Energy-efficient disk caching for content delivery

Aditya Sundarrajan
University of Massachusetts Amherst
asundar@cs.umass.edu

Mangesh Kasbekar
Akamai Technologies
mkasbeka@akamai.com

Ramesh K. Sitaraman
University of Massachusetts Amherst & Akamai Technologies
ramesh@cs.umass.edu

ABSTRACT

Content delivery networks (CDNs) operate hundreds of thousands of servers that cache content and deliver them to users with high performance. Each CDN server has multiple spinning disks that are used for caching content. These disks account for 40-55% of the total server energy usage of a CDN. Reducing the energy consumption of a CDN by shutting down some of the disks during off-peak hours is the main focus of our work. The primary challenge with this approach is that shutting down disks decreases the size of the content cache, potentially lowering the cache hit rates, and resulting in a degradation in user-perceived performance. Our main contribution is developing and evaluating algorithms for cache sizing, disk shutdown, content placement and eviction that allow disks to be shut down without significantly impacting cache hit rates and user-perceived performance. We empirically evaluate the energy-performance tradeoff for our algorithms using extensive request traces from the world's largest CDN. We show that it is feasible to obtain a 30% disk energy savings with a 6.5% decrease in the normalized server hit rate and a mere 3% reduction in the normalized cluster hit rate. This work establishes disk shutdown as a key mechanism for energy savings in CDNs, paving the way for its future rollout on production networks.

Keywords

Content Delivery Network, Web Caching, Energy Optimization, Disk Drives

1. INTRODUCTION

Content delivery networks (CDNs) are Internet-scale distributed systems that deploy a large number of servers around the world to cache and serve web pages, videos, and other content to billions of users around the world. *Content providers* such as web portals, SaaS application providers, e-commerce sites, news outlets, media companies, social networks, and movie distribution services use CDNs to host and deliver their content. CDNs are now ubiquitous and are key to the functioning of the Internet, as most of the content accessed by users are served by such networks. To provide high performance for users accessing content, CDNs deploy *clusters* of servers in hundreds of data centers located at the "edges" of the Internet, so as to be proximal to users around the world. For instance, Akamai's CDN [16] deploys over 200,000 servers in over 1500 data centers around the world and serves 15-30% of the global web traffic.

Why focus on energy reduction of CDNs? While providing better performance in the form of fast downloads is the primary goal of a CDN, energy minimization has become critical in the past few years for two reasons. Deployed servers in data centers now account for more than 1.5% of the global power consumption [10], consuming more than mid-sized countries such as Argentina, and growing at a rapid pace commensurate with growth of the Internet. With greater awareness of climate change, the CDN industry is increasingly focused on making their systems more sustainable. For CDNs, reducing the energy usage of server deployments is a major part of their sustainability goals.

A second motivator for the reduction of the energy usage of CDN servers is the rising cost of energy. The operational expenditure (OPEX) of a CDN can be divided into two broad categories: bandwidth cost and colocation cost. Colocation cost is the cost of the datacenter space, racks, the energy needed to power the servers up and cool them down. While bandwidth costs have been the dominant factor in a CDN's OPEX in the past, bandwidth prices have fallen sharply each year in the past decades. The bandwidth cost of delivering 1 MByte has fallen from \$0.15 in 1998 to \$0.00005 per MB today, a drop of $1.8x$ per year. In stark contrast, the cost of energy has been rising over the past decade [2]. Due to these long-term price dynamics, colocation cost is now comparable to the bandwidth cost. The cost structures at most data centers are such that energy cost presently ranges between 30-50% of the total colocation cost, and is expected to only rise further in this decade.

Why focus on disk energy reduction? In each server, there are several components that consume energy. Energy consumed by the CPU, disks, fans, memory chips and motherboard chipset account for most of the consumed energy. CDN server models vary widely from each other. The number of disk drives per server is a model-specific parameter and can vary from 2 to as high as 64. Across most production server models that CDNs use, the average fraction of energy consumption attributed to the spinning disks is estimated to range from 40% to 55%. Therefore, disk energy represents a sizable chunk of the CDN's energy consumption, making

e-Energy'16, June 21 - 24, 2016, Waterloo, ON, Canada

ISBN 978-1-4503-4393-0/16/06…$15.00

DOI: http://dx.doi.org/10.1145/2934328.2934348

it the main focus of our study.

Why is disk shutdown uniquely well-suited for CDNs? The primary mechanism to save energy that we explore in our work is disk shutdown. For hard disks, the two natural options for energy reduction are shutting down disks entirely or reducing their rotational speed, the former providing more drastic energy reduction than the latter. In most IT systems, disks store *original* data that could be accessed at any time and shutting down disks means that the data is completely unavailable, an unacceptable outcome that must be avoided at all costs. In fact, most server software would need to be re-designed significantly to handle the unavailability of data from disk shutdowns. Therefore, disk shutdown has seldom been explored or implemented in industry. However, disk shutdown is a viable energy saving mechanism for a CDN because the disk cache of a CDN server only stores a *copy* of the content that is stored persistently at the content provider's origin servers. Thus, the unavailability of a cached copy is easily rectified by retrieving it from a peer server or origin. While this causes performance degradation for the user, it is less severe than content unavailability. Thus, if disk shutdown provides significant energy saving in exchange for a small performance degradation, that is an interesting possibility from the stand point of a CDN operator. Our work is focused on understanding this energy-performance tradeoff, allowing a CDN operator to choose an acceptable operating regime in that tradeoff.

Why not shutdown the entire server? A complementary approach studied in the literature is turning off servers entirely [5, 18, 11, 12, 14]. However, turning off servers has the disadvantage of complicating network management in a global CDN. If servers are unreachable for extended periods of time, they miss real-time reporting, software updates and control messages for that duration. This may upend network management guarantees and operational practices of the CDN platform. Thus, shutting down disks as proposed here, while the servers are live and serving content, represents an attractive complimentary option worth exploring.

What are the challenges in shutting down disks in a CDN? An important determinant of user-perceived performance is the cache hit rate which we define as the percentage of content bytes served to users from the server's cache, as opposed to being retrieved from a peer server or the origin[1]. When disks are shut down, the content stored on them become unavailable, leading to a decrease in the cache hit rate. To scope out the impact of disk shutdown on performance, we first implemented a simple baseline scheme that shuts down disks in proportion to the disk load, e.g., when the disks are loaded at x% of its I/O capacity, we turn off roughly x% of the disks chosen at random. The baseline scheme was tested on a simulated CDN server with traffic logs from a live server in the Akamai CDN. Figure 1 shows the hit rate for a server with the baseline disk shutdown algorithm (labeled `Load/Random/LRU`) in comparison with hit rate for the same server when all disks on (labeled `NOOFF`). The observed 15-20% drop in cache hit rate with disk shutdown would be unpalatable to the CDN operator for performance reasons, even if the energy savings were significant. *Our main challenge is to evolve algorithms that are smarter than the baseline scheme, so that the performance penalty incurred to obtain energy savings is not steep.*

[1]This metric is also called the *byte* hit rate. A similar alternate metric used in the literature is *object* hit rate which is the percentage of requests that are served from cache.

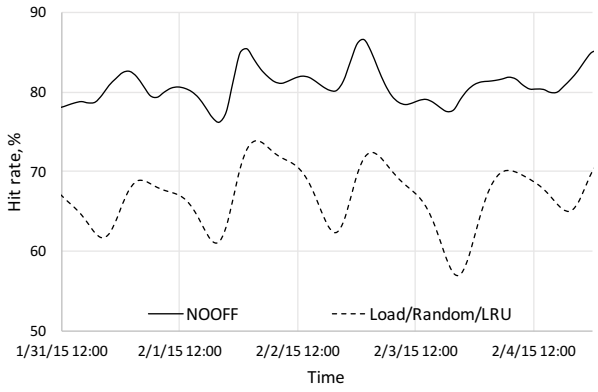

Figure 1: Hit rate decreases significantly when disks are shutdown using a simple baseline scheme.

