Hibernator: Helping Disk Arrays Sleep through the Winter

Qingbo Zhu, Zhifeng Chen, Lin Tan, Yuanyuan Zhou
Department of Computer Science
University of Illinois at Urbana-Champaign
{qzhu1,zchen9,lintan2,yyzhou}@cs.uiuc.edu

Kimberly Keeton and John Wilkes
Hewleet-Packard Laboratories
Palo Alto, CA
{kkeeton, wilkes}hpl.hp.com

Abstract

Energy consumption has become an important issue in high-end data centers, and disk arrays are one of the largest energy consumers within them. Although several attempts have been made to improve disk array energy management, the existing solutions either provide little energy savings or significantly degrade performance for data center workloads.

Our solution, Hibernator, is a disk array energy management system that provides improved energy savings while meeting performance goals. Hibernator combines a number of techniques to achieve this: the use of disks that can spin at different speeds, a coarse-grained approach for dynamically deciding which disks should spin at which speeds, efficient ways to migrate the right data to an appropriate-speed disk automatically, and automatic performance boosts if there is a risk that performance goals might not be met due to disk energy management.

In this paper, we describe the Hibernator design, and present evaluations of it using both trace-driven simulations and a hybrid system comprised of a real database server (IBM DB2) and an emulated storage server with multi-speed disks. Our file-system and on-line transaction processing (OLTP) simulation results show that Hibernator can provide up to 65% energy savings while continuing to satisfy performance goals (6.5–26 times better than previous solutions). Our OLTP emulated system results show that Hibernator can save more energy (29%) than previous solutions, while still providing an OLTP transaction rate comparable to a RAID5 array with no energy management.

Categories and Subject Descriptors

D.4 [**Operating Systems**]: Storage Management

General Terms

Algorithms, Management, Performance, Experimentation

Keywords

Energy management, Disk layout, Performance guarantee, Storage system, Disk array

SOSP'05, October 23–26, 2005, Brighton, United Kingdom.

1. Introduction

"What matters most to the computer designers at Google is not speed but power – low power, because data centers can consume as much electricity as a city." – Eric Schmidt, CEO, Google

"One of the biggest consumers of power within the computer technology industry is storage, and the little magnetic disk drive is one of the worst power hogs in the business. The magnetic disk drive is very similar to a honeybee. One is no problem. You can even have dozens, but when you reach hundreds or thousands then you have a swarm." – Chuck Larabie, Computer Technology Review

Data centers used to support modern enterprises and Internet service providers are getting larger, and their energy consumption is increasing, too, as power densities increase. Typical values for service-provider data center power densities are 150-200 W/ft^2 today, and will be 200-300 W/ft^2 in the near future [27]. Meanwhile, some are already designing 500 W/ft^2 data centers [5]. With the latter figure, a medium-sized 30,000 ft^2 data center requires 15 MW to power, one third of which is spent on cooling [33]; this is $13 million per year of electricity. In aggregate, US data centers were projected to cost $4 billion/year to power in 2005 [11].

Power represents about 19% of a data center's Total Cost of Ownership (TCO) [8], and disk drives are a major contributor to that. For example, disk drives contribute 86% of the energy consumption in a typical EMC Symmetrix 3000 storage system configuration [3]. In a larger system context, disks consumed 71% of the power for the 2003 Dell PowerEdge6650 benchmark system that set a price/performance record [4] – eighteen times as much as the processors, and 13% of the TCO. In the much larger HP Integrity

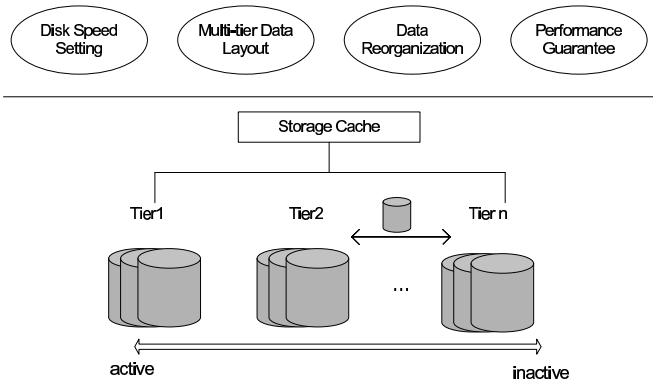


Figure 1: The Hibernator design

RX5670 Cluster TPC-C system [6], disk power represents about 10% of the TCO.

Extrapolating for the medium-sized data center above, disk drives would represent an electricity budget of $7–9 million per year. Reducing this by 50% would save $4–4.5 million/year, or 5–6% of the TCO. This paper describes a system capable of doing so.

1.1 Other observations

As research efforts on energy management for high-end processors, main memory, network interface cards and switches reduce the energy consumption of high-end servers, storage system energy consumption will become even more important. This trend is exacerbated by the projected annual growth rate of 60% [28] in storage requirements, as well as the use of higher-performance, faster-spinning disks, with their higher power requirements.

Reducing the speed at which a disk spins reduces its power consumption; stopping it completely reduces it still further. Unfortunately, both techniques reduce performance, so a good energy conservation solution needs to balance energy savings against performance degradation. This is particularly important for data center applications, where service level agreement (SLA) performance goals often need to be met. These typically include average or maximum response times for transactions [41]. There may be significant penalties for failing to comply with an SLA, so energy conservation cannot be bought at the expense of performance.

Although several attempts have been made to improve disk array energy management, the existing solutions either provide little energy savings or significantly degrade performance for data center workloads (Details are discussed in Section 2).

1.2 Our contributions

In this paper, we take a significant step towards practical disk array energy management for performance-sensitive data center environments, by describing and evaluating the design of *Hibernator* (Figure 1), a disk array that minimizes disk drive energy consumption, while still meeting response-time performance requirements. Hibernator achieves this goal by combining the following ideas:

1. *Leveraging multi-speed disk drives*, such as Sony's [29, 42], which can run at different speeds (and hence power levels), but have to be shut down to transition between the different speeds.

2. A *disk-speed-setting algorithm* that we call **Coarse-grain Response (CR)**, which uses the observed workload to determine optimal disk speed settings that minimize energy consumption without violating the performance goals. (This algorithm also works with previous data layouts such as RAID5.)

3. CR is used to size the tiers in a *multi-tier data layout*, where each tier consists of a set of multi-speed disks operating at the same speed. This layout requires no extra disks and does not sacrifice reliability compared to RAID5.

4. An *energy- and time-efficient data migration* scheme, called **randomized shuffling**, that performs reconfiguration quickly and allows the data layout to adapt to workload changes.

5. An algorithm to *meet response-time goals*, by boosting disk speed if the performance goal is at risk. This method also works with previously proposed energy control algorithms.

It is important to use appropriate, realistic workloads in determining system performance. Most previous studies on disk array energy management used trace-driven simulations in their evaluation, and many used synthetic workloads. As a result, it is unclear whether their results are representative of real data center workloads. We address this by evaluating Hibernator using traces of real systems and by constructing a hybrid system of a real database server (IBM DB2) and an emulated storage server.

Our file-system and on-line transaction processing (OLTP) simulation results show that Hibernator can satisfy the specified performance goals and still provide up to 65% energy savings, 6.5–26 times more than previous solutions. Our OLTP emulated system results show that Hibernator has the highest energy savings (29%) across five evaluated approaches, while still providing an OLTP transaction rate comparable to RAID5 without any energy management. To the best of our knowledge, our study is the first to evaluate the impact of disk energy management on data center application performance (transaction rate) in a commercial database server (IBM DB2).

This paper is organized as follows. Section 2 describes background material. Section 3 discusses our Hibernator solution. Section 4 describes our simulation methodology and emulated system evaluation. Section 5 presents simulation results, followed by hybrid system results in Section 6. Section 7 concludes the paper.

2. Background and related work

In this section, we first discuss disk power models and algorithms to control disk energy adaptation based on the disk models. Then we discuss disk layouts that also affect disk energy adaptation, followed by previously proposed methods to provide performance guarantees.

2.1 Disk power models

Most modern disks have two power modes: *active*, where the disk spins at full speed and *standby*, where the disk stops spinning completely. Disks in standby mode use considerably less energy than disks in active mode, but have to be spun up to full speed before they can service any requests. This incurs a significant energy and time penalty (e.g., 135 Joules and 10.9 seconds for IBM Ultrastar 36Z15 disks [21]). To justify this penalty, the energy saved by putting the disk in standby mode has to be greater than the energy needed to spin it up again – which will only be true if the next request arrives after a *break-even time*. Unfortunately, this is rarely the case in intense, enterprise workloads.

Gurumurthi et al. [19] and Carrera et al. [10] have proposed a dynamic multi-speed disk model, which has the capability of dynamically changing the disk speed while spinning. Additionally, such a disk could service requests at low speeds without transitioning to full speed. Unfortunately, such disks do not exist yet, and it is also unclear whether such a disk is mechanically feasible to build.

A more practical approach may be to use disk drives that are designed to operate at a small set of different rotational speeds, but can only change spin speed while they are in standby mode [29, 42]. Sony makes commercial versions of such disk drives that support two speeds, although there appears to be no fundamental obstacle to supporting more speeds. We assume this style of multi-speed disk for our experiments.

