Energy Proportionality for Disk Storage Using Replication

Jinoh Kim and Doron Rotem
Lawrence Berkeley National Laboratory
University of California, Berkeley, CA 94720, USA
{jinohkim,d_rotem}@lbl.gov

ABSTRACT

Saving energy for storage is of major importance as storage devices (and cooling them off) may contribute over 25 percent of the total energy consumed in a datacenter. Recent work introduced the concept of energy proportionality and argued that it is a more relevant metric than just energy saving as it takes into account the tradeoff between energy consumption and performance. In this paper, we present a novel approach, called FREP (Fractional Replication for Energy Proportionality), for energy management in large datacenters. FREP includes a replication strategy and basic functions to enable flexible energy management. Specifically, our method provides performance guarantees by adaptively controlling the power states of a group of disks based on observed and predicted workloads. Our experiments, using a set of real and synthetic traces, show that FREP dramatically reduces energy requirements with a minimal response time penalty.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

Keywords

Energy management, fractional replication, workload adaptation

1. INTRODUCTION

Among the many components in the datacenter, storage is the next largest consumer of energy after servers and cooling systems. It is currently estimated that disk storage systems consume about 25–35 percent of the total power [14]. This percentage of power consumption by disk storage systems will only continue to increase, as data intensive applications demand fast and reliable access to online data resources. This in turn requires the deployment of power hungry faster (high RPM) and larger capacity disks.

Several energy saving techniques for disk-based storage systems have been introduced in the literature [17, 26, 13, 28, 27]. Most of these techniques use the idea of spinning down the disks from their usual high energy consumption mode into a lower energy mode (sleep/standby mode) after they experience a period of inactivity whose length exceeds a certain threshold (*idleness threshold*). The reason for this is that typical disks consume about one tenth of the power in standby mode as compared with their power consumption when spinning. There are several challenges associated with these spin-down techniques when applied to individual disks:

- *Energy and response time penalty*: Disks can only service requests while they are spinning, in case a request arrives when the disk is in sleep mode there is a response time penalty (typically 10–15 seconds) before the request can be serviced. In addition, considerable amount of energy is required to spin up the disk, in some cases this can exceed the energy saved by transitioning the disk to standby mode.
- *Expected length of inactivity periods*: Under many typical workloads found in scientific and other applications, individual disks do not experience long enough periods of inactivity (longer than the idleness threshold) thus limiting the opportunities to save energy.

Achieving *energy proportionality* in datacenters, rather than just energy saving, has been recently getting attention from industry and researchers and proposed as an important design metric [2]. The core principal behind energy proportionality is that computing equipment (storage, servers, networks, etc.) should consume power in proportion to their load level, i.e., a computing component that consumes x watts at full load, should consume $x \cdot \frac{p}{100}$ when running at p-% load.

The energy management approach we consider in this paper promotes energy proportionality and is different from existing approaches, as it is based on handling energy management in a *group* of disks rather than controlling disks individually. We show that our energy management is scalable to large datacenters with thousands of disks and preserves important features of the storage system such as parallelism and fault tolerance. As explained later, our approach exploits *data replication* which is used in many datacenters for reasons such as fault tolerance and load balancing. Popular distributed file systems, such as HDFS (Hadoop Distributed File System) [15] and GFS (Google File System) [12], also automatically replicate data by default. Data replication can help saving energy because when a data item is replicated several times, there is often an opportunity to select a replica found on a currently spinning disk, thus avoiding the costs (spin-up energy) involved in accessing a replica found on a disk which is currently in sleep mode.

Although replication requires additional storage space, it is a relatively cheap resource as it is reported that storage resources in datacenters are often considerably under-utilized and use only a small fraction of the total available capacity (less than 25% according to several studies) [14, 19, 21]. In this paper, we present a novel

EDBT 2011, March 22–24, 2011, Uppsala, Sweden.

replication strategy that achieves energy benefits while maintaining performance and fault tolerance. In particular, our *fractional replication* enables flexible gradual energy management based on workloads which promotes energy proportionality.

Another important challenge is to determine when to transition the disks to lower or higher power states. One typical approach for this is to use a set of thresholds for a load metric. For example, Lang et al. [19] use CPU utilization as their load metric, and assume that disk power states are changed at some threshold points of the load metric. Similarly, PARAID [26] monitors disk utilization and spins disks up/down based on some thresholds. In addition, workload characteristics can be significantly different among datacenters due to the type of applications they run, and also can change over time in the same datacenter.

Our solution, called FREP (Fractional Replication for Energy Proportionality), considers the disks in each disk array as a single unit which can be spun down or up. We refer to spinning up and down of such units as "gear-shift". As we will discuss, our gear-shift mechanism incorporates both predictive and reactive mechanisms, rather than simply relying on static thresholds. To improve performance, we maintain load balancing among the disk arrays that are currently spinning by access request re-direction. For *downshifts* (for energy saving), we make *predictions* based on past historical information. For *upshifts* (for performance guarantees), we utilize *reactive* information in order to respond to any performance degradation quickly. As a basis for these, FREP provides a replication strategy (and its associated functions). We believe that FREP is an important tool for achieving *energy proportionality* in storage systems. Our main contributions include the following:

- We present basic functions and strategies for FREP energy management, including replication strategy, load distribution, and update consistency.
- We present a prediction model based on past historical observations with de Bruijn graphs [20] to enable probabilistic decisions. In addition, we present our constraint-based gear-shift mechanism, by which FREP can shift gears for energy management.
- We provide extensive evaluation results with a diverse set of traces, including two Cello99 traces [5] 6-month apart from each other, two OLTP traces [24], and synthetic traces with different workload characteristics.

The paper is organized as follows. In Section 2, we discuss some related work on energy management in datacenters. In Section 3, we introduce the FREP replication strategy, and a series of functions for I/O service and energy management in such a replicated environment. Our prediction model with de Bruijn graphs is introduced in Section 4, where the FREP gear-shift mechanism based on the prediction model is also described. Our extensive experimental results with several workload sets are presented in Section 5. Conclusions and some directions for future work are presented in Section 6.

2. RELATED WORK

Our work is inspired by Power-aware RAID (PARAID) [26] which was the first work to introduce the concept of gear-shifting based on system load as reconfiguration. It provides a replication strategy, called skewed striping, for disk energy management without service disruption. The main difference is that, PARAID shifts gears within a RAID unit by spinning up/down one or more disks in the array, while we do it across multiple RAID arrays. Another difference is the conditions leading to a gear-shift. PARAID relies on disk utilization to make its gear-shift decisions, while FREP monitors the degree of SLA satisfaction for reconfiguration.

Lang et al. [19] used mirroring for disk energy management. Traditionally, mirroring gives two options — running all the disks (100%) or half of disks (50%). The authors present gradual disk power control combined with load balancing techniques by using a new replication strategy, called chained declustering. Although this new technique provides more flexibility, energy saving is still limited to 50%, since at most 50% of the disks can be spun down. However in real systems the degree of load variation can be more dramatic. For example, in [6], the authors observed a high degree of load variation (over a factor of three) in a commercial web site. Since datacenters usually tend to over provision resources to satisfy peak loads, there may be many opportunities to save energy by factors much greater than 50%.

Similar to our approach, Rabbit [1] provides a skewed data placement strategy for energy proportionality in a MapReduce type cluster. For a dataset with B blocks, the primary replica is spread evenly among the first p nodes (B/p blocks per node). The rest of the nodes hold the remaining $r-1$ replicas where a node with index i ($i > p$) holding B/i blocks. The number of nodes (s) needed to hold the $r-1$ non-primary replicas satisfies $s \geq pe^{r-1}$. Rabbit provides energy proportionality by allowing one-by-one deactivation of nodes along an expansion chain. Each dataset may have its own expansion chain. In power saving mode, a load-balancing algorithm ensures that data accesses are distributed evenly among the active nodes even though their data load is highly skewed. While sharing some similarities with FREP in terms of skewed replication and energy proportionality, there are also some differences between the two techniques. FREP is designed to be used in large datacenters where performance requirements are often dictated by service level agreements (SLA). For that reason, one of the main concerns of FREP is to save energy while satisfying these SLAs. This is done by observing workloads and SLA violations and then adjusting the number of active disks using probabilistic prediction heuristics. Rabbit does not employ such mechanisms. FREP also provides some more implementation details such as analysis of storage requirements for replication and the determination of a feasible range of values for the number of primary nodes. Another important difference is that in Rabbit the number of nodes holding additional replicas grows exponentially and also depends on p (i.e., the number of primary nodes), while FREP is more flexible and does not impose such restrictions.

