Energy-Aware Load Balancing in Content Delivery Networks

Vimal Mathew[†], Ramesh K. Sitaraman[†‡] and Prashant Shenoy[†]
[†]University of Massachusetts, Amherst [‡]Akamai Technologies Inc.
Email: {vml, ramesh, shenoy}@cs.umass.edu

Abstract*—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 CDNs to incorporate energy optimizations as a first-order principle. We propose techniques to turn off CDN servers during periods of low load while seeking to balance three key design goals: maximize energy reduction, minimize the impact on client-perceived service availability (SLAs), and limit the frequency of on-off server transitions to reduce wear-and-tear and its impact on hardware reliability. We propose 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. We evaluate our algorithms using real production workload traces from a large commercial CDN. Our results show that it is possible to reduce the energy consumption of a CDN by 51% while ensuring a high level of availability that meets customer SLA requirements and incurring an average of one on-off 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 and minimize any impact on availability SLAs. Finally, we show that redistributing load across highly proximal data centers can enhance service availability significantly, but has only a modest impact on energy savings.

I. Introduction

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) [4]. The environmental implications are equally profound. 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 [11]. These factors necessitate a complete rethinking of the fundamental architecture of Internet-scale systems to include energy optimization as a first-order principle.

An important Internet-scale distributed system to have evolved 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 [8]. A large CDN, such as that of a commercial provider like Akamai, 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 [14]. 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 in a major metro area. A CDN's servers cooperatively deliver content and applications to optimize the *availability* and *performance* experienced by the clients. Specifically, each client request is routed by the CDN's *load balancing system* to an "optimal" server that can serve the content with high availability and performance. Content and applications can typically be replicated on demand to any server of the CDN. The load balancing system ensures high availability by routing each client request to an appropriate server that is both *live* and *not overloaded*. Further, the load balancing system ensures good performance by routing each client request to a cluster that is *proximal* to that client. For instance, a client from a given metro area would be routed to a server cluster in the same metro area or perhaps even the same last-mile network. The proximity (in a network sense) of the client and the server ensures a communication path with low latency and loss. A comprehensive discussion of the rationale and system architecture of CDNs is available in [14].

Problem Description. In this paper, we focus on reducing the energy consumption of large Internet-scale distributed systems, specifically CDNs. Energy reduction in CDNs is a multi-faceted problem requiring advances in the power usage effectiveness (PUE) of data centers, improvements in server hardware to make them more "energy proportional" [4], as well as advances in the architecture of CDN itself. Our focus is on the CDN architecture, and more specifically, on its load balancing system. 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

978-1-4673-0775-8/12/$31.00 ©2012 IEEE

idle servers [7], [13]. 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 [4]. 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 [14] as well as in other types of Internet-scale services [2], [7]. 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 [15].

Research 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 [14]. 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 $\mathrm{Hibernate}$ 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. $\mathrm{Hibernate}$ 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, $\mathrm{Hibernate}$ 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, $\mathrm{Hibernate}$ 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.

Roadmap. After formulating our models and methodology (Section II), we study local load balancing (Section III) in an offline setting with the assumption that the entire traffic load pattern is known in advance (Section III-A), and then extend it to the more realistic online situation where future traffic is unknown (Section III-B). Then, we explore the gains to be had by moving traffic between clusters via global load balancing (Section IV). Finally, we discuss related work (Section V) and offer conclusions (Section VI).

II. 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 [14]. 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 [4] where the power (in Watts) consumed by a server serving load λ is

$$power(\lambda) \triangleq P_{idle} + (P_{peak} - P_{idle})\lambda, \qquad (1)$$

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 [14], 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 1, the precise manner in which load is distributed to the live servers makes

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

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 [2], [7] to tackle some of these issues.

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 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 $100 * (total\ served\ load)/(total\ input\ load)$. 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.

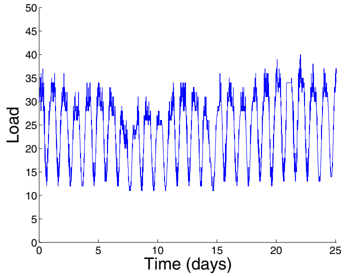


Fig. 1: Average load per server measured every 5 minutes across 22 Akamai clusters in the US over 25 days. Note load variations due to day, night, weekday, weekend, and holidays (such as low load on day no. 8, which was Christmas).

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 1).

III. 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 III-A). Then, we derive practical online algorithms and evaluate them on realistic load traces from a CDN (Section III-B), paying particular attention to how well they do in comparison to the theoretical baselines provided by the offline algorithms.