2.2 Energy control algorithms

A commonly used energy control algorithm is to transition a disk into a low power mode after the disk is idle for a while. When a request arrives at a disk in a low power mode, the disk immediately transitions to the active mode to service the request. This control algorithm has been used in many previous studies on energy management for disk arrays [14, 37, 44] and disks in mobile devices [15, 17, 18, 20, 30, 39]. We refer to it as Traditional Power Man-

agement (**TPM**). Since several previous studies [10, 19, 44] have shown that this algorithm performs worse than other energy control algorithms in data-center-like workloads, we do not consider it further here.

Carrera et al. [10] and Pinheiro et al. [32] proposed exploiting dynamic multi-speed disks by switching speeds based on the observed load. When the disk load becomes lighter than 80% of the disk throughput of a low speed, the disk spins down to the low speed mode; if the load is heavier than the same threshold, the disk spins up to the high speed. We refer to this as Load Directed (**LD**).

Gurumurthi et al. [19] suggested using changes in the average response time and the length of the disk request queue to drive dynamic disk-speed transitions. Periodically, each disk checks the number of pending requests in its queue. If this number is less than a threshold N_{min} representing light load, the disk spins down its speed by one level. Meanwhile, the controller tracks average response times for fixed-sized windows of requests and calculates the percentage change in average response time over the past two windows. If the percentage change exceeds an upper tolerance, the controller spins up all disks to full speed. If it is less than a lower tolerance, a disk may spin down to a lower speed. We refer to this scheme as Dynamic RPM (**DRPM**).

Even though the energy control algorithms listed above do consider performance in various ways, they do not attempt to provide performance guarantees, and in many cases they degrade performance so much that they are unusable in many data center applications, as we shall see in Sections 5 and 6.

Disk energy management can be complemented by processor and memory energy management techniques. Typical disk operations take milliseconds, while processors can scale their voltage in a few tens of microseconds [25] and shutdown micro-architectural resources such as functional units within a few CPU cycles [7]. This implies that any disk energy savings will result in whole-system energy savings. Of course, the delays themselves may not be acceptable.

2.3 Disk array layouts

We discuss both traditional performance-oriented disk layouts and recently proposed energy-efficient disk layouts in this section.

2.3.1 Performance-oriented disk array layouts

RAID techniques are a long-standing solution for improving disk array performance and reliability [31]. Many studies have been conducted on disk array layouts, but almost all have been directed toward improving performance in the absence of energy conservation considerations.

RAID5 is a common disk array data layout that interleaves data blocks and distributes parity blocks evenly across all disks in the array. It offers a good balance of storage efficiency and good performance for reads and large writes, but suffers from poor performance for small writes [12].

The HP AutoRAID [40] employs a dynamic, adaptive data layout, mixing RAID5 and mirrored storage (RAID1), in order to achieve space efficiencies comparable to RAID5 and performance comparable to mirrored storage. Hibernator uses some of the same ideas, but for the purpose of achieving the best tradeoff between energy and performance.

2.3.2 Energy-efficient disk array layouts

Attempts to trade off availability against energy by powering down "unnecessary" disk drives in disk arrays [22] result in little benefit: if a disk array can survive p disk failures, these energy management schemes can power down at most p disks on average. Since p in modern disk arrays is usually small compared to the total number of disks, the energy saved by such methods is also small (e.g., 2–7%). Worse, when disks are powered down to save energy, the system's reliability is significantly reduced.

More success has been achieved with schemes that concentrate disk array workloads onto a subset of their disks so that the other disks can stay in low power modes. We discuss three here.

Son et al. [37] proposed a method to determine the striping parameters (the number of disks, the stripe block size, etc.) for a RAID5 layout to minimize disk energy consumption for scientific applications with regular data access patterns. Using the SPEC95 floating-point benchmarks as a test case, only limited energy savings resulted (19%), even with aggressive compiler cooperation and access to the applications' source code. Since this approach is targeted at scientific applications and cannot adapt to workload changes, we do not consider it further.

Massive Array of Idle Disks (MAID) [14] uses a few additional always-on cache disks to hold recently accessed blocks to reduce the number of accesses to other disks. Unfortunately, this layout, which was designed for archiving workloads, is not energy-efficient for data center workloads, because the extra cache disks consume energy [32]. We verified this ourselves: for one of our test workloads (an OLTP trace collected from IBM DB2), we found that 3 additional cache disks increased the total energy usage by about 13% in a 25-disk MAID design over a straightforward RAID5 baseline. Therefore, we do not consider MAID further here.

Popular Data Concentration (PDC) [32] concentrates loads by taking advantage of heavily skewed file access frequencies. Periodically, PDC migrates files based on their access frequencies: the most popular files are migrated to the first disk until the disk is full or the expected load on this disk approaches its maximum bandwidth, and the next most popular files are migrated to the second disk, and so on. However, as shown in Section 5, it can incur substantial performance degradation due to load concentration, even when all disks stay in the active mode.

2.4 Performance guarantees

In a previous study [23], we proposed a technique to provide performance guarantees for energy control algorithms in main memory and disks, with a focus on memory. A user would supply a limit on the acceptable percentage *execution time slowdown* of the application. By tracking the performance effects of energy adaptation, the technique was able to decide when to disable the underlying energy management and go to full speed (and full-power) mode to avoid exceeding the slowdown limit. After the measured slowdown had returned to sufficiently below the limit, the energy management could be re-enabled. (A similar technique was used in the AFRAID disk array to manage behavior to availability bounds [35].)

Unfortunately, although the goal is attractive, the technique may be impractical: many applications issue multiple outstanding asynchronous disk I/Os to hide I/O latency, and it is difficult to understand the effects of the energy adaptation scheme without the application's cooperation, which is rarely forthcoming.

We take a simple, practical approach here, and assume the existence of a storage-system level SLA, with an average I/O response time (R_{limit}) for the storage system itself [41]. We also assume that the SLA is practical, and that a RAID5 disk array is able to deliver an average response time within the specified limit. (How to ensure this property is related to resource provisioning, which is beyond the scope of this paper. See [9] for one approach.)

Other work [13], conducted in parallel with ours, also attempts to dynamically optimize energy and operational costs while meeting performance-based SLAs by using three techniques based on

steady state queuing analysis, feedback control theory and a hybrid between the two. However, it focuses on energy adaptation at server (node) granularity in a cluster comprised of identical servers and is evaluated using web server workloads, where a node can access all data even when all other nodes are powered down. This is usually not the case in storage systems.

3. Hibernator

Our goals for Hibernator were to design a system that could provide a controllable balance between energy savings and performance impacts for a RAID5-like disk array. The first component of the Hibernator design is the use of multi-speed disks. The remaining components are the subject of this section.

3.1 Disk-speed setting

Since multi-speed disks take a significant amount of time to transition from one speed to another (12.4s for the SONY drive [29, 42], and 6.9s for the dynamic multi-speed disk [19]), requests that arrive during this period can be significantly delayed. Therefore, from performance perspective, it is desirable to make such speed changes infrequently – i.e., at a coarse time granularity.

Moreover, frequently starting and stopping disks is suspected to affect disk drive longevity. Even though drive reliability has been significantly improved by using load/unload technology to prevent head-to-disk interaction and start-up wear, the number of start/stop cycles a disk can tolerate during its service life time is still limited, and many disk specifications provide an expected lifetime value (e.g., the IBM Ultrastar 36Z15 can handle a minimum of 50,000 start/stop cycles [21]). Making disk speed changes infrequently reduces the risk of running into this limit. For example, if the disk speed changes only 25 times a day, it would take 6 years to reach the minimum number, slightly longer than the maximum service life time for which most disks are designed (typically 5 years or 20,000 operating hours [2]).

As a result, the main idea of the Hibernator disk-speed setting algorithm is to adapt the disk speed infrequently, and keep it constant during a relatively long *epoch*. We call the algorithm *coarse-grain response* (CR), to emphasize that it makes large-granularity decisions based on predicted I/O response times. CR chooses a disk speed configuration at the beginning of each epoch that minimizes energy consumption, while still satisfying response-time goals; it works with any underlying disk layout, including RAID5, PDC, and the new Hibernator disk layout described in Section 3.2.

At the beginning of an epoch, CR determines the best speed configuration for each disk, using the predicted workload for each disk and the specified average response time limit R_{limit}. Each disk stays at the assigned speed throughout the entire epoch unless the observed average response time exceeds the specified limit due to unpredicted workload changes, in which case the performance guarantee method described in Section 3.3 takes over. During each epoch, the CR algorithm monitors the load on each disk to provide the workload statistics needed for the next epoch.

The epoch length, T_{epoch}, should be long enough to amortize the disk transition cost and short enough to be responsive to workload changes. In practice, our sensitivity analysis results in Section 5 show that Hibernator is insensitive to a broad range (one to four hours) of epoch lengths.