Although energy management is a crucial problem for datacenters, performance guarantees may be even more important. Hence, energy management needs to be performed within acceptable performance bounds. As pointed out in [3], simple dynamic energy management techniques, such as timeout-based disk spin-down may pay significant performance penalty. This makes administrators of datacenters reluctant to employ such approaches in these cases where system performance is a crucial requirement. To provide energy saving within a controlled performance environment, several research works have taken system SLAs into account. Hibernator [28] employs response time constraint, and considers an optimization problem to minimize energy subject to a given constraint. Similarly, eRAID [21] uses a response time constraint in addition to a system throughput constraint for their energy saving problem. However, we observed that average response times can experience a very high degree of variance, sometimes exceeding three orders of magnitude. Elnozahy et al. [10] employ a "percentile-based response time" to specify the performance constraint for Web servers. In this work, we also employ this to define system SLAs.

Both static and dynamic techniques have been studied for workload-

adaptive energy management. A well known static technique, which we call *2-competitive* algorithm [17], is based on transitioning the disk to sleep mode whenever it experiences a period of inactivity greater than $\frac{\beta}{P_\tau}$ where β is the energy penalty (in Joules) for having to serve a request while the disk is in sleep mode (i.e., spinning the disk down and then spinning it up in order to serve a request) and P_τ is the rate of energy consumption of the disk (in Watts) when spinning. This technique does not attempt to predict the workload and may sometimes lead to unstable performance. Dynamic techniques include employing a multiple set of "experts" [16, 8]. In [16], a set of timeout values are combined to determine the next idleness timeout based on associated weights varied over time, based on the past history. In [8], rather than using an aggregated result, one expert is chosen for energy management, whenever needed, based on the weight values. In updating weight values, this work considered both energy saving and response time latencies. Chung et al. [7] established Markov chains for dynamic energy management, and calculated state transition probabilities based on observations for non-stationary workloads. Energy management actions are determined based on the probabilities. Our prediction model is also probabilistic and refers to past observations for workload adaptability. In fact, the de Bruijn graph used by FREP (Section 4) corresponds to a special type of Markov chain.

3. THE FREP SYSTEM MODEL

We are particularly interested in *read-many, write-rare* environments, as many datacenters use write off-loading [22] or Log Structured Files techniques [11] to batch together write transactions and minimize their effect on power consumption.

FREP manages power states on the basis of a group of disks (e.g., a RAID array). In other words, a group of disks (which form a RAID array) are transitioned together to either *standby* or *active* state in the course of energy management. We assume that the entire disks in a group are either in a standby state (non-spinning), or they are all spinning in the active state.[1]

Formally, we define *node* as an array of disks managed together with respect to energy management. Thus, a node is a collection of disks and there is no disk sharing between nodes. For example, a node can be a RAID-5 array that includes data and parity disks. For scalability, a large storage system can be divided into multiple disjoint *partitions*, each of which consists of its own set of participating nodes. In the rest of this paper, all functions for energy management and analysis refer to a single partition. A partition of a storage system consists of a set of nodes $N = \{N_i\}$. We distinguish between two classes of nodes: Covering Set (*CS*) nodes that are always spinning and contain between them a copy of each data item in the partition, and *non-CS* nodes that can change their power states for energy management purposes. For ease of exposition, we assume n nodes in total, where the first m ($1 \leq m < n$) nodes with the lowest indexes are CS nodes, i.e., $\{N_1, N_2, .., N_m\}$, and the rest are non-CS nodes, i.e., $\{N_{m+1}, N_{m+2}, .., N_n\}$. Table 1 summarizes notations used in this paper.

FREP reconfigures the system based on the current workload, we also call this process "gear-shift", from the entire set of nodes active (the highest gear level) to only the CS nodes active (the lowest gear level). Naturally, the lower the gear level, the greater energy saving can be achieved. Figure 1 illustrates our gear-shift model, from the lowest gear level to the highest gear level. In the figure, filled nodes are active, whereas non-filled nodes are standby. In the lowest gear level, only CS nodes are active (disks associated with CS nodes are spinning) while at the highest gear level all the nodes are active. As explained later, for any node in standby mode, all requests to its data are redirected and serviced (in a balanced fashion) by other active nodes that hold an associated replica. Our replication enables continuous service regardless of energy management with minimal storage requirement, as discussed in the next section.

[1]We interchangeably use "standby/active", "spun-down/spun-up", and "powered-off/powered-on" for disk array state.

Table 1: Notation

Symbol	Description
N_i	A node with index i ($N_i \in N$)
D_i	Original data for N_i
$D_i(\frac{a}{b})$	a/b fraction of D_i
V_i	Storage volume of N_i
$\mathcal{W}_i$	Replica storage for N_i
n	Total number of nodes ($= \|N\|$)
m	Number of CS nodes
$n-m$	Number of non-CS nodes
w	Number of active nodes
$n-w$	Number of standby nodes
C	Storage capacity
ρ	Storage utilization
$L_i(w)$	Load for node i with w active nodes
LIF	Load imbalance factor; $LIF = 0$ means balanced load
$R(p)$	p-% response time in ascending order in a time frame
$p\_threshold$	minimal probability for satisfying down-shift condition

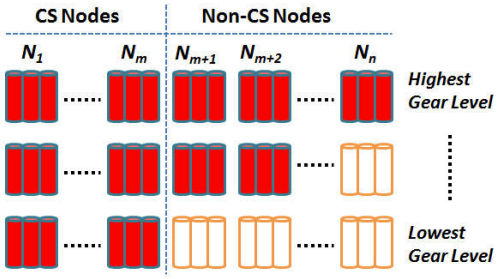


Figure 1: FREP gear-shift model: Gear-shift is based on current workload. At highest gear level, all nodes are active (best performance), at lowest gear level only a fraction of the nodes (CS nodes) are active (greatest energy saving). Filled and empty nodes denote active and standby nodes respectively.

3.1 Replication strategy

The main idea behind FREP is to utilize data replication in order to avoid performance penalties. We will show that our replication scheme can achieve continuous data availability even in energy saving mode. On the other hand, energy management without replication usually causes severe latencies because of the need to spin up disks from standby node, an operation that can take tens of seconds to get back to service (hence unacceptable to datacenters in general). For example, as shown in the evaluation section, a simple energy management technique based on the *2-competitive algorithm* [17] mentioned earlier, may sometimes incur response time penalty causing performance degradation by a factor of 10.

Next we outline the general structure of our replication scheme. We assume that before replication is introduced each node N_i has some original data denoted by D_i. We denote by, $D_i(\frac{a}{b})$ an a/b fraction of D_i. After replication each node will hold some replicated data in addition to its original data as follows: (a) Each CS node gets an equal fraction of the replicated data from each non-CS

	CS nodes		non-CS nodes			
Node	N_1	N_2	N_3	N_4	N_5	N_6
Orig.	D_1	D_2	D_3	D_4	D_5	D_6
Repl.	$D_3(\frac{1}{2})$	$D_3(\frac{1}{2})$				
	$D_4(\frac{1}{2})$	$D_4(\frac{1}{2})$	$D_4(\frac{1}{3})$			
	$D_5(\frac{1}{2})$	$D_5(\frac{1}{2})$	$D_5(\frac{1}{4})$	$D_5(\frac{1}{4})$		
	$D_6(\frac{1}{2})$	$D_6(\frac{1}{2})$	$D_6(\frac{1}{5})$	$D_6(\frac{1}{5})$	$D_6(\frac{1}{5})$	
			$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$	$D_1(\frac{1}{4})$
			$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$	$D_2(\frac{1}{4})$

Figure 2: FREP replication scheme with 6 nodes

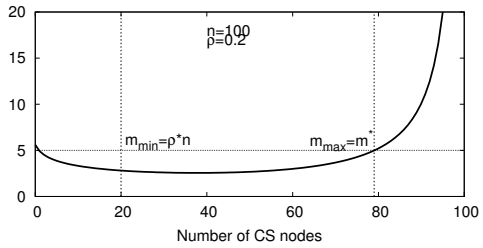


Figure 3: Constraints on the number of CS nodes: Based on the constraints $\rho n \leq m \leq m^*$ (Proposition 3.2), it is possible to compute the min/max number of CS nodes for any configuration.

node; (b) For fault tolerance and load balancing purposes, non-CS nodes maintain replicas of original data associated with CS nodes. Again, each non-CS node gets an equal fraction of the replicated data from each CS node; (c) Non-CS nodes keep replicas of original data (D_i's) from specific other non-CS nodes in a skewed way as explained later. We call cases (a) and (b) *balanced-replication* and (c) *skewed-replication*.