1.1 Our contributions

The main contribution of our work is to show that unlike most enterprise class servers, CDN servers are able to save energy through disk shutdown without major software redesign or major performance degradation. We develop energy-efficient cache management schemes to address three key questions.

- *Cache sizing.* How large a disk cache does a server need to hold the "working set" of the content that is being accessed by users?
- *Disk shutdown.* Which disks must be shut down (or woken up) to realize the cache size that is required?
- *Content placement & eviction.* Where should content be placed and what is to be evicted if the cache is full?

For each of the above questions, we explore simple and implementable algorithms to understand their impact on both energy savings and cache hit rates. Using extensive traces from Akamai's CDN servers, we derive the energy-performance tradeoff for our algorithms. As shown in Figure 2, our algorithms achieve significant energy savings while incurring only a modest degradation in performance. For instance, our algorithms achieve a 30% energy reduction with only a 3% reduction in the normalized cluster hit rate and 6.5% reduction in the normalized server hit rate.

A key reason behind the effectiveness of our algorithms lies in the very nature of how content on the Internet is accessed by users. As shown in Figure 3, of the 25 million objects accessed on an Akamai CDN server over a period of 9 days, over 16 million were "one-hit-wonders" accessed only once! In fact, only 6.6% of the objects were accessed more than 10 times over the 9-day span. Further, as shown in Figure 6, 80% of the requests are for 1% of the objects. Our algorithms migrate the more popular content to a subset of disks within the server, allowing the other disks to be shut down. As long as our algorithms place at least one copy of the small fraction of popular objects on an active disk, the loss of the remaining "long tail" of less popular content due to disk shutdown has only a modest impact on hit rates.

In summary, our work is the first to establish disk shutdown as a key mechanism for energy savings in CDNs, paving the way for its future roll-out on the production network.

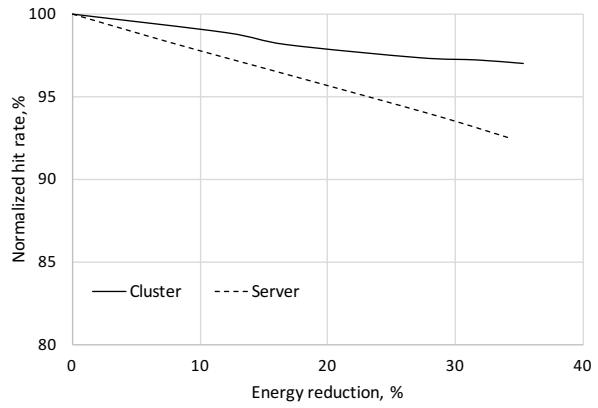

Figure 2: Energy-Performance tradeoff: significant energy reduction is possible by judiciously shutting down disks with only a modest reduction in hit rates.

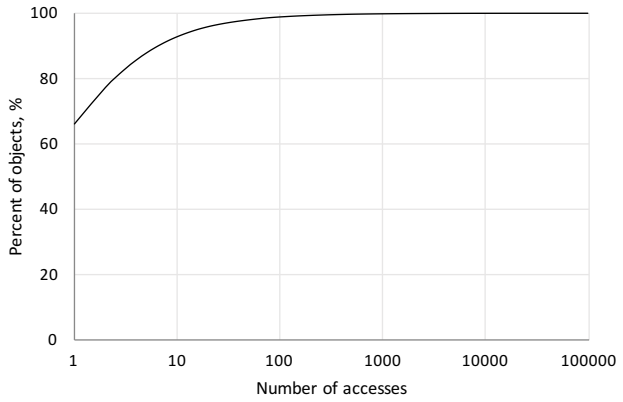

Figure 3: Popularity of content accessed by users on a CDN server.

1.2 Roadmap

The rest of the paper is organized as follows. We review background information, describe the data sets, and provide an overview of the simulator used in our empirical work in Section 2. Then, in Section 3, we describe the typical cache management function of the edge-server, and how we modify it to be energy efficient. The three main components of energy efficient cache management are cache sizing algorithm, disk shutdown algorithm, and content placement and eviction algorithm. We describe and evaluate the performance of these components in Sections 4, 5, and 6 respectively. The evaluation is performed in the context of the hit rate of a single server. In Section 7, we evaluate the different cache management schemes in the context of a cluster of servers and show how the hit rates for an entire cluster differ from that of individual servers. Finally, we review related work in Section 8 before concluding in Section 9.

2. BACKGROUND AND METHODOLOGY

We introduce the basics of CDN architecture in brief. A key component of a CDN server is its *content cache* that stores content in its disks and serves it to users. When a user accesses an *object*[2] such as a web page or a video hosted by the CDN, the CDN's mapping system directs the user to a server that is "nearby" in the network sense (cf. Figure 4). If the content requested by the user is in the server's cache (i.e. a "cache hit"), then the user experiences superior performance in the form of a fast download of the content. However, if the content is not to be found in the cache (i.e. a "cache miss"), then the server requests the content from "peer"' servers that are deployed in the same server cluster. If the content is found in one of the peer server caches, then it is fetched from that peer server and then served to the user. If requested content is found in either the server itself or its peer, we call that a "cluster hit". If the requested content is not found in either the chosen server or its peers, i.e., a cluster miss, then the content is fetched from a remote origin location that is operated by the content provider, and served to the requesting user. The server also caches the content that it fetched from the origin. The origin stores the original copy of all the published content.

When a user's request results in a cluster hit, the response time for serving the object is not as good as a cache hit due to the additional overhead of fetching the content from a peer. The latency of content fetch between peer servers is fast and is of the order of several hundred microseconds, while fetching content from the origin over the WAN is slow and could add hundreds of milliseconds of latency to each download. Cluster hit is therefore still desirable, and an origin fetch is the least desirable.

The two main metrics of performance we consider in this work are the *hit rate*[3] which is percentage content bytes served to users directly from the server cache and the *cluster hit rate* which is the percentage of content bytes served to users from the cache of either the server or one of its peers within the same data center.

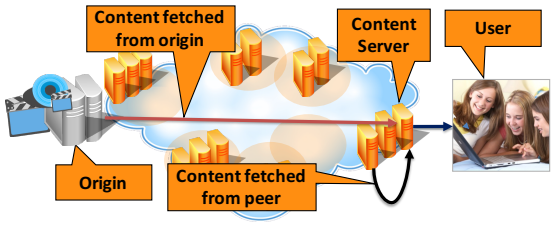

Figure 4: Content is served from the cache of the CDN server, from one of its peers within the same datacenter, or from the content provider's origin.

2.1 Content request traces

The extensive data set used in this paper was collected from one of the server clusters in Akamai's commercially-deployed CDN. The data set contains anonymized access logs for content requested by users. Each log line corresponds to a single request and contains a timestamp, the requested URL (anonymized), object size, and bytes served for the request. The access logs were collected over a period of 9 days from a cluster containing 5 CDN servers. Each server has a disk configuration that is typical for deployed

[2]We use the term *object* for pieces of content that users access using an URL.

[3]We also use the term *server* hit rate when it is necessary to distinguish it from cluster hit rate.