3.1.1 Problem formalization

The goal of the CR algorithm is to choose, at the start of each epoch, for each disk i, a speed j that minimizes the total predicted energy consumption subject to a constraint that the average response time be no greater than R_{limit}. More formally, CR needs to solve for j in the following optimization problem:

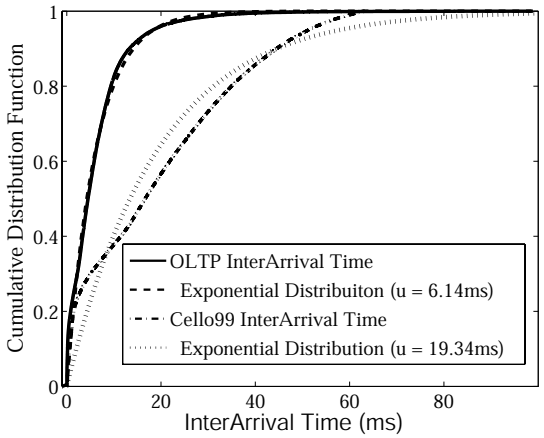


Figure 2: Observed inter-arrival time distributions of OLTP and Cello99 workloads, fitted against exponential distributions.

$$\textbf{minimize} \quad \sum_{i=0}^{n-1} E_{ij}$$

$$\textbf{subject to} \quad \sum_{i=0}^{n-1} (N_i \times R_{ij})/N \leq R_{limit}$$

where E_{ij} is a prediction of the energy that would be consumed by disk i if spinning at speed j in the epoch, R_{ij} is a prediction of the average response time in the epoch, n is the total number of disks, N_i is the number of requests at disk i and N is the total number of requests.

3.1.2 Solving for response time R_{ij}

Clearly, the estimation of R_{ij} depends on the workload on disk i in the epoch. Since most data center workloads are relatively stable, we use the last epoch's workload characteristics as a predictor for the next epoch's. In most cases, the prediction is correct; when it is not, the performance guarantee method described in Section 3.3 is invoked. Our results (in Section 5) validate these claims.

We use a M/G/1 queuing model to estimate the average response time for each disk, and extend this with a model for the delay caused by any disk transition at the beginning of the epoch.

The M/G/1 queuing model, which represents Poisson arrivals and a general service time distribution, has been widely used in modeling the performance of disk systems [26, 36]. Figure 2 displays the reasonable match achieved between the observed I/O request inter-arrival time for the OLTP and Cello99 traces used in our experiments and exponential distributions that were fitted to them using maximum likelihood estimation. (The Cello99 trace is slightly less good a match, perhaps because of the variety of performance behaviors seen in Cello's storage system.) Importantly, an exact match is not necessary: we are only using the performance model to determine a "good enough" disk speed level; errors only result in slightly higher energy usage than the ideal.

Let us assume that the most recent observed average request arrival rate at disk i is α_i. The service time for a request at a disk spinning at speed j is t_{ij}, which can be measured at run time. Let $Exp(t_{ij})$ and $Var(t_{ij})$ be the mean and variance of the service time. The disk utilization ρ_{ij} can be calculated as $\rho_{ij} = \alpha_i Exp(t_{ij})$.

Suppose disk i needs to change its speed in the new epoch. While it is changing its spin speed, the disk cannot service requests. Let us denote length of the transition period as T_i; if disk i does not

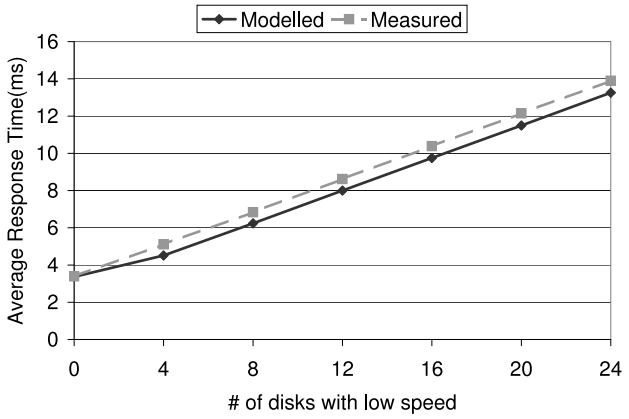


Figure 3: The measured and modeled average disk response time as a function of the number of low-speed disks. The workload used is the OLTP trace (see Section 4.1).

need to change its speed, $T_i = 0$. During this period, the number of requests that will arrive is $\alpha_i \times T_i$. Since T_i is large compared to t_{ij}, it dominates those requests' response time, and the average delay for those requests is $T_i/2$, for an arrival process with a Poisson distribution. The average response time of the requests affected by a disk speed transition is thus

$$R'_{ij} = \frac{T_i}{2} + \frac{\alpha_i T_i Exp(t_{ij})}{2} = \frac{T_i}{2}(1+\rho_{ij})$$

where the second term in the formula represents the average queuing and service delay of those requests held up during T_i. (In practice, it will be slightly less, as the longer queue will likely be serviced more efficiently by the disk.)

For requests that arrive after the disk transition finishes, the disk is in a stable state. Therefore, according to the M/G/1 queuing model, these requests' average response time is

$$R''_{ij} = \alpha_i Exp(t_{ij}) + \frac{\alpha_i^2 (Exp^2(t_{ij}) + Var(t_{ij}))}{2(1-\alpha_i Exp(t_{ij}))}$$

Combining the two formulas, the average response time during the entire epoch is

$$R_{ij} = \frac{\alpha_i T_i R'_{ij} + \alpha_i (T_{epoch} - T_i) R''_{ij}}{\alpha_i T_{epoch}}$$

Requests that arrive immediately after a speed-transition ends (i.e., just after T_i) will see a queue of the requests held up during T_i, and will themselves experience an additional delay. However, since T_{epoch} (one or multiple hours) is several orders of magnitude larger than T_i (5–10 seconds), the effects of such delays should be negligible, and we do not consider them in our R_{ij} calculation.

If the disk does not need to change its speed in the new epoch, the average response time for this disk R_{ij} is just R''_{ij}. Figure 3 shows that our analytical model matches the measured average response time reasonably well.

3.1.3 Solving for energy E_{ij}

Next, we estimate the energy consumption of keeping disk i at speed j in the next epoch. Suppose the active power, idle power at speed j and transition power are P'_{ij}, P''_{ij} and P'''_{ij}, respectively. The transition time T_i is defined as above. The active time during which disk i is servicing requests is $T_{epoch} \times \rho_{ij}$ because the request arrival rate is independent of the disk speed transition. The remaining time is idle time when disk i is spinning at speed j, but does not service requests. Therefore, the total energy for disk i is

$$\begin{aligned} E_{ij} &= P'_{ij} \times T_{epoch} \times \rho_{ij} \\ &\quad + P''_{ij} \times (T_{epoch} - T_{epoch} \times \rho_{ij} - T_i) \\ &\quad + P'''_{ij} \times T_i \end{aligned}$$

3.1.4 Finding a solution

Now we have materialized every term in the disk-speed selection problem, and it can be easily converted into an integer programming problem and be solved using the CPLEX solver [1]. The computational cost is small, especially for the two-speed SONY disk [29, 42] used in our experiments.

3.2 Adaptive layout

While the CR algorithm can determine a good disk-speed configuration from the workload, the amount of energy that can be conserved is still related to the underlying disk layout because the layout directly affects the loads to each disk. As we discussed in Section 2, RAID5 layouts provide good performance but are not energy-efficient, whereas previously proposed energy-efficient layouts such as PDC save energy but provide less performance than RAID5, even when all disks are in active mode. To maximize energy conservation while still meeting the performance goal, ideally we need a "polymorphic" disk array that provides performance similar to traditional performance-oriented layouts such as RAID5 and can save the same or more energy than previous energy-efficient layouts such as PDC.

One naive layout might be using two set of disks, one being organized as the RAID5 and the other as PDC, and dynamically switch between them based on the load characteristics. But this layout has several limitations. First, it requires doubling the number of disks without doubling the performance. Second, the switch granularity is too big. Basically, it is either RAID5 or PDC. There is no intermediate point to gradually increase the performance as the load increases. As such, it loses opportunities to conserve energy at those intermediate points.

Therefore, we designed a self-adaptive, performance-directed and energy-efficient disk layout. There are three goals in our layout design: (1) *energy goal*: minimizing disk energy consumption under any load: light, heavy, or intermediate; (2) *performance goal*: satisfying the average response time constraint; and (3) *self-adaptiveness goal*: dynamically adapting to workload changes. In addition, our layout should not require any extra disks or sacrifice reliability.

To achieve the above goals, Hibernator:

- uses the CR algorithm to determine a layout that will minimize disk energy consumption while still satisfying the predicted performance requirement;
- uses two levels of reorganizations to adapt to workload changes with little effect on foreground requests; and
- extends the performance model in the CR algorithm to consider the delay due to reorganizations.

3.2.1 Choosing the disk configuration

Hibernator uses a multi-tier organization, as shown in Figure 1. All disks in a tier spin at the same speed; different tiers run at different speeds. Each tier uses a data-protection organization similar to RAID5, with one parity block per data stripe, and so achieves a similar level of availability. The tiers are exclusive: data lives in only one tier, unlike a typical multi-level storage hierarchy. Thus, Hibernator needs no more disks than RAID5.

By convention, Tier 1 disks store the most active data and so are set to run at full speed to maximize performance. Conversely, the highest-numbered tier stores the least active data and is set to spin at the lowest speed to save energy. Within each tier, Hibernator strives to achieve uniform load-balancing to maximize performance.

The number of disks in each tier is obtained simply by running the CR algorithm described in Section 3.1 at the beginning of each epoch. This results in a speed for each disk, and we simply label the set of disks with the same spin speeds a tier.

Note that this scheme adapts dynamically to workload changes. If the load becomes heavier, CR will put more disks into high speed tiers to satisfy performance requirements, and if the load becomes lighter, CR will put more disks into low speed tiers to conserve energy.