Figure 2 illustrates an example of our replication scheme with 6 nodes (in a partition), two of which are CS nodes. Since there are two CS nodes in this setting, they each keep a disjoint half of non-CS node data. Non-CS nodes also maintain disjoint replicas of CS node data. As there are four such nodes each gets a disjoint quarter of the data. This replication is done strictly for fault tolerance and performance as will be discussed in Section 3.6. In addition, non-CS nodes maintain a part of other non-CS node replicas based on the gear-shift principle. The replication between non-CS nodes helps to distribute the request load in energy saving mode (i.e., a non-highest gear level).

We next explain the skewed-replication, the replication scheme used between non-CS nodes. The original data D_i of a non-CS node N_i ($i > m+1$) are replicated equally to other non-CS nodes with lower indexes, i.e., N_j for $m+1 \leq j < i$, in a random and disjoint fashion. This is done as follows. For each j for ($m+1 \leq j < i$), we randomly select $\frac{1}{i-1}$ of the blocks of D_i (original data of non-CS node N_i) without replacement, and copy them to node N_j.

3.2 Storage requirements

We denote the storage requirement for replicas at each node by $\mathcal{W}_i$. Let V_i be the size in bytes of D_i, i.e., the data volume of N_i. The following equation shows the storage requirement for replicated data on each node.

$$\mathcal{W}_i = \begin{cases} \sum_{k=m+1}^{n} V_k/m & \text{if } 1 \leq i \leq m; \\ \sum_{k=1}^{m} V_k/(n-m) & \text{if } i = n; \\ \mathcal{W}_{i+1} + V_{i+1}/i & \text{otherwise.} \end{cases} \quad (1)$$

In the equation, the first case is for CS nodes, and the second case is for the last non-CS node N_n. These nodes only hold replication data resulting from balanced-replications. The third case is a recursive expression for the storage requirements resulting from the skewed-replication for non-CS nodes described above (except for non-CS node N_n).

We next discuss the total storage requirement for FREP. For simplicity, from now on, we assume that each node holds the same volume of original data, i.e., $\forall N_i\ V_i = V$, so that the total storage for original data is nV. Clearly storage requirements are at least $2nV$ as each item is replicated at least once, the next proposition shows that it is less than $3nV$.

PROPOSITION 3.1. *The total storage requirement $\mathcal{W}$ for FREP is $\mathcal{W} \approx 3nV - mV(1 + \ln \frac{n}{m})$.*

PROOF. Omitted due to space limitations. See our extended paper [18] □

Now, let us consider possible CS/non-CS node configurations from the storage perspective. For simplicity, we assume that all disks in a partition have an equal disk capacity C.

PROPOSITION 3.2. *The number of CS nodes m is bounded by: $\rho n \leq m \leq m^*$, where ρ is storage utilization and m^* is the largest m that satisfies the inequality: $1 + \frac{m}{n-m} + \ln\left(\frac{n}{m+1}\right) \leq \frac{1}{\rho}$.*

PROOF. Omitted due to space limitations. See our extended paper [18] □

Figure 3 shows an example for possible min/max number of CS nodes for a system with $n = 100$ and $\rho = 0.2$. In the figure, the minimum required number of CS nodes is 20 and the maximum is 79. In other words, the number of CS nodes cannot be smaller than 20 or greater than 79 to successfully accommodate the FREP replication. From the perspective of energy management, the maximum energy saving can be obtained up to 80% (i.e., $1 - \frac{20}{100}$) with the minimal CS node setting in this example.

3.3 Load distribution

We next discuss the impact of energy management on load balancing and how we distribute the load to active nodes by taking advantage of the replication. In FREP, active nodes take up load from standby nodes, based on the location of replicas (the details on implementation of request redirection for standby nodes are described in Section 3.5). For example, in Figure 2, nodes N_1–N_5 service requests for N_6 when N_6 is in standby, since they have N_6 replicas. Similarly, when N_5 transitions to standby, N_1–N_4 take up N_5 load evenly. Figure 4 shows an example of load distribution as a function of the number of active nodes (w). For simplicity, we assume that the load generated by accessing the original data in each node is uniform and normalize it to 1 (i.e., load=1). Note that numbers in parentheses in the figure represent the adjusted load achieved by our optimization algorithm to mitigate load imbalance between CS and non-CS nodes. We will discuss it later in this section.

Next we show how to compute the load for each node based on the number of active nodes. Let $L_i(w)$ be the load for node i where the number of active nodes is w ($m \leq w \leq n$). By definition, $L_i(w = n) = 1$ and $\sum_i L_i(w) = n$. We can then compute load as follows:

$$L_i(w) = \begin{cases} 0 & \text{if } i > w; \\ L_i(w+1) + 1/w & \text{if } m < i \leq w; \\ 1 + \frac{\sum_{k=w+1}^{n}(1 - \frac{w-m}{k-1})}{m} & \text{otherwise.} \end{cases} \quad (2)$$

	CS nodes		non-CS nodes			
w	N_1	N_2	N_3	N_4	N_5	N_6
6	1	1	1	1	1	1
5	1.2	1.2	1.2	1.2	1.2	—
4	1.55 (1.5)	1.55 (1.5)	1.45 (1.5)	1.45 (1.5)	—	—
3	2.11 (2)	2.11 (2)	1.78 (2)	—	—	—
2	3	3	—	—	—	—

Figure 4: Example of load distribution: This example shows load before and after (in parenthesis) optimization, as a function of the number of active nodes (w).

In Equation 2, the first case is for standby nodes, and thus the load is necessarily zero. The second case is for active non-CS nodes, and can be defined recursively with the newly introduced load at every node spin-down ($1/w$). Alternatively, it can be defined (non-recursively) as $L_i(w) = 1 + \sum_{k=w+1}^{n} \frac{1}{k-1}$. The last case is for CS nodes. Since each active non-CS node takes $\sum_{k=w+1}^{n} \frac{1}{k-1}$ load from standby nodes, the rest of the load on standby nodes which is equal to $\sum_{k=w+1}^{n}(1 - \frac{w-m}{k-1})$ should be handled by CS nodes. Therefore, each CS node takes $\frac{\sum_{k=w+1}^{n}(1-\frac{w-m}{k-1})}{m}$ in addition to its own load (i.e., 1).

In Figure 4, we see some degree of load imbalance between CS and non-CS nodes, as non-CS nodes transition standby. This is the inherent characteristic of our load distribution algorithm. More accurately, we define a metric *load imbalance factor (LIF)* to express how the load deviates from the ideal balanced state: $LIF = given\_load - balanced\_load$. Thus, $LIF > 0$ means overloaded, while $LIF < 0$ means under-loaded and $LIF = 0$ indicates perfect load balance. $LIF_i(w)$ represents LIF for node i where the number of active nodes is w. Hence, LIF can be expressed as $LIF_i(w) = L_i(w) - n/w$, since n/w is the ideal balanced load. Figure 5 shows LIF for CS nodes varying with the number of active nodes in a system with $n = 100$ and $m = 20$. We can see that load is balanced for the two extremes, i.e., $w = m$ or $w = n$. In between these extremes, we see some degree of load imbalance in the figure.

We now present an optimization technique to reduce LIF for CS nodes, thus bringing the system closer to a balanced state. The basic idea is to have non-CS nodes service CS node load based on replicated data they maintain. To achieve this, it is possible to redirect requests accessing CS node data to any active non-CS nodes probabilistically if the corresponding replicas are kept in such active non-CS nodes. In our optimization, we compute the redirection probability (θ) as follows:

$$\theta = \min\left(1, \frac{\sum_{k=1}^{m} LIF_k(w)}{m} \times \frac{n-m}{w-m}\right) \quad (3)$$

In the equation, $\frac{\sum_{k=1}^{m} LIF_k(w)}{m}$ is simply $LIF_1(w)$, since each CS node has the same LIF.

The intuition behind this is to redirect requests for any CS node data more aggressively if either the LIF for CS node is greater or the number of active non-CS nodes are smaller (or both). For any request accessing CS node data, we can probabilistically redirect the request based on the computed θ but only if any of active non-CS nodes keeps the replica. For example, suppose $\theta = 0.5$ and active non-CS nodes keep $1/2$ of CS node replicas. In that case, 50% requests to CS node data can be redirected to non-CS nodes (probabilistically), but 50% of them can actually be serviced by active non-CS nodes. Consequently, it can reduce CS node load by 0.25. Figure 5 shows LIF for CS nodes with and without the optimization in a system with $n = 100$ and $m = 20$. We can see that our optimization can significantly mitigate load imbalance compared to the basic one. Except for states with fairly small number of active non-CS nodes (less than 50 in the graph), load is almost balanced for active nodes.

