" smaller than 1 to provide some capacity headroom within each server to account for the inability to accurately estimate small load variations. In Figure 3.5, we quantify the tradeoffs associated with Λ as it pertains to our three metrics. Running the CDN "hotter" by increasing Λ would increase the system capacity and the server utilization. Note that as Λ increases, the effective capacity of each live server increases, resulting in fewer live servers being needed to serve the load. This results in increased energy reduction (Figure 3.5a) as well as a smaller number of transitions (Figure 3.5b). However, increasing Λ also decreases availability (Figure 3.5c) and potentially increases customer SLA violations. Note that availability decreases when there is less unutilized live server capacity in the cluster that can serve as a headroom for absorbing unpredictable load spikes. When Λ is increased, the unutilized live server capacity in the cluster decreases both due to the fact that fewer servers are kept live and due to the fact that each live server is loaded closer to its capacity, resulting in a loss of headroom and more load being dropped. Note also that requiring 10% spares ($\kappa = 0.1$) allows the CDN operator to run the system hotter with a larger Λ than if there were no spares ($\kappa = 0$) for the same availability SLA requirements. Thus, there

is a relationship between the target load/utilization Λ and the spares κ, since both paramaters permit some capacity "headroom" to handle workload variations. The hotter the system (higher Λs), the more κ needs to be to achieve the same SLA.

Handling Large Flash Crowds: A particular worry of CDN operators from the standpoint of powering off servers is the global flash crowd scenario where there is a large unexpected load spike across most clusters of the CDN. Note that a local flash crowd scenario that only affects some of the clusters, say just the northeastern US, is often easier to deal with, since the global load balancing system will redistribute some of the traffic outside that local region at some cost to performance. Global flash crowds that matter to a large CDN are rare but do occur from time to time. Some examples include 9/11, and the Obama inauguration. Since it is critical from the standpoint of a CDN operator to understand the behavior of any load balancing algorithm in a global flash crowd situation and since our actual CDN traces lacked a true global flash crowd event, we modified the traces to simulate one. To pick a worst-case scenario, we chose a low traffic period in the night when servers are likely to be turned off and introduced a large spike measuring 30% percent of the capacity of the cluster and lasting for a 1 hour period (Figure 3.6a.) Further, to simulate a global event we introduce the same spike at the same time in all the 22 clusters distributed across the US. A critical factor in a flash crowd is the *spike rate* ρ at which the load increases (or, decreases) in one time interval (Recall that the time interval models the "reaction time" of the load balancing system which in our case is 300 seconds). We ran algorithm Hibernate for different settings of the spike rate ρ and the spare capacity threshold κ with the results summarized in Figure 3.6. As κ increases, more servers need to be held live and the energy reduction decreases in a roughly linear fashion in all the simulated scenarios (Figure 3.6b). The average transitions also stayed within the accepted range of less than 1 transition per server per day in all cases. However, a direct relationship was observed between the spike rate ρ and spare



(a) A simulated load spike in the Ashburn cluster

(b) % Energy reduction decreases with additional live spares

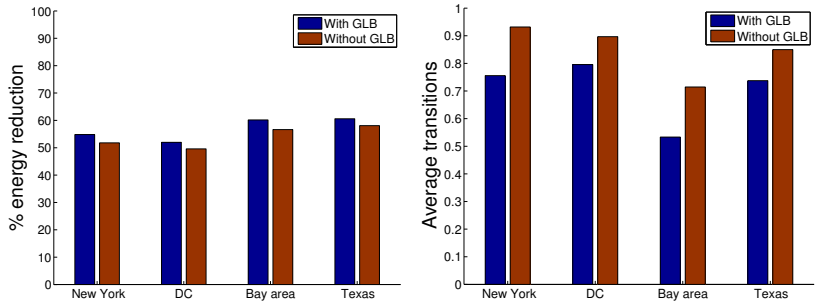
(c) More live spares help increase availability

(d) Spares (κ) needed to absorb a spike rate ρ with 99.999% availability

Figure 3.6: The behavior of Hibernate during a simulated global flash crowd

capacity threshold κ where a larger ρ was tolerable only with a corresponding larger value of κ to sustain the required levels of service availability and meeting customer SLAs (Figure 3.6c). To absorb a spike rate of ρ with at least five nine's of availability (99.999%) a commensurately large value of κ is required (Figure 3.6d). Since the spike rate can be deduced from prior global flash crowds, this gives clear guidance to CDN operators on how much spare capacity must be held live at all times to absorb even large flash crowds.

3.5 Global Load Balancing



(a) % Energy reduction

(b) Average Transitions (per server per day)

Figure 3.7: Energy reduction and transitions show only modest improvements with global load balancing

Cities	With GLB	Without GLB
New York	100%	99.9993%
DC	100%	99.9996%
Bay Area	100%	99.9995%
Texas	100%	99.9997 %

Figure 3.8: Availability improves drastically with global load balancing

In prior sections, we devised energy-aware schemes for *local* load balancing that redistribute load across servers within the *same cluster*. A natural question is what can be gained by energy-aware *global* load balancing that can redistribute load across *different clusters* of the CDN. An important requirement for global load balancing is that each request is served from a cluster that is "proximal" to the client, so as to ensure good network performance. However, a large CDN with wide deployments may have several clusters that can all provide equivalently good performance to a given client. Thus, global load balancing typically has numerous choices of clusters to serve a given portion of the incoming load. While there are other considerations such as bandwidth costs[4] that come into play, we focus on energy consumption and ask the following key question. Does redistributing load across clusters that can provide equivalent performance further help optimize energy reduction, transitions, and availability?

To answer the above question empirically using our CDN trace data, we create *cluster sets* from the 22 clusters for which we have load traces. Each cluster set consists of clusters that are likely to have roughly equivalent performance so as to allow global load balancing to redistribute load between them. To form a cluster set we choose clusters that are located in the same major metropolitan area, since network providers in a major metro area tend to peer well with each other and can likely to provide equivalently good performance to clients from the same area. For instance, our Bay Area cluster set consisted of clusters located in Palo Alto, San Francisco, San Jose, and Sunnyvale, our DC metro area cluster set consists of clusters in Ashburn

and Sterling, and our New York metro area cluster set consists of clusters in New York and Newark. Further, since a large CDN is likely to have more than a dozen clusters in each major metro area and since we only a have trace data for a subset of the clusters of a large CDN, we simulate eight clusters from each actual cluster by dividing up the traces into eight non-overlapping periods of 3-days each and aligning the 3-day traces by the local time of day. To simulate the baseline scenario with no energy-aware global load balancing, we ran our algorithm Hibernate individually on each cluster. Note that in this case the incoming load to a cluster as represented in the traces is served by the same cluster. Now, to simulate energy-aware global load balancing, we viewed each cluster set as a single large cluster with the sum total of the capacities of the individual clusters and sum total of the incoming load. We then ran Hibernate on the large cluster. Thus we allow the incoming load to be redistributed in an arbitrary fashion across the clusters within a cluster set.

The results of our evaluation are summarized in Figures 3.7 and 3.8. The additional energy savings due to global load balancing were modest in the 4% to 6% range. The reason is that clusters within the same cluster set are *broadly* similar in the their load patterns, with the peak and off-peak loads almost coinciding in time. Thus, global load balancing is not able to extract significantly more energy savings by moving load across clusters (Figure 3.7a), over and above what can be saved with local load balancing. However, a 10% to 25% reduction in the average transitions can be achieved by global load balancing, since there are occasions where load spikes in one cluster can be served with live spare capacity in a different cluster by redistributing the load rather than incurring server transitions (Figure 3.7b). But, perhaps the key benefit of global load balancing is the increased availability (Figure 3.8). The enhanced availability is due to an "averaging" effect where an unpredictable upward load fluctuation that would have caused some load to be dropped within a single cluster can be routed to a different cluster that happened to have a corresponding

downward load fluctuation leaving some spare live capacity in that cluster. In fact, in our simulations, the availability was nearly 100% with global load balancing in all cluster sets.

3.6 Related Work

Our work differs from prior work on energy management by the focus on CDNs with a particular emphasis on the interplay between energy management and the local/global load balancing algorithms in the CDN. We also examine the impact of shutting servers on client SLA as well as the impact of server transitions on wear and tear.

A recent effort related to our work is [23]. Like us, this paper also presents offline and online algorithms for turning servers on and off in data centers. While [23] targets data center workloads such as clustered mail servers, our focus is on CDN workloads. Further, [23] does not emphasize SLA issues, while in CDNs, SLAs are the most crucial of the three metrics since violations can result in revenue losses. Two recent efforts have considered energy-performance or energy-QoS tradeoff in server farms [17, 19]. Our empirical results also show an energy-SLA tradeoff with a primary focus on choosing system parameters to obtain five 9s of availability in CDNs.

3.7 Conclusions

In this chapter we proposed energy optimization techniques to turn off CDN servers during periods of low load while seeking to balance the interplay of three key design objectives: maximize energy reduction, minimize the impact on client-perceived availability (SLAs), and limit the frequency of on-off server transitions to reduce wear-and-tear and its impact on hardware reliability. We proposed an optimal offline algorithm and an online algorithm to extract energy savings both at the level of local load balancing within a data center and global load balancing across data

centers. Our evaluation using real production workload traces from a large commercial CDN showed that it is possible to reduce the energy consumption of a CDN by 51% while ensuring five nine's of service availability and an average of just 1 transition per server per day. Further, we show that keeping even 10% of the servers as hot spares helps absorb load spikes due to global flash crowds with little impact on availability SLAs. Our future work will focus on the incorporation of workload prediction techniques into our Hibernate algorithm, further optimizations of the global load balancing algorithm from an energy perspective and techniques for managing footprint (disk state) of CDN customers while turning servers on and off.

CHAPTER 4

ENERGY EFFICIENCY THROUGH CLUSTER SHUTDOWN

In this chapter we propose and evaluate a novel CDN-specific technique called *cluster shutdown* where an entire cluster of servers in a CDN data center can be turned off. Cluster shutdown is easily integrated into the global load balancer (GLB) that will now have the ability to move all load away from a cluster and shut it down. However, since the granularity of energy management is to turn off entire clusters or leave them entirely on, the technique does not have the ability to turn off individual servers (e.g., a fraction of a cluster). In contrast, the server shutdown technique studied in [29] has the ability to shutdown individual servers within the cluster depending on the load, but has has no ability to control how much load enters a cluster. Therefore, in this sense, the two techniques are complementary and may be implemented together. While cluster shutdown has not been studied before in the CDN context, it has certain natural advantages that make it worthy of consideration for CDN energy reduction.

(1) Redundant deployments. Large CDNs such as Akamai can have over a thousand clusters deployed in data centers around the world [35] with more than a dozen redundant deployments in any given geographical area. Thus, when some clusters near a user are shutdown during off-peak hours, other nearby active clusters can continue to provide CDN service to users and ensure good availability and performance. In fact, one of the contributions of this work is determining the impact of cluster shutdowns on user performance.

(2) Cluster shutdown is consistent with the original CDN architectural design. Each cluster in a CDN is often architected to be a self-sufficient unit with enough processing and disk storage to serve the content and application domains that are assigned to it [35]. In particular, there is limited data dependency and resource sharing across clusters. Thus, cluster shutdown can be implemented with little or no changes to the CDN's original architecture. In contrast, servers within a cluster are closely linked in a fine-grained fashion and they cooperatively cache and serve the incoming requests. For instance, servers *within* the same cluster cooperatively store application state and content for user requests served by that cluster. Thus, shutting down individual servers for energy savings requires greater migration of state and content between servers in a cluster at levels not customary in a CDN today. Cluster shutdown, in contrast, does not require state migration and cached content is already replicated across clusters for fault-tolerance purposes, which ensures that availability is not impacted by shutting down a cluster. In this sense, cluster shutdown is a better architectural design choice for energy management than server shutdown.

(3) Cluster shutdown has the potential to save on cooling power in addition to IT power. A key advantage of cluster shutdown is that the *all* of the energy consumed by a cluster, which includes energy consumed by the servers, the network equipment, and the cooling within that cluster, can be saved when a cluster is turned off. In contrast, a server shutdown technique will typically turn off a fraction of the servers within the cluster and will require the networking and cooling equipment to stay on. The cooling equipment is not energy proportional—thus turning off a fraction of the servers only saves energy consumed by those servers and does not yield a proportionate reduction in cooling costs.

For cluster shutdown to be effective, a CDN would need to have control over *all* of its energy consumption, i.e., both IT (such as servers) and cooling equipment. Such a scenario is reasonable given the trend for CDN's to opt for self-contained, modular

[45], or containerized [41] deployments. With such deployments a CDN can manage the power consumption of its own cluster, independent of other tenants in the data center – an advantage for a CDN that wants manage its power consumption closely. The savings that can be obtained from reducing cooling costs can have a significant impact on the total energy expenditure of a cluster. The key reason is that the energy consumed by cooling equipment is a significant fraction of the energy expended by the IT equipment[1] such as servers. The ratio of total energy to IT energy is a standard metric called PUE (Power Usage Effectiveness) that has a typical value[2] of about 2 implying cooling energy is roughly equal to IT energy in typical data center deployments. But in more recent energy-efficient designs, PUE is smaller but cooling energy is still a significant fraction of the IT energy. Further, cooling energy consumption is *not* power-proportional since cooling still takes a significant amount of energy even when the servers have low utilization and are not producing much heat, resulting in disproportional energy savings when cooling is shutdown entirely (cf. Figure 4.1a).

Despite these advantages, a cluster shutdown technique is not without disadvantages when compared to server shutdown [29]. Shutting down a cluster and moving all its users to other clusters might degrade performance for users if they have to go "farther away" in the network sense for their content. Further, moving traffic across clusters has the potential of increasing the bandwidth cost, even if it reduces energy. A primary focus of our work then is to evaluate the energy reduction provided by cluster shutdown and how it trades off against potential degradation in performance and increases in bandwidth costs.

---

[1]IT energy expenditure is primarily the energy consumed by the servers, since the networking equipment consume significantly less. Likewise, cooling energy expenditure is dominated by the energy consumed by the chillers[39].

[2]In a survey by the Uptime Institute [48] in July 2012 , data centers reported an average PUE between 1.8 to 1.89. Other estimates place PUEs even higher.

4.1 Contributions

We propose algorithms for incorporating cluster shutdown in the GLB of a CDN and quantify the energy savings achievable by this technique. Our evaluation uses extensive real-world traces collected from 22 geographically distributed clusters over 25 days from one of the world's largest CDNs. We show how energy savings are impacted by the energy characteristics of servers, cooling equipment, and data centers. Further, we quantify the tradeoffs between three goals of CDN architecture: saving energy, reducing bandwidth costs, and enhancing end-user performance. Finally, we compare the relative efficacy of cluster shutdown with the well-studied and complementary approach of server shutdown. Our specific key contributions are as follows.

- We propose a GLB algorithm that minimizes energy by routing traffic away from certain clusters and switching them off. On production CDN workloads with typical assumptions for server and cooling efficiencies, our algorithm achieved a significant system-wide reduction in CDN energy consumption of 67%.

- When servers and cooling equipment are energy inefficient, the energy savings from cluster shutdown can be as large as 73%. These savings can decrease to 61% if the servers become perfectly power proportional, and can further become almost zero if the cooling also becomes perfectly efficient.