3.2.2 *Efficient small-scale reorganization*

The granularity of movement between tiers is whole disks, but many workloads have smaller-granularity hot spots, or there may be portions of the workload that become more (in)active than they were before. Although the performance-bounding algorithm of Section 3.3 ensures that the performance goals will be met if this happens, it is desirable to trigger it as infrequently as possible in order to save energy, and so Hibernator provides a *temperature-based* algorithm to migrate such data between tiers.

Similarly to HP's AutoRAID [40], we organize data into fixed-sized relocation blocks (RBs) that represent the minimal unit of migration in the system. Within a tier, the number of RBs that a RAID stripe contains on a single disk is defined as a stripe unit.

Hibernator translates SCSI logical block numbers (LBNs) into the corresponding RB address via an RB-map that stores the physical location of every RB. The map is stored at fixed, redundant locations on disk so that it can easily be referenced, and cached in memory in its entirety for speed. The map is small: in a 10 TB disk array with 256 KB RBs, the RB-map occupies 320 MB, or only 0.0032% of the total storage, which is small compared to a typical disk array cache. A small portion of the disk array's non-volatile memory (NVRAM), normally used to provide fast write commits, is used to preserve RB-map updates, from where they are periodically written to disk.

Hibernator places an RB in a tier based on the RB's "temperature", which can be calculated in a number of ways. For example: recency (how recently it is accessed), popularity (the number of times it is accessed), or a combination of these metrics. In our experiments, we used an aged-weighted frequency metric that combines both recency and popularity. Formally speaking, after every k accesses (typically $k = 10$) to an RB, the RB's temperature is adjusted as follows:

$$Temp_{new} = (1 - \xi) \times Temp_{old} + \xi \times k/T_{last}$$

where $Temp_{old}$ is the previous temperature, T_{last} is the time period of the last k accesses, and k/T_{last} is the average access frequency (the number of accesses per time unit) during the last k accesses. The history factor ξ specifies how strongly the most recent measurement contributes to the temperature estimation. We used $\xi = 0.8$ in our experiments.

Hibernator monitors the temperature of each RB in the RB-map and compares it to the temperature of RBs in other tiers to determine if rearranging data is beneficial. RBs that are accessed frequently and recently can be promoted if their temperature values become larger than those of RBs in higher speed tiers. Conversely, seldom accessed RBs may be demoted if their values fall below those of RBs in lower speed tiers. This algorithm runs continuously: it is not bound to epoch boundaries.

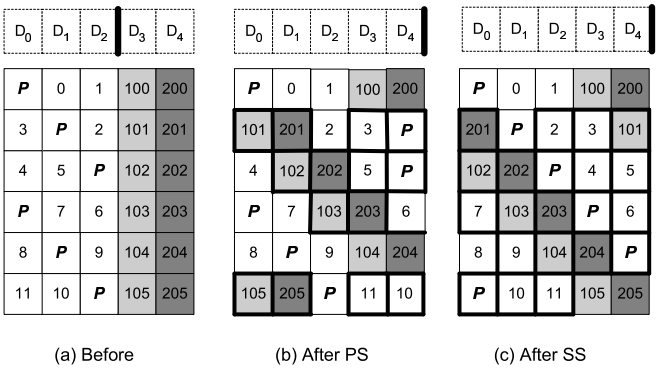


Figure 4: Large-scale reorganization solutions: PS and SS. Blocks are shaded and numbered by their temperature (lighter = smaller = hotter). Migrated blocks are surrounded by a thick border. In the column headings, D_i means disk i, and thick bars represent tier boundaries. The figure shows the migration of two disks (D_3 and D_4) from a low-speed tier into a high-speed tier with three disks (D_0, D_1 and D_2).

To reduce interference with client (foreground) requests, Hibernator only performs migration when there are no foreground requests to service, and makes use of the disk array's NVRAM to enable recovery from a power failure during a migration. Parity and the RB-map are updated as part of an RB migration. In the future, techniques such as free block scheduling [24] could be used to reduce the impact of RB migrations on foreground work further.

3.2.3 *Efficient large-scale reorganization*

At the beginning of each epoch, Hibernator uses the CR algorithm to determine the best disk tier configuration for this epoch. If the configuration is different from the last epoch, disk reconfiguration is needed to achieve a balanced load inside each tier, and whole disks must be migrated from one tier to another.

Because disk migration may need to move a large amount of data quickly, block-at-a-time solutions (e.g., disk cooling [38]) are undesirable, and we need to develop efficient algorithms that approach near-sequential disk access rates. We developed three such algorithms:

- *Permutational shuffling (PS)* scans the stripes in a tier sequentially, as shown in Figure 4(b). It exchanges data blocks to achieve an even balance of RB temperatures on each disk but it only considers exchanges of RBs from the newly-added disks to other disks in a permutational fashion. This means that the maximum number of migrated blocks per stripe is small: $2m$ per stripe, for m migrated disks, and it is independent of the stripe size. However, the load and parity blocks are not always balanced uniformly. For example, in Figure 4(b), disk D_3 has no parity blocks and disk D_4 has no 100-numbered blocks.

- *Sorted shuffling (SS)*, is like PS but provides better load balance by uniformly distributing parity blocks and data blocks of different temperature among all disks in the tier. It does this by first *sorting* the blocks in a stripe based on their temperatures and then rotationally shifting all blocks in a stripe by one more position than the previous stripe (Figure 4(c)). However, SS incurs greater overhead than PS, because most blocks are relocated.

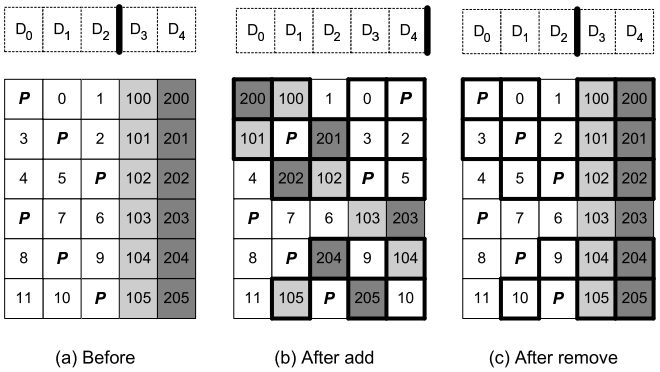


Figure 5: Randomized Shuffling. The graphics have the same meanings as in Figure 4. The transition from (a) to (b) shows the addition of disks D_3 and D_4 to a fast (hot) tier. The transition from (b) to (c) shows the result of moving those same disks into a slower (cooler) tier.

- *Randomized shuffling (RS)* fixes both problems, by distributing load and parity blocks in a statistically uniform fashion, while relocating fewer blocks than SS (Figure 5(b)). In each stripe, RS exchanges each block in the new disks with a randomly-chosen block from the stripe, regardless of whether they are data or parity blocks; if the target disk is the same as the one it came from, no move occurs, and only the parity needs to be updated. The randomization ensures a balanced block-temperature and parity distribution, since the number of stripes on a disk is large. The number of migrated blocks per stripe is essentially the same as with PS (a bit less than $2m$), but the result is a better layout. For a large stripe size, disk reorganization costs less than disk reconstruction upon failure in a traditional RAID5 array.

  Hibernator uses RS, as it is an improvement over both PS and SS.

Since all the disks in a tier are in a parity group, all the parity blocks in the tier need to be updated whenever a disk is moved in or out of a tier. This is done at the same time as the blocks are being reorganized. If a tier gets too large, it can be divided into smaller parity groups, although our experiments never did it. For exchanges within a tier, the large-write optimization, which calculates the parity from the data block values rather than using the old parity value, is only useful if all the data blocks are involved in the exchange. In this case, it saves one read – that of the parity block.

Since a block-exchange causes two blocks to move, a straightforward implementation that performs the data reorganization at the block granularity would require six I/O operations per stripe (reading and writing the exchanged blocks and the parity). To improve the reconfiguration efficiency, a simple optimization is to combine multiple small I/O operations on a migrated disk into a large one to take advantage of disks' high sequential access bandwidth. This method can leverage the large storage cache of modern storage systems(e.g., up-to 128GB for EMC Symmetrix Systems [3]) to batch multiple blocks that are either read sequentially from the migrated disk, or are read from other disks but will be placed sequentially into the migrated disk. Obviously, we cannot batch as many blocks as possible into one large sequential I/O because doing so could introduce significant delays to foreground requests.

Additionally, Hibernator performs two further optimizations in disk migration. Before migrating a disk into a higher-speed (hotter) tier, Hibernator first migrates the hottest data blocks in each stripe of the old tier onto the disk being moved; if the new tier is colder, then the coldest blocks are moved to the disk being migrated. By doing this, Hibernator attempts to move active data to high-speed tiers and inactive data to low-speed tiers. For example, in the transition from Figures 5(b) to 5(c), all 100-numbered and 200-numbered blocks are moved to disks D_3 and D_4 respectively before those disks are removed from the hot tier. It also has one other important benefit: the migrating disk will contain no parity blocks, so disk migrations do not change the number of parity blocks in a stripe.

To further reduce the reconfiguration cost, we coalesce the migrations generated by the pre-migration temperature concentration step and the actual migration, so that no block needs to be moved twice. For example, suppose a block had to first move to a disk to be migrated, but was then going to be exchanged with another block in its new tier. This optimization ensures that it is moved directly to its new position in a single operation.