A. 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

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

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:

$$\begin{aligned} E(t,m) &= \min_{0 \leq m' \leq M} \{E(t-1, m') + \delta m \cdot power(\lambda_t/m) \\ &+ \alpha \cdot |m - m'|\}, \ if\ \lambda_t \leq \Lambda m \qquad (2) \\ &= +\infty, \ otherwise \end{aligned}$$

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 2. The first term in Equation 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.*

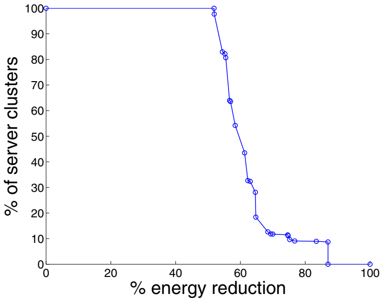


Fig. 2: Optimal Offline Energy Reduction. The median cluster achieved a 58% reduction.

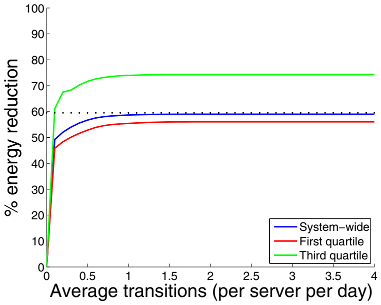


Fig. 3: Energy reduction attainable with bounded server transitions. About 98.6% of the optimal reduction can be achieved with just 1 transition per server per day. The dotted-line represents the optimal reduction with unbounded transitions.

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:

$$\begin{aligned} E(t,m,k) &= \min_{m-k \leq m' \leq m+k} \{E(t-1, m', k - |m - m'|) \\ &+ \delta m \cdot power(\lambda_t/m) \\ &+ \alpha \cdot |m - m'|\}, \ if\ \lambda_t \leq \Lambda m \qquad (3) \\ &= +\infty, \ otherwise \end{aligned}$$

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 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.

Next, we study how much energy reduction is possible if the server transitions are bounded and are required to be infrequent. Figure 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 $\mathrm{OPT(k)}$ 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 transitions. Besides system-wide energy reduction, Figure 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 $\mathrm{OPT(k)}$ 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 $t - 1$.

B. Online Algorithms

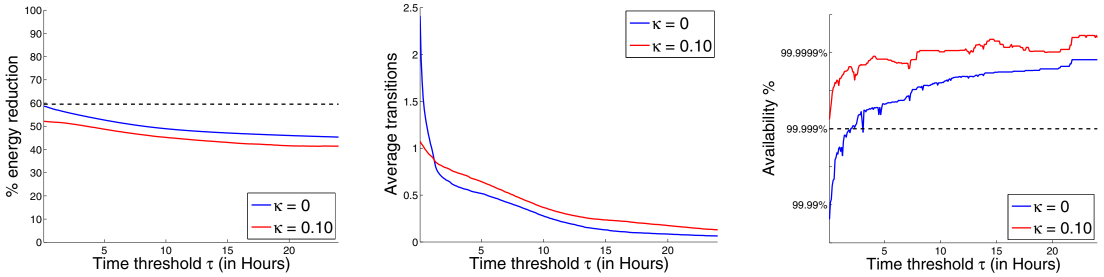
In contrast to offline algorithms, an online algorithm knows only the past and current load but has no knowledge of the future load. This accurately models any real-life load balancing system. At time t, an online algorithm does not know load λ_{t+1} and must estimate the number of servers to transition 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 $\mathrm{Hibernate}$ that achieves the "sweet spot" with respect to all three metrics, both for typical CDN traffic and flash crowds. While $\mathrm{Hibernate}$ 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 $\mathrm{Hibernate}$. The design of such a predictive Hibernate is future work.

Algorithm $\mathrm{Hibernate}$ 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 $\mathrm{Hibernate}$ 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 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 tradeoff 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, $\mathrm{Hibernate}$ 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 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 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 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 5, we quantify the tradeoffs associated with Λ as it

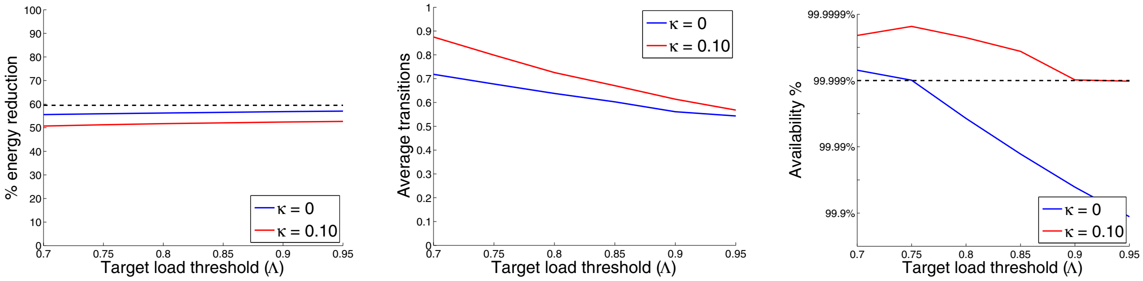


(a) Energy reduction decreases as τ increases as spare servers are kept live longer.