- The outside air temperature has an impact on cooling efficiency and hence influences the energy savings achievable by cluster shutdown. Energy savings are stable at about 67% for outside temperatures less than $85°F$ but tapers off as the temperature rises to 44% at $100°F$.

- To obtain the maximum possible energy savings, bandwidth costs of the CDN would have to increase by a factor of 2. However, 73% of the maximum energy savings are obtainable with no change in bandwidth costs at all. Likewise, 93% of the maximum

energy savings is obtainable with no significant performance degradation with each user served from clusters within an average distance of 500 km.

- Frequent cluster shutdowns and the operational overheads that it would entail are not necessary to achieve significant energy savings. Our technique is able to extract 79% of the maximum savings even when limiting each cluster to at most one shutdown per day and even when the incoming load is not known in real-time and must be predicted.

- Realistic CDNs are required to operate under multiple constraints. We identify a sweet spot where our technique provides 22% of maximum savings while limiting each cluster to at most one shutdown per day, allowing no increase in bandwidth costs and serving users from clusters within an average distance of 800 km.

- Cluster shutdown does better than server shutdown within a broad operating range of outside air temperatures from $40°F$ to $90°F$, while server shutdown is better outside of this range. In general, cluster shutdown performs better during lower periods of CDN utilization, while server shutdown has the edge at higher utilization.

- Augmenting cluster shutdown with server shutdown has limited impact under relaxed performance or bandwidth constraints because the CDN is already nearly power proportional under these conditions with just cluster shutdown. However, if either latency or bandwidth costs need to be kept low, server shutdown can provide significant additional gains over a pure cluster shutdown strategy. If low latency is required, server shutdown can provide an additional 46% in energy savings. Likewise, if no increase in bandwidth costs are allowed, the additional energy savings is 34%.

4.2 Background, Models, and Methodology

4.2.1 Content Delivery Networks

Our work assumes a global content delivery network (CDN) that comprises a very large number of servers that are grouped into thousands of clusters. Each cluster is deployed in a single data center and its size can vary from tens to many thousands of servers. The incoming requests are forwarded to a particular server in a particular cluster by the CDN's load balancing algorithm. As outlined earlier, load balancing in a CDN is performed in two stages: global load balancing (GLB) that routes a user's request to an "optimum" cluster, and local load balancing (LLB) that assigns the user request to a specific server within the chosen cluster. Load balancing can be implemented using many mechanisms such as IP Anycast, load balancing switches, or most commonly, the DNS lookup mechanism [35]. We do not assume any particular mechanism, but we do assume that those mechanisms allow load to be arbitrarily re-divided and re-distributed among servers, both within a cluster (local) and across clusters (global). This is a good assumption for typical web workloads that form a significant portion of a CDN's traffic.

Our proposed technique of cluster shutdown is implemented in the GLB of a CDN. First, GLB moves away all the traffic from a cluster, typically by setting the cluster capacity to zero. Then, the cluster is shutdown by turning off all the relevant components, inclusive of servers and cooling equipment. Since our focus is on GLB algorithms that incorporate cluster shutdown, unless mentioned otherwise, we assume that the LLB evenly distributes the incoming load assigned by the GLB across servers within that cluster. In contrast, the server shutdown mechanism studied in [29] is incorporated within the LLB system that turns off individual servers within a cluster.

4.2.2 Workload Model

The workload entering a CDN is generated by users around the world accessing web pages, video content, and Internet-based applications. To model the spatial distribution of the users, we cluster them according to their geographical location. In particular, we define M *client locations* where each location is a compact geographical area, example, Massachusetts, USA. The workload entering the CDN is modeled as a discrete sequence[3] $\lambda_{t,i}$, $1 \leq t \leq T$ and $1 \leq i \leq M$, where $\lambda_{t,i}$ is the average load in the t^{th} time slot from users in client location i. We always express load in the normalized unit of actual load divided by peak server capacity.[4] Further, we assume that each time slot is δ seconds long and is large enough for the decisions made by the global load balancing algorithm to take effect. Specifically, in our experiments, we consider $\delta = 5$ minutes.

4.2.3 Algorithmic Model for Load Balancing

While a real-life load balancing system is complex [35], we model only those aspects of such a system that are critical to energy usage. For simplicity, our load balancing algorithms redistribute the incoming load rather than explicitly route incoming requests from clients to servers. The major determinant of energy usage is the number of clusters that need to remain active (i.e., turned on) at each time slot to effectively serve the incoming load. Unless we mention otherwise, we assume that local load balancer is not energy aware and does not turn servers on and off on its own accord. But, rather, the LLB simply distributes the load assigned to each cluster evenly among the servers in that cluster. However, the GLB is energy aware and can

[3]When the time slot is implicit, we often drop the time subscript from our notation. For instance, we describe the incoming load simply λ_i, $1 \leq i \leq M$.

[4]For simplicity, we assume that the servers in the CDN are homogeneous with identical capacities, though our algorithms and results can be easily extended to the heterogeneous case.

turn clusters on or off. Therefore a cluster is either active with all servers turned on, or inactive with all servers turned off.

At each time slot, an energy aware GLB takes as input the incoming load λ_i, $1 \leq i \leq M$. The global load balancing algorithm of a CDN routes the incoming load from each client location i to clusters that are active at that time step, i.e., GLB determines the values μ_{ij} that represents the load induced by client location i on a server in the j^{th} cluster, $1 \leq j \leq N$, such that

$$\sum_{1 \leq j \leq N} \mu_{ij} c_j = \lambda_i, \forall i,$$

where c_j is the number of servers in that cluster. Servers are typically not loaded to capacity. But rather a *target load threshold* μ_{max}, $0 < \mu_{max} \leq 1$, is set such that the load balancing algorithm attempts to keep the load on each server of the CDN to no more than μ_{max}. Mathematically,

$$\sum_{1 \leq i \leq M} \mu_{ij} \leq \mu_{max}, \forall j.$$

We assume a typical value of $\mu_{max} = 0.75$ in our work, i.e., the target load for each server is 75% of its capacity.

4.2.4 Power consumption of clusters

We model both the power consumed directly by the servers (IT power) and the power consumed for cooling those servers (cooling power). By convention, we indicate power draw for a single server by using a superscript "server", while variables without that superscript represent the power draw for the entire cluster. Also, note that while we mostly discuss power draw (in Watts), integrating power draws over time provides us the energy consumed (in Joules).

4.2.4.1 Server power model

First, we model the power consumed by a single server as a function of its load. Based on our own testing of typical off-the-shelf server configurations used by CDNs, we use the standard linear model[9] where the power (in Watts) consumed by a server is

$$P^{\mathbf{IT,server}} = \left[P_{idle}^{\mathbf{IT,\ server}} + (P_{peak}^{\mathbf{IT,\ server}} - P_{idle}^{\mathbf{IT,\ server}})\lambda\right] \tag{4.1}$$

where the load ($0 \leq \lambda \leq 1$) is the server load, and $P_{peak}^{\mathbf{IT}}$ is power consumed by the server when it is loaded to its capacity (i.e., $\lambda = 1$). $P_{idle}^{\mathbf{IT,server}}$ is the power consumed by an idle server when it has no load (i.e., $\lambda = 0$). We define a quantity $0 \leq \alpha \leq 1$ called the *server power proportionality factor* where

$$\alpha \stackrel{\Delta}{=} 1 - P_{idle}^{\mathbf{IT,\ server}}/P_{peak}^{\mathbf{IT,server}}.$$

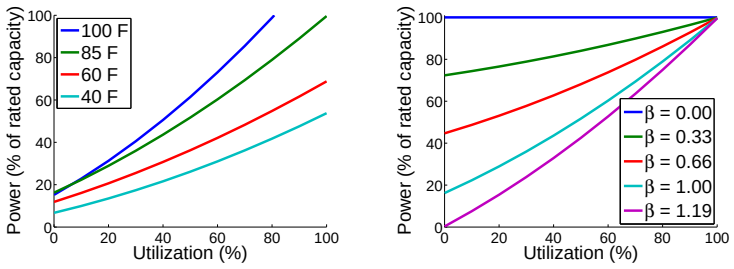
Note that $\alpha = 1$ represents a perfectly power proportional server—the ideal case for an energy-efficient server—while $\alpha = 0$ represents the opposite extreme. In our empirical work, unless mentioned otherwise, we use $P_{peak}^{\mathbf{IT,server}} = 92$ Watts, α =0.31, and $P_{idle}^{\mathbf{IT,server}} = 63$ Watts as typical values based on our measurements of a typical deployed server today. However, we also vary α over a wide range to study the impact of server power proportionality on our conclusions.

4.2.4.2 Cooling power model

The cooling systems deployed to cool a server cluster consist of a number of components. An air handler transfers heat out of the server room. An air or water based chiller cools down the hot air before it is pumped back into the server room.

The coolant, usually a combination of water and glycol[5] is transferred from the chiller to cooling towers that exchange heat with the outside air before returning it back to the chiller. The chiller plant's compressor accounts for the majority of the cooling cost in most data centers [39].

To make our model assumptions realistic, we use a set of benchmark regression curves provided by the California Energy Commission (CEC) [11] to model our cooling power requirements. Assuming efficient heat exchange at the cooling towers, we take the outside air temperature as a proxy for the temperature of the coolant on return. The power consumed by the chiller $P^{\mathbf{COOL}}$ is a *quadratic* function of its utilization u



(a) The chiller power $P^{\mathbf{COOL}}$ gets larger and steeper as outside temperature increases.

(b) As cooling efficiency β increases the cooling power required $P^{\mathbf{COOL}}(\beta)$ is smaller.

Figure 4.1: Cooling power and its dependence on outside air temperature and cooling efficiency.

as shown below [11], where $u \triangleq \frac{Q}{Q_{peak}}$, Q is the heat removed by the chiller, and Q_{peak} is maximum heat removal that the chiller is rated for.

$$P^{\mathbf{COOL}} = P^{\mathbf{COOL}}_{peak} \times \left(A + B \cdot u + C \cdot u^2\right) \tag{4.2}$$

[5]For the purposes of modeling a typical cooling system, we assume that the chilled water coolant is at $44°F$.

where u is the utilization of the chiller and the constants A, B, and C are dependent on the capacity correction factor (CAP_FT) and the efficiency correction factor (EIR_FT) that vary quadratically with the outside air temperature. Given a value for the outside air temperature the constants A, B, and C can be derived from the regression curves provided in the CEC manual [11]. *It is worth noting that a chiller consumes disproportionately more power at higher utilization than lower ones due to the quadratic nature of the curve.* Also, as shown in Figure 4.1a, as the outside air temperature gets higher the power required $P^{\mathbf{COOL}}$ gets larger and curve becomes more non-linear and steeper.

The chillers deployed in practice vary greatly in terms of their efficiency, ranging from less efficient older systems to highly efficient next-generation ones. To study this wide variation, we propose a family of chiller models that have the same quadratic functional form for the relationship between utilization and power consumed as the CEC chiller described in Equation 5.1 but different values for the constants. Specifically, we define a factor β that we call the *chiller efficiency factor* and each value of β provides a different curve for the chiller power consumption $P^{\mathbf{COOL}}(\beta)$ as described in the equation below.

$$P^{\mathbf{COOL}}(\beta) = P^{\mathbf{COOL}}_{peak} \times \left(A_\beta + B_\beta \cdot u + C_\beta \cdot u^2\right), \tag{4.3}$$

where $A_\beta = \max\{(\beta A + 1 - \beta), 0\}$, $B_\beta = \beta B$, and $C_\beta = \beta C$.

We study five chillers by setting β to five different values as shown in Figure 4.1b. The first three curves ($0 \leq \beta < 1$) represents chillers that are less efficient than CEC's chiller. As can be seen from Equation 4.3, the fourth curve with $\beta = 1$ models the CEC chiller exactly. And, the fifth curve with $\beta > 1$ models a next-generation chiller that is more efficient than the CEC chiller and has power consumption of zero when idle.

4.2.4.3 Total power consumed by a cluster

The total power consumed by a cluster is defined to the power needed to run the servers and associated equipment and the power needed to cool the servers. We define the IT power $P^{\mathbf{IT}}$ of a cluster to be the aggregate power consumed by the c servers of the cluster. In addition to the servers themselves, a cluster includes other IT equipment such as network switches and power distribution units. Typically the power consumed by networking and power distribution equipment is a small fraction of that consumed by the servers of the cluster (studies have shown this portion to be around 5-10% [39]). Our power model currently ignores the contribution of this other IT equipment to the total IT power, but it is straightforward to extend our models and algorithms to incorporate its contribution through a small multiplicative constant.

Thus, the IT power $P^{\mathbf{IT}}$ consumed by a server cluster consisting of c servers, each running at utilization λ, is

$$P^{\mathbf{IT}} = c \times P^{\mathbf{IT,server}} \tag{4.4}$$

where $P^{\mathbf{IT,server}}$ can be computed using Equation 4.1. And, the peak IT power of a cluster $P^{\mathbf{IT}}_{peak} = c \times P^{\mathbf{IT,server}}_{peak}$. Given the PUE of the data center in which the cluster is deployed, we determine the peak cooling power $P^{\mathbf{COOL}}_{peak} = (\text{PUE} - 1) \times P^{\mathbf{IT}}_{peak}$. Since the chiller removes the heat produced by the servers, the utilization of the chiller $u = \frac{P^{\mathbf{IT}}}{P^{\mathbf{IT}}_{peak}}$. Now, given the value of β that determines the cooling efficiency, we can compute the total power consumed by the chiller $P^{\mathbf{COOL}}(\beta)$ using Equation 4.3. The total power P consumed by the cluster is simply the sum of its IT and cooling power:

$$P = P^{\mathbf{IT}} + P^{\mathbf{COOL}}(\beta) \tag{4.5}$$

Note that the quadratic and non-energy proportional nature of the chiller-based cooling model has interesting implications on cluster and server shutdown techniques.

When a server shutdown technique switches off a fraction of the servers within a cluster, the non-energy proportional nature of the curve works "against" it and does not yield a proportional reduction in cooling energy usage, while a full cluster shutdown reduces the cooling costs to zero for that cluster. In contrast, cluster shutdown "packs" the load from a region onto a smaller number of clusters (and turns off the remaining clusters), but the quadratic nature of the curve yields *more than linear* increase in cooling costs for the clusters that stay on; the higher the cluster utilization due to such packing, the greater the increase in cooling cost due to the quadratic nature of the curve. A similar effect comes into play due to the outside air temperature, where increasing cluster utilization in hotter outside temperature causes a disproportionately larger increase in cooling costs due to the quadratic curve.

4.3 GLB Algorithms with Cluster Shutdown

We now describe our algorithm for global load balancing that routes traffic from client locations to clusters while turning clusters on or off with the goal of minimizing the total energy consumed by the CDN. At any given time, the algorithm takes as input the load λ_i from each individual client location i. Here we make the simplifying assumption that the GLB knows precisely the load that it needs to route at each time step and that it can instantaneously turn clusters on or off to minimize energy usage. This is clearly not strictly true in practice where both sensing the load and shutting down clusters incur a small delay. However, our algorithm provides a baseline on what is achievable with the cluster shutdown technique, leaving a more complex model that incorporates delays for future work. The output of our algorithm is two-fold. First, it computes a binary variable u_j that indicates whether the j^{th} cluster should be turned on ($u_j = 1$) or turned off ($u_j = 0$) in that time step. Next, it computes a quantity μ_{ij} that represents the load from client i that must be routed to cluster j at the given time step.