As with single-RB migration, Hibernator only executes reconfiguration operations if there is no foreground work to perform. Thus foreground requests are delayed by at most one reconfiguration request. Experimental results in Section 5 confirm that a disk reconfiguration only slightly increases the average response time of foreground requests, and the reorganization can be done almost as quickly as sequentially scanning the disk twice.

3.2.4 The performance model revisited

Since disk reconfiguration may impact performance, we need to incorporate it into the response time estimation of Section 3.1. To do this, we need to calculate what performance penalty it imposes on the foreground requests and how long the reconfiguration period is. Unfortunately, building a precise model is complicated because there are two simultaneous access streams that may have different access patterns. For example, with the OLTP workload, there exist random foreground I/Os and sequential background I/Os – and the priority scheduling further adds to the complexity. We thus develop a simple, but practical, method to estimate the performance penalty and the length of the disk reorganization phase.

Let us first consider disk i in tier j. From the foreground request's point of view, besides the normal queuing and service delay, in the worst case each request has a delay caused by servicing one background request. Since a foreground request arrives randomly according to a Poisson distribution, the average additional delay in the worse case is $Exp(t_{ij})/2$. Therefore, conservatively, the average response time during this period for accesses to disk i is calculated as below:

$$R'''_{ij} = \alpha_i Exp(t_{ij}) + \frac{\alpha_i^2 (Exp^2(t_{ij}) + Var(t_{ij}))}{2(1-\alpha_i Exp(t_{ij}))} + \frac{Exp(t_{ij})}{2}$$

Let T_R represent the length of this reconfiguration phase. The expected number of idle periods is $\alpha_i T_R(1-\rho_{ij})$, where $\rho_{ij} = \alpha_i Exp(t_{ij})$ is the disk utilization. A seek and rotational delay occurs for the background requests during each idle period. Because all the background traffic occurs when the disk is idle, we have the following equation,

$$\alpha_i T_R(1-\rho_{ij})(Seek + Rotation) + 2Scan = T_R(1-\rho_{ij})$$

where $Scan$ is the disk sequential scan time when there is no foreground requests. As $Seek + Rotation \approx Exp(t_{ij})$, the disk reconfiguration time with foreground requests is

$$T_R \approx \frac{2Scan}{(1-\rho_{ij})^2}$$

Table 3 in Section 5 shows that our estimates of average response time and phase length are fairly accurate, compared to the experimental measurements.

The performance model described in Section 3.1 can be easily modified to take this phase into account, in a similar way to how the disk transition phase was handled.

To make Hibernator work, we also need to address other common but important issues such as disk failures during data reorganization, online storage capacity expansion and controller failover. These problems have similar solutions to those used by HP AutoRAID [40].

3.3 Response time performance guarantees

Even though our CR algorithm strives to determine disk speed settings based on the specified response-time requirement, an average response time higher than the specified limit is still possible, because of workload mispredictions from one epoch to the next. We thus designed a simple algorithm to ensure that performance goals would continue to be met even if the misprediction occurs. Note that we only provide soft guarantees on response-time averages, rather than hard guarantees for each request.

Our performance guarantee algorithm is a straightforward modification of our previous work [23]. Instead of keeping track of application execution time slowdown, the Hibernator disk array controller dynamically measures the average response time. If the observed average response time is larger than R_{Limit}, the energy management scheme is disabled and all disks spun up to full speed until the observed average response time is less than the specified limit R_{Limit}. Then the energy management scheme is re-enabled to conserve energy.

Our experimental results show that this method works well for almost all cases, including previous algorithms that are not performance-aware, as long as it is possible for the layout's maximum performance (without any energy management constraints) to meet the specified performance requirement.

4. Evaluation methodology

We evaluated Hibernator using trace-driven simulations with traces of real file system and OLTP workloads, and using a hybrid system comprised of a real database server (IBM DB2) and a storage server with emulated multi-speed disks driven by a live OLTP workload.

In our experiments, we compare Hibernator's energy savings over a baseline case (RAID5 without energy management) to previous layouts, such as RAID5 and PDC. In addition, since our performance guarantee method and CR algorithm are general, we also apply them to previous solutions. The baseline results give us the upper bound for energy consumption and the lower bound for average response time. The energy savings for each scheme is computed relative to this baseline case. All the evaluated schemes are listed in Table 1.

Scheme name	*Constituents*
Baseline	RAID layout without power management
$RAID_{DRPM}$	RAID layout + DRPM algorithm [19]
PDC_{LD}	PDC layout + LD algorithm [10, 32]
$RAID^{+}_{DRPM}$	RAID layout + DRPM algorithm +PG
PDC^{+}_{LD}	PDC layout + LD algorithm + PG
$RAID^{+}_{CR}$	RAID layout + CR algorithm + PG
Hibernator	adaptive layout + CR algorithm + PG

Table 1: The evaluated schemes. The parameters for previous energy control algorithms were taken from the original papers that proposed these algorithms. PG = performance guarantee.

4.1 Trace-driven simulation

We enhanced the widely used DiskSim simulator [16] and augmented it with a multi-speed disk power model. The specifications for the disk used in our study are similar to that of the IBM Ultrastar 36Z15, which, although now a little old, has the benefit of having been used in many previous studies [10, 14, 19, 32, 44]. The parameters are taken from the disk's data sheet [21]; some of the important ones are shown in Table 2. We modeled a two-speed disk (3000 and 15000 RPM) in our experiments. To calculate the parameters for each mode, we use the quadratic model [19]. Since we use a static multi-speed disk model, the disk speed transitioning cost is the sum of the costs for powering down and subsequently powering up the disk. The simulated storage system uses 2GB NVRAM as the storage cache in all cases. The number of disks is the same as specified by the corresponding trace.

Attribute	*Value*	
Individual disk capacity	18 GB	
Transitioning time	12.4 s	
Transitioning energy	152 J	
Attribute	*at 3000 RPM*	*at 15000 RPM*
Average seek time	3.4 ms	3.4 ms
Average rotational delay	10 ms	2 ms
Transfer rate	16 MB/s	80 MB/s
Active power (R/W)	6.1 W	13.5 W
Seek power	13.5 W	13.5 W
Idle power	2.8 W	10.2 W

Table 2: Disk simulation parameters

Our simulation evaluation used two traces. The first one is a two-week subset of the HP Cello99 trace, which is a more recent trace of the successor system to Cello92 system [34]. Cello99 was collected on a timesharing and file server at HP Labs; it was an HP 9000 K570 class machine (4 PA-RISC CPUs) running HP-UX 10.20 with 2GB of main memory and (roughly) the equivalent of 25 high-end SCSI disks. The trace was collected at the SCSI device driver level after filtering from the file system cache.

The other trace, OLTP, was collected on a storage server connected to an IBM DB2 database server via a storage area network (SAN) driven by queries from a TPC-C-like benchmark, as shown in Figure 6.[1] The detailed setup is similar to the one described in Section 4.2. The storage system had 24 IBM Ultra SCSI 15000 RPM disks. This one-day trace includes all I/O accesses from the IBM DB2 database server to the storage system.

In the experiments, we used the first half of the traces to warm up the storage system's cache and adaptive layout. For the CR algorithm, we set the epoch length to be 1 hour by default.

We use both *stable* workloads and *dynamic* workloads to demonstrate the performance and energy effects of these schemes. These workloads are defined by how Hibernator's CR algorithm responds: for stable workloads, CR produces the same disk tier membership result for each of the epochs in the trace; for dynamic workloads, the disk tier membership varies between epochs. To get stable workloads, we selected particular 4-hour segments from the second half of each trace that had the desired property. The mean loads of the two selected 4-hour segments from OLTP and Cello99 are 140 and 50 req/s, respectively. To evaluate Hibernator and previous schemes under different loads, we replayed the 4-hour segments at various speeds.

[1] Because our experiments are not audited, the Transaction Processing Performance Council does not let us refer to our workload as "TPC-C".

The dynamic Cello99 workload is simply the second half of the two-week Cello99 trace, while the dynamic OLTP workload is assembled by sequentially replaying three different traces collected from the real system, generated by running 10, 55 and 100 clients. Figure 9(a) and 9(b) show how the load characteristics of the dynamic Cello99 and OLTP workloads change with time.

4.2 Emulated system evaluation

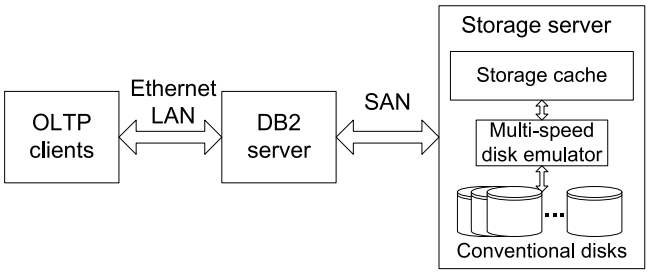


Figure 6: The emulated system.

To evaluate Hibernator in actual use, we implemented the Hibernator algorithms in our previously built storage server [43], and used 15000 RPM disks to emulate multi-speed disk drives. As shown on Figure 6, the resulting storage system emulator was connected to, and driven by, an IBM DB2 database server via a high speed SAN with a peak bandwidth of 113 MB/s and a one-way latency of 5.5 μs for a short message. Each server had dual 2.4 GHz Pentium IV processors with 512 KB L2 caches and 2.5 GB of main memory, and ran Windows 2000 Advanced Server. Due to the limited number of disks available, each layout (RAID5, PDC, Hibernator) used ten 15000 RPM IBM Ultra SCSI disks.