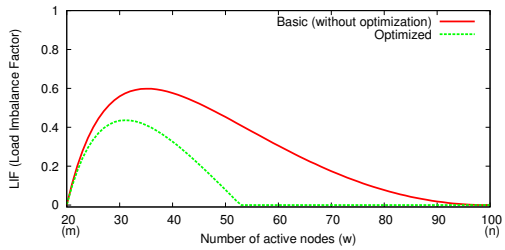


Figure 5: Load imbalance factor (LIF) for CS node ($LIF = given\_load - balanced\_load$): With the optimization, LIF is significantly reduced.

3.4 Update consistency

Since our focus is more on *read-dominant* environments, FREP provides simplified functionality for new writes and update consistency. The main principle for writes is that we redirect write requests for data on standby non-CS nodes to the replica held on CS nodes, and perform synchronization later in the *reorganization phase* (discussed below). Hence, the basic idea is similar to write off-loading [22], in which all writes to powered-off disks are redirected to other disks temporarily. This can slightly increase the load on CS nodes, but we assume write requests occur infrequently.

More specifically, for an update request to a block of D_i residing on N_i (the node holding the updated block), if all replicas of D_i are on active nodes, all of them are updated. Otherwise (i.e., if one or more of the nodes holding replicas of D_i is standby), FREP off-loads the updated block to the CS node holding its replica, and the updated block in D_i is marked as *stale* (in the meta data) to prevent subsequent accesses. For new writes on D_i, we distinguish between two cases (a) a new write to a CS node and (b) a non-CS node. In case (a), we write it on N_i. We then randomly select a non-CS node N_j. If N_j is active, we write the new block on it; otherwise, we mark the appropriate block on N_j as stale for later synchronization. In case (b), if N_i is active, the new block is added to D_i and replicated based on our replication scheme (described in Section 3.1); otherwise, one of CS nodes is randomly selected, the new block is off-loaded to it, and the block on N_i is then marked *stale* for later synchronization. The off-loaded information can also be duplicated to another separate place (e.g., a centralized cache) to handle unexpected CS node failures, and cleared whenever reorganization is done.

Whenever the gear goes up to the highest level (i.e., performance mode), a background *reorganization* process is scheduled. That is, any subsequent down-shift condition enables reorganization to be executed. During the reorganization, down-shift is postponed. In this phase, all stale blocks in non-CS nodes are synchronized with the corresponding copies of CS nodes. New blocks that are off-loaded in CS nodes are also copied to non-CS nodes, and replication takes place using our *skewed-replication* scheme, as discussed in Section 3.1. After completing reorganization, the gear is then shifted down if the down-shift condition is still in effect. In case an up-shift condition arises in the course of reorganization, it will take priority over it, and the system will resume reorganization when a down-shift event occurs.

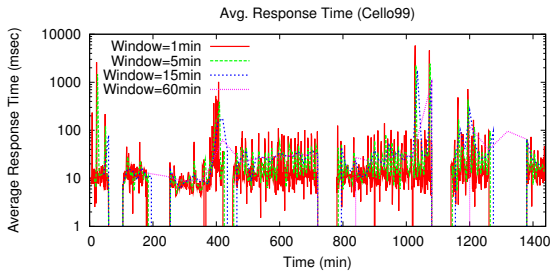


Figure 6: Variation of Average Response Time (Cello99)

3.5 Request redirection

Requests can be redirected to one of replica nodes, (1) when the node with the original block is now in the standby, (2) when the original block is updated (hence stale), or (3) when we need load optimization as discussed in Section 3.3. For (3), we use a probabilistic approach as discussed. For (1) and (2), we maintain a mapping table that holds meta data for replica locations and data status (e.g., stale). For each request, FREP tests redirection first, and based on the test result, the request is dispatched to a node. More information about this redirection procedure can be found in our extended paper [18].

3.6 Fault tolerance

The storage system in many datacenters consists of RAID arrays. We therefore assume each node is a RAID group of disks, and thus, most disk failures can be handled by the RAID fault tolerance functions. For failures that cannot be dealt with by RAID functions, FREP immediately stops energy management functions, and standby nodes are spun up. If a failed node is a non-CS node, all requests to it are redirected to CS nodes (similar to the case when it is in standby). In the case of CS node failure, non-CS nodes take over all requests to that CS node, since non-CS nodes maintain CS node replicas (as seen in Figure 2). In that case, load for non-CS nodes is $1 + \frac{1}{(n-m)}$, while active CS nodes have load 1. As mentioned, off-loaded information can be cached to a separate location, and the information can be used for synchronization in case of CS node failure.

4. GEAR-SHIFT MECHANISM

FREP shifts gears for energy management, and as a result, its energy benefits and response time performance critically rely on the gear-switch mechanism. In this section, we present our gear-switch mechanism.

4.1 Service constraints

Our main goal in designing gear-switching mechanism is to maximize energy benefits, such that the system SLA is met. There are various SLA metrics in the literature. Average response time is one typical example [28, 21]. However, in real life trace logs we observed a high degree of variation for this metric (Cello99). As shown in Figure 6, variation of over 3 orders of magnitude is possible. The figure plots average response time observed with different window sizes from 1 minute to 1 hour. Even with a large time window, we can still see drastic changes across time. Such a high degree of variation makes it difficult to use this metric in defining system SLAs For this reason, we alternatively consider *percentile* for system SLAs.

Percentile is widely employed in definitions of system SLAs. For example, *availability* requirement (for data, node, etc) is often defined with x-nines, where x-nines refers to the number of '9' in percentile value. Thus, 5-nines refers to 99.999% availability. In this paper, we use percentile of request response time for the system SLA. For instance, we can specify a service constraint: "*99% of requests should be serviced within 500 msec.*" This is a safer metric than average response time, particularly for such environments with a high degree of variations, in which only smaller number of delayed completions can critically affect the aggregated result. Formally, a system SLA is defined:

$$SLA : R(p) \leq \tau \quad (4)$$

Here, p is a percentile, $R(p)$ is p-% response time in ascending order (observed in a given time interval), and τ is the response time constraint. Thus, we need to provide values for p and τ to specify an SLA. With the specified SLA, FREP checks whether p-% requests lie within τ. For this, FREP uses a predictive approach, and makes gear-shift decisions based on prediction. Before discussing our prediction model, we first discuss workload diversity with real and synthetic traces, and then continue to discuss our prediction model based on de Bruijn graphs.

4.2 Sensitivity to Workload Characteristics

Workload characteristics can be widely different for systems or even in a single system over time. Figure 7 compares request arrival rates from two traces, a Cello99 trace and an OLTP trace, each of which is from HP Storage Research Lab [5] and University of Massachusetts [24], respectively. We call these traces "Cello99" and "umass". The details of workload traces used in this paper are described in Section 5. As can be seen in the figure, the traces have fairly distinctive patterns. The Cello99 trace looks highly bursty going over to 1,000 requests in a second, while the umass trace is relatively uniform moving up and down between arrival rate 0 to 200 per second over time. Another recent observation also reported significant I/O workload differences for server systems, including mail, web, and file servers [25].

Figure 8 shows disk idle time distributions for the two traces described above and three new synthetic traces, each of which has an exponential distribution with μ=6ms, μ=20ms, and μ=50ms, respectively, for inter-arrival time. The first two synthetic traces were characterized in Hibernator [28] for OLTP workload (μ=6ms) and Cello99 workload (μ=20ms). We additionally create the third synthetic trace to represent a relatively light workload. Note that inter-arrival time for our cello trace is 29 ms, and for the umass trace is 8 ms. We assume that the disk is *idle* if it does not perform any action over 10 seconds. Here, T_{BE} refers to *break-even time*, a time interval where the energy consumption in idle mode is equal to the sum of energy for disk spin-down, standby, and spin-up, and is computed based on our disk model used in Section 5.

In the figure, the Cello99 trace shows a heavy tail, indicating some devices experienced very long idle times. The umass trace looks similar to Cello99, but shows a slightly shorter tail. The synthetic traces show relatively short lengths of idle times and non-heavy tails compared to the real-world traces, and provide lesser opportunities for energy savings for the 2-competitive algorithm (which is based on a fixed idleness threshold). One interesting observation is that the synthetic trace with μ=50ms shows smaller opportunities than the real traces despite heavier arrival rates. These observations suggest that in order to get better results, we need energy savings strategies that consider workload characteristics.