Computing the assignment of load to clusters can be stated as a convex optimization problem as follows. The IT power required by cluster j is

$$P_j^{\mathbf{IT}} = c_j \left[P_{idle}^{\mathbf{IT,server}} \times u_j + \sum_{1 \leq i \leq M} (P_{peak}^{\mathbf{IT,server}} - P_{idle}^{\mathbf{IT,server}}) \mu_{ij} \right],$$

where the value of u_j is used to determine if the cluster is turned on and idle power should be incurred. The chiller utilization of cluster j can be computed as $\frac{P_j^{\mathbf{IT}}}{P_{peak}^{\mathbf{IT}}}$. The corresponding cooling energy for cluster j denoted by $P_j^{\mathbf{COOL}}$ can be computed using Equation (4.3), given the chiller efficiency β. Our objective function is simply the total power drawn by the CDN summed across all its N clusters and is stated below.

$$\min \sum_{1 \leq j \leq N} \left(P_j^{\mathbf{IT}} + P_j^{\mathbf{COOL}} \right) \tag{4.6a}$$

$$s.t. \sum_{1 \leq j \leq N} \mu_{ij} c_j = \lambda_i, \qquad \forall i \tag{4.6b}$$

$$\sum_{1 \leq i \leq M} \mu_{ij} \leq u_j \mu_{max}, \quad \forall j \tag{4.6c}$$

The quantities that are varied in the minimization are the output variables $\mu_{i,j}$ and u_j. Equation 4.6b ensures that the all of the incoming load at the given time step is assigned to some cluster. Further, Equation 4.6c ensures that no server is loaded by more than the threshold μ_{max}. We pick a typical value of $\mu_{max} = 0.75$ that implies that no server is loaded to more than 75% of its capacity.

Besides the above constraints that always apply, we also study tradeoffs between energy savings, performance and bandwidth costs by adding one or both of the constraints below.

$$\frac{\sum_{1\leq i\leq M}\sum_{1\leq j\leq N}\mu_{ij}c_j d_{ij}}{\sum_{1\leq i\leq M}\lambda_i} \leq D, \tag{4.7a}$$

$$\sum_{1\leq i\leq M} B_i \frac{\mu_{ij}c_j}{\lambda_i} \leq BW_{max}(j), \quad \forall j \tag{4.7b}$$

Equation (4.7a) states that the average distance between the users and the cluster to which they are assigned (weighted by load) is no more than some specified value D, where d_{ij} is the geographical distance between client location i and cluster j. For smaller values of D, this equation constrains the global load balancer to assign users to server clusters that are proximal to them, so as to ensure good performance. By making D larger, we are loosening the performance requirement by allowing the users to be assigned to clusters that are farther away. We are particularly interested in how the performance requirement D impacts energy savings. Note that as was assumed in earlier work [42], we use geographical proximity as a rough proxy for "network proximity" that governs user performance. Our formulation could equally well accommodate network latency instead of geographical distance without significant changes, though our empirical CDN traces do not provide the required network information for such an evaluation.

A CDN pays for the bandwidth that their deployed servers utilize. Typically, CDNs use 95/5 contracts for paying for their bandwidth use which works as follows [4]. For each cluster j, the traffic from the CDN's servers in the cluster is averaged over 5 minute intervals. Then the 95^{th} percentile of those 5-minute averages over the billing interval is computed. The cost of bandwidth for that cluster is proportional to the computed 95^{th} percentile. Since 95^{th} percentile cannot be modeled and constrained within a convex programming framework, we use the maximum value instead as a proxy. Equation (4.7b) above is used to constrain the maximum bandwidth sent out of cluster j to be no more than $BW_{max}(j)$, hence also constraining the bandwidth cost that is incurred by the CDN in cluster j. Choosing higher values for $BW_{max}(j)$ is tantamount to increasing the allowable bandwidth cost at cluster j. We use the

bandwidth constraint to study the impact of varying the bandwidth costs on energy savings.

Converting the convex program to a mixed integer program (MIP). Note that as currently stated the objective of the optimization function in Equation 4.6 contains the term $P_j^{\mathbf{COOL}}$ that is quadratic in variables μ_{ij}. However, since MIPs are more tractable than convex programs, we used a linear piecewise approximation of $P_j^{\mathbf{COOL}}$ to rewrite the optimization with only linear constraints. The domain for the function $P_j^{\mathbf{COOL}}(u)$ was split into equal sized segments. For each such segment $[x_i, x_{i+1}]$ we sampled the value of the function at its endpoints $[y_i, y_{i+1}]$. Computing the slope m_i and intercept k_i, the linear approximation between the points (x_i, y_i) and (x_{i+1}, y_{i+1}) takes the form $P_{j,(x_i,x_{i+1})}^{\mathbf{COOL}}(u) = P_{j,peak}^{\mathbf{COOL}} \times (m_i \cdot u + k_i)$. For each such segment we added a constraint

$$P_j^{\mathbf{COOL}} \geq P_{j,(x_i,x_{i+1})}^{\mathbf{COOL}}$$

with cluster j running at a chiller utilization of $u = \frac{P_j^{\mathbf{IT}}}{P_{peak}^{\mathbf{IT}}}$. $P_j^{\mathbf{COOL}}$ is present in the objective and lower bounded by the piecewise linear approximation. The absence of any other constraint on the variable ensures that it equals its lower bound in the optimal solution. Our implementation used 5 linear segments for an approximation error of 0.25% at $85°F$.

4.4 Combining Cluster and Server Shutdown

Server shutdown is a complementary technique to cluster shutdown and turns off individual idle servers within each cluster to save energy [55, 29]. We now devise a combined approach of using server shutdown in conjunction with our cluster shutdown algorithm to potentially provide even more energy savings. Our combined approach first explores the possibility of shutting down entire clusters, thereby saving both

the IT and cooling energy consumed by those clusters. Note that a cluster shutdown algorithm must maintain a distributed set of clusters in an active state at all times for reasons of user-perceived performance. For instance, if all clusters in a geographical region are shut down, GLB will be forced to assign users from that region to distant clusters resulting in larger latencies and degraded performance. Server shutdown can provide additional energy savings within clusters that are kept active by the cluster shutdown algorithm. In particular, not all of the servers in an active cluster may be required to serve its assigned load and a subset of these servers can be turned off to save more energy.

To capture the additional benefit of server shutdown, we enhance the cluster shutdown algorithm of Section 5.2 by incorporating server shutdown algorithms within the LLB of individual server clusters. We propose a hierarchical strategy that consists of the following two steps.

1. GLB decides which clusters should remain active and which need to be turned off using the algorithm described in Section 5.2. GLB then reroutes global traffic away from clusters that can be turned off and reassigns that traffic to clusters that remain active.

2. The server shutdown algorithm is run independently and in parallel by the LLB in each active cluster at each time step. For each active cluster, the LLB of that cluster consolidates the load assigned to that cluster into the fewest number of servers possible and turns off the remaining servers. Specifically, for a cluster of c servers, a target load threshold μ_{max} and load λ, LLB computes the optimal number of live servers $c_t = \left\lceil \frac{\lambda}{\mu_{max}} \right\rceil$ that is required to serve the load. The algorithm keeps $c - c_t$ servers inactive while keeping c_t servers active to serve the load λ.

In step (2) of our above algorithm, we make the simplifying assumption that servers can be shutdown in one time step, providing a baseline for the savings possible. A more complex server shutdown algorithm that takes into account the delay for transitioning servers between active and inactive states is provided in [29].

4.5 Evaluation

To evaluate the benefits of integrating cluster shutdown in a CDN's global load balancer we used extensive traces from Akamai, perhaps the largest commercial CDN, and ran the algorithms presented in Section 5.2. In our experiments, unless otherwise indicated, we model chillers with $\beta = 1$, i.e., the same as CEC's chiller model, and we assume that the outside air temperature is $85°F$. Later, we vary these parameters and show how energy savings vary with different parameter values.

4.5.1 Empirical Data from the Akamai Network

We used extensive load traces collected over 25 days from a large set of Akamai clusters deployed in data centers in the US. The 22 clusters captured in our traces are distributed widely within the US and had 15439 servers in total, i.e., a representative sampling of Akamai's US deployments. Our load traces account for a peak traffic of 800K requests/second and an aggregate of 950 million requests delivered to clients. The traces consist of a snapshot of total load served by each cluster collected every 5-minute interval from Dec 19th 2008 to January 12th 2009, a time period that includes the busy holiday shopping season for e-commerce traffic (Figure 5.1). In the figure, one may note load variations due to day, night, weekday, weekend, and holidays (such as low load on day no. 8, which was Christmas) Since the clusters are restricted to the US, we also restricted the trace to clients from North America. The trace consists of samples taken every 5 minutes indicating the current load on each cluster, along with a breakup of traffic from each client location. Specifically, for every 5 minutes,

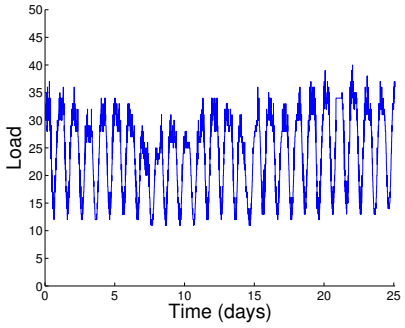
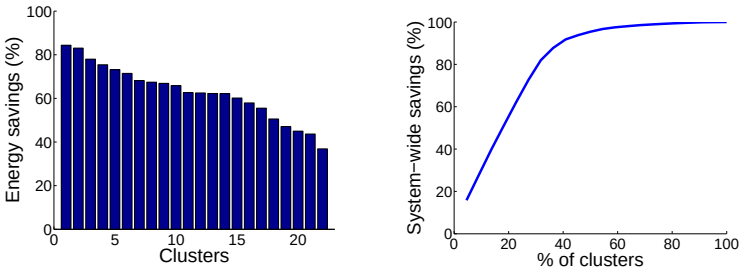


Figure 4.2: Average load per server measured every 5 minutes across 22 Akamai clusters in the US over 25 days.

we measured the load induced by client location i on cluster j and the corresponding bytes served by cluster j to users in client location i, for all relevant pairs of i and j. In addition, we also measured the number of servers present and total capacity of each cluster. In the course of our optimization, we assume that the load from a client can be shifted to any cluster as long as the capacity constraints are met and no server is overloaded. Our traces also capture the geographic location (city, state, and country) of both the client location and cluster, which lets us estimate the geographical distance between the users at a particular client and location the cluster from which they are served. The geographical distance computed in this fashion is used as a proxy for performance. The byte information captured in our traces is used to compute the bandwidth usage of the CDN in each cluster that in turn determine the bandwidth costs incurred by the CDN that we study in our work.

4.5.2 Overall energy savings

We emulated the GLB-based cluster shutdown algorithm in Section 5.2 on the CDN traces described above. The algorithm minimizes the energy consumption of the CDN in each time step by orchestrating which clusters should be on and which clusters should be turned off. Then the total energy consumed by the CDN is computed by adding the energy consumed at each time step across the entire trace. As a basis



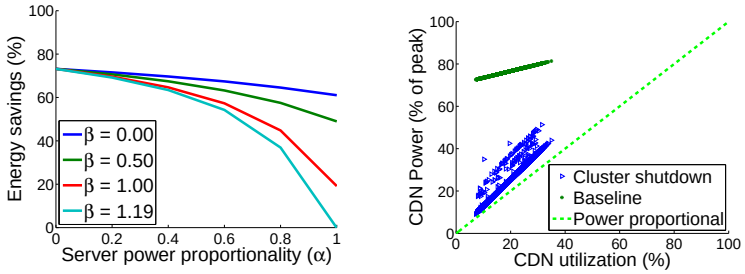
(a) Individual clusters save between 37% to 84%. The system-wide energy savings is 67%.

(b) Applying cluster shutdown to the top 45% clusters is sufficient to obtain 94% of the system-wide energy savings.

Figure 4.3: CDN energy savings obtainable by cluster shutdown.

for comparison, we used as a baseline the energy consumed by the user-to-cluster assignment in the trace with no cluster shutdown, i.e., all clusters are assumed to be on throughout the trace which is consistent with how CDNs operate today.

The system-wide energy savings that is possible with cluster shutdown incorporated into the CDN's GLB is 67% in comparison with the baseline where all clusters are always turned on. In performing this analysis, we make typical assumptions about the energy efficiency of the data centers (PUE = 2), servers ($\alpha = 0.31$) and chillers ($\beta = 1$). We also do not constrain performance and bandwidth costs. Therefore, these are the best case savings possible. However, we vary each of these assumptions in subsequent sections to examine how these savings change under different scenarios. To further breakdown the savings, in Figure 4.3a we show savings obtained by individual server clusters. Savings vary between 37% to 84% with the median cluster saving 63%. Further, most of the savings can be obtained by performing cluster shutdown in a few key clusters. As shown in Figure 4.3b, applying cluster shutdown to top 45% of the clusters is sufficient to obtain 94% of the optimal energy savings.



(a) Energy savings decrease as servers and cooling equipment become more energy efficient.

(b) Cluster shutdown makes the CDN power proportional by aligning power values close to the ideal 45-degree line.

Figure 4.4: Energy savings and power proportionality

4.5.3 Impact of server and cooling efficiency

CDNs operate with a wide range of server hardware and are deployed in a wide range of data center facilities. Further, both server and cooling efficiencies are constantly being improved over time. To capture these effects, we varied the power proportionality factor of the servers (α) as well as the cooling efficiency of the chillers (β) to study how energy savings vary with these parameters (cf., Figure 4.4a). When both the servers and cooling are energy-inefficient ($\alpha = \beta = 0$), the cluster shutdown technique provides the most energy savings of 73%.

As servers become more energy-efficient the idle power usage gets lower, and thus lowers cooling energy. This results in energy savings from cluster shutdown dropping to 61% when servers are perfectly power proportional ($\alpha = 1$). In fact for any chiller efficiency β, energy savings decrease as servers become more efficient.

Likewise, for any given server efficiency α, increasing cooling efficiency β reduces the energy savings. For perfectly power proportional servers ($\alpha = 1$) energy savings fall as β increases, dropping from 61% when $\beta = 0$ to 19% for $\beta = 1$. In the ideal world with highly-efficient servers and cooling, e.g., $\alpha = 1$ and $\beta > 1$, the energy savings from cluster shutdown approaches zero, i.e., if the "hardware" is itself highly-efficient there is no need for an explicit shutdown mechanism to reduce energy.