We emulated the 15000 RPM mode of the two-speed disk by simply passing requests through to the real one. We emulated the 3000 RPM mode by introducing an average delay of 8.4 ms before issuing a request to the real disk. We emulated disk speed transitions by delaying one request by the transition time. The active, idle and transitioning energy were charged on a per-request basis, using the parameters listed in Table 2.

We use a TPC-C-like benchmark as the OLTP workload in all experiments. The benchmark generates mixed transactions based on the TPC-C specification, and we set the think time to zero. Each run starts from the same database content, so different runs do not interfere with each other. Each experiment runs the benchmark for one hour, which is long enough for the system to yield at least 30 minutes of stable performance. To examine the end performance impact by various disk energy management solutions and layouts, we compare the transaction rate (transaction per minute). For the energy consumption, we compare the energy consumption per transaction, which is the total energy consumption divided by the total number of transactions committed within one hour. We also compare the average power (i.e., energy consumption per time unit) at each small time interval.

5. Simulation results

Since different layouts and energy control algorithms have different performance degradations, directly comparing them is difficult. Thus, we first evaluate the performance of all schemes, and then compare energy only among those schemes that can provide performance guarantees. Finally, we evaluate the cost of data shuffling to adapt to workload changes and the effects of epoch length and response time limit.

5.1 Performance guarantees

Figure 7 shows the average response time for the two traces with the seven schemes under various stable loads. In all experiments, the user-specified average response time limit is set to be 10 ms unless mentioned specifically. Here, we show only the results with stable workloads. The high-level results with dynamic workloads are similar.

As shown in Figure 7, previous schemes, such as PDC_{LD} and $RAID_{DRPM}$, do not provide average response times within the specified limit; in many cases, they can significantly increase the average response time. For example, with the default load of 140 req/s in OLTP, both PDC_{LD} and $RAID_{DRPM}$ have an average response time of 120 ms, 12 times larger than the specified limit. Comparing $RAID_{DRPM}$ with PDC_{LD}, the performance degradation by PDC_{LD} increases with increasing loads because PDC_{LD} distributes the data in an extremely uneven fashion, so that a small subset of disks become hot spots. Even if all disks spin at full speed, PDC_{LD} cannot provide the same level of performance as the baseline. With $RAID_{DRPM}$, the average response time increases gradually with increasing loads because of the increased number of disk transitions. But if the load is heavy, all disks in $RAID_{DRPM}$ will stay at full speed and thus provide the same performance as the baseline.

The results also show that our performance guarantee method works well for all cases except PDC^{+}_{LD}. For example, with OLTP, $RAID^{+}_{DRPM}$, $RAID^{+}_{CR}$ and Hibernator have average response times smaller than the specified limit. Therefore, our performance guarantee method is an effective way to control performance degradation by energy control algorithms, as long as the underlying disk layout is good enough to deliver a peak performance comparable to the baseline.

The performance guarantee method does not work with PDC_{LD} under heavy loads, because the PDC layout is unable to provide an average response time comparable to the baseline, even with all disks at full speed. For this reason, we do not compare with PDC schemes in our other simulation results. However, we do compare Hibernator with the PDC and RAID schemes in our emulated-system evaluation.

5.2 Energy consumption

We begin with a study of what happens with steady-state workloads, and then look at the more realistic situation of dynamically-varying workloads.

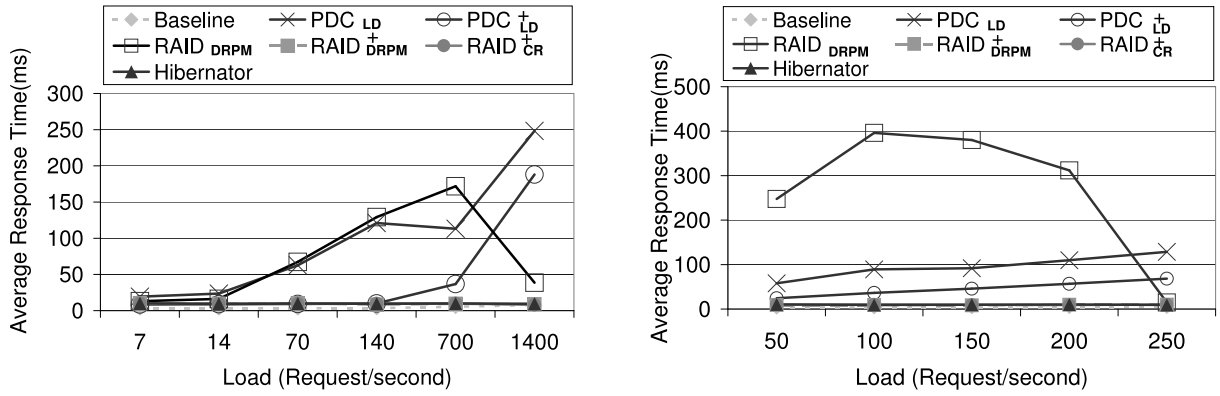
5.2.1 *Stable workloads*

Figure 8 shows the energy savings of the three performance-guaranteed schemes, namely $RAID^{+}_{DRPM}$, $RAID^{+}_{CR}$ and Hibernator, for the stable workloads. The energy savings are computed over the baseline case.

First, the energy savings from $RAID^{+}_{DRPM}$ are small: less than 10% for OLTP and almost zero for Cello99. DRPM changes the disk speeds frequently and so can significantly degrade performance, due to the long disk speed transition time. As a result, $RAID^{+}_{DRPM}$ is frequently disabled by the performance guarantee method and all disks are forced to full speed to meet performance requirement, leading to poor energy savings.

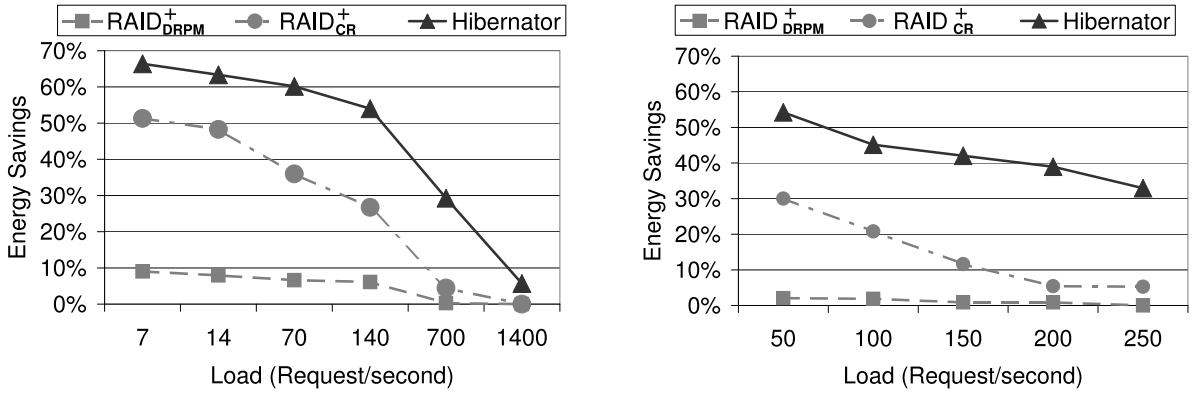
The two CR schemes, $RAID^{+}_{CR}$ and Hibernator, provide significantly more energy savings than $RAID^{+}_{DRPM}$. For example, for OLTP with 14 req/s, these two schemes can save 48% and 63% energy, respectively, over the baseline. This savings is because the CR algorithm adapts energy at coarse granularity, and considers workload characteristics and performance requirements to determine the best disk speed settings to minimize energy consumption.

Among all schemes, Hibernator provides the most energy sav-



(a) OLTP (b) Cello99

Figure 7: Average response time for different schemes under various stable loads. (Note the x-axis scale in (a) is not linear.)



(a) OLTP (b) Cello99

Figure 8: Energy savings for different schemes with performance guarantee under various stable loads. (Note the x-axis scale in (a) is not linear.)

ings. For example, for OLTP with 7 req/s, Hibernator can save 65% energy, whereas $RAID^{+}_{CR}$ and $RAID^{+}_{DRPM}$ provide only 50% and 10% energy savings, respectively. Hibernator saves more energy than $RAID^{+}_{CR}$ because Hibernator uses a multi-tier layout with active data evenly distributed across first-tier full speed disks to provide good performance. Such a layout allows more disks to stay in low speed tiers without significantly affecting the performance. As $RAID^{+}_{CR}$'s energy savings is always between Hibernator and $RAID^{+}_{DRPM}$, we do not present its results in subsequent sections, in order to make our figure presentation clearer.

5.2.2 *Dynamic workloads*

Figure 9 shows how Hibernator and $RAID^{+}_{DRPM}$ adapt to workload changes. For both OLTP and Cello99, Hibernator has lower average power than $RAID^{+}_{DRPM}$. For example, in the time period 1–4 hours in the OLTP workload, Hibernator has an average power of 50–75% of the baseline, whereas $RAID^{+}_{DRPM}$'s average power is more than 90% of the baseline. Thus, Hibernator provides a 15% energy savings for OLTP, and a 48% energy savings for Cello99, whereas $RAID^{+}_{DRPM}$'s energy savings are only 3% and 16%, respectively. $RAID^{+}_{DRPM}$ cannot save as much energy as Hibernator, since all disks are often forced to full speed to meet the performance requirement, whereas this does not happen often with Hibernator.