Variations in workload characteristics indicates that static techniques should be ruled out. For example, if we simply apply the 2-competitive algorithm for the synthetic traces (particulary for the first two synthetic traces), there will be severe performance and en-

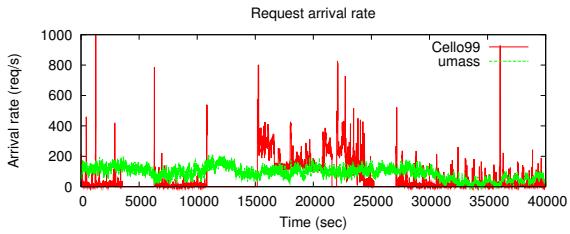


Figure 7: Request arrival rates for real traces

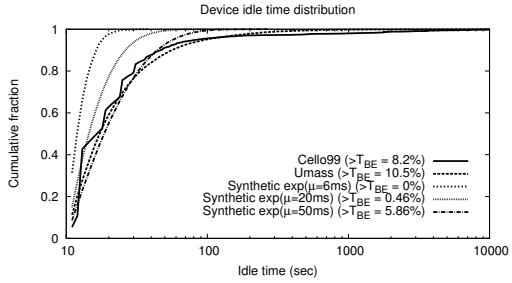


Figure 8: Disk idle time distribution

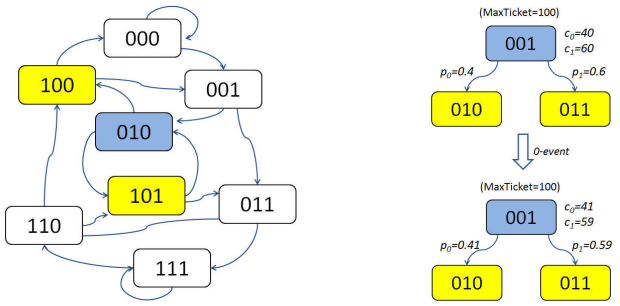
ergy penalties. Although the static parameters can be tuned for each workload, it is usually difficult to capture workload characteristics *a priori*. A *predictive* approach can be an option for dynamic adaptation to different workload characteristics. In this paper, we consider a predictive approach based on *probability* that is constructed based on past observations, as discussed next.

4.3 Prediction with de Bruijn graphs

We use "state-based" predictors to probabilistically predict future states by using a de Bruijn graph. In a de Bruijn graph with k bits, there exist 2^k states represented in binary, each of which has 2 incoming edges and 2 outgoing edges [20]. For each state, one of two events can take place , *0 or 1*, based on how the current state transitions to the next state along with the corresponding outgoing edge. With this property, each state tells us what has happened over k time frames. For example, if the current state is '100', there was *1*-event before 2 time frames, followed by 2 consecutive *0*-events. Thus, we can be aware that the most recent event was *0*-event. Figure 9(a) illustrates a 3-bit de Bruijn graph with 8 possible states. In the figure, the current state is '010', and the next possible state is either '100' or '101' according to the next event.

On a de Bruijn graph, we construct edge probabilities based on historical information. To achieve this, each node has two counters, c_0 for the number of *0*-events and c_1 for the number of *1*-events. These counters are incremented based on the corresponding event. Figure 9(b) shows a snapshot of the counters for state '001'. Based on the counter values, edge probabilities are computed, as shown in the figure.

By configuring max number of tickets (*MaxTicket*), it is possible to limit the window length we wish to monitor. The window length should be equivalent to *time frame length* $\times$ *MaxTicket*, where time frame length is an observation interval, in which a single event (zero or one) is generated based on the observation. If the total number of tickets is smaller than *MaxTicket*, the counters are simply incremented according to the event. After the total number of tickets reaches *MaxTicket*, however, one ticket in a counter is transferred to the other counter, instead of incrementing



(a) A de Bruijn graph

(b) Counters in a de Bruijn node

Figure 9: A de Bruijn graph and ticket transfer

the counter. Thus, there will be no change in the total number of tickets (=*MaxTicket*) after this happens. In Figure 9(b), an *0*-event occurs, and we see that a ticket in c_1 is transferred to c_0 and the probabilities are recomputed accordingly. Note that the values of the counters can be zero or a positive integer.

4.4 Gear-shift algorithm

Now, we present the FREP gear-shift algorithm. For down-shift, FREP relies on the above prediction model, while it uses a reactive model for up-shift. We first discuss how FREP determines down-shift, and then discuss the case of up-shift.

FREP maintains a de Bruijn graph for each partition. To construct edge probabilities, we assume a *0*-event happened if the measured information meets the service constraint (e.g., 99% of requests are less than 500 ms) in the time frame. On the other hand, if the percentage of violations is greater than the given percentile, we assume *1*-event happened. That is, we give zero if $R(p) \leq \tau$; otherwise give one for each time frame.

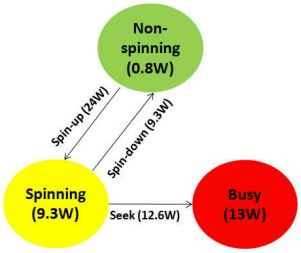
To determine down-shift, we calculate the probability of consecutive k zeros (i.e., the probability that the service constraint will be met for the following k time frames) at the end of each time frame. If the computed probability is greater than a certain threshold (or *p_threshold*), we consider that the down-shift condition is satisfied, and the node with the highest index among active non-CS nodes will be sent to standby mode. Naturally, no down-shift test is performed at the lowest gear level.

For clarity, we formally describe this procedure as follows. Let S_i be state i in the graph configured with k bits. Hence, there are max 2^k states, and we assume that i is the state number; for example S_0 indicates state '000', while S_7 is for state '111'. Let $P_{i,0}$ be the probability of *0*-edge at S_i, and similarly $P_{i,1}$ be the probability of *1*-edge at S_i. If we suppose the current state is S_a, the probability for consecutive k-zeros means the probability of transition from S_a to S_0, right after k time frames. Then, the probability $\mathcal{P}$ is defined:

$$\mathcal{P} = \prod_{i=0}^{k-1} P_{a \ll i,0} \tag{5}$$

where '$\ll$' is the bitwise shift-left operator. Based on the computed probability, we decides down-shift. The gear goes down if $\mathcal{P} \geq p\_threshold$; otherwise, nothing takes place.

We determine k (i.e., the number of bits in the graph) from the break-even time (T_{BE}). For a given time frame size T_W, we set $k = \lceil T_{BE}/T_W \rceil$. The intuition behind this is that there is no energy penalty if the following k consecutive time frames satisfy the service constraint. Prediction can sometimes fail due to rea-



Description	Value
Spinning power	9.3 W
Busy power	13 W
Non-spinning power	0.8 W
Spin-up power	24 W
Spin-down power	9.3 W
Spin-up time	15 sec
Spin-down time	10 sec

Figure 10: Disk power model (from Seagate Barracuda 7200)

Table 2: Synthetic traces

Trace	# req.	Arrival dist.	Disk access dist.	# block dist.
S11	1 M	exp(6)	uniform	exp(20)
S12	1 M	exp(20)	uniform	exp(20)
S13	1 M	exp(50)	uniform	exp(20)
S21	1 M	exp(6)	Zipf(α=1.0)	exp(20)
S22	1 M	exp(20)	Zipf(α=1.0)	exp(20)
S23	1 M	exp(50)	Zipf(α=1.0)	exp(20)
S31	1 M	exp(6)	Zipf(α=1.8)	exp(20)
S32	1 M	exp(20)	Zipf(α=1.8)	exp(20)
S33	1 M	exp(50)	Zipf(α=1.8)	exp(20)

sons such as a sudden change in workload characteristic. In such a case, we give a *penalty*, and edge probabilities are totally recomputed. With a penalty, all *0*-counters in the graph are dropped by half of their original values, and the corresponding number of tickets are transferred to the associated *1*-counters. This decreases *0*-probabilities, resulting in more conservative down-shift decisions thereafter. For T_W and *p_threshold*, we explore the impact of those parameters in Section 5.5.

Making up-shift decisions relies on *reactive* information. In our mechanism, FREP immediately up-shifts the gear whenever it sees l consecutive misses against the service constraint, so as to prevent undesired performance degradation. In this paper, we used $l = 2$ by default to prevent any impulsive up-shift decision due to a temporal degradation. However, l may have a certain correlation with time frame size T_W. Investigation of this would be interesting and planned for future work.