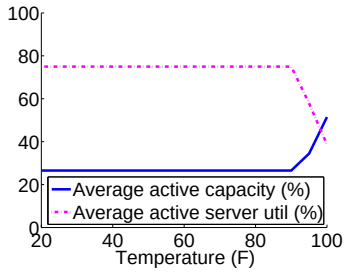
4.5.4 CDN Power Proportionality

To visualize how server shutdown makes a CDN more power proportional, it is instructive to view the instantaneous power consumption of the entire CDN as a function of its overall utilization. Specifically, in Figure 4.4b, we plot the CDN's total power consumption (as a percentage of its peak) and its overall utilization at each time step as a single point of a scatter plot. Note that these plots are the exact analogue of server proportionality described in Equation 4.1 that relates power to utilization, but computed for the CDN as a whole. A perfectly power proportional system would have all its points aligned along the 45-degree line shown in the figure. The scatter plot of the total CDN power without cluster shutdown deviates from the ideal 45-degree line significantly as the CDN consumes a lot of power even during periods of low utilization during the non-peak hours. However, cluster shutdown makes the scatter plot of the total CDN power much more closely aligned to the ideal 45-degree line, i.e., cluster shutdown makes the CDN significantly more power proportional.

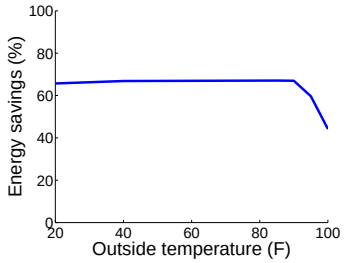
4.5.5 Impact of Outside Air Temperature

The cooling equipment transfers heat from inside the server room to the external atmosphere. Physical laws suggest that the heat transfer rate through convection is larger when the temperature differential between the inside and outside air temperatures are greater. Thus, it takes less energy to cool when the outside temperature is cooler (say, in the winter) than when the outside temperature is hotter (say, in the summer). Further, as we saw in Figure 4.1a, the required power for cooling rises more sharply in a quadratic fashion with increasing utilization when the outside air temperature is hotter.

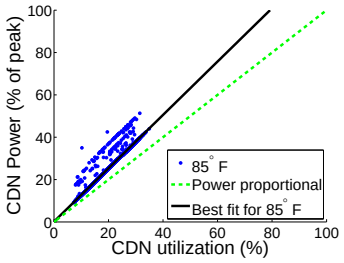
The interplay of outside air temperature with cooling power impacts what energy savings are achievable by GLB via cluster shutdown. Specifically, as outside air



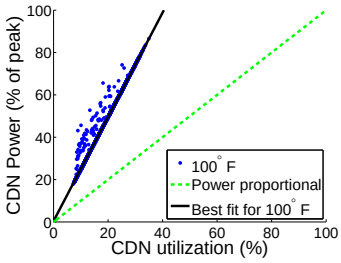
(a) Avg. active server utilization falls as temperature rises



(b) Energy savings drop from 67% at $85°F$ to 44% at $100°F$



(c) At $85°F$, the CDN with cluster shutdown is roughly power proportional.



(d) At $100°F$, cluster shutdown is less effective.

Figure 4.5: Cluster shutdown is more effective in saving energy at lower temperatures than higher ones.

temperature increases, the cluster (and server) utilization have to be kept low since there is a greater cooling power penalty associated with higher utilization. Thus, as shown in Figure 4.5a, at low temperatures the algorithm runs all active servers at the maximum allowed utilization of $\mu_{max} = 75\%$. At high temperatures cooling costs rise rapidly with utilization, and the optimal solution at $100°F$ corresponds to active servers running at 39% utilization. Note that to continue to serve the same incoming load, a lower cluster (or, server) utilization means more clusters (and, servers) need to remain active. Thus, the fraction of total CDN capacity that is kept active, rises

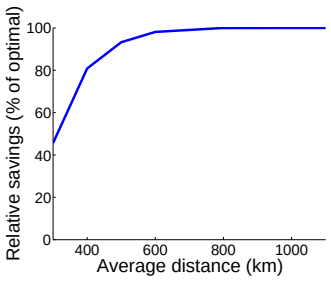


Figure 4.6: Relaxing performance results in greater energy savings. 46%, 93% and 99.9% of the optimal energy savings are obtained at D values of 300 km, 500 km and 795 km respectively

from 27% at low temperatures, to 51% of total capacity at $100°F$. The increase in active capacity with rising temperatures combined with lower utilization of active servers has a negative impact on savings. Figure 4.5b shows that energy savings drop from 67% at $85°F$ to 44% at $100°F$. The energy savings achieved by cluster shutdown at different outside air temperatures can also be viewed as a scatter plot of the total CDN power versus its utilization. The scatter plots in Figures 4.5c and 4.5d correspond to $85°F$ and $100°F$ respectively. At $85°F$ the best linear fit to the power-utilization curve has a slope of 1.26, closer to the ideal 45-degree line with a slope of 1, i.e., the CDN with cluster shutdown is roughly power proportional. At $100°F$ the slope almost doubles to 2.46.

4.5.6 Tradeoff between Energy and Performance

CDNs host a wide range of applications. Some applications such as dynamic web sites are highly sensitive to network latency, with even small increases in latency causing significant degradation in the performance experienced by the user. Other applications such as software downloads are weakly sensitive to latency and can even be performed in the background. As in [42], we use geographical distance as a rough proxy for the network latency between a user and the cluster assigned to that user by GLB. To study the tradeoff between performance requirement and energy savings we add Equation (4.7a) as a constraint where different latency requirements can be

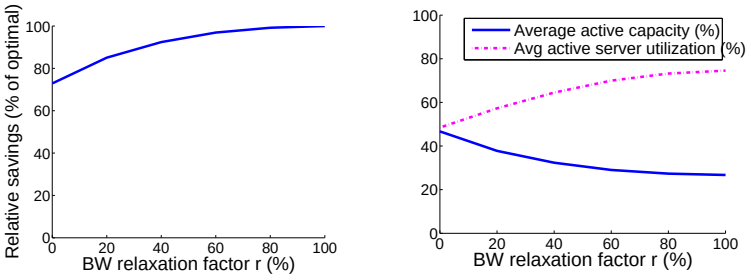
modeled by varying the distance bound D. Specifically, larger values of D allow a larger load-weighted average distance between the users and their assigned clusters. Allowing larger user-cluster distances (and latencies) has the effect of degrading performance, but allows for potentially more cluster-shutdown opportunities for GLB and greater power savings. Figure 4.6 illustrates this tradeoff where setting $D = 300$ km provides 46% of optimal savings. Note that this distance bound is roughly the distance between Boston and New York with network latencies often in the 10-15 ms range that is adequate for even applications with higher latency sensitive. When $D = 500$, one can achieve 93% of the energy savings. This distance bound is roughly the distance between Boston and Philadelphia where typical latencies are in the 20 ms range, suitable for most moderately latency-sensitive applications. Finally, when $D = 795$ km, a suitable limit for weakly latency-sensitive applications such as background downloads, we achieve 99.9% of optimal savings.

4.5.7 Tradeoff between Energy and Bandwidth Costs

The operating expenditure (OPEX) of a CDN includes two major components: the energy costs for powering the servers and the bandwidth cost for the traffic from the server clusters to the users. Reducing energy usage by packing traffic into fewer server clusters could cause increased bandwidth usage in those clusters, which in turn could drive up the bandwidth cost at those clusters. The primary question is whether energy savings can be achieved without significant increase in the bandwidth cost. Note that if energy savings are only obtainable by significantly increasing the bandwidth cost, that would serve as a disincentive for a CDN to implement cluster shutdown.

As noted in Section 5.2, the bandwidth cost incurred by the CDN at each cluster can be approximated by the maximum over all 5-minute time slots in the billing

period[6] of the average traffic (in Mbps) transmitted in that time slot. We constrain (through Equation (4.7b)) the maximum bandwidth for each cluster j to be at most $(1+r)BW_{max}(j)$, where $BW_{max}(j)$ is the maximum bandwidth value observed in the trace and r is the BW relaxation factor that determines how much extra bandwidth costs we are willing to allow. Figure 4.7a shows energy savings relative to optimal as the bandwidth constraints are relaxed by varying r. With no increase in bandwidth cost ($r = 0$), cluster shutdown can still achieve 73% of optimal savings. 47% of the total CDN server capacity remains turned on, with active servers running at an average utilization of 48%. Relaxing bandwidth constraints allows active server utilization to rise to $\mu_{max} = 75\%$ at $r = 100\%$. This allows the CDN to run with 27% of its server capacity turned on and achieve optimal energy savings. Overall, our results indicate that cluster shutdown can still achieve significant energy savings with little or no increase in bandwidth costs.

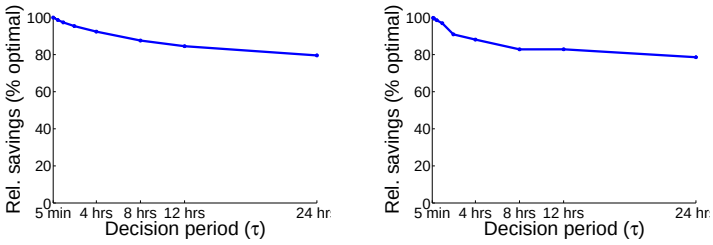


(a) We get 73% of optimal energy savings with no increase in bandwidth cost

(b) Average active server utilization increases from 49% to $\mu_{max} = 75\%$ as BW cost doubles (r=100%)

Figure 4.7: Energy savings versus Bandwidth cost

[6]In our simulations, we assume that the billing period is length of the trace which is 25 days, though in reality a billing period is typically one month.



(a) Switching clusters once a day still achieves 80% of optimal savings

(b) With load prediction we achieve 79% of optimal savings switching clusters once a day

Figure 4.8: Impact of decision period and traffic prediction

4.5.8 Impact of Limiting the Cluster Transitions

Frequently switching server clusters on and off can impact the overall lifetime and reliability of the equipment. Further, the mechanical nature of cooling equipment limits the rate at which it can be switched on and off. Chillers, for example, require a warm up at partial load before they can be incrementally ramped up to full capacity. Thus it is neither desirable nor feasible to frequently turn entire clusters on and off, and we study the amount of energy savings that can be extracted when limiting the frequency of cluster shutdowns.

Suppose that cluster transitions are allowed to occur only once every τ time slots, where τ is defined as the *decision period* and is required to be an integral multiple of δ. In our experiments we vary τ from 5 minutes to 1 day. In Figure 4.8a the left-most point in the graph corresponds to $\tau = 5$ minutes which is the smallest time granularity at which the trace data is collected. It is nearly infeasible to turn clusters on or off every 5 minutes. However, the $\tau = 5$ minutes measurement provides the theoretical optimal of how much energy savings is possible in the best case that can serve as a benchmark for comparing other values of τ. Increasing τ could decrease energy savings as GLB has a lesser ability to turn clusters on or off in response to load

variations. However, as we see in Figure 4.8a, even with $\tau = 1$ day where clusters are transitions just once a day, we achieve 80% of the optimal savings possible. Thus, we establish that frequent cluster transitions are not necessary for obtaining most of the benefits of cluster shutdown.

4.5.9 Impact of inaccurate real-time load information

Thus far, we have assumed that the load for the current decision period τ is accurately available and can be used for decision making for that period. This is a reasonable assumption for smaller decision periods (say $\tau \leq 30$ minutes) but not so much when the decisions are more infrequent and decision periods are longer. Therefore we consider the situation where our algorithm does not know the current load but would have to predict it for the purpose of deciding which clusters are transitioned. When cluster transitions are made based on a prediction of load over any extent of time there always exists the chance of insufficient active capacity and users being denied service. We allow active CDN clusters to run to 100% utilization before they drop incoming workload. We define *availability* as the ratio of workload served to total workload. Under these assumptions, we define a simple algorithm that predicts the load and computes the optimal cluster allocation under this prediction. The predicted load equals the load at the previous decision period, for small decision periods ($\tau \leq 1$ hour), or the load at the same decision period from the previous day, for larger periods ($\tau > 1$ hour). Using this simple prediction algorithm, Figure 4.8b shows energy savings for decision period 5 minutes $\leq \tau \leq 1$ day. Energy savings dropped from 100% to 79% of optimal over this range. In each case, the algorithm provided at least "three nines" of availability (i.e. 99.9%).

4.5.10 Finding a sweet-spot

So far we looked at the impact of individual parameters on the energy savings obtained through cluster shutdown. In a realistic situation, we would expect CDNs

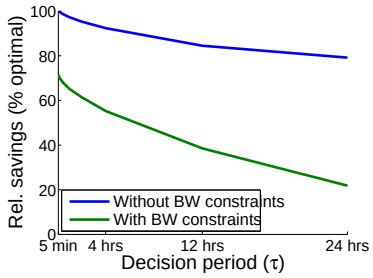


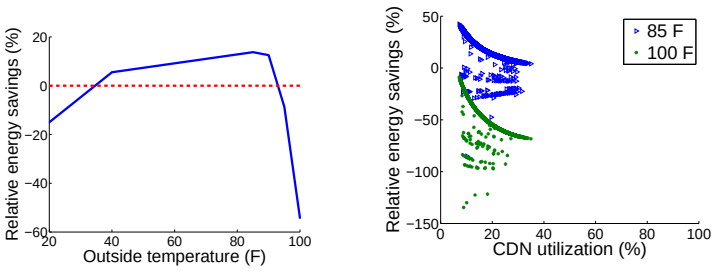
Figure 4.9: We can achieve 22% of the optimal savings even with switching each cluster no more than once a day, allowing no increase in bandwidth costs, and limiting the average distance from the user to the cluster to be no more than 800 km.

to operate under multiple constraints. In this section we look at the combined impact of cluster transitions, performance and bandwidth constraints on energy savings. Figure 4.9 shows energy savings as a function of the decision period when the average user-cluster distance is upper bounded at $D = 800$ km. With no increase in bandwidth costs (corresponding to $r = 0$), for a decision period (τ) of 5 minutes, and a performance constraint of 800 km we obtain 71% of optimal savings. This compares favorably with the 73% savings without the performance constraints (Section 4.5.7). Savings fall to 22% of optimal as the decision period (τ) increases to 1 day.

4.5.11 Cluster vs Server shutdown

We look at the relative energy savings of two complementary techniques: GLB that incorporates cluster shutdown and an LLB that incorporates server shutdown. We assume that, given a cluster with c servers getting incoming load λ, LLB always keeps the exact number of servers $\lceil \lambda/\mu_{max} \rceil$ required to serve the incoming load for that cluster and at every time step. This is of course an optimistic assumption but it helps understand the best possible savings achievable using LLB. However, unlike GLB, LLB is unable to move traffic across clusters to shutdown entire clusters. Figure 4.10 plots the difference between the energy savings of implementing cluster shutdown

in GLB and the corresponding savings from implementing server shutdown in LLB. In Figure 4.10a, we see that at low outside air temperatures when cooling is relatively inexpensive (cf., Fig 4.1a), LLB with server shutdown performs better due to its greater impact on server energy. At high temperatures GLB with cluster shutdown runs active clusters at lower utilization to reduce cooling energy. The limited ability of GLB to shutdown clusters at higher temperatures implies that it performs worse than LLB. Thus, GLB outperforms LLB at moderate temperatures outside of these two extremes. The relative performance of GLB versus LLB also depends on the CDN utilization. Figure 4.10b shows that when the CDN is lightly loaded, GLB has greater flexibility to move traffic around and switch off clusters. There are fewer such opportunities at higher system utilization, where larger clusters need to be kept active for serving the incoming CDN load. At $85°F$, GLB out performs LLB in all cases. But the additional energy savings drop from 42% to 4% as CDN utilization increases from 7% to 35%. This trend is exaggerated when the temperature increases to $100°F$. In this case, LLB is better than GLB but the additional savings provided by LLB increases from 9% to 68% over the same range of utilization.