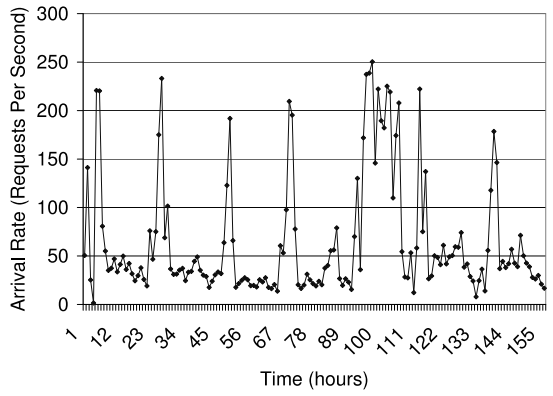
Figure 9(c) shows that when the workload becomes heavy, Hibernator can quickly adapt its speed setting and layout to meet the performance goal, while achieving maximal energy conservation. At a few points in Cello99 when the load prediction is inaccurate, the CR algorithm alone cannot deliver the expected performance. In this case, the performance guarantee mechanism immediately forces all disks to full speed to satisfy the performance requirement, resulting in the high sharp spikes in Figure 9(c).

The energy savings for OLTP are smaller than that for Cello99 because OLTP's loads are much heavier than Cello99. Under heavy loads (e.g., the time period 4–10 hours), all disks need to stay at full speed in order to meet the performance goal. Therefore, there are no energy savings at all during this period, and since this is 50% of the entire workload, the overall energy savings are small, too.

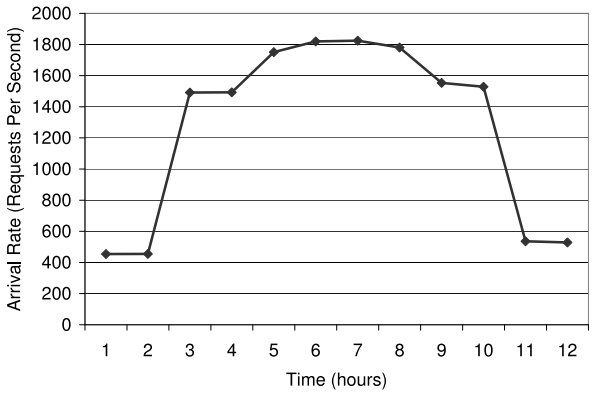
5.3 The cost of data shuffling

Foreground traffic	*Phase length*		*Mean response time*		
	measured	*model*	*no shuffling*	*measured*	*model*
none	22.1 min	22.1 min	–	–	–
1400 req/s	27.8 min	30.0 min	6.35 ms	8.82 ms	9.1 ms

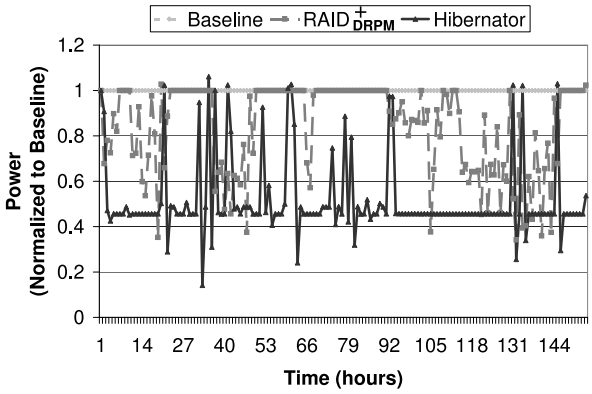
Table 3: The cost of data shuffling with and without foreground OLTP workloads. The "no shuffling" column represents the average response time during this period if there is no background shuffling.



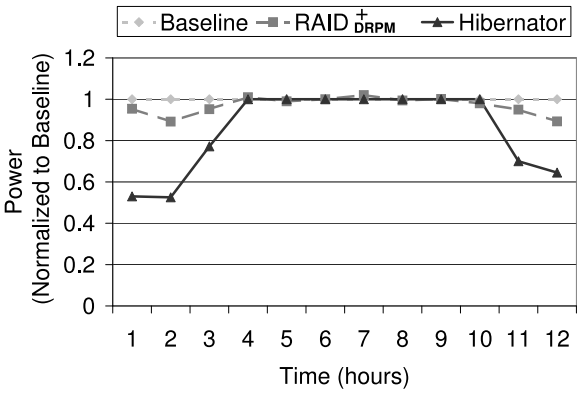
(a) Cello99



(b) OLTP



(c) Cello99



(d) OLTP

Figure 9: Top: trace workload characteristics. Bottom: energy adaptation for dynamic OLTP and Cello99 workloads for a performance goal of 10ms. Energy savings achieved for OLTP were: $RAID^{+}_{DRPM}$: 3%, Hibernator: 15%; and for Cello99: $RAID^{+}_{DRPM}$: 16%, Hibernator: 48%.

To evaluate the effects of our data shuffling approach for workload adaptation on the response time of foreground requests, we conduct an experiment that moves a disk from a low-speed tier to a 22-disk full-speed tier. The random shuffling algorithm was used to ensure that future loads could be evenly distributed across the 23 full-speed disks.

Table 3 shows that the average response time of foreground requests only increases by 2.5 ms (from 6.35 ms to 8.82 ms) due to background shuffling traffic. It also shows that data shuffling for an entire 18 GB disk can be done almost as quickly as sequentially scanning the disk twice. With foreground requests, the shuffling duration is increased slightly from 22.1 minutes to 27.8 minutes. In summary, random shuffling takes advantage of disks' high sequential bandwidth and can be done quickly with only a small overhead on the average response time of foreground requests.

5.4 The effect of a response time goal

Figure 10 shows the effects of the response time limit on energy savings by Hibernator. In one extreme, the response time limit is small so that all disks need to spin at full speed. In this case, no energy savings are possible. At the other extreme, when performance is not critical (the response time limit is large), all the disks can be spun down, reaching the maximum energy savings of 70%. In between, Hibernator saves energy by putting more disks into the low-speed tiers as the performance restrictions are eased. The effects of the response time limit under the Cello99 are similar.

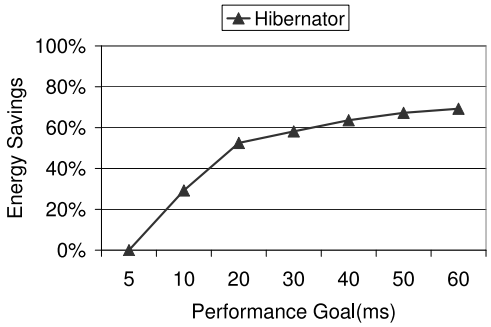


Figure 10: The energy effects of response time limit under OLTP workload of 700 req/s.

5.5 The effect of epoch length

Figure 11 shows the energy effects of epoch length on Hibernator under the dynamic Cello99 workload. Once the epoch length is large enough to accommodate the disk speed transition and reorganization period, the energy savings are insensitive to epoch length across a broad range. This is because most of time the Cello99

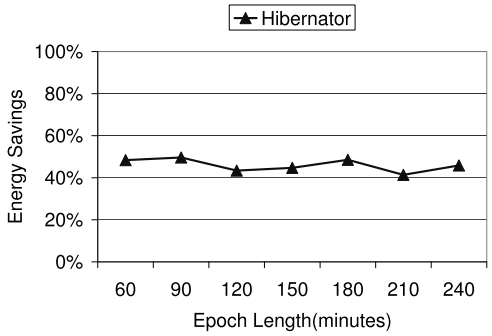


Figure 11: The energy effects of epoch length on Hibernator under the dynamic Cello99 workload.

workload is relatively steady and at a few points where the load prediction is inaccurate, regardless of the epoch length, the performance guarantee method forces all disks up to meet the performance requirement. The energy effects of epoch length on Hibernator under the OLTP workload are similar.

6. Emulated system results

Figure 12 shows the results from an emulated storage system that is connected to IBM DB2 running the TPCC-like benchmark. Besides validating our simulation results, we used this environment to evaluate the impact of energy management on the transaction rate of our database application.

The emulated-system results demonstrate that only Hibernator can save significant energy, while providing performance comparable to the baseline (energy-unaware RAID5). For example, Hibernator provides an average response time of 6.8 ms, only 3% higher than the baseline (6.6 ms) and thereby has an OLTP transaction rate similar to the baseline. Moreover, Hibernator saves the highest percentage (29%) of energy among all five energy-aware schemes. This suggests that Hibernator is a practical and energy-efficient scheme to be used in data center disk arrays.

The two RAID schemes, $RAID_{DRPM}$ and $RAID^{+}_{DRPM}$, produce little energy savings. $RAID_{DRPM}$ does not have a performance guarantee and degrades the database transaction rate by 30%. Even though it allows disks to stay at lower speeds more often than Hibernator (as shown on the left figure of Figure 12(b)), $RAID_{DRPM}$ still consumes almost the same energy as the baseline, due to the increased execution time to finish the transactions (energy = power $\times$ time). In contrast, $RAID^{+}_{DRPM}$ does not degrade performance significantly, but it saves little energy over the baseline because disks are often forced to full speed.