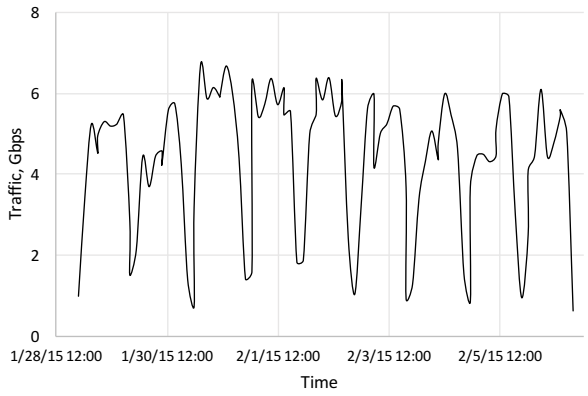

Figure 5: Content traffic served to users from our cluster (in Gbps), averaged hourly.

CDN servers: 8 spinning disks with a capacity of 600GB each with a content cache. We chose a busier cluster for our analysis to provide a conservative bound on the energy savings. *Disk load* is a function[4] of the number and the amount of read and write I/O operations that are being performed on the disk and is expressed as a fraction of its I/O capacity[5]. The average disk load of the servers in our cluster was 39.4% of capacity, this average includes both peak and off-peak periods. The average disk load of a typical CDN server cluster tends to be lower than 20%, and is likely to provide an even greater opportunity for energy savings from disk shutdown. The traffic served in Gbps captured in our data set is shown in Figure 5. We can see that the cluster has a short off-peak duration of about 6 hours each day.

Total requests	3 billion
Total bytes served	429 TB
Total distinct objects	162 million
Total distinct bytes served	67 TB

Table 1: Characteristics of content access data for the Akamai cluster.

Table 1 lists some of the characteristics of the content request traces used in this work. As shown in Figure 3, the content popularity distribution exhibits a "long tail" with nearly 70% of the objects accessed only once. Further, as in Figure 6, 80% of the requests are for 1% of the objects.

2.2 Cache simulator for disk shutdown

We evaluate our algorithms using a custom event-driven simulator. This simulator simulates all the necessary hardware and functional details of an CDN server as shown in Figure 7. The simulator mimics the content placement and eviction algorithms used by the live CDN servers. Every incoming request is placed on one of the disks and existing content is evicted from disks when necessary. In addition to the above fundamental functionality, the simulator implements new architectural components, the cache sizing and

[4]See linux utility iostat for a description of how disk load is computed[1].

[5]We conservatively use 80% of the hardware-rated I/O capacity as our available I/O capacity in our experiments.

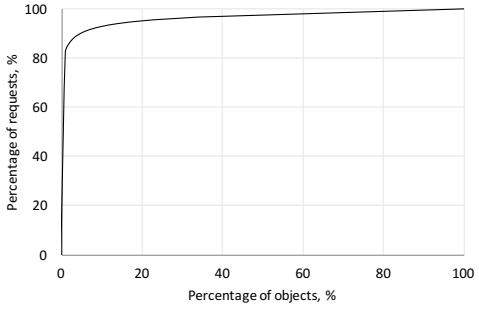

Figure 6: A large fraction of the requests are for a small fraction of the objects.

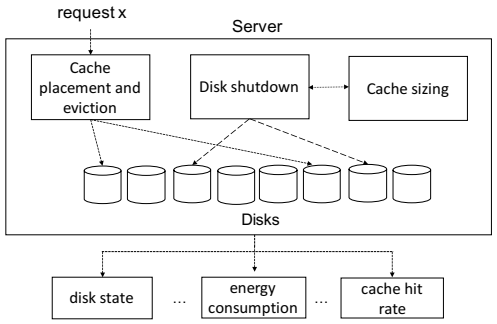

Figure 7: The architectural components of a content cache that uses disk shutdown.

disk shutdown components. These components do not exist presently in the live production CDN servers. The cache sizing component estimates the number of active disks needed to store the working set of content that is currently being accessed by users. The disk shutdown component chooses the precise set of disks to be shutdown or woken up. For each of these components, we implement multiple algorithms described in Table 2 in Section 3. The simulator also carefully tracks the I/O request rates that every disk receives. It plugs these I/O request statistics into the disk power model outlined in Appendix A to compute per-disk, per-server and cluster-wide energy consumption.

From the configuration provided, the simulator constructs the multi-layer cache hierarchy within each server: a number of simulated disks of the given size, the filesystem buffer cache, and the web-server software's hot-object memory cache. In addition to a single-server environment, the simulator is also able to create a cluster of servers. In this mode, it mimics the Internet Cache Protocol (ICP)-based [20] intra-cluster content sharing among all the servers of a cluster, same as the mechanism used in production servers. As its input, it accepts content access logs from live CDN servers in production network. It periodically outputs a rich set of metrics such as traffic volumes, cache hit rates (broken down into the hit-rate seen at every cache hierarchy layer within the server and cluster hit rates when ICP is used). The simulator was validated by replaying 9 days of a production cluster's logs and matching the simulator output metrics with the production cluster's traffic and hit rate statistics.

Cache Sizing	Disk Shutdown	Content Placement & Eviction
Hybrid (Load* & storage based)	Random*	LRU* (Random placement & LRU)
	Fixed	SLRU (Segmented placement & LRU)
	LRU-DS LRU-ordered disk shutdown	

Table 2: Algorithms for energy-efficient cache management. The starred algorithms are simple options that we use as a baseline that we improve upon.

3. CACHE MANAGEMENT SCHEMES

We describe the cache management schemes that we propose, implement, and study in our work.

3.1 A typical algorithm without disk shutdown

We describe a typical cache management scheme that is often used by CDNs that we call NOOFF. NOOFF does not shutdown disks and hence performs only content placement and eviction. Each server has multiple disks. Each requested object that is not already in cache is placed on a randomly-selected disk so that load and space utilization are uniform across all disks. The entire cache space is part of one single Least-Recently-Used (LRU) stack. Eviction occurs when the cache is more than 95% full, at which time ∼5% of the least recently used bytes are evicted from cache.

We implement NOOFF to assess the hit rates that current CDN servers achieve when no energy savings are in place and no disks are shutdown. The hit rates of the algorithms we propose for disk energy reduction must be viewed in relation to the hit rate of NOOFF. The decrease in the hit rate of an energy-efficient algorithm in comparison to NOOFF is the performance penalty that is paid in exchange for the energy savings. In particular, for our energy-efficient algorithms, we compute the *normalized* hit rate which is simply the ratio of the algorithm's hit rate and that of NOOFF.

3.2 Energy-efficient cache management

Besides content placement and eviction, our energy-efficient cache management schemes incorporate two other components (see Table 2). A *cache sizing algorithm* determines the number of active disks required for storing and serving the current working set of content that is being accessed by users. We describe our cache sizing algorithm, Hybrid, in Section 4. Once cache sizing sets a target number of active disks, a *disk shutdown algorithm* chooses the precise set of disks that must be shutdown or woken up. We study two algorithms, Random and Fixed, that we describe in Section 5. Finally, we study content placement and eviction algorithms and compare the baseline scheme of LRU with a sophisticated scheme of segmented placement and LRU (SLRU) as shown in Table 2.

In the rest of the paper, we represent our energy-efficient cache management solutions as a triple, specifying what algorithm was used for cache sizing, disk shutdown, and content placement & eviction. We start out with the simple baseline solution of Load/Random/LRU whose poor hit rate performance we outlined earlier Figure 1. We progressively improve each of the three algorithms to show that Hybrid/Fixed/SLRU outperforms other combinations and provides the excellent energy-performance tradeoffs that we outlined earlier in Figure 2.