(a) GLB is better within a broad temperature range

(b) GLB is better at lower utilization and outside temperatures

Figure 4.10: GLB (cluster shutdown) vs LLB (server shutdown)

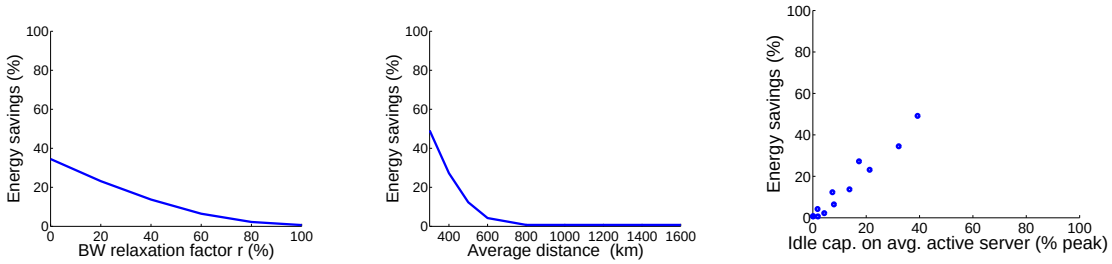
4.5.12 Integrating Server shutdown with Cluster shutdown

We evaluate the hierarchical strategy described earlier in Section 4.4 that incorporates energy-awareness at both the local and global load balancer by implementing cluster shutdown and server shutdown. A pure cluster shutdown strategy is taken as the baseline, and we study the incremental benefit of adding server shutdown.

We saw earlier in Section 4.5.7 that with no increase in bandwidth costs ($r = 0$), a pure cluster shutdown strategy kept more clusters active with servers running below the allowable peak utilization ($\mu_{max} = 75\%$). Relaxing bandwidth constraints allowed servers to run at higher utilizations and thus keeping a smaller fraction of its clusters active. In fact, the CDN approached power proportionality for $r = 100\%$. To study the impact of adding server shutdown, we plot the incremental gains obtained in Figure 4.11a. With no increase in bandwidth cost (r = 0), the combined strategy saves 34% over pure cluster shutdown. Relaxing bandwidth constraints causes savings to drop to a negligible 0.72% at twice the bandwidth cost (r = 100%).

Figure 4.11b shows incremental gains obtained as a function of performance. If low latency is required, the energy savings over a pure cluster shutdown strategy is 46%, with an average user-cluster distance of 300 km. These gains taper off as performance constraints are relaxed and cluster shutdown approaches power proportionality.

Tight constraints limit the performance of the pure cluster shutdown strategy by requiring the CDN to keep more clusters active and run at higher idle capacity. Server shutdown targets this idle capacity to obtain additional gains. We quantify this in Figure 4.11c by plotting savings against average idle capacity of an active server (as a percentage of peak utilization μ_{max}). The roughly power proportional nature of the CDN after adding server shutdown implies that any idle capacity previously present is converted directly into savings. This explains the approximate linear nature of the graph.



(a) Additional energy savings over pure cluster shutdown falls off as bandwidth constraints are relaxed. 34% savings are achieved without any increase in BW costs

(b) 46% additional energy savings over pure cluster shutdown can be achieved at an average distance of $D = 300$ km.

(c) Energy savings obtained by adding server shutdown are roughly linear to the idle capacity of an active server under pure cluster shutdown

Figure 4.11: Integrating server shutdown with cluster shutdown

4.6 Related Work

Data center energy management has emerged as an active area of research in recent years. Several approaches have emerged for reducing the energy consumption of data centers, including server shutdown during off-peak periods [12, 50, 21, 13], the use of low-power server nodes [7], OS-level energy management through methods such as DVFS, the use of renewable energy [46, 18], and routing requests to locations with the cheapest or greener energy [42]. Separately, there has also been work on designing cooling-aware or thermal-aware algorithms for data centers. Cooling-aware workload management techniques have been studied in [24]. Thermal-aware workload placement techniques that place load on cool portions of the data center have been studied in [33, 51]. Models for air- or chiller-based cooling data centers have been studied in [24, 39]; the cooling models used here are inspired by this work and also the data published by the California Energy Commission [11].

A key difference between the prior work and our work is our focus on content delivery networks; the design choices made by a CDN require these ideas to be cus-

tomized to the CDN case, for instance by integrating energy management with the CDN's load balancing algorithms. Another key CDN-specific issue is to design energy saving methods that minimize the impact on user performance and bandwidth costs. Specifically we use realistic power and cooling models for clusters, based on prior work, and use them to design cluster shutdown algorithms that can be implemented in the CDN's global load balancing algorithms. In this sense the approach also differs from, and is complementary to, prior work on server shutdown technique for CDN energy management [29].

4.7 Conclusions

We focused on the design of energy-efficient CDNs. Since a CDN could comprise thousands of server clusters across the globe consuming a significant amount of energy, we propose a new technique called cluster shutdown to turn off entire clusters to save energy. Our experimental results using extensive traces from a commercial CDN shows that cluster shutdown can reduce system-wide energy usage by 67% in the optimal case, and most of these savings can be achieved without sacrificing end-user performance and bandwidth costs. In addition, the technique works well even when shutdown is limited to once per day for each cluster and when the load is not known in real-time and must be predicted. We believe that cluster shutdown is a strong candidate for implementation in an actual CDN, especially since it fits in more easily with current CDN architectural principles in comparison with server shutdown techniques studied in the past.

CHAPTER 5

REDUCING ENERGY COSTS USING DEMAND RESPONSE

A recent trend is the emergence of the smart electric grid that supports many technologies and features to encourage greater adoption of energy-efficiency techniques. These include the availability of novel electricity pricing models to encourage greater energy efficiency, the deployment of smart meters for fine-grain metering and billing needed by such pricing models, and automated demand-response where the grid provides explicit signals to consumers to reduce their usage during peak periods of supply-demand imbalances. While demand-response involves explicit requests to users to reduce usage, we note that variable pricing schemes provide an *implicit* form of demand-response by discouraging users from using "too much" electricity when the electricity prices are high.

In this chapter, we study how Internet-scale distributed systems can exploit smart grid features such as demand response to reduce their energy costs. There are two possible methods for reducing energy usage in an IDS in response to explicit or implicit demand-response signals. Both methods involve reducing the load at the data center that receives such a signal and then shutting down a subset of the servers to reduce the total energy usage. One possible approach to reducing energy use is to move a portion of the load to other nearby data centers and then shutting down a portion of the servers; this is achieved by having the IDS redirect some of the incoming requests to other nearby data centers and ensure that data is already replicated to service these requests from alternate sites. This approach was studied in [42] where this mechanism was employed to reduce electricity bills by redirecting load from data centers with

higher electricity prices to others with lower prices. This approach, and related ones, implicitly assume that the incoming requests need to be serviced immediately (i.e., in "real-time"). We study an alternate approach that moves load in the temporal dimension (rather than spatially or geographically, as has been done in prior work [42]) in order to reduce energy costs. Our approach assumes that *not all of the incoming requests need to be serviced immediately.* While requests to interactive services such as web requests do need immediate service, there are other classes of requests that are elastic and can be delayed if necessary. Examples of such elastic requests include background downloads of software updates by operating systems, distribution of OS-level or security patches and content prefetching for local caching.[1] In addition to elastic content requests, Internet-scale distributed systems also see elastic requests for computation—such as batch jobs like transcoding of videos [5], analytics processing, nightly backups, or book-keeping operations such as accounting and billing. Thus we assume that an IDS sees two types of requests: interactive requests that require immediate service and elastic requests that can be delayed if necessary. We study how such a system can respond to demand-response signals from the smart grid by delaying elastic requests and shutting down some of the servers, thereby temporarily reducing energy usage (and thus, energy costs).

We make the following contributions:

- In the offline context where the full load sequence is known ahead of time, we derive provably optimal algorithms for demand-response that delay load to minimize the overall cost.

- We evaluate our algorithm on a large CDN workload using an extensive set of pricing contracts that include time-of-use energy pricing and peak demand pricing. We

[1] All major OS platforms—Mac, Windows and Linux—as well as many phone-based OSes routinely download software updates in the background.

achieve savings of 12% even when only 40% of the load is elastic and off-peak usage is charged at half the rate of on-peak usage. We also demonstrate that almost all the energy savings can be attained with no increase in the bandwidth costs.

- For a peak demand pricing contract the algorithm does significantly better, achieving 32% savings under similar constraints.

- For hybrid contracts where both energy usage and demand charges are included in the energy costs, we show that 23% savings are possible for the case when energy and demand contribute almost equally to the total cost.

- We find that upper-bounding the service delay by 6 hours is sufficient to achieve the maximum possible savings for 40% elastic load under all the contracts evaluated with our workload.

The rest of this chapter is structured as follows. Section 5.1 presents some background and the models assumed for the workload, power consumption and electricity pricing. Our algorithm for optimizing energy costs via demand-response is presented in Section 5.2. Results from our experimental evaluation are presented in Section 5.3. We present related work in Section 5.4 and conclude in Section 5.5.

5.1 Background

Internet-scale Distributed Systems: Our work assumes an Internet-scale Distributed System (IDS) that provide service delivery or content delivery to its users. Content distribution networks (CDNs) are an example of an Internet-scale distributed system, and so are distributed cloud-based service delivery networks. A large IDS employs tens of thousands of servers that are spread across a large number of data centers; each data center houses a cluster of servers and the size of each cluster can vary from hundreds to many thousands of servers [35]. Incoming requests for service are assumed to be forwarded to an appropriate cluster by the IDS, and the request

is then serviced by one of the servers within that cluster. Our work assumes that a request can be one of two types: interactive requests that require immediate service and elastic requests that can be delayed if needed by the system. In this work, we assume that each request, whether interactive or elastic, is always serviced by the cluster to which it is sent by the IDS. That is, we do not consider the ability of the IDS to redirect some of the load to other nearby clusters, and only look at temporal load optimizations for elastic requests. While it is possible to combine techniques for moving load across clusters with those that move load across time, we leave the design of such hybrid techniques to future work.

We are interested in quantifying the potential energy savings that can result by delaying elastic requests when performing smart-grid demand response. Demand response (DR) is a technique by which a customer temporarily reduces electricity usage in response to a signal from the grid; in our context, demand response refers to any technique that the IDS can employ to reduce or defer its energy usage in response to signals from the grid. We assume that the smart grid exposes variable electricity prices to each customer; the exact pricing models considered in this study are detailed later in this section. Since price of electricity is no longer flat, the varying prices serve as implicit signals for demand-response When the electricity price is high or when higher electricity usage will result in higher costs, the consumer (which, in our case, is the IDS) is incentivized to temporarily curtail usage or shift usage to lower-price periods, and thereby reduce costs. We study an optimization approach for performing such demand-response in an IDS. Our work focuses only on implicit demand-response (that responds to pricing signals) and we do not consider explicit demand response here. Temporary deferral of elastic requests in response to an explicit DR signal from the grid is an easier problem and it is straightforward to incorporate such DR signals into our current work.

Workload Model: The workload of an IDS is generated by users and applications around the world. The global load balancer of the IDS partitions the load and directs a part of the load to each cluster of the IDS. Since our energy cost optimizations do not move load across clusters, we model and optimize the load arriving at each cluster independently. For each cluster, we model the load arriving at that cluster by an *arrival sequence* $\lambda = \langle \lambda_0, \lambda_1, \ldots, \lambda_{T-1} \rangle$, where λ_t is the load that *arrives* at the cluster at time step t. We assume that a fraction κ of the incoming load is elastic and that the elastic load can be served in a delayed fashion. Specifically, we assume that the maximum allowed service delay for elastic load is τ. As a result of our optimizations, the loads are processed by the servers in the cluster at times that are potentially different from when they arrived. The output of our optimization is a *service sequence* that we represent by $\hat{\lambda} = \langle \hat{\lambda}_0, \hat{\lambda}_1, \ldots, \hat{\lambda}_{T-1} \rangle$, where $\hat{\lambda}_t$ represents the load that will be *served* by the cluster at time t.

Power consumption model for clusters: A power consumption model is used to derive the instantaneous power drawn by the cluster, given its service load sequence $\hat{\lambda}$. Our cluster power model is based on our earlier work in [31]. We assume that the cluster is fully power proportional and consumes power that equals $u \cdot P_{peak}$, where the u is the utilization of the cluster defined as the ratio of the load served by the cluster and its peak capacity. P_{peak} is the maximum power that can be drawn by the cluster that equals the product of the number of servers in the cluster and the peak power draw of each server. Based on a typical deployed server used by IDNs, we assume that each server can draw 97W of power at peak. Note that we assume that the cluster is power proportional since a number of techniques such as server shutdown [29] are known to make clusters close to power proportional. We also model the power required for cooling the cluster as below.

$$P^{\mathbf{COOL}} = P^{\mathbf{COOL}}_{peak} \times \left(A + B \cdot u' + C \cdot u'^2\right)$$

where u' is the utilization of the chiller and the constants A, B, and C can be derived from the regression curves provided by the California Energy Commission [11]. We refer to our earlier work [31] for more details on our cooling model.

Electricity Pricing Models: The cost of electricity is often computed on the basis of the four generic metrics described below. These metrics are themselves computed from "instantaneous" measurements of electricity consumption made throughout the billing period that is typically a month. Each metric below is either a demand metric that is based on peak KW measurements or an energy usage metric that is based on the energy consumed in KWHs. Further, some parts of the day are denoted as peak, when energy consumption is usually high, and other parts of the day are denoted as off-peak, when the energy consumption is usually low. We first derive the integrated thirty-minute values by partitioning the billing period into 30-minute intervals and computing both the average demand (KW) and the energy KWHs) in each 30-minute interval. We then compute the four metrics below.

1. On-peak demand (D_{on}): The maximum integrated thirty-minute demand (in KWs) during on-peak periods.

2. Off-peak demand (D_{off}): The maximum integrated thirty-minute demand (in KWs) during off-peak periods.

3. On-peak energy usage (E_{on}): Energy consumed (in KWHs) during on-peak periods.

4. Off-peak energy usage (E_{off}): Energy consumed (in KWHs) during off-peak periods.

We consider three commonly used pricing models in our work. Let the cost of electricity to serve a load sequence λ under a particular pricing model π be denoted by $cost_\pi(\lambda)$. We compute $cost_\pi(\lambda)$ as follows. First we apply the cluster power model

to determine how much instantaneous power is drawn by the cluster to serve a given load sequence. We then compute the four metrics above using the instantaneous power draw and use it as follows.

1) The first model we consider is the *time-of-use (TOU) pricing model*[2] where the utility computes the electricity bill based only on energy usage and does not explicitly impose a demand price that depends on the peak consumption. If π is a tariff that uses the TOU model then

$$cost_\pi(\lambda) = \alpha_{on}E_{on} + \alpha_{off}E_{off},$$

where α_{on} the on-peak unit price (in \$/KWH) and is more expensive than the off-peak unit price α_{off}. Of particular interest is the ratio of off-peak to on-peak energy prices $\rho_E = \frac{\alpha_{off}}{\alpha_{on}}$. Small values of ρ_E imply a cheap off-peak price, while $\rho_E = 100\%$ is equivalent to flat pricing.