5. EVALUATION OF FREP

5.1 Experimental Setup

For evaluation, we augmented Disksim [9], which is widely used for studying storage systems, with energy metrics. We considered Seagate Cheetah 15K.5 enterprise disks.[2] For this disk model, however, some power information, such as standby power and spin up/down power, is missing in the associated documents. For this reason, we alternatively chose power parameters from Seagate Barracuda specification.[3] Since the main purpose of our experiments here is to see applicability of FREP in terms of both performance and power, we believe that comparing FREP with existing techniques with identical power parameters is a fair comparison. The power model we used in this paper is shown in Figure 10.

We assumed a datacenter environment with 120 disks. Although our model has no dependency on any specific RAID organization, we used RAID-5 structure as a unit of energy management where a RAID-5 array is a node in our terminology. Each array has 4 data disks and 1 parity disk. Thus, there are 24 RAID arrays in the system (i.e., 24 nodes). We divided the system into 4 partitions, each of which consists of 2 CS nodes and 4 non-CS nodes. However, we also conducted experiments with different partition configurations in order to examine configuration effects.

We used multiple traces, including real and synthetic workloads: 2 Cello99 traces from HP Storage Research Lab [5]: a 1-day trace on May 1st (labeled "cello-1") and a 3-day trace between November 15th–17th (labeled "cello-2"); 2 financial traces provided by University of Massachusetts [24], these are labeled "umass-1" and "umass-2", respectively. The average inter-arrival time for Cello99 traces is 29.6 ms and 20.9 ms for cello-1 and cello-2, while it is 8.2 ms and 11.1 ms for umass-1 and umass-2, respectively.

To map the Cello99 traces to our configuration with 120 disks, we assumed that each disk in the traces is mapped into a single RAID array. Thus, 24 disks in the traces are mapped to 120 disks in 24 RAID arrays. The umass traces have no disk information. To use these traces in our experiments, we assumed that each application runs with a dedicated RAID array. Similar to the Cello99 traces, umass requests are mapped to RAID addresses.

We additionally created 9 synthetic workloads with different characteristics, as summarized in Table 2. In the table, "exp(μ)" stands for exponential distribution with mean μ. For example, exp(6) for arrival distribution represents an exponential distribution with μ=6ms. The exp(6) and exp(20) arrival rates were used in Hibernator [28], and we added one more arrival distribution with exp(50) to consider an environment with a relative light load . It is observed in the literature that Internet data access patterns are related to a Zipf distribution with skewness α=1.0 [4]. In addition, the authors in [23] observed more heavily skewed accesses with α=1.8. We modeled synthetic traces based on those observations, in addition to uniform access distribution.

We evaluated 4 different systems: NPS (No Power Saving) is a base system for comparison without energy management; FTH (Fixed Threshold) is a system employing a fixed idleness threshold based on the 2-competitive algorithm; PARAID(k, l) is a PARAID configuration with a total of l disks with gear shifting down to k (thus, $l-k$ disks can go standby); and FREP(n, m) is an FREP configuration with a total of n nodes and m CS nodes in a partition. We set up two PARAID systems (PARAID(5,3) and PARAID(5,2)) and multiple FREP systems with different configurations, but mainly discuss the FREP(6,2) configuration. Thus, there are 4 partitions for 24 nodes for FREP(6,2) setting. By definition, PARAID systems can spin down disks with up to 40% (for PARAID(5,3)) and 60% (for PARAID(5,2)) of the total disks spun down, while FREP(6,2) can spin down a maximum of 67% disks. We observed that PARAID(5,1), that allows spinning down of up to 80% disks, is severely degraded in terms of response time performance. For example, PARAID(5,1) increased average response time by a factor of 5 as compared to NPS with cello-1 trace. We thus excluded this configuration from our experiments.

As discussed in Section 4, FREP maintains de Bruijn graphs for gear-switch decisions. For the graphs, we used 5-second time frame (i.e., $T_{TF} = 5s$). Since we configure the number of bits based on the break-even time (i.e., number of bits = $\lceil T_{BE}/T_{TF} \rceil$), the graph is configured with 11 bits, since $T_{BE} = 54s$ based on the power model (Figure 10). In addition, we conservatively chose *p_threshold=0.9* for down-shift, and set consecutive miss counter *l=2* for up-shift to consider temporal performance degradation. We discuss the effects of time frame size and *p_threshold* in Section 5.5.

[2] http://www.seagate.com/www/en-us/products/enterprise-hard-drives/cheetah-15k/

[3] http://www.seagate.com/support/disc/manuals/sata/100402371a.pdf

5.2 Relaxed service constraints

As discussed, FREP makes gear-shift decisions under consideration of service constraints. Here, we first consider *relaxed constraints*. To give a relaxed constraint, we reverse-engineered NPS logs, then we set service constraints with numbers greater than double of the numbers in the logs. For example, 99% response time in the NPS results with cello-1 is 1,485 ms, and we used a number greater than 2×1485 ms for the 99% constraint for relaxation. Thus, FREP focuses more on energy saving in this case.

Figure 11 compares those two metrics with the real traces. Overall, FTH is fairly sensitive to workloads with respect to energy saving, and it shows very poor response times due to spin-up delays. PARAID and FREP consistently save energy for different workloads. However, PARAID shows a higher degree of variation in response time, while FREP shows fairly stable results by adaptively shifting gears to the given workload. Interestingly, we can see that FREP yields better response time than NPS with the cello-2 trace. This can be explained by the effect of request redirection. Bursty requests to a single node can be smoothed out by redirecting them to multiple nodes when FREP is operating in energy saving mode. For the umass traces, FTH shows an average response time of 77–173ms, while the others show quite negligible time ($<$ 4ms) for these non-bursty workloads.

Figure 12 shows the experimental results with a set of synthetic workloads. With these traces, FTH could make 0%–20% energy saving, but the mean response time is very poor. Although not shown in the figure, mean response time in the worst case was 389 ms for *S33*, which is 100 times greater than NPS. The replication-based solutions (i.e., PARAID and FREP) consistently yield significant energy saving with little performance loss.

5.3 Tight service constraints

We next examine FREP under tight service constraints. We assume the following three types of constraints, based on our constraint model: $R(p) \leq \tau$, where p is a percentile and τ is a response time constraint.

- *C1*: $(p = 90\%) \wedge (\tau =$ 90% NPS response time);
- *C2*: $(p = 95\%) \wedge (\tau =$ 95% NPS response time);
- *C3*: $(p = 99\%) \wedge (\tau =$ 99% NPS response time).

For comparison, we call relaxed constraint *C0*.

Figure 13 shows the experimental results under tight constraints. In the figure, P(n,m) $\equiv$ PARAID(n,m). We can see that significant percentages of violations occurred for PARAID in order to obtain energy benefits for all three constraints. PARAID(5,2) shows heavy violations between 37%–57%. Even in the case of PARAID(5,3), the violations were over 20% for those real-world traces. FTH also shows some degree of violations greater than the constraints but smaller than PARAID. In contrast, FREP largely satisfies the given constraints. With tight constraints, FREP operates energy management more conservatively. Nonetheless, we can see that FREP still achieves non-trivial energy savings. FREP yields 15%–60% energy saving for the cello traces and 3%–15% for the umass traces. The reason why FREP achieves relatively greater energy saving with the cello traces is that they have greater NPS response times as compared with the umass traces. Thus, the service constraints for the umass workloads are more restrictive and hence difficult to meet than these of the cello workloads, leading to the results shown in the figure. For example, the 99% of NPS response time for the one-day cello trace is 1.4s, while for the umass-1 it is only 6ms.

Figure 14 shows the results with the synthetic traces for *C3* constraints. FREP successfully maintains violation rates to less than the 1% constraint. However, it yields energy saving only with skewed traces (i.e., *S31* and *S32*). We found that skewed access patterns provide more opportunities to save energy as compared with uniform access patterns. The reason for that is that with more skewness a smaller number of disks (and therefore only some partitions) become more heavily utilized allowing the remaining partitions to shift to lower gear thus save more energy. Additionally, as explained before, the reason why *S31* had the greatest power saving compared to the others is because it had the greatest NPS response time and hence the least restrictive constraint. For example, 99% NPS response times we observed were *S31*=15ms, *S32*=10ms, *S33*=8ms.

Summarizing, although energy benefits achieved by FREP in this case may be reduced, it still provides strong performance guarantees. We observed only a single case that slightly exceeds the given constraint (5.4% for 5% requirement for cello-1 *C2*) out of 21 experimental cases.

5.4 Impact of FREP partition configuration

In this experiment, we configured 4 additional FREP settings with different numbers of nodes and CS nodes for each partition: FREP(4,1), FREP(4,2), FREP(6,1), and FREP(6,2).