(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.

Fig. 4: The three key metrics for algorithm $\mathrm{Hibernate}$ on typical CDN load traces.



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

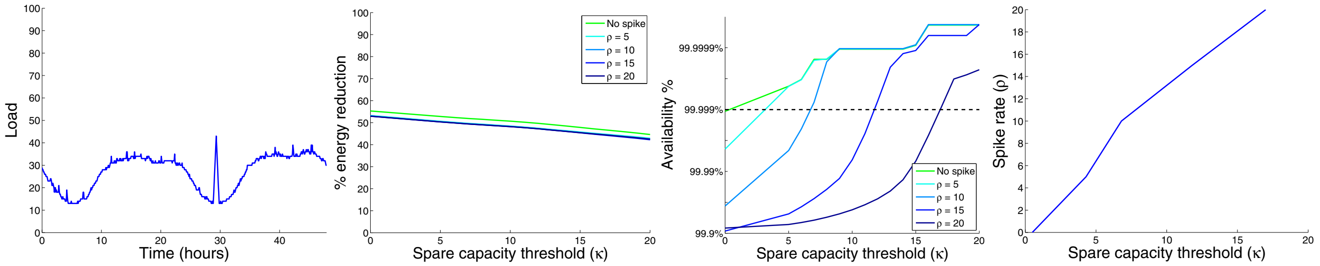
(b) Average transitions (per server per day) decreases with Λ as there is more effective capacity in each server.

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

Fig. 5: Variation of the three metrics with the target load threshold Λ

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 5a) as well as a smaller number of transitions (Figure 5b). However, increasing Λ also decreases availability (Figure 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 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 $\mathrm{Hibernate}$ for different settings of the spike rate ρ and the spare capacity threshold κ with the results summarized in Figure 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 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 capacity threshold κ where a larger ρ was tolerable only with a corresponding larger value of κ to sustain the



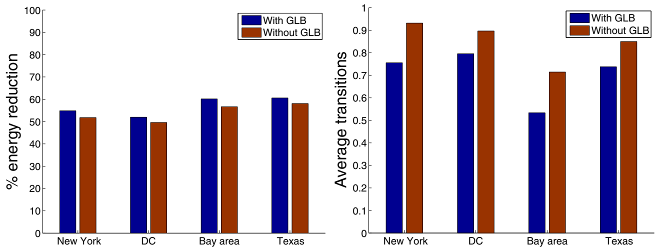
(a) A simulated load spike in the Ashburn cluster

(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

Fig. 6: The behavior of Hibernate during a simulated global flash crowd



(a) % Energy reduction

(b) Average Transitions (per server per day)

Fig. 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 %

Fig. 8: Availability improves drastically with global load balancing

required levels of service availability and meeting customer SLAs (Figure 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 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.

IV. 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[1] 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 7 and 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 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 7b). But, perhaps the key benefit of global load balancing is the increased availability (Figure 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.

V. Related Work

Energy management in data centers has been an active area of research in recent years [6]. Techniques that have been developed in this area include, use of DVFS to reduce energy, use of very low-power servers [3], routing requests to locations with the cheapest energy [15] and dynamically activating and deactivating nodes as demand rises and falls [5], [16], [12]. A key difference between much of this prior work and the current work is our 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 [13]. Like us, this paper also presents offline and online algorithms for turning servers on and off in data centers. While [13] targets data center workloads such as clustered mail servers, our focus is on CDN workloads. Further, [13] 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 [9], [10]. 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.

VI. Conclusions

In this paper, 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.

Acknowledgment

The authors would like to acknowledge the support of NSF awards CNS-0519894, CNS-0916972, and CNS-1117221. We would like to thank Rick Weber and Bruce Maggs for their help with data collection, and Tim Dunn and Nicole Peill-Moelter for stimulating discussions.

References

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

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

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

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

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

[6] A. Chen, W. Das, A. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. 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.

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

[8] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *Internet Computing, IEEE*, 6(5):50–58, 2002.

[9] A. Gandhi, V. Gupta, M. Harchol-Balter, and M. Kozuch. 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.

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

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

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

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

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

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

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