2) The second model we consider is the *demand pricing model* where the utility computes the electricity bill based only on the demand and does not explicitly charge for the energy consumed. If π is a tariff that uses demand pricing then

$$cost_\pi(\lambda) = \beta_{on}D_{on} + \beta_{off}D_{off},$$

where β_{on} the on-peak unit price (in \$/KW) is more expensive than the off-peak unit price β_{off} (in \$/KW). Of particular interest is the the ratio of off-peak to on-peak demand prices $\rho_D = \frac{\beta_{off}}{\beta_{on}}$. Small values of ρ_D imply a much cheaper off-peak price, while $\rho_D = 100\%$ is equivalent to flat demand pricing.

3) In the most general model which we call the *hybrid pricing model* [1, 3] all four metrics above are used to compute the energy cost. In particular, $cost_\pi(\lambda) = \alpha_{on}E_{on} + \alpha_{off}E_{off} + \beta_{on}D_{on} + \beta_{off}D_{off}$. We define the *mixing coefficient* as the ratio $\rho_M = \frac{\beta_{on}}{\alpha_{on}}$, where a value of 0 implies a pure energy usage pricing, while ∞

implies a pure demand pricing. Note that we can rewrite the incurred cost as $\beta_{on}\left(D_{on} + \rho_D D_{off} + \rho_M \left\{E_{on} + \rho_E E_{off}\right\}\right)$.

5.2 An Optimal Algorithm for Demand Response

We describe our algorithm for demand response that optimally delays load to minimize the total energy cost of an IDS. The algorithm works individually for each cluster of the IDS and does not move load across clusters. Let the incoming load at a cluster be represented by an arrival sequence $\lambda = \langle \lambda_0, \lambda_1, \ldots, \lambda_{T-1} \rangle$, where λ_t is the load that arrives at the cluster at time step t. Further, let the fraction of the incoming load that is elastic be κ and let the maximum allowed service delay for elastic load be τ.

Our algorithm works in two steps. First, our algorithm creates a modified load sequence called the service load sequence that we represent by $\hat{\lambda} = \langle \hat{\lambda}_0, \hat{\lambda}_1, \ldots, \hat{\lambda}_{T-1} \rangle$, where $\hat{\lambda}_t$ represents the load that will be *served* by the system at time t. Note that $\hat{\lambda}$ represents the load sequence obtained after the algorithm moves around the load to optimize energy costs. (For simplicity, assume that $\lambda_t = \hat{\lambda}_t = 0$, for $t < 0$ and $t \geq T$.) Next, our algorithm uses the service sequence $\hat{\lambda}$ and produces a set of specific load movements $L_{t,t'} \geq 0$ that transforms the arrival sequence λ to the service sequence $\hat{\lambda}$. Specifically, $L_{t,t'}$ is the amount of elastic load that is moved from time t to time t', for all $0 \leq t \leq T-1$ and $t \leq t' \leq t + \tau$. We describe each step in detail below.

5.2.1 Constructing the service load sequence $\hat{\lambda}$

The algorithm delays processing some of the elastic load to minimize the energy cost, while ensuring that no elastic load is delayed more than τ time steps and further the cluster's capacity bounds are met. Let f_t be the elastic load that arrived at time step t but was postponed to be processed at a later step by our algorithm. Since the amount of elastic load arriving at time t is at most $\kappa\lambda_t$, the following holds.

$$f_t \leq \kappa \cdot \lambda_t, \forall t \tag{5.1}$$

The load that is delayed at a time step is assigned by the algorithm to be processed at a later time step. Let p_t represent the total elastic load that arrived at the cluster at some time in the past but is assigned to be served at time t. We can write the load served by the cluster at time t as

$$\hat{\lambda}_t = \lambda_t + p_t - f_t, \forall t \tag{5.2}$$

For simplicity, for values of t outside of our time window we set both p_t and f_t to be zero, i.e., $p_t = f_t = 0$ for $t < 0$ and $t \geq T$. Since the algorithm can only move elastic load to a future time slot and never back to a past time slot, we require that the total load served in every prefix in the service load sequence is upper bounded by the corresponding load from the arrival load sequence. In other words,

$$\sum_{i=0}^{t} \hat{\lambda}_i \leq \sum_{i=0}^{t} \lambda_i, \forall t \tag{5.3}$$

By substituting for $\hat{\lambda}_i$ from Equation 5.2, we get

$$\sum_{i=0}^{t} f_i - \sum_{i=0}^{t} p_i \geq 0, \forall t \tag{5.4}$$

Since service delay is at most τ, we require that the load in the arrival sequence $\lambda_1, \cdots, \lambda_t$ should be served by the cluster within time $t + \tau$. In other words

$$\sum_{i=0}^{t+\tau} \hat{\lambda}_i \geq \sum_{i=0}^{t} \lambda_i, \forall t. \tag{5.5}$$

Substituting for $\hat{\lambda}_i$, we get

$$\sum_{i=0}^{t+\tau} f_i - \sum_{i=0}^{t+\tau} p_i \leq \sum_{i=t+1}^{t+\tau} \lambda_i, \forall t \tag{5.6}$$

Let cluster capacity C represent the maximum load that a cluster can serve at any given time . The cluster capacity is a function of server resources available at each cluster. Since the served load cannot exceed C at any time step, we have

$$\hat{\lambda}_t \leq C, \forall t \tag{5.7}$$

Finally, we need the following variables to be non-negative.

$$\hat{\lambda}_t, p_t, f_t \geq 0, \forall t \tag{5.8}$$

Let $cost_\pi(\hat{\lambda})$ represent the energy cost of serving load sequence $\hat{\lambda}$ using energy pricing policy π. We minimize $cost_\pi(\hat{\lambda})$ subject to the linear constraints represented in Equations 5.1, 5.2, 5.4, 5.6, 5.7, and 5.8. Since the constraints are linear and we know that the cost function $cost_\pi$ described in Section 5.1 is also linear for the tariffs π that we consider, we can solve the minimization problem as a linear program (LP).

Theorem 3. *For a given arrival load sequence λ, our linear program produces a feasible service load sequence $\hat{\lambda}$ that has the minimum energy cost.*

Proof. Our LP formulation has a feasible solution since the input arrival sequence λ satisfies the capacity constraints of Equation 5.7. Here we make the reasonable assumption that the load balancer of the IDS distributes load to each cluster such that arriving load satisfies the capacity constraint. Thus, $\hat{\lambda}_t = \lambda_t$ and $p_t = f_t = 0$, for all t, is a feasible solution for the LP. It follows that our algorithm yields a feasible service sequence with minimum cost. □

5.2.2 Constructing the load movement schedule L

The first step of our algorithm does not explicitly produce a schedule for how much elastic load moves from each time t to each time t', $t' > t$. However, such a

schedule $L_{t,t'}$ can be computed given the output service sequence $\hat{\lambda}$ and the input arrival sequence λ as follows. We create a directed graph $G = (V, E)$ with capacities assigned to each edge as follows. The vertex set $V = \{s\} \cup U \cup V \cup \{s'\}$, where s is a source node, s' is a sink node, $U = \{u_0, u_1, \cdots, u_{T-1}\}$, and $V = \{v_0, v_1, \cdots, v_{T-1}\}$. The edge set E has an edge (s, u_t) for each $u_t \in U$ with capacity $w(u, s_t) = \lambda_t$. Likewise, it has an edge (v_t, s') for each $v_t \in V$ with capacity $w(s_t, s') = \hat{\lambda}_t$. Finally, we add edges $(u_t, v_{t'})$ with capacity $+\infty$ as long as $t \leq t' \leq t + \tau$. We then compute the maximum flow from source s to sink s' in graph G and compute the required load movement schedule $L(t, t')$ to equal the flow routed on edge $(u_t, v_{t'})$.

Theorem 4. *The above process finds a valid load movement schedule L that corresponds to the arrival sequence λ and service sequence $\hat{\lambda}$ in time $O(\tau T^2)$.*

Proof. First, we establish that all the load is successfully reassigned without any being dropped. That is, the maximum flow routed equals the total load $\sum_i \lambda_i$ that arrived at the cluster. Since the maximum flow equals the minimum capacity of a cut that separates the source s and sink s' vertices, we compute the capacity of the minimum cut of G. Note that the minimum cut will not contain any edge in $U \times V$ since those edges have infinite capacity. Therefore, it suffices to consider cuts that place vertices $\{s\} \cup \{u_0 \cdots u_t\} \cup \{v_0, \cdots, v_{t+\tau}\}$ on one side and rest of the vertices on the other side, for some $0 \leq t \leq T - 1$. Such a cut has capacity

$$\sum_{i=t+1}^{T-1} \lambda_i + \sum_{i=0}^{t+\tau} \hat{\lambda}_t,$$

which using Equation 5.5 is at least $\sum_{i=0}^{T-1} \lambda_i$. Now noting there exists a cut of size $\sum_{i=0}^{T-1} \lambda_i$, namely the cut with source s on one side and all other vertices on the other side, we can conclude that the capacity of the minimum cut is $\sum_{i=0}^{T-1} \lambda_i$ which in turn equals the routed flow through G. Thus, all load that arrived at the cluster is routed through G. Further, note that L constructed in this fashion obeys the delay

bound of τ, since we added only edges from a vertex u_t to vertices $\{v_t, \cdots, v_{t+\tau}\}$ when constructing G. Thus, the load movement schedule L is valid and when L is applied to the arrival load sequence λ we obtain the service load sequence $\hat{\lambda}$. Finally, note that using Orlin's max flow algorithm, computing the load assignment L takes $O(|V||E|) = O(\tau T^2)$ time. □

5.3 Evaluating the Benefits of Demand Response

To evaluate the cost benefits of demand response (DR) in an IDS we used extensive traces from Akamai [35], the largest commercial CDN, and ran the optimal demand response algorithm presented in Section 5.2 for each Akamai cluster. We used each of the three electricity pricing models described in Section 5.1 and analyzed the energy cost benefits for the IDS. For all our evaluations, we report on system-wide cost savings for the IDS by aggregating our results across all clusters. The system-wide metrics capture the situation where demand response is implemented in all the clusters of the IDS. As a baseline we compute the energy cost incurred by the IDS when no demand response is implemented in any of the clusters, i.e., in the baseline no load is shifted and the arrival load sequence and service load sequence are identical for each cluster. Energy cost savings is the percent reduction in cost due to DR, i.e.,

$$100 \times ((\text{baseline cost}) - (\text{cost with DR})/(\text{baseline cost})).$$

5.3.1 Empirical Data from the Akamai Network

For our analysis, we used extensive load traces collected over 25 days from a large set of Akamai clusters deployed in data centers in the US. The 22 clusters captured in our traces are distributed widely within the US and had 15439 servers in total, i.e., it is a representative sampling of Akamai's US deployments. Our load traces account for a peak traffic of 800K requests/second and an aggregate of 950 million requests

delivered to clients. The traces consist of a snapshot of total load served by each cluster collected every 5-minute interval from Dec 19th 2008 to January 12th 2009, a time period that includes the busy holiday shopping season for e-commerce traffic (Figure 5.1). In the figure, one may note load variations due to day, night, weekday, weekend, and holidays (such as low load on day no. 8, which was Christmas).

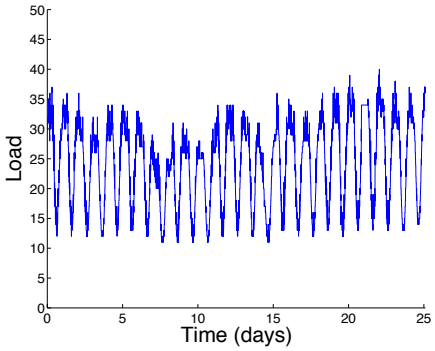
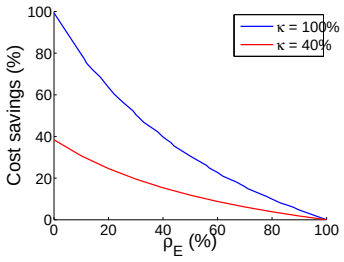


Figure 5.1: Average load per server measured every 5 minutes across 22 Akamai clusters in the US over 25 days.

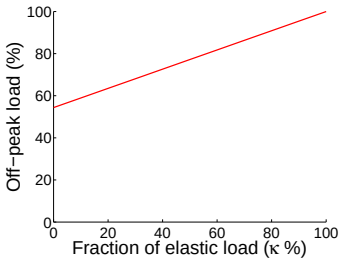
5.3.2 Time-of-use (TOU) Pricing Model

We evaluate energy cost benefits of DR on a typical time-of-use energy contract where the energy usage charge is a function of the time of day. The energy consumed between 9 AM to 9 PM on weekdays is charged at the on-peak energy rate of α_{on} dollars per kWh. The energy consumed during the remaining duration is charged at the off-peak rate of α_{off} dollars per kWh.

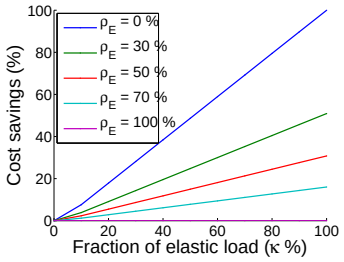
Varying ρ_E. Electric utility companies incentivize off-peak usage by providing discounted pricing. We capture this through $\rho_E = \frac{\alpha_{off}}{\alpha_{on}}$, the ratio of off-peak to on-peak energy usage charge. $\rho_E = 1$ corresponds to flat pricing where the energy charge is independent of the time of day. $\rho_E = 0$ corresponds to the case where off-peak usage



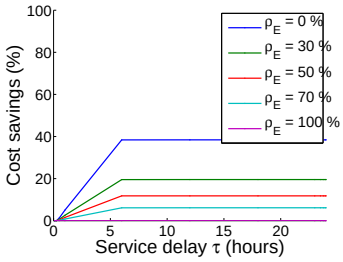
(a) Energy cost savings as a function of ρ_E for different fractions of elastic load (κ). 12% savings when $\rho_E = 50\%$ for 40% elastic load with $\tau = 12$ hours.



(b) 72% of the total load is served at off-peak hours when $\kappa = 40\%$ of the load is elastic with $\tau = 12$ hours.



(c) Cost savings increase linearly with the fraction of elastic load.



(d) At 40% elastic load, max service delay $\tau = 6$ hours is sufficient to get maximum savings.

Figure 5.2: Time-of-use pricing

is free (such as in underutilized renewable sources of energy). To study the impact of discounted pricing, we varied ρ_E and plotted it against the savings obtained by our algorithm for $\tau = 12$ hours. (Figure 5.2a). A service delay of half a day allows us to move almost the entire load from peak periods to off-peak hours ($\rho = 0$), saving 99% when $\kappa = 100\%$, and 38% savings with $\kappa = 40\%$. The savings drop to 0 when the

incentive is removed and off-peak is charged at the same rate as on-peak ($\rho_E = 1$). For a typical value of $\rho_E = 50\%$ where off-peak energy charge is half of the on-peak charge we are able to save 12% even when only 40% of the load is elastic.