Figure 15 shows the experimental results with cello-1 under different partition settings. We can see that FREP saves energy from 40% to 76% compared to NPS. In terms of performance penalty, defined as the ratio of the mean response time to the NPS's, it is only 14% at the worst case. Interestingly, 2 out of 5 cases showed negative penalty, which implies that FREP showed better response time than NPS. This is a result of request redirection as discussed in Section 5.2.

5.5 Impact of time frame size and p_threshold

The time frame size (T_W) determines the number of bits in the de Bruijn graph ($= \lceil T_{BE}/T_W \rceil$), while *p_threshold* (denoted by ψ) enables probabilistic decisions for down-shifting on the graph. A greater T_W requires less space complexity for graph representation with smaller number of bits. As mentioned, we used $T_W = 5s$, and thus, the number of bits was 11. Additionally, we set up a graph with $T_W = 30s$ resulting in three bits for the graph. With those time frame sizes, we conducted experiments along different *p_threshold* values from $\psi = 0.1$ to $\psi = 0.9$ incrementing by 0.1 (we used $\psi = 0.9$ by default) under *C1*.

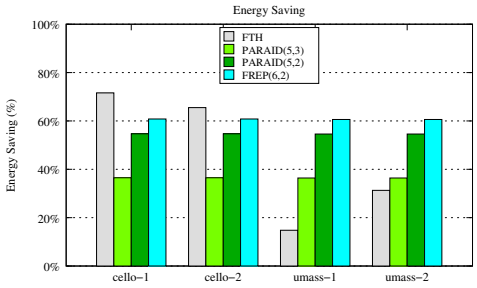
Figure 16 shows the results. We compare the results of the two time frame sizes. As expected, a smaller time frame size yields greater energy saving, but worse response time. In contrast, a greater time frame size makes more conservative gear-shift decisions, thus yielding less energy saving but better performance.

Choosing a *p_threshold* value largely affects both performance and energy saving. Increasing *p_threshold* gradually reduces the level of energy saving due to more conservative gear-shifts. With respect to performance, average response time becomes stable as $\psi \geq 0.8$ for $T_W = 5$. In the case of $T_W = 30$, $\psi = 0.9$ improves response time dramatically. With less than that, there are no significant differences.

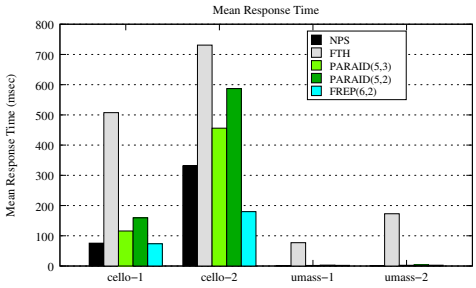
Turning our attention to the number of down-shifts. Naturally, smaller *p_threshold* tends to make more down-shifts. We observed 78 down-shifts with $\psi = 0.1$ and 20 with $\psi = 0.9$ in the setting of $T_W = 5s$. At the same time, the number of down-shifts for $T_W = 30s$ was 496 with $\psi = 0.1$ and 34 with $\psi = 0.9$. As expected, this shows that a greater time frame size (i.e., a smaller number of bits) makes more frequent gear-shift decisions.

5.6 Impact of the number of tickets

It is possible to configure *MaxTicket* (i.e., the maximum number

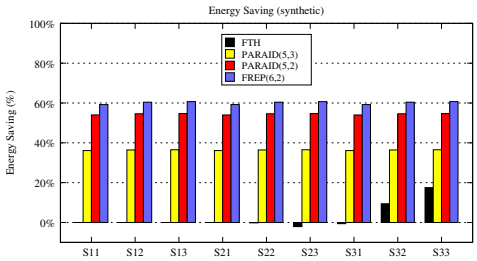


(a) Energy saving

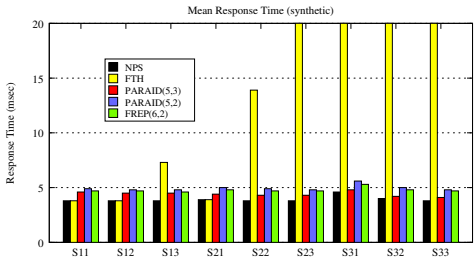


(b) Average response time

Figure 11: Energy saving and average response time (real traces)

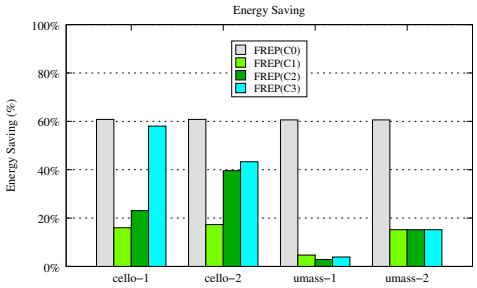


(a) Energy saving

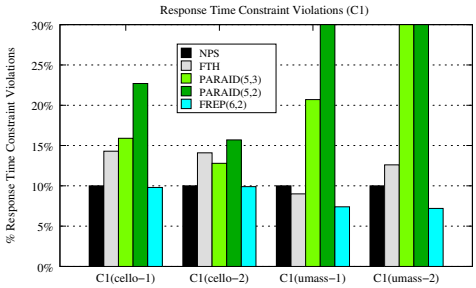


(b) Average response time

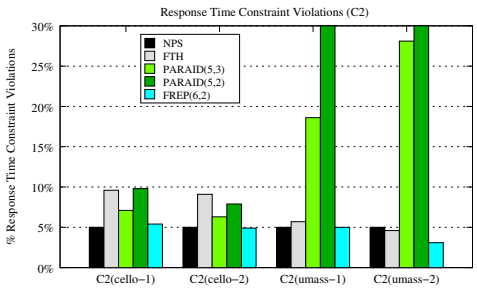
Figure 12: Energy saving and average response time (Synthetic traces)



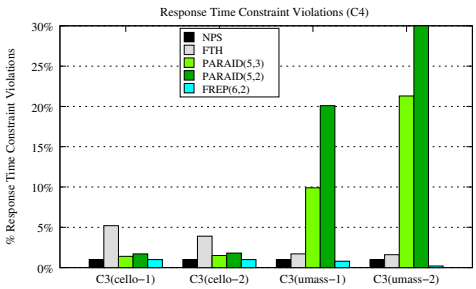
(a) Energy saving



(b) Percentage of response time constraint violation (*C1*)

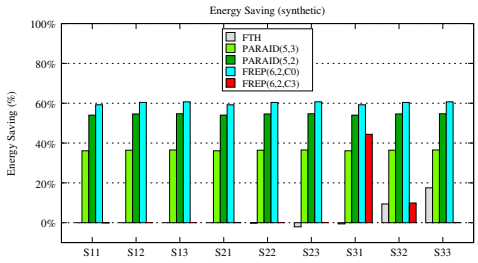


(c) Percentage of response time constraint violation (*C2*)

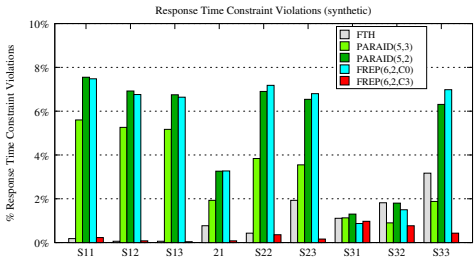


(d) Percentage of response time constraint violation (*C3*)

Figure 13: Energy saving and performance guarantee under specified SLAs

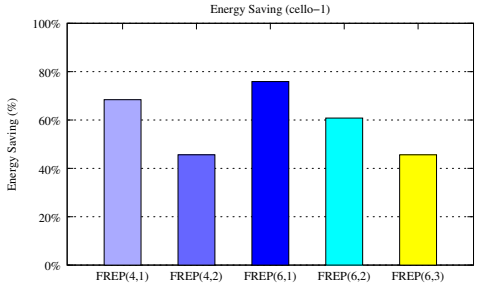


(a) Energy saving

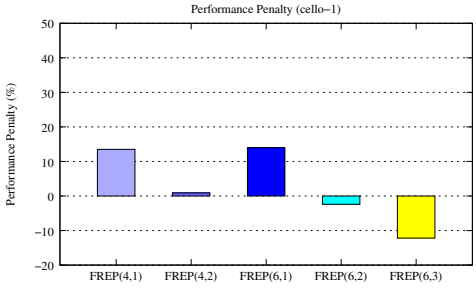


(b) Percentage of response time constraint violation

Figure 14: Energy saving and performance guarantees (Synthetic traces)