PDC_{LD} performs as expected: it is able to put many disks into low speeds as shown in Figure 12(b), because the load distribution across disks is extremely uneven – but this also means that it significantly degrades the database transaction rate by 47%. Combining both effects, it has the same amount of energy consumption per transaction as Hibernator, but it fails the performance requirement. (As in the simulation results, PDC cannot always meet the performance goal even if all disks spin at full speeds, and the same happens to PDC^{+}_{LD}.)

Our emulated-system results for $RAID^{+}_{DRPM}$ and Hibernator accurately match those in simulation. Both the average I/O response time and the energy savings by both Hibernator and $RAID^{+}_{DRPM}$ measured from emulated systems are very similar to the corresponding simulation results for OLTP with the default load (140 req/s) shown in Figure 7 and Figure 8.

However, the performance degradation of $RAID_{DRPM}$ in the emulated system is less than the corresponding result in simulation because of the inter-request dependency seen from the database. When requests are delayed in the emulated system, subsequent requests are also delayed – an effect not seen in the open-loop trace replay we used in the simulations. This tends to lower the offered I/O load in a system suffering slower performance due to energy management. The discrepancies for performance guarantee schemes such as $RAID^{+}_{DRPM}$ and Hibernator are much less pronounced because those schemes do not add much delay, so the inter-request arrival rate does not change significantly in the emulated system.

7. Conclusions

Current disk array energy management schemes are impractical for use in real data centers because they do not balance performance and energy conservation well. Our solution to this problem is the Hibernator disk array design, which combines several new ideas for disk array energy management: (1) a coarse-grain disk-speed selection algorithm (CR) that supports multiple data layouts; (2) an energy-efficient data layout that requires no extra disks and sacrifices no reliability; (3) efficient, dynamic data reorganization algorithms for both macro- and micro-level adjustments; and (4) a response-time performance guarantee method that can be coupled with many energy control algorithms.

Using both simulations with real system traces and a multi-speed disk emulation with a commercial database engine (IBM DB2), our results show that Hibernator is effective in overcoming the current limitations in disk array energy management, thereby making it practical in real systems. Specifically, Hibernator saves the most amount of energy when compared to previous solutions, while providing OLTP transaction rate comparable to traditional performance-oriented layouts such as RAID5 without any energy management.

There are many potential areas where our results could be extended. For example: performing real system energy measurement and performance evaluation by using SONY's multi-speed disks; broadening the workloads used, to include decision support systems (DSS), mixed workload types, and larger-scale ones; extending CR to consider response-time variance as well as its mean; leveraging intelligent disk scheduling algorithms, such as free block scheduling [24], to improve the efficiency of disk reorganization and data migration; and coupling Hibernator's algorithms with other disk array redundancy schemes and other optimized techniques, such as parity declustering and parity logging.

In summary, we believe that Hibernator is an attractive disk array energy management design: one that offers significant energy savings, while meeting performance guarantees, using simple, effective algorithms that themselves have wider applicability.

Acknowledgments

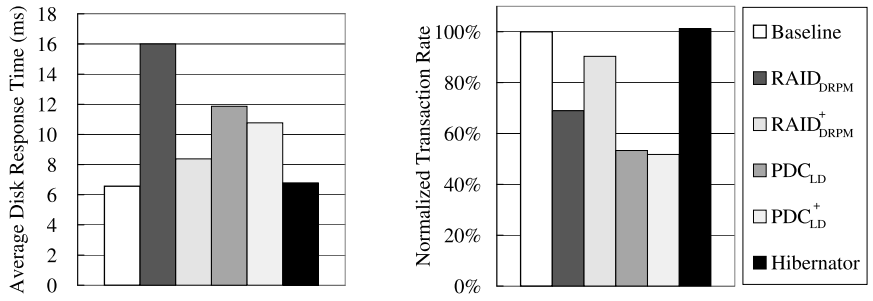
The authors would like to thank the shepherd, Lorenzo Alvisi, and the anonymous reviewers for their feedback. This research is supported by NSF CCR-03-12286, NSF Career Award (CNS 03-47854), NSF EIA 02-24453, IBM SUR grant, and IBM Faculty award.

References

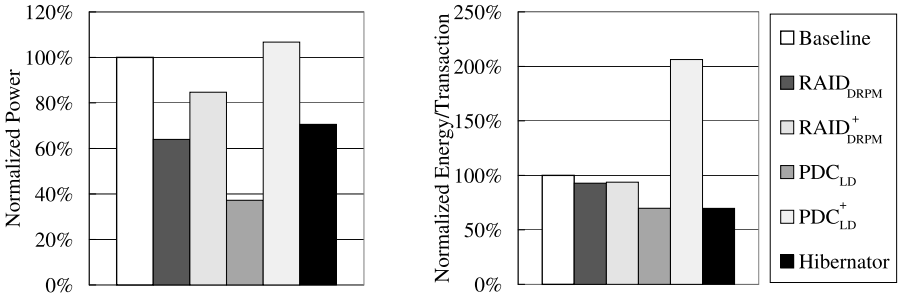
[1] ILOG CPLEX 7.0 documentation home page. http://www.ise.ufledu /ilog/cplex70/.

[2] Panel computer hard disk drive precautions. http://www.pro-face. com/support/technical/00apr3.htm.

[3] Symmetrix 3000 and 5000 Enterprise Storage Systems product description guide. http://www.emc.com/, 1999.



(a) Performance



(b) Energy Consumption

Figure 12: Performance and energy consumption comparison on an emulated storage system. The database transaction rate, power (energy/time), and energy consumption are normalized to the baseline. The energy consumption is evaluated for the specified number of transactions.

[4] Dell PowerEdge 6650 executive summary. http://www.tpc.org/results/individual_results/Dell/dell_6650_010603_es.pdf, March 31 2003.

[5] HP announces "smart" cooling solution for data centers. http://www.hp.com/hpinfo/newsroom/press/2003/030304b.html, March 4 2003.

[6] HP Integrity RX5670 Cluster 64P Executive Summary. http://www.tpc.org/results/individual_results/HP/HP%20Integrity%20rx5670%20Cluster%2064P_ES.pdf, Dec. 2003.

[7] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *Proc. of the 32nd Annual International Symposium on Microarchitecture*, 1999.

[8] American Power Convension, Write Paper. Determining total cost of ownership for date centers and network room infrastructure. ftp://www.apcmedia.com/salestools/CMRP-5T9PQG_R2_EN.pdf, 2003.

[9] E. Anderson, M. Hobbs, K. Keeton, S. Spence, M. Uysal, and A. Veitch. Hippodrome: running circles around storage administration. In *Proc. of the First USENIX Conference on File and Storage Technologies*, Jan. 2002.

[10] E. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *Proc. of the 17th International Conference on Supercomputing*, June 2003.

[11] J. Chase and R. Doyle. Balance of power: energy management for server clusters. In *Proc. of the 8th Workshop on Hot Topics in Operating Systems(HotOS)*, May 2001.

[12] P. Chen, E. Lee, G. Gibson, R. Katz, and D. Patterson. RAID: high-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

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

[14] D. Colarelli and D. Grunwald. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, pages 1–11, June 2002.

[15] F. Douglis, P. Krishnan, and B. Bershad. Adaptive disk spindown policies for mobile computers. In *Proc. of the 2nd USENIX Symposium on Mobile and Location Independent Computing*, Apr. 1995.

[16] G. Ganger, B. Worthington, and Y. Patt. The DiskSim simulation environment - version 2.0 reference manual. http://www.pdl.cmu.edu/DiskSim/disksim2.0.html.

[17] C. Gniady, Y. C. Hu, and Y.-H. Lu. Program counter based techniques for dynamic power management. In *Proc. of the 10th International Symposium on High Performance Computer Architecture*, 2004.

[18] R. Golding, P. Bosch, C. Staelin, T. Sullivan, and J. Wilkes. Idleness is not sloth. In *Proc. of the USENIX Winter Technical Conference*, Jan. 1995.

[19] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. DRPM: dynamic speed control for power management in server class disks. In *Proc. of the 30th International Symposium on Computer Architecture*, June 2003.

[20] D. P. Helmbold, D. D. E. Long, T. L. Sconyers, and B. Sherrod. Adaptive disk spin-down for mobile computers. *Mobile Networks and Applications*, 5(4):285–297, 2000.

[21] Ultrastar 36Z15 Datasheet. http://www.hitachigst.com/hdd/ultra/ul36z15.htm, Jan. 2003.

[22] D. Li, P. Gu, H. Cai, and J. Wang. EERAID: energy-effi cient redundant and inexpensive disk array. The 11th ACM SIGOPS European Workshop, Sept. 2004.

[23] X. Li, Z. Li, F. David, P. Zhou, Y. Zhou, S. Adve, and S. Kumar. Performance directed energy management for main memory and disks. In *Proc. of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.

[24] C. Lumb, J. Schindler, G. Ganger, D. Nagle, and E. Riedel. Towards higher disk head utilization: Extracting "free" bandwidth from busy disk drives. In *Proc. of the Fourth USENIX Symposium on Operating Systems Design and Implementation*, Oct. 2000.

[25] G. Magklis, M. L. Scott, G. Semeraro, D. H. Albonesi, and S