3.3 An ideal energy-efficient variant of LRU

LRU is known to be an effective technique for content eviction, variants of which are implemented in most real-world CDNs. We propose a simple extension of LRU to incorporate disk shutdown which we call LRU-ordered disk shut down (LRU-DS). LRU-DS keeps content on disk as per the LRU ordering, i.e., the content on disk $i+1$ is less recently used than the content on disk i for all $1 \leq i < n$. When the cache sizing algorithm requires k disks to be shut down, LRU-DS marks the k lowered numbered disks as being inactive, i.e., the ones with content that was least recently used are marked inactive. Likewise, when k disks need to be woken up, the higher numbered disks are marked active, i.e., the disks that have the more recently accessed content are marked active. If a requested object is present on a disk that is marked active, that request is considered a cache hit. If a requested object is not present or if it is present only on an inactive disk, that request is deemed a cache miss.

Note that LRU-DS is *not* an implementable algorithm as it requires the content on *both* active and inactive disks to be *always* ordered in an LRU fashion. It is not possible to maintain that property since inactive disks cannot be read or written into. However, LRU-DS in combination with a cache sizing algorithm such as Hybrid provides an idealized upper bound on hit rates that our algorithms can attempt to reach. For this reason, we plot the energy-performance tradeoff of Hybrid/LRU-DS as a point of comparison to the tradeoff achieved by our algorithms, though the former may not be achievable by any implementable algorithm.

4. CACHE SIZING ALGORITHMS

A cache sizing algorithm determines how many active disks are required for storing and serving the current working set of content that is being accessed by users. The number of disks that need to be active depends on the current incoming requests for content, i.e., potentially more active disks are required when large volumes of content are being accessed than during periods when access volumes are smaller.

A cache sizing algorithm must consider two different types of resource constraints. First, each disk has an I/O capacity that determines the number of input/output operations (IOPS) that it can sustain. We define *disk load demand* L as the amount of disk IOPS required to serve the incoming content requests, expressed in the units of the maximum IOPS that a single disk can support. For instance, if $L = 6.5$, then the number of IOPS that need to be supported is 6.5 times that of the IOPS of a single disk. Thus, we need to have at least $\lceil L \rceil$ active disks to serve the content requests, since otherwise the load of some active disk will exceed 100%, resulting in very slow response times for retrieving content from disk. A second constraint is the disk storage capacity that dictates how much content you can store in the disk. Shutting down too many disks results in a decrease in the active disk storage capacity, resulting in an increase in cache misses, leading to poor performance. The goal of the cache sizing algorithm is to determine the number of active disks, taking into consideration both disk I/O and storage constraints.

As shown in Figure 8, our cache-sizing algorithm takes as input the disk load demand L that needs to be supported

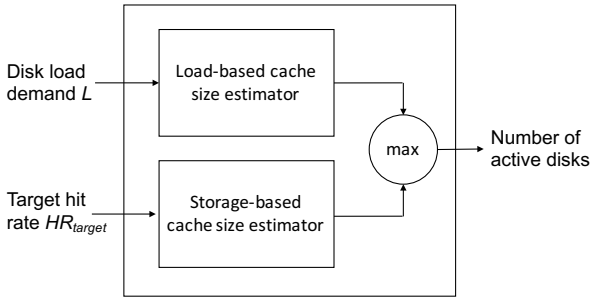

Figure 8: The `Hybrid` cache-sizing algorithm.

by the active disks and a target hit rate HR_{target} that must be sustained and computes the number of disks that must stay active. We call our algorithm "`Hybrid`" as it uses two different estimators, a load-based cache size estimator and a storage-based cache size estimator, and takes the maximum of these two estimates.We describe each estimator in turn.

Load-based cache size estimator. Load-based sizing determines the number of active disks using the disk load demand L that must be supported to serve the incoming content requests. Specifically, it estimates that $\lceil L \rceil$ disks need to be active. In our trace-based simulations, at time t, we record the average disk load x_t% of a CDN server with n (active) disks and estimate the disk load demand L to equal $\frac{x_t}{100} \cdot n$. To provide more stability to the algorithm, the average disk load x_t is computed as the average of the instantaneous disk load values for all the disks during the time interval $[t - \tau_l, t]$. Note that the choice of τ_l presents a recency-stability tradeoff, smaller values of τ_l could lead to more recent but spiky estimates, and larger τ_l could lead to less recent but more stable estimates.

Storage-based cache size estimator. Storage-based cache sizing uses the recent content access sequence to predict how much cache storage is required to achieve a target cache hit rate HR_{target}. It performs the following two steps.

i) Compute the hit rate curve (HRC). The HRC relates cache size with hit rate. Figure 9 shows an example HRC computed for a segment of our CDN content access trace. Given a request sequence $R = \langle r_1, r_2, \cdots, r_n \rangle$, we first compute a metric we call *stack size* s_i for each request r_i, $1 \leq i \leq n$. Stack size is a measure of temporal locality of the requested content and is a variant of the classical notion of stack distance [15]. If the object requested by r_i was never requested before, its stack size s_i is infinite. Otherwise, let j be the largest integer such that $j < i$ and r_j is a request for the same object as r_i, i.e., r_j is the previous request for the same object as r_i. The stack size of r_i is simply the number of *unique* bytes accessed in the request sequence $\langle r_j, \cdots r_i \rangle$. The hit rate for request sequence R and cache size C is computed by assuming that every request $r \in R$ that has stack size less than or equal to C is a cache hit and every request $r' \in R$ that has stack distance greater than C is a cache miss. This computation is repeated for different values of C to obtain the HRC. A keen reader will note that HRC is the hit rate achieved on request sequence R by an idealized cache of size C that is maintained in LRU order.

ii) Given a target hit rate HR_{target}, estimate the number of active disks required to achieve that target using the HRC. The timeline is divided into segments of length of τ_{hrc} hours. At the end of each time segment, a new HRC is computed by setting the request sequence R to be all the requests received in that segment. The choice of τ_{hrc} presents a tradeoff. Larger τ_{hrc} records more history but varies slowly with time and smaller τ_{hrc} records lesser history but is more sensitive to time-varying traffic. The variability in input traffic can be used to decide a suitable value for τ_{hrc}.

Once the HRC is computed, the "ideal" cache size estimate C_{ideal} is the value that corresponds to HR_{target} in the HRC (cf. Figure 9). One key aspect of our algorithms not modeled by the idealized LRU cache is that *multiple* copies of the same object may be stored on a server, albeit the copies must appear on different disks within the server. Such object replication occurs when an object stored in a currently inactive disk is accessed by a user, resulting in a new copy being created on an active disk. When the first disk becomes active again, we may end up with multiple copies of an object in the active disks. The degree of object replication depends on the algorithms being used for disk shutdown. To account for this replication, we compute a replication factor ρ which is simply the actual total bytes currently in cache divided by the total unique bytes. The required cache size C is set to be equal to $\rho \cdot C_{ideal}$. Thus, the number of active disks is $\min\{\lceil C/C_s \rceil, n\}$ disks, where C_s is the storage capacity of a single disk and n is the total number of disks in the server.

Example. Suppose that $HR_{target} = 75\%$ and the HRC is as shown in Figure 9. C_{ideal} is 3000GB requiring 5 active disks of 600 GB each. If $\rho = 1.15$, then $C = 3450$GB, requiring an additional disk to account for the replication. Our storage-based sizing algorithm recommends 6 active disks.

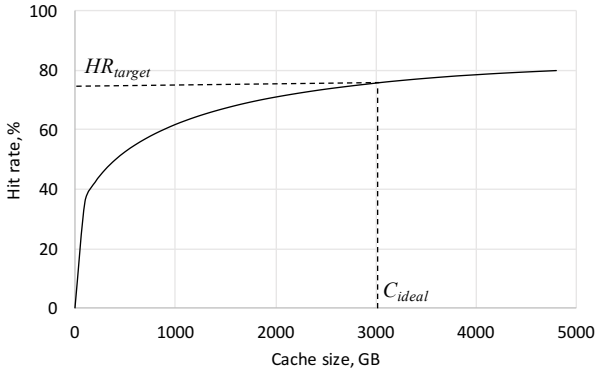

Figure 9: Hit rate curve (HRC) shows the relation between cache size and hit rate.

After the load-based and storage-based cache size estimates have been determined, our `Hybrid` algorithm computes the maximum of the two estimates to satisfy both the load and storage constraints.

4.1 Performance evaluation

We evaluate the performance of our cache sizing algorithm `Hybrid`, with two other algorithms `Load` that uses only a load-based cache size estimator and `Storage` that uses only a storage-based cache size estimator. We use baseline algorithm of `Random` for disk shutdown and `LRU` for content placement & eviction (cf., Table 2), i.e., we pick the disks to be shut down (or woken up) randomly and we place new objects on a random disk and use LRU eviction when the cache

is full. We evaluate all three sizing algorithms and NOOFF using our Akamai content access traces for a CDN server that had $n = 8$ disks, each disk having capacity $C_s = 600GB$. We use $\tau_l = 60s$ in the simulations. We set $\tau_{hrc} = 6\ hrs$ to account for the day/night variations in the traffic and we set HR_{target} to equal the current hit rate of NOOFF.

In Figure 10, we see that Load/Random/LRU has an average hit rate that is nearly 15% lower than NOOFF, since the load-based scheme underestimates the needs for cache space and turns off more disks than it should during off-peak hours when the incoming content request volume is low. The load-based scheme keeps only ∼2 disks (out of the 8) active during off peak hours, leading to a higher rate of eviction. Eviction age, which is the average amount of time that an object spends in cache before being evicted, also tells a similar story. Eviction age for the load-based cache sizing scheme was ∼2.5 times lower than that of NOOFF. Thus,

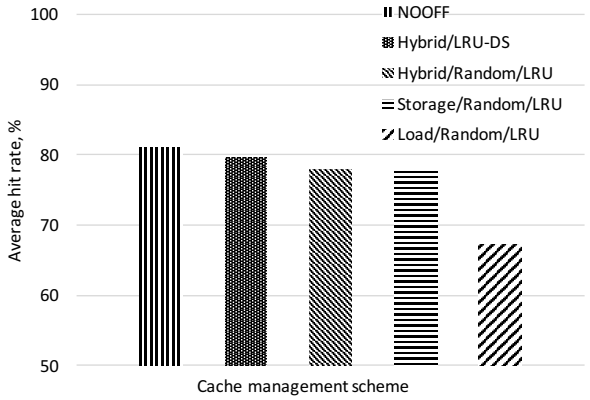

Figure 10: Comparing the hit rate performance of the different cache sizing algorithms with the hit rate of NOOFF that does not shutdown disks.

a load-based scheme alone does not provide good hit rate performance, since cache sizing also depends on the actual content access patterns that cause a given disk load. For instance, if the disk load was caused by very few popular objects in cache, that implies that a smaller cache size with fewer active disks is sufficient. However, if the same disk load was distributed over a large number of less popular objects, more active disks are required to hold the working set. Thus, we must consider the actual content access patterns to determine the number of active disks.

In Figure 10, note that the Hybrid/Random/LRU has a higher cache hit rate when compared to Load/Random/LRU, since Hybrid also accounts for the storage constraints. But, the cache hit rate of the Hybrid/Random/LRU is still ∼3% lower than the ideal target hit rate that was set to equal NOOFF. This discrepancy is in part due to the inefficiency of random disk shutdown that could make frequently accessed objects in the randomly-selected disks inactive. To remove this effect, we compare the performance of Hybrid/LRU-DS with NOOFF. We see that Hybrid/LRU-DS has a hit rate performance that is ∼1.5% less than NOOFF. This small difference indicates that Hybrid is a good cache sizing algorithms that is able to closely match the target hitrate of NOOFF.

What about storage-based *only* cache sizing? The hit rate of storage-based cache sizing, Storage/Random/LRU, is nearly the same as using Hybrid/Random/LRU as shown in Figure 10. However, the former has the drawback of occa-

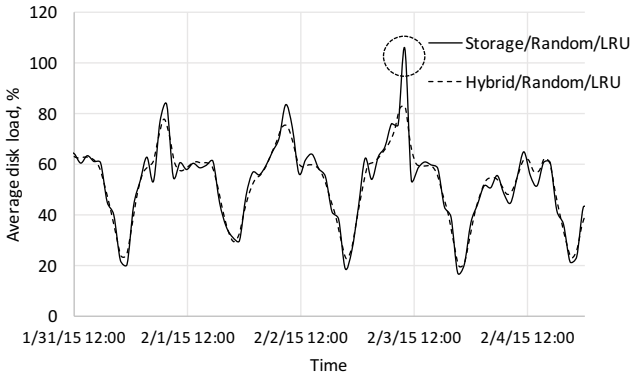

Figure 11: The storage-based cache sizing algorithm occassionally overloads the disks, since it does not factor in disk load. This deficiency can be corrected with a hybrid scheme.

sionally overloading the disks. The disk load occasionally goes over the 100% mark as highlighted in Figure 11. To avoid such overloading, the hybrid cache sizing algorithm proposed above should be used to account for both the load and storage constraints of the server disks.

Concluding Remark. The Hybrid algorithm works the best for cache sizing and we use this algorithm as the default option in all our future experiments where we investigate disk shutdown, content placement, and eviction algorithms.

5. DISK SHUTDOWN ALGORITHMS

Once the cache sizing algorithm outputs a target number of disks that need to be active, the disk shutdown algorithm decides precisely *which* disks should be shutdown (or woken up) to meet that target. Let $dcount_t$ be the number of active disks at time t and suppose that the cache sizing algorithm produces a target $dtarget_t$ of active disks. Then, if $dtarget_t > dcount_t$, the disk shutdown algorithm wakes up $|D| = dtarget_t - dcount_t$ disks, where the set D is the set of all disks that need to be woken up, and if $dtarget_t < dcount_t$, the disk shutdown algorithm shuts down $|D| = dcount_t - dtarget_t$ disks, where the set D is the set of all disks to be shutdown. There are a number of ways in which the set of disks D can be chosen and we review two shutdown algorithms below.

1) Random disk shutdown. Algorithm Random is a simple baseline scheme where the required number of disks to be shut down (resp., woken up) are randomly selected from among the active (resp. inactive) disks.

2) Fixed disk shutdown. The disks are ordered in sequence from 1 through n. Algorithm Fixed shuts down disks in the increasing order starting from 1, i.e., if k disks are to be made inactive the disks 1 to k are shut down. When the disks are made active, they are woken up in the decreasing order, i.e., disk k is woken up, followed by $k-1$, and so on till disk 1.

The objects may get replicated on two or more disks within the same server in both shutdown schemes. If an object that is currently accessed is on an inactive disk, a new copy is

made on an active disk. When both disks are active at a later time, we have more than one copy of the object.

5.1 Performance evaluation

We empirically evaluate the algorithms `Random` and `Fixed` by simulating them on the CDN content request traces. In particular, we use `Hybrid` for cache sizing and the baseline content placement & eviction algorithm of `LRU` with our two shutdown algorithms. We also compare the energy-performance tradeoff of these two algorithms with that of the idealized algorithm `LRU-DS` described in Section 3.3.

To explore different ranges for the energy-performance tradeoff, we use an internal disk shutdown aggressiveness knob. The knob lowers the hitrate target HR_{target} given to the cache sizing algorithm with respect to `NOOFF`, and controls how aggressively that algorithm turns disks off to save energy. In order to plot the tradeoff curve between energy savings and the corresponding hit rates, we run the simulation five times, each time with a higher agressiveness than the previous run. In each run, the hitrate target HR_{target} is lowered in steps of 5%. This creates five scenarios of gradually increasing energy savings.

In Figure 12, we plot the energy-performance tradeoff for algorithms `Random` and `Fixed`. For comparison, we also plot the tradeoff for the idealized algorithm `LRU-DS`. Note that an algorithm that provides a larger hit rate for a given reduction in energy can be deemed to be better. We see that `Fixed` offers a better energy-performance tradeoff when compared to `Random`. For instance, for a 30% energy reduction, `Random` has a 88% normalized hit rate, while `Fixed` has a larger normalized hit rate of 91.5%. (Recall that normalized hit rate is the actual hit rate divided by the hit rate of `NOOFF`.)

The reasons for the superior hit rate performance of `Fixed` in comparison with `Random` are two-fold.

1) As noted in Figure 6, 80% of requests are for a small fraction of 1% of popular objects. In the case of `Fixed`, the popular objects that get accessed throughout the day tend to get replicated on higher disks that are seldom ever shutdown. Thus, popular objects that account for almost all of the user requests are eventually always available on an active higher disk in cache, even when the lower disks are shutdown. However, in the case of `Random`, there is a probability that copies of popular objects are made inactive, since the disks are shutdown randomly, generating cache misses for future requests for them.

2) `Random` also has a greater replication factor ρ than `Fixed`. The reason is that when `Fixed` replicates an object to a higher disk, more copies are not likely to be needed since that copy is likely to be available at all times. However, `Random` could continue to make more copies with some probability, since the existing copies could be made inactive by the random choice of disks for shutdown. Figure 13, shows the higher replication factor for `Random` in comparison with `Fixed`, for all five settings of the disk shutdown aggressiveness knob. Higher replication factor means that fewer unique objects are stored for a given cache size, resulting in a less efficient use of the cache space.

Figure 12 also plots the tradeoff for the `LRU-DS` that shows tradeoff that is better than fixed, for instance, a 30% energy savings can be had with a hit rate of 96%. This suggests that further improvements are possible, motivating our quest for better content placement & eviction algorithms in Section 6.

Concluding Remark. `Fixed` disk shutdown algorithm should be used to select disks that need to be switched on and off since it offers a better energy-performance tradeoff. We use `Fixed` as our default disk shutdown algorithm in our experimental results in the future sections.

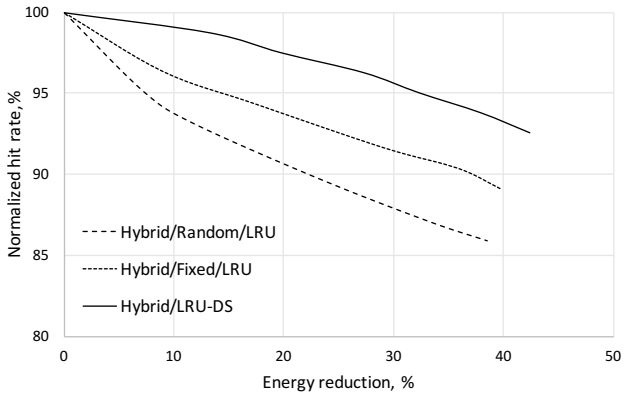

Figure 12: `Fixed` disk shutdown provides a better energy-performance tradeoff than `Random` shutdown.

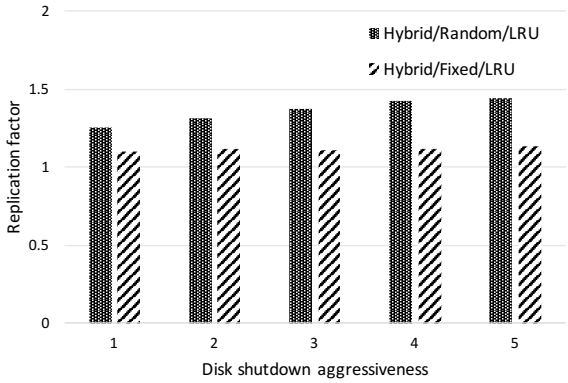

Figure 13: `Fixed` replicates content less than `random` resulting in a more efficient use of the cache space.

6. CONTENT PLACEMENT & EVICTION

Cache management schemes comprise content placement algorithms and eviction algorithms that work together to manage the objects in cache. The cache sizing algorithms and the disk shutdown algorithms described in Sections 4 and 5 control the number of disks, and the actual disks that are shutdown; but they have no control over the placement of objects on those disks. Content placement algorithms choose one among all the active disks in a server, to place the requested object on. Content eviction algorithms select the objects to be evicted from active disks to make space for new ones. In this section, we describe two algorithms below for content placement and eviction. We assume that `Hybrid` sizing and `Fixed` shutdown are used with the two algorithms studied in this section.

1) Random object placement with LRU eviction (`LRU`). `LRU` is a baseline scheme and all the algorithms evaluated in the Section 5 used `LRU`. Each requested object is placed on a randomly selected active disk. All objects are part of

one LRU stack. During eviction, the least recently accessed objects on active disks are evicted until enough cache space is reclaimed.

2) Segmented placement and LRU (`SLRU`). `LRU` is oblivious to the fixed order in which disks are shutdown. `SLRU` avoids the drawbacks of `LRU` by placing more popular objects on higher numbered disks that are less likely to be shutdown. Specifically, `SLRU` divides the cache space into k equally sized segments, where segment 1 contains the least popular objects and segment k contains the most popular objects. Every incoming object is first placed in segment 1. Each subsequent request for that object migrates it one segment up, until it reaches segment k. Objects in each segment are evicted independently using the LRU eviction policy. Hence, we have k LRU stacks, one for each segment. In this work, we assume that that the maximum number of segments in a cache is the number of disks in the server, $k \leq n$. Since the available cache space in the server is reduced or increased at the granularity of a disk, $k > n$ provides no benefit from the perspective of disk energy savings.

6.1 Performance evaluation

We empirically evaluated the hit rate performance of `LRU` and `SLRU` by performing simulations using the CDN content request traces. In particular, as we experiment with `LRU` and `SLRU`, we use `Hybrid` and `Fixed` as the cache sizing and disk shutdown algorithms respectively. To explore different ranges for the energy-performance tradeoff, we used 5 different settings for disk shutdown aggressiveness knob as before. Recall that for higher values of the knob, `Hybrid` cache sizing will lower the target hit rate HR_{target}, resulting in fewer disks being active, saving more energy. As in prior experiments, we set $\tau_l = 60s$, $\tau_{hrc} = 6\ hrs$. Further, we simulate the simplest form of `SLRU` that has $k = 2$ segments, i.e., there is a segment with the 4 higher-ordered disks and a segment with the 4 lower-ordered disks,

From Figure 14, we see that `SLRU` has higher hit rate than `LRU` for any given value of the energy reduction. For instance, for a 30% reduction in energy, `SLRU` has a normalized hit rate of 93.5%, while `LRU` has a normalized hit rate of 91.5%. This is due to the fact that `SLRU` is cognizant of the manner in which algorithm `Fixed` shuts down disks and places popular objects accessed more than once in the higher segment. Since the higher segment resides in the higher half of the disks, it is unlikely to be shutdown.

6.2 Disk power cycles and impact on lifetimes

One of the major impacts on disk lifetime is the number of disk power cycles, in other words the number of times the disk is switched off and switched on. Disks used in current CDN servers are limited to anywhere from 7 to 35 in the number of disk power cycles per day, given that servers are upgraded every 4-5 years. We measure the average number of disk power cycles per day for all 5 simulations and see that both `LRU` and `SLRU` have ∼2-4 disk transitions per day. This is within the bounds of manufacturer specifications and hence disk shutdown is feasible without sacrificing disk lifetimes.

Concluding Remark. The `Hybrid/Fixed/SLRU` scheme provides the best energy-preformance tradeoffs from the standpoint of a single server. In addition, the significant energy savings are obtainable without an impact on disk lifetimes.

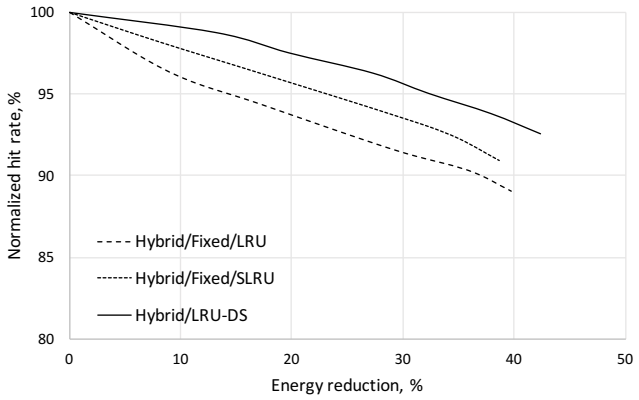

Figure 14: Energy-performance tradeoff for content placement & eviction algorithms

7. UNDERSTANDING CLUSTER HIT RATES

In a typical CDN cluster, popular objects are often stored in more than one server. Algorithms such as consistent hashing [9, 13] that replicate content within a cluster were originally designed to better balance the load within the cluster. Here we study how disk shutdown interacts with load balancing and content replication within a cluster by simulating an entire CDN server cluster consisting of 5 servers for 9 days.

When the requested object is unavailable on the chosen server, due to disk shutdown or otherwise, the server attempts to fetch a copy from a peer server within the same cluster via ICP [20]. If no copy of the object is found within the cluster, the server fetches it from the origin server over the WAN. While server cache hits are the most desirable scenario, cluster hits also often provides adequate performance, since latencies between servers in the same datacenter are quite small. Further, ICP transfers within the same cluster incur no cost for the CDN, since such ICP traffic does not exit the datacenter. The least desirable scenario is when the object experiences a cluster miss, and the object will have to be fetched from a distant origin server over the WAN. Such WAN traffic incurs additional bandwidth costs for the CDN. This cluster hit rate that we study in this section is important both from a performance and cost perspective.

7.1 Performance evaluation

In this paper, thus far, we have looked at (server) hit rates. This section evaluates the energy-performance tradeoff from the standpoint of cluster hit rates. To perform the evaluation, we ran cluster-wide simulations where every server independently shuts down disks using one of the proposed cache management schemes: `Hybrid/Random/LRU`, `Hybrid/Fixed/LRU` and `Hybrid/Fixed/SLRU`. For each cache management scheme, we ran the simulation five times, each time with a higher disk shutdown aggressiveness knob than the previous run. The normalized cluster hit rates for the three cache management schemes are shown in Figure 15. We see that the difference in the normalized cluster hit rates for the three cache management schemes are not as significant as the differences in the (server) hit rates observed for these schemes in Figures 12 and 14. From a cluster hit rate perspective, simple disk shutdown algorithms such as `Hybrid/Random/LRU` and `Hybrid/Fixed/LRU` provide

energy-performance tradeoffs that are comparable to `Hybrid/Fixed/SLRU`. This is due to the following reasons: *1)* since popular objects are replicated across servers, a cluster miss occurs only when all the copies of on objects within the cluster are inaccessible, and *2)* content placement and eviction are performed independently on each server within a cluster. Hence, the probability that *all* the copies of an object will be on inactive disks is now much smaller for all three cache management schemes.

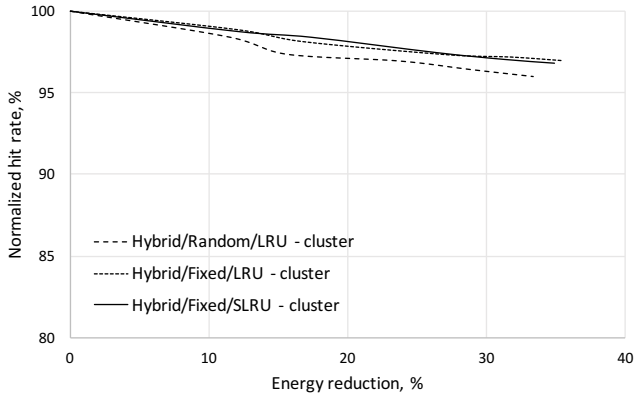

Figure 15: Energy-performance tradeoff of disk shutdown within a cluster

We also compare the normalized cluster hit rate for `Hybrid/Fixed/LRU`) with its normalized (server) hit rates in Figure 2. We see that cluster hit rates have a better energy-performance tradeoff when compared to a single server hit rates. For instance, for an energy reduction of 30%, the normalized cluster hit rate reduces by a mere 3%, in comparison with the normalized server hit rate that reduces by 6.5%. In this case, the *absolute* hit rate reduction for the cluster hit rate and the server hit rate were 2.5% and 5% respectively. Thus, shutting down disks to save energy has a much smaller impact on cluster hit rates than server hit rates.

Concluding Remark. For CDNs primarily concerned with maximizing cluster hit rates to minimize bandwidth costs, our simple scheme `Hybrid/Fixed/LRU` is an attractive option, as it saves significant amounts of energy with only a modest performance loss and a small implementation overhead. However, CDNs interested in maximizing both server and cluster hit rates must invest in more sophisticated algorithms such as `Hybrid/Fixed/SLRU`.

8. RELATED WORK

First we review work that reduce disk energy consumption by the use of multi-speed disks that consume less energy by rotating at lower speeds at periods of low load. The authors in [8] propose the use of multi-speed disks to reduce disk power consumption, instead of shutting down disks. The rotation speed is chosen proportional to the disk load. The work in [4] focusses on reducing the energy consumption of disks in network servers that serve web traffic. The authors in this work, also use multi-speed disks, to conserve energy while maintaining the server's throughput. Hibernator [23], is another work that uses multi-speed disks to reduce energy consumption. This work is targeted at conserving disk energy in disk arrays that serve transaction workloads. Our work differs from these approaches in that we consider typical CDN servers that are commodity hardware with fixed speed disks, and typical CDN workloads. Further, our approach saves additional energy by shutting down disks entirely, an approach that is feasible for CDNs but not for other enterprise networks that store original content.

Other work that attempt to switch disks to a lower power mode include [17] were popular objects are migrated to a subset of disks so that the other disks can be switched to a low-power mode without affecting the performance of the server. The decision to switch disks to low-power modes is based on the incoming request rate. In [3], the size of the front end cache is optimized to reduce the energy consumption of back-end disks in a storage system, while the front end cache consumes power. The cache size is estimated as being proportional to the disk load. However, in our CDN context, we identify that the disk load alone is not a good indicator of the cache space requirement.

Complementary to our work, there is significant work on saving energy in other components of the server such as CPU, including dynamic power scaling and dynamic component deactivation [19, 21, 6, 7].

Prior work also study turning off servers entirely in the context of a data center [5, 18, 11, 12] and in the context of a CDN [14]. However, turning off servers makes network management difficult in a global CDN. If servers are unreachable for extended periods of time, they may miss real-time reporting, software updates and control messages for the duration. This may upend some of the network management guarantees and operational practices of the CDN platform. Therefore, shutting down disks as proposed in this paper, while the servers are still live and serving content, represents an attractive alternative worth exploring.

9. CONCLUSION

CDNs are ubiquitous and carry much of the traffic on the Internet. Reducing the energy consumption of CDNs is an important problem, both from the standpoint of sustainability and OPEX cost reduction. The energy consumed by spinning disks constitute a significant portion of a CDN's energy usage. Our work explores the possibility of reducing the disk energy usage by shutting down disks, a possibility that is particularly well-suited for CDNs since these disks do not store original copies of the content. Our main contribution is developing and evaluating algorithms for cache sizing, disk shutdown, content placement and eviction that allow disks to be shut down without significantly impacting cache hit rates and user-perceived performance. We empirically evaluate the energy-performance tradeoff for our algorithms using extensive request traces from the world's largest CDN. We show that it is feasible to obtain 30% disk energy savings with only a 6.5% decrease in the normalized server hit rate and a mere 3% reduction in the normalized cluster hit rate. This work establishes disk shutdown as a key mechanism for energy savings in CDNs, making it a prime candidate for implementation on the production network.

Acknowledgements. The research was supported in part by NSF grant CNS-1413998. We thank Ming Dong Feng of Akamai for his help with our trace-based simulations. Opinions expressed in this paper are solely that of the authors and not necessarily that of Akamai or UMass or any other organization.

APPENDIX

A. DISK POWER MODEL

In the simulator, we used a power model similar to the 2-parameter disk-power model described in Dempsey [22]. In this model, the energy consumed by the disk is modeled as $E_{total} = E_{idle} + E_{active}$. The values of the two components of the total energy were empirically determined as follows.

Measurement of E_{idle}: We used a typical CDN server that comes equipped with a power-supply unit that has a PMBus interface that allows us to query the server's power consumption at any time. We used the CDN server in the lab for disk power measurement in which four of the disks had no files or directories. A script running on this server queried the PMBus interface every 10 seconds to record a time-series of server power. All the processes other than the bare minimum needed to keep the server up were stopped. Then the four disks were accessed for a duration of time, then left idle for a duration of time, and then spun-down using the SCSI stop command. The time series of server power measurement collected during the time allows us to identify $P_{rotation}$ and $P_{electronics}$. Since the disks in CDN servers never get a chance to shut their electronics down, we use $P_{rotation} + P_{electronics}$ as P_{idle}, which is used to compute E_{idle}. Further, in the manufacturer's detailed data sheet, we located the maximum observed power consumption of the disk model. We call this P_{max}. $P_{max} - P_{idle}$ gives P_{iomax}, the maximum power that the disk's I/O activity can consume. These observations are listed below in Table 3.

Component	Power consumption (W)
P_{iomax}	2.25
$P_{electronics}$	0.75
$P_{rotation}$	3
P_{max}	6

Table 3: Disk power consumption.

Model for E_{active}: E_{active} is the sum of the products of T_{active} and P_{active} for all the I/O operations, where T_{active} is the time consumed by an I/O operation, and P_{active} is the power consumption of that operation. In the typical CDN servers that we studied, the disk I/O pattern caused by serving web traffic has a very narrow range of bytes transferred per request. Over 85% of the I/O requests to the disks have transfer sizes of less than 100KB. Since most transfer sizes are clustered in such a narrow band, we do not create a generalized fine-grain model for P_{active} covering a wide range of I/O sizes, but assume that P_{active} is narrowly clustered around a mean. Due to the low variance property, P_{active} can be said to scale linearly with T_{active}.

We model T_{active} as a function of four disk activity parameters collected at the block layer. To create this model, we collected a large archive of disk statistics using the linux iostat command. The archive contains data points from each machine in this cluster of edge servers for 5 days. Each data point is of the form (read operations/sec, average read size, write operations/sec, average write size, T_{active}), each providing the average of a 30 second observation period. The value of P_{active} for every data point in the archive was estimated as $P_{max} \times T_{active}/\text{observation} - \text{duration}$.

Using linear regression, the 172,800 data points so collected were converted to a power model, which expresses T_{active} and P_{active} as a piecewise linear function of the four disk activity parameters.

Ideally, the power model should also address the energy consumed by disk spin-up phase, which is easily obtained from measurement. But as we saw earlier, the number of power cycles per day per disk is low (approximately 3). Therefore, this component has a minor impact on the total energy consumption, and is excluded from the power model.

B. REFERENCES

[1] *iostat*. http://sebastien.godard.pagesperso-orange.fr/man_iostat.html.

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

[3] Cai, L., and Lu, Y.-H. Power reduction of multiple disks using dynamic cache resizing and speed control. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)* (2006).

[4] Carrera, E. V., Pinheiro, E., and Bianchini, R. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing (ICS)* (2003).

[5] Chase, J., Anderson, D., Thakar, P., Vahdat, A., and Doyle, R. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)* (2001).

[6] Elnozahy, E., Kistler, M., and Rajamony, R. Energy-efficient server clusters. *Power-Aware Computer Systems* (2003), 179–197.

[7] Gandhi, A., Harchol-Balter, M., Das, R., and Lefurgy, C. Optimal power allocation in server farms. In *Proceedings of the 11th International Joint Conference on Measurement and Modeling of Computer Systems* (2009), ACM, pp. 157–168.

[8] Gurumurthi, S., Sivasubramaniam, A., Kandemir, M., and Franke, H. Drpm: Dynamic speed control for power management in server class disks. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA)* (2003).

[9] Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., and Lewin, D. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC)* (1997).

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

[11] Krioukov, A., Mohan, P., Alspaugh, S., Keys, L., Culler, D., and Katz, R. Napsac: Design and implementation of a power-proportional web cluster. In *Proceedings of the ACM SIGCOMM Workshop on Green Networking* (2010).

[12] Lin, M., Wierman, A., Andrew, L., and Thereska, E. Dynamic right-sizing for power-proportional data centers. In *Proceedings of the 30th IEEE International Conference on Computer Communications (INFOCOM)* (2011).

[13] Maggs, B. M., and Sitaraman, R. K. Algorithmic nuggets in content delivery. *ACM SIGCOMM CCR 45*

(2015).

[14] Mathew, V., Sitaraman, R., and Shenoy, P. Energy-aware load balancing in content delivery networks. In *Proceedings of the 31st IEEE International Conference on Computer Communications (INFOCOM)* (2012).

[15] Mattson, R. L., Gecsei, J., Slutz, D. R., and Traiger, I. L. Evaluation techniques for storage hierarchies. *IBM Systems journal 9*, 2 (1970).

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

[17] Pinheiro, E., and Bianchini, R. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th Annual International Conference on Supercomputing (ICS)* (2004).

[18] 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 *Proceedings of the Workshop on Power-aware Computing Systems* (2008).

[19] Weiser, M., Welch, B., Demers, A., and Shenker, S. Scheduling for reduced cpu energy. *Mobile Computing* (1996), 449–471.

[20] Wessels, D., and claffy, k. IETF RFC 2186: Internet Cache Protocol (ICP), version 2, 1997.

[21] Wierman, A., Andrew, L., and Tang, A. Power-aware speed scaling in processor sharing systems. In *Proceedings of the 28th IEEE International Conference on Computer Communications (INFOCOM)* (2009), IEEE, pp. 2007–2015.

[22] Zedlewski, J., Sobti, S., Garg, N., Zheng, F., Krishnamurthy, A., and Wang, R. Modeling hard-disk power consumption. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)* (2003).

[23] Zhu, Q., Chen, Z., Tan, L., Zhou, Y., Keeton, K., and Wilkes, J. Hibernator: Helping disk arrays sleep through the winter. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP)* (2005).