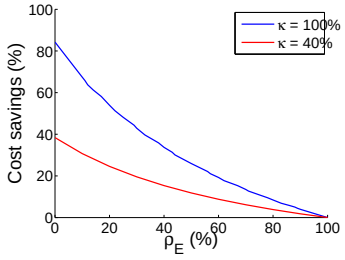
Varying elastic load fraction κ. Any increase in the fraction of elastic load κ is exploited by the algorithm by moving a larger fraction of the overall load to off-peak hours. Figure 5.2b quantifies this by plotting κ against the fraction of overall traffic served during off-peak hours over the duration of the entire trace. For interactive loads, where $\kappa = 0$, about 55% of the entire load is handled during off-peak hours. With increasing flexibility to delay load, the fraction of off-peak load increases linearly with κ. For typical values of $\tau = 12$ hours, $\kappa = 40\%$ the algorithm serves 72% of the entire load during off-peak hours.

The linear relation between κ and the off-peak load gets reflected in the cost savings as well, as seen in Figure 5.2c. Individual curves in the figure correspond to different values of the energy usage pricing ratio ρ_E. The lower the value of ρ_E, the higher the discount for off-peak usage and thus the greater savings.

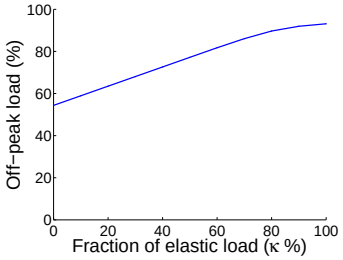
Varying maximum allowable delay τ. Different elastic tasks processed by an IDS have different delay sensitivities. A task such as billing is relatively insensitive to delay, since it suffices that the monthly bills for customers of the IDS is ready by the end of the month. However, other elastic tasks like a software update or video transcoding is expected to complete within hours.The relation between maximum allowable service delay τ and cost savings obtained by the algorithm are shown in Figure 5.2d for $\kappa = 40\%$. Individual curves in the figure correspond to different values of the energy usage pricing ratio ρ_E. It is interesting to note that increasing τ beyond a threshold provides little additional cost savings. In particular, a service delay $\tau =$ 6 hours is sufficient to obtain the maximum possible savings. Thus, adding elastic loads with more laxity than 6 hours does not provide larger benefits. The six hour

threshold is a consequence of the time duration of the on-peak and off-peak time periods in the TOU pricing.

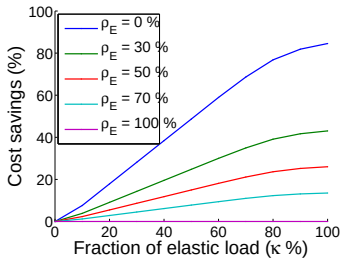
Optimizing electricity costs without increasing bandwidth costs: The TOU pricing



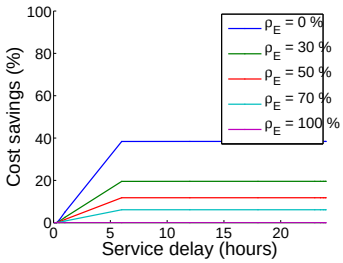
(a) Energy cost savings as a function of ρ_E for different fractions of elastic load (κ). 12% savings when $\rho_E = 50\%$ for 40% elastic load with $\tau = 12$ hours.



(b) 72% of the total load is served at off-peak hours when $\kappa = 40\%$ of the load is elastic with $\tau = 12$ hours.



(c) Cost savings increase linearly for $\kappa < 70\%$ and then slowly plateaus



(d) At 40% elastic load, max service delay $\tau = 6$ hours is sufficient to get maximum savings.

Figure 5.3: Energy cost optimization without increasing bandwidth costs using the max-load constraints.

model does not explicitly charge for the maximum power demand of a cluster. So

the cost optimizations we saw earlier in this section could potentially create new load peaks when moving load from on-peak to off-peak hours. In fact, such peaks could cause the maximum load of the service load sequence to be higher than that of the arrival load sequence! Such a situation is untenable from the standpoint of other operational costs incurred by an IDS. Besides electricity, a primary operating cost for an IDS is bandwidth. Bandwidth is often priced using a 95/5 contract where the billing period is divided into 5-minute intervals and the average bandwidth used by the cluster is computed over each such interval. The bandwidth cost of the cluster is then proportional to 95^{th} percentile of the 5-minute averages [4]. We use the maximum load of the service load sequence of a cluster as a reasonable proxy for bandwidth costs incurred in that cluster. In particular, we assume more load means proportionally more bandwidth usage. Further, as we did in [4], we use "maximum" as a proxy for the "95^{th} percentile" as the latter is difficult to analyze and optimize. Note that if our energy cost optimization increases the bandwidth cost, that could negate the economic incentive[2] for the IDS performing such an optimization.

We now optimize demand response in the TOU pricing model with the additional constraint that the bandwidth costs are not increased. To achieve this we add a new constraint to our optimization algorithm mandating that the maximum load of the output service load sequence $\hat{\lambda}$ is no more than the maximum load of the input arrival load sequence λ. Specifically, let the maximum load in the arrival sequence be $\lambda_{max} = \max_{i=0}^{T-1} \lambda_i$. We require that $\forall i, \hat{\lambda}_i \leq \lambda_{max}$.

A limit on the maximum load decreases the ability to run at higher utilization and thus exploit energy discounts effectively. Therefore we would expect cost savings to decrease with the max-load constraints. Figure 5.3a shows that savings drops to

[2]It is also worth noting that any scheme that increases maximum load also increases the maximum power demand, instead of decreasing it. This negates a primary purpose of an utility offering TOU pricing to incentivize reduction in peak power demand.

84% when off-peak energy is free ($\rho_E = 0$) for pure elastic load ($\kappa = 100\%$). It is interesting to note however that the additional constraints have no impact for a lower fraction of elastic load ($\kappa = 40\%$). Comparing figures 5.2c and 5.3c we see that the max-load constraint has no impact on the behavior of the algorithm for $\kappa < 70\%$.

5.3.3 Demand Pricing

Demand pricing is an important component of most realistic electricity pricing contracts, allowing electric utilities to directly manage the peak power demand by charging on the basis of it. A demand pricing contract consists of an on-peak demand charge β_{on} and an off-peak demand charge β_{off}. The on-peak charge is applied to the maximum integrated thirty-minute demand during on-peak periods (D_{on}) seen over the billing period. Similarly the off-peak charge is applied to the maximum integrated thirty-minute demand during off-peak periods (D_{off}). The electricity cost for a demand pricing policy π for a load sequence λ is

$$cost_\pi(\lambda) = \beta_{on} D_{on} + \beta_{off} D_{off}.$$

Varying relative off-peak ratio ρ_D. Electric utilities are underutilized during off-peak hours and can support higher demands from individual consumers and incentivize them by discounted off-peak pricing. We capture this discounting through $\rho_D = \frac{\beta_{off}}{\beta_{on}}$, the relative price of off-peak demand. $\rho_D = 0$ corresponds to free usage during off-peak hours, and $\rho_D = 1$ corresponds to time-insensitive demand pricing. Figure 5.4a plots cost savings as a function of the relative off-peak price ρ_D. For a maximum service delay of half a day the savings resemble those seen earlier for pure energy usage contracts when $\rho_D = 0$. But for $\rho_D = 100\%$ savings are still possible by smoothing out the peaks. When the entire load is capable of withstanding service delays of $\tau = 12$ hours, we see savings of 37%. For a lower value of $\kappa = 40\%$, we still

get savings of 27% at $\rho_D = 100\%$. For typical values of $\rho_D = 50\%$ and $\kappa = 40\%$ we get 32% savings.

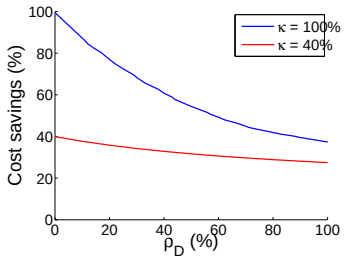
Varying percent of elastic load κ. Since pricing depends on peak demand, substantial savings can be obtained by smoothing out the largest peaks with relatively low movement in load. As the peaks and valleys get shallower, more load needs to be moved for incremental savings. We see this in Figures 5.4b and 5.4c where for low values of κ, savings grow rapidly without moving load from on-peak to off-peak hours. As κ increases beyond 30%, the gains obtained by local valley filling are exhausted and additional gains are obtained by moving traffic to off-peak hours.

Varying maximum allowable service delay τ. The relationship between the maximum allowed service delay and cost savings are shown in Figure 5.4d for $\kappa = 40\%$. As in the case for time-of-use contracts, we see that maximum possible savings are achieved by a service delay of at most 6 hours.

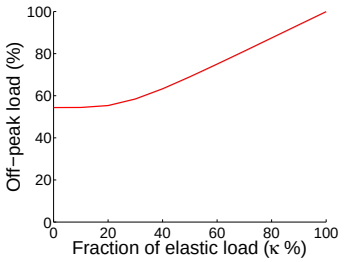
5.3.4 Hybrid Pricing

Electric utilities use a combination of energy usage and demand charges to increase the usage during off-peak hours and at the same time decrease the peak power usage. We capture this through a *mixing coefficient* $\rho_M = \frac{\beta_{on}}{\alpha_{on}}$, the ratio of on-peak demand charge to on-peak energy charge. $\rho_M = 0$ corresponds to a pure energy usage contract such as time-of-use, while as ρ_M tends to infinity the contract gets closer to a pure demand pricing model.

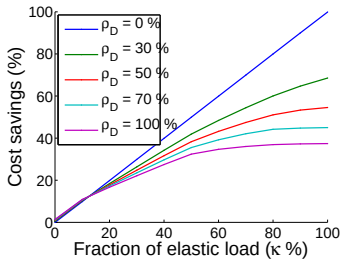
Varying mixing coefficient ρ_M. In Figure 5.5a, we study the impact of demand response as ρ_M is increased with $\kappa = 100\%$. When energy usage costs dominate at low values of ρ_M we see savings as observed earlier in Figure 5.2a with 0 savings for $\rho_E = 100\%$ and 31% for $\rho_E = 50\%$. When the contribution of demand charges dominates for large values of ρ_M we see savings rise to roughly 37% and 53% respectively for $\rho_D = 100\%$ and $\rho_D = 50\%$ respectively, comparable to values seen in Figure 5.4a. It



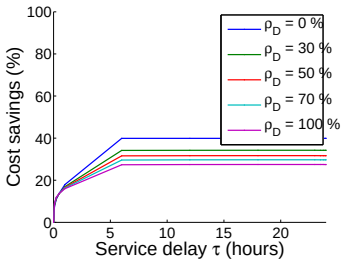
(a) 32% savings when $\rho_D = 50\%$ for $\kappa = 40\%$ elastic load with $\tau = 12$ hours.



(b) 63% of the total load is served at off-peak hours for 40% elastic $\tau = 12$ hours



(c) Cost savings flatten out as the fraction of elastic load increases

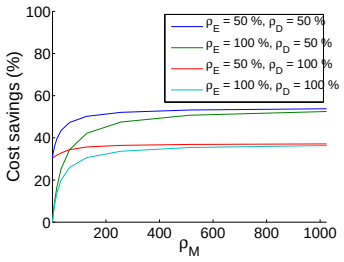


(d) At 40% elastic load, 6 hours of service delay is sufficient to get maximum savings

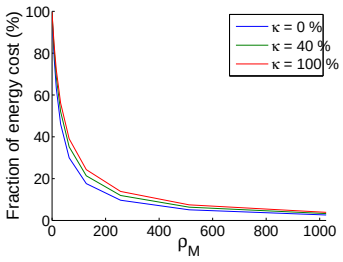
Figure 5.4: Demand Pricing

is interesting to note that savings increase as the contract tends towards a demand pricing model.

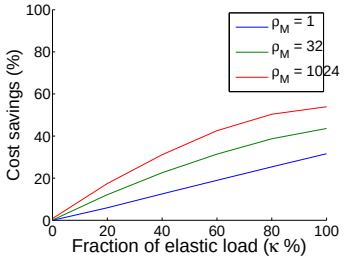
We consider a typical hybrid contract where $\rho_E = \rho_D = 50\%$ in greater detail. Figure 5.5b shows the contribution of energy charges as a fraction of the total cost paid to the utility. We see the curve drop-off asymptotically from 99.6% to 3.3% as ρ_M increases from 0.1 to 1024. Energy utilization charges contribute about the same as demand charges when $\rho_M = 32$.



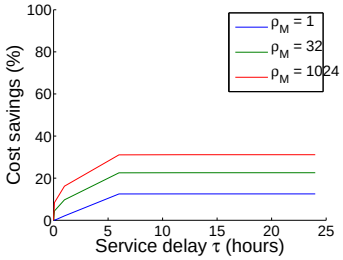
(a) 43% savings when $\rho_M = 32$ for $\rho_D = 50\%$, $\rho_E = 50\%$, $\kappa = 100\%$ elastic load with $\tau = 12$ hours.



(b) Energy costs contribute roughly half of the total cost when $\rho_M = 32$ for $\kappa = 40\%$ with $\tau = 12$ hours



(c) The relation between savings and the fraction of elastic load κ becomes non-linear as ρ_M increases



(d) At 40% elastic load, 6 hours of service delay is sufficient to get maximum savings of 23% at $\rho_M = 32$

Figure 5.5: Hybrid Pricing

Varying fraction of elastic load κ. Figure 5.5c shows the relation between the fraction of elastic κ and cost savings for different values of the mixing coefficient ρ_M. Low values correspond to linear relation, similar to pure energy contracts (Figure 5.2c) while high values mirror the non-linear relation observed in Figure 5.4c.

Varying maximum allowable delay τ. A high service delay τ allows greater freedom to the algorithm to postpone load and thus increase savings. It is interesting to note

though that a maximum service delay of 6 hours is sufficient to obtain the maximum possible savings through demand response. The savings obtained increases with the value of ρ_M when the demand pricing component begins to dominate. Savings increase from 13% to 31% as ρ_M increases from 1 to 1024 for $\kappa = 40\%$ elastic load.

5.4 Related Work

Recently the area of energy-aware ("green") distributed system design has seen significant research attention. Design of energy-aware techniques for data centers has involved power management mechanisms at a server level [19] as well shutting down servers when not needed [12, 13, 17]. Thermal-aware placement of workloads across servers to reduce energy and cooling costs has also been studied [33]. FAWN uses "wimpy" nodes to serve simple content and reduce cluster energy costs [7]. More recent work has studied how to incorporate intermittent renewable energy to power data center clusters [46, 18]. Design of energy-aware Internet-scale systems has also seen recent attention. The use of server shutdown and cluster shutdown have been proposed as mechanisms to turn off less utilized servers or clusters in a CDN and reduce energy costs [29, 25, 31]. Separately techniques to move incoming load to other nearby data centers with lower electricity prices has been proposed as a mechanism to reduce the energy bills of an IDS [42]. Our approach is complementary since we propose moving the load in the temporal dimension—by delaying elastic requests—and thereby reducing electricity bills.

The use of automated demand response techniques have been studied in the context of the smart grid [28]. However such techniques have been designed for smart buildings where a building reduces its energy footprint by automatically switching off less important loads upon receiving a DR signal from the grid. Integration of demand-response directly into data centers or distributed systems is relatively new idea and our approach takes an initial step in that direction.