(a) Energy saving



(b) Performance penalty

Figure 15: Impact of FREP configurations with different number of CS and non-CS nodes (cello-1)

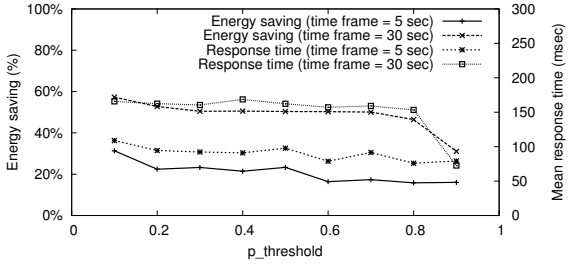


Figure 16: Impact of time frame size and *p_threshold* (cello-1/*C1*)

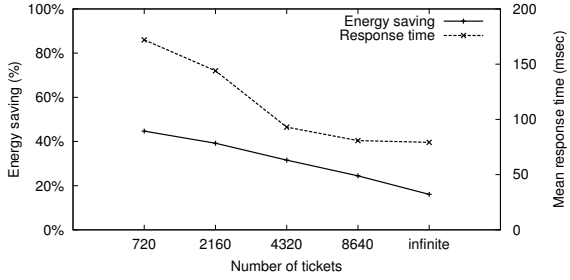


Figure 17: Impact of number of tickets (cello-1/*C1*)

of tickets) to set window size, since it limits the number of observations used for edge probabilities on the de Bruijn graph. Intuitively, any larger window may allow us to construct a more accurate graph with many more observations (due to a greater time length to collect), but may not be helpful for frequently varying workloads over time (because each observation has less impact on a larger collection of observations). With a smaller window, in contrast, it can better react to the recent workload characteristics.

Figure 17 shows the impact of *MaxTicket* with the cello-1 trace under *C1*, as in the above experiment. Overall, using a small window achieves better energy saving but worse performance, and vice versa. Thus, *MaxTicket* can be adjusted based on system goals and the expected degree of workload variations. The number of down-shifts observed was 179 with $MaxTicket = 720$, while it was only 20 for $MaxTicket = \infty$ (used in the other experiments).

5.7 Evaluation of down-shift decisions

In this section we discuss the quality of gear shifting decisions using our four workloads. In general, three main approaches are employed in the literature for gear down-shift decisions. MAID based products simply perform a down-shift when the disk idle period exceeds some fixed threshold (FTH), PARAID uses disk utilization thresholds, whereas FREP determines when to perform a down-shift using probabilistic prediction based on analysis of past historical events (by using de Bruijn Graphs).

The advantage of using FREP's de Bruijn graph prediction is illustrated in Figure 11 where it is shown that FREP achieves excellent energy savings while paying a minimal price in terms of response time. Similarly, in Figures 13(b)–13(d) and 14(b), FREP consistently shows the smallest number of response time constraint violations. In Figure 18, we analyze the quality of FREP's predictions based on the percentage of down-shifts that were penalized due to service constraint violations for the four workloads. Recall that a penalty event occurs in FREP if the service constraint is not met during a period of break-even time right after the down-shift. Maximizing the number of down-shifts is essential in order to maximize energy saving, while at the same time we wish to minimize the number of penalty events. As shown in the figure, prediction accuracies are over 80% for most cases (*C0* results show almost

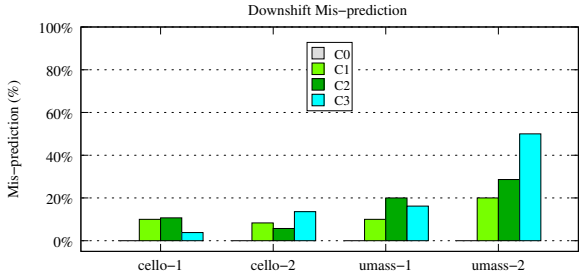


Figure 18: Prediction accuracy

100% accuracy). However, we can see that *C2* and *C3* for umass-2 have somewhat higher inaccuracy. Nonetheless, we have seen that FREP successfully manages the performance level even for umass-2, as shown in Figure 13. In the case of umass-2 *C3*, FREP reduced the number of down-shifts to 60% of *C0*'s (due to the penalties).

6. CONCLUSION

Energy proportionality is one of key metrics for future datacenters for both energy conservation and performance guarantees. In this work, we presented a technique called FREP (Fractional Replication for Energy Proportionality), for energy management that enhances energy proportionality in large datacenters. FREP includes a replication strategy and basic functions to enable flexible energy management. Specifically, our method provides performance guarantees by adaptively controlling the power states of a group of disks based on observed and predicted workloads. Our extensive experimental results with a broad set of traces showed that our energy management technique can achieve energy saving of over 90% of theoretical limits with little performance loss. With tight service constraints, we showed that FREP satisfies service constraints in diverse settings. Future planned work includes extending our data placement algorithm and mathematical analysis for allowing variable replication factors to different parts of the data based on their access frequencies.

7. ACKNOWLEDGMENTS

We are grateful to the anonymous reviewers for their constructive comments and suggestions. This work was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

8. REFERENCES

[1] 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*, SoCC '10, pages 217–228, New York, NY, USA, 2010. ACM.

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

[3] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management. pages 231–248, 2002.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: Evidence and implications. In *Proceedings of INFOCOM (INFOCOM '99)*, pages 126–134, 1999.

[5] Tools and traces, http://www.hpl.hp.com/research/ssp/software/.

[6] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116. ACM, 2001.

[7] E.-Y. Chung, L. Benini, A. Bogliolo, Y.-H. Lu, and G. De Micheli. Dynamic power management for nonstationary service requests. *IEEE Trans. Comput.*, 51(11):1345–1361, 2002.

[8] G. Dhiman and T. S. Rosing. Dynamic power management using machine learning. In *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, pages 747–754, New York, NY, USA, 2006. ACM.

[9] Disksim, http://www.pdl.cmu.edu/disksim/.

[10] M. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 8–8. USENIX Association, 2003.

[11] L. Ganesh, H. Weatherspoon, M. Balakrishnan, and K. Birman. Optimizing power consumption in large scale storage systems. In *HotOS*, 2007.

[12] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.

[13] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: dynamic speed control for power management in server class disks. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 169–181. ACM, 2003.

[14] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Reducing disk power consumption in servers with DRPM. *Computer*, 36(12):59–66, 2003.

[15] D. Borthakur. The Hadoop Distributed File System: Architecture and Design., http://hadoop.apache.org/ core/docs/current/hdfs design.pdf.

[16] D. P. Helmbold, D. D. E. Long, and B. Sherrod. A dynamic disk spin-down technique for mobile computing. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 130–142. ACM, 1996.

[17] S. Irani, G. Singh, S. K. Shukla, and R. K. Gupta. An overview of the competitive and adversarial approaches to designing dynamic power management strategies. *IEEE Trans. VLSI Syst.*, 13(12):1349–1361, 2005.

[18] J. Kim and D. Rotem. Energy proportionality for disk storage using replication. Technical Report LBNL-3936E, Lawrence Berkeley National Laboratory, September 2010.

[19] W. Lang, J. M. Patel, and J. F. Naughton. On energy management, load balancing and replication. *SIGMOD Rec.*, 38(4):35–42, 2009.

[20] F. T. Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes (Section 3)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[21] D. Li and J. Wang. eRAID: a queueing model based energy saving policy. In *MASCOTS '06: Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation*, pages 77–86, Washington, DC, USA, 2006. IEEE Computer Society.

[22] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. *Trans. Storage*, 4(3):1–23, 2008.

[23] V. N. Padmanabhan and L. Qiu. The content and access dynamics of a busy web site: findings and implications. *SIGCOMM Comput. Commun. Rev.*, 30(4):111–123, 2000.

[24] Umass Trace Repository: OLTP Application I/O, http://traces.cs.umass.edu/index.php/storage/storage.

[25] A. Verma, R. Koller, L. Useche, and R. Rangaswami. SRCMap: energy proportional storage using dynamic consolidation. In *FAST*, pages 267–280, 2010.

[26] C. Weddle, M. Oldham, J. Qian, A.-I. A. Wang, P. Reiher, and G. Kuenning. PARAID: A gear-shifting power-aware RAID. *Trans. Storage*, 3(3):13, 2007.

[27] T. Xie. SEA: a striping-based energy-aware strategy for data placement in RAID-structured storage systems. *IEEE Trans. Comput.*, 57(6):748–761, 2008.

[28] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: helping disk arrays sleep through the winter. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 177–190. ACM, 2005.