5.5 Conclusions

In this chapter we studied techniques for reducing the energy costs in an IDS by performing demand-response to respond to variable electricity prices. Our proposed demand response approach consists of moving a portion of the incoming load—comprising elastic requests—to a later point in time, thereby temporarily curtailing the server demand and reducing energy costs. Such an approach is best suited for elastic requests such as background downloads of software updates or background computational tasks that do not always require immediate service. We presented an optimization-driven algorithm for our demand-response approach and evaluated the potential benefits of this approach for realistic workloads from a commercial CDN and realistic electricity pricing models. Our results showed that our algorithm can achieve 12% savings in the presence of time-of-use electricity pricing when only 40% of the demand is elastic. The savings increase to 32% under peak-based demand pricing and to 23% under a combination of time-of-use and demand pricing.Further, most if not all of energy savings can be obtained without an increase in bandwidth costs.

As part of future work, we plan to study hybrid techniques that combine the ability to move load in the spatial dimension (by moving some load to nearby data centers) as well as the temporal dimension (by deferring a portion of the load) to achieve greater energy savings. It is likely that geographically separated data centers will differ not just in the price of power but also the type of contract imposed by the utility, which can provide greater scope for cost savings.

CHAPTER 6

SUMMARY AND FUTURE WORK

This thesis demonstrates that large distributed systems like CDNs can save on energy costs, by reducing energy usage or by exploiting differences in electricity pricing, without increasing operating costs while still satisfying client-side SLAs.

We proposed an optimal offline algorithm and an online algorithm to extract energy savings at the level of local load balancing within a data center. We show that it is possible to reduce the energy consumption of a CDN by 51% while ensuring five nines of service availability and an average of just 1 transition per server per day. Further, we show that keeping even 10% of the servers as hot spares helps absorb load spikes due to global flash crowds with little impact on availability SLAs.

Next we proposed a new technique called cluster shutdown to turn off entire clusters to save energy. Our experimental shows that cluster shutdown can reduce system-wide energy usage by 67% in the optimal case, and most of these savings can be achieved without sacrificing end-user performance and bandwidth costs. In addition, the technique works well even when shutdown is limited to once per day for each cluster and when the load is not known in real-time and must be predicted. We believe that cluster shutdown is a strong candidate for implementation in an actual CDN, especially since it fits in more easily with current CDN architectural principles in comparison with server shutdown techniques studied in the past.

Finally we studied techniques for reducing the energy costs in a CDN by performing demand-response to respond to variable electricity prices. Our proposed demand response approach consists of moving a portion of the incoming load - comprising

elastic requests - to a later point in time, thereby temporarily curtailing the server demand and reducing energy costs. We presented an optimization-driven algorithm for our demand-response approach and evaluated the potential benefits of this approach for realistic workloads from a commercial CDN and realistic electricity pricing models. Our results showed that our algorithm can achieve 12% savings in the presence of time-of-use electricity pricing when only 40% of the demand is elastic. The savings increase to 32% under peak-based demand pricing and to 23% under a combination of time-of-use and demand pricing. Further, most if not all of energy savings can be obtained without an increase in bandwidth costs.

6.1 Future work

Recent developments in the area of renewable energy indicate that the combination of solar and energy storage is gaining traction as an alternative to fossil fuels [15, 38]. While it may not be economically viable to move completely off-grid, renewables are being used to clip the peak demand from industrial customers [38]. Batteries help smooth out the intermittent nature of renewable power generation but there still remain significant challenges to their adoption at a commercial scale. Initial attempts have been made to redesign data centers and scheduling algorithms to use renewables [8, 18, 36].

A direction of future work would be to reduce energy cost in CDNs where a portion of server clusters are powered through renewables. The use of battery storage can provide a lower bound on workload that can be handled at any specific deployment. Further, the geographically distributed nature of data centers deployed reduces the risk of renewable energy generation. The challenge would be to maintain SLAs while reducing energy from the grid.

Another direction of future work would be to use *net-metering* contracts to reduce energy costs. When renewable energy production is significant, a data center can

deliver energy to the grid and use it to offset energy use over the billing period. Time-of-use pricing provide an opportunity to sell excess energy during peak hours and reduce the load on the grid. The addition of energy storage and local energy generation would be an extension to our earlier work on demand-response techniques for CDNs.

BIBLIOGRAPHY

[1] *Duke Energy : Schedule OPT.* http://www.duke-energy.com/pdfs/scscheduleopt.pdf.

[2] *Ontario Hydro Rates.* http://www.ontario-hydro.com/index.php?page=current_rates.

[3] *Wisconsin Electric Rates.* http://www.we-energies.com/pdfs/etariffs/wisconsin/elecrateswi.pdf.

[4] Adler, Micah, Sitaraman, Ramesh K., and Venkataramani, Harish. Algorithms for optimizing the bandwidth cost of content delivery. *Computer Networks 55*, 18 (2011), 4007–4020.

[5] Akamai Technologies. *VOD Transcoding*, 2013. http://www.akamai.com/dl/brochures/sola_vision_transcoding_brief.pdf.

[6] Amur, H., Cipar, J., Gupta, V., Ganger, G.R., Kozuch, M.A., and Schwan, K. Robust and flexible power-proportional storage. In *Proceedings of the 1st ACM symposium on Cloud computing* (2010), ACM, pp. 217–228.

[7] Anderson, D., Franklin, J., Kaminsky, M., Phanishayee, A., Tan, L., and Vasudevan, V. Fawn: A fast array of wimpy nodes. In *Proceedings of ACM SOSP* (October 2009).

[8] Arlitt, Martin, Bash, Cullen, Blagodurov, Sergey, Chen, Yuan, Christian, Tom, Gmach, Daniel, Hyser, Chris, Kumari, Niru, Liu, Zhenhua, Marwah, Manish, et al. Towards the design and operation of net-zero energy data centers. In *Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm), 2012 13th IEEE Intersociety Conference on* (2012), IEEE, pp. 552–561.

[9] Barroso, L.A., and Holzle, U. The case for energy-proportional computing. *Computer 40*, 12 (2007), 33–37.

[10] Beloglazov, Anton, Buyya, Rajkumar, Lee, Young Choon, and Zomaya, Albert Y. A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in Computers 82* (2011), 47–111.

[11] California Energy Commission. *Nonresidential Alternative Calculation Method (ACM) approval manual for the 2008 building energy efficiency standards*, December 2008.

[12] Chase, J., Anderson, D., Thakar, P., Vahdat, A., and Doyle, R. Managing energy and server resources in hosting centers. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles (SOSP)* (October 2001), pp. 103–116.

[13] Chen, A., Das, W., Qin, A., Sivasubramaniam, A., Wang, Q., and Gautam, N. Managing server energy and operational costs in hosting centers. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (June 2005).

[14] Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., and Zhao, F. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation* (2008), USENIX Association, pp. 337–350.

[15] Citigroup. *Energy Darwinism II*, September 2014. `http://citi.us/1vJooWQ`.

[16] Douglis, Fred, Krishnan, Padmanabhan, Bershad, Brian, et al. Adaptive disk spin-down policies for mobile computers. *Computing Systems 8*, 4 (1995), 381–413.

[17] Gandhi, A., Gupta, V., Harchol-Balter, M., and Kozuch, M. Optimality analysis of energy-performance trade-off for server farm management. In *Proc. 28th Intl. Symposium on Computer Performance, Modeling, Measurements, and Evaluation (Performance 2010) Namur, Belgium* (November 2010).

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

[19] Kant, K., Murugan, M., and Du, D H. C. Willow: A control system for energy and thermal adaptive computing. In *Procedings of the 25th IEEE IPDPS* (2011).

[20] Koomey, J.G. Worldwide electricity used in data centers. *Environmental Research Letters 3* (Sept 2008).

[21] Krioukov, A., Mohan, P., Alspaugh, S., Keys, L., Culler, D., and Katz, R. Napsac: Design and implementation of a power-proportional web cluster. In *Proc. of ACM Sigcomm workshop on Green Networking* (August 2010).

[22] Kusic, Dara, Kephart, Jeffrey O, Hanson, James E, Kandasamy, Nagarajan, and Jiang, Guofei. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing 12*, 1 (2009), 1–15.

[23] Lin, M., Wierman, A., Andrew, L.L.H., and Thereska, E. Dynamic right-sizing for power-proportional data centers. *Proc. IEEE INFOCOM, Shanghai, China* (2011), 10–15.

[24] Liu, Z., Chen, Y., Bash, C., Wierman, A., Gmach, D., Z. Wang, M. Marwah, and Hyser, C. Renewable and cooling aware workload management for sustainable data centers. In *Proceedings of ACM Sigmetrics* (2012).

[25] Liu, Z., Lin, M., Wierman, A., Low, S., and Andrew, L. Greening geographical load balancing. In *Preprint. Extension of a paper that appeared in ACM Sigmetrics, 2011* (2012).

[26] Liu, Zhenhua, Liu, Iris, Low, Steven, and Wierman, Adam. Pricing data center demand response. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems* (2014), SIGMETRICS '14, pp. 111–123.

[27] Liu, Zhenhua, Wierman, Adam, Chen, Yuan, Razon, Benjamin, and Chen, Niangjun. Data center demand response: Avoiding the coincident peak via workload shifting and local generation. *Performance Evaluation 70*, 10 (2013), 770 – 791. Proceedings of {IFIP} Performance 2013 Conference.

[28] Longbo, H., Walrand, J., and Ramchandran, K. Optimal demand response with energy storage management. In *Proceedings of IEEE SmartGrid Comm* (December 2012).

[29] Mathew, Vimal, Sitaraman, Ramesh K, and Shenoy, Prashant. Energy-aware load balancing in content delivery networks. In *INFOCOM, 2012 Proceedings IEEE* (2012), IEEE, pp. 954–962.

[30] Mathew, Vimal, Sitaraman, Ramesh K., and Shenoy, Prashant. Energy-efficient content delivery networks using cluster shutdown. *Sustainable Computing: Informatics and Systems*, 0 (2014).

[31] Mathew, Vimal, Sitaraman, Ramesh K., and Shenoy, Prashant J. Energy-efficient content delivery networks using cluster shutdown. In *International Green Computing Conference, IGCC 2013, Arlington, VA, USA, June 27-29, 2013, Proceedings* (June 2013), pp. 1–10.

[32] Mathew, Vimal, Sitaraman, Ramesh K., and Shenoy, Prashant J. Reducing energy costs in internet-scale distributed systems using load shifting. In *Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on* (Jan 2014), pp. 1–8.

[33] Moore, J., Chase, J., and Ranganathan, P. Making scheduling "cool": Temperature-aware workload placement in data centers. In *Proc. USENIX ATC (USENIX '05)* (2005).

[34] Nathuji, Ripal, and Schwan, Karsten. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review* (2007), vol. 41, ACM, pp. 265–278.

[35] Nygren, E., Sitaraman, R.K., and Sun, J. The Akamai Network: A platform for high-performance Internet applications. *ACM SIGOPS Operating Systems Review 44*, 3 (2010), 2–19.

[36] Oró, Eduard, Depoorter, Victor, Garcia, Albert, and Salom, Jaume. Energy efficiency and renewable energy integration in data centres. strategies and modelling review. *Renewable and Sustainable Energy Reviews 42* (2015), 429–445.

[37] Pallipadi, Venkatesh, and Starikovskiy, Alexey. The ondemand governor. In *Proceedings of the Linux Symposium* (2006), vol. 2, pp. 215–230.

[38] Parkinson, Giles. *How battery costs could plunge below $100/kWh*. REneweconomy, October 2014. `http://reneweconomy.com.au/2014/battery-storage-costs-plunge-below100kwh-19365`.

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

[40] Pinheiro, Eduardo, Bianchini, Ricardo, Carrera, Enrique V, and Heath, Taliver. Load balancing and unbalancing for power and performance in cluster-based systems. In *Workshop on compilers and operating systems for low power* (2001), vol. 180, Barcelona, Spain, pp. 182–195.

[41] Pitchaikani, Bala. *Strategies for the Containerized Data Center*, September 2011. `http://www.datacenterknowledge.com/archives/2011/09/08/strategies-for-the-containerized-data-center/`.

[42] Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., and Maggs, B. Cutting the electric bill for internet-scale systems. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (2009), ACM, pp. 123–134.

[43] Raghavendra, Ramya, Ranganathan, Parthasarathy, Talwar, Vanish, Wang, Zhikui, and Zhu, Xiaoyun. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News* (2008), vol. 36, ACM, pp. 48–59.

[44] Rao, Lei, Liu, Xue, Xie, Le, and Liu, Wenyu. Minimizing electricity cost: optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM, 2010 Proceedings IEEE* (2010), IEEE, pp. 1–9.

[45] Rath, John. *DCK Guide To Modular Data Centers: Why Modular?*, October 2011. `http://www.datacenterknowledge.com/archives/2011/10/20/dck-guide-to-modular-data-centers-why-modular/`.

[46] Sharma, Navin, Barker, Sean Kenneth, Irwin, David E., and Shenoy, Prashant J. Blink: managing server clusters on intermittent power. In *ASPLOS* (2011), pp. 185–198.

[47] Srikantaiah, Shekhar, Kansal, Aman, and Zhao, Feng. Energy aware consolidation for cloud computing. *Proceedings of the 2008 conference on Power aware computing and systems 10* (2008).

[48] Stansberry, Matt, and Kudritzki, Julian. *Uptime Institute 2012 Data Center Industry Survey*. Uptime Institute, 2012. `http://www.uptimeinstitute.com/images/stories/Uptime_Institute_2012_Data_Industry_Survey.pdf`.

[49] Stillwell, Mark, Schanzenbach, David, Vivien, Frédéric, and Casanova, Henri. Resource allocation using virtual clusters. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (2009), IEEE, pp. 260–267.

[50] Tolia, N., Wang, Z., Marwah, M., Bash, C., Ranganathan, P., and Zhu, X. Delivering energy proportionality with non energy-proportional systems-optimizing the ensemble. In *Proc of Workshop on Power-aware Computing Systems, San Diego, CA* (December 2008).

[51] Tolia, N., Wang, Z., Ranganathan, P., Bash, C., Marwah, M., and Zhu, X. Unified thermal and power management in server enclosures. In *Proceedings of the ASME/Pacific Rim Technical Conference and Exhibition (InterPACK '09)* (July 2009).

[52] Verma, Akshat, Ahuja, Puneet, and Neogi, Anindya. pMapper: power and migration cost aware application placement in virtualized systems. In *Middleware 2008*. Springer, 2008, pp. 243–264.

[53] Wang, Hao, Huang, Jianwei, Lin, Xiaojun, and Mohsenian-Rad, Hamed. Exploring smart grid and data center interactions for electric power load balancing. *SIGMETRICS Perform. Eval. Rev. 41*, 3 (2014), 89–94.

[54] Wendell, Patrick, Jiang, Joe Wenjie, Freedman, Michael J, and Rexford, Jennifer. Donar: decentralized server selection for cloud services. *ACM SIGCOMM Computer Communication Review 41*, 4 (2011), 231–242.

[55] Wierman, A., Andrew, L.L.H., and Tang, A. Power-aware speed scaling in processor sharing systems. In *INFOCOM 2009, IEEE* (2009), IEEE, pp. 2007–2015.

[56] Wierman, A., Liu, Zhenhua, Liu, I., and Mohsenian-Rad, H. Opportunities and challenges for data center demand response. In *Green Computing Conference (IGCC), 2014 International* (Nov 2014), pp. 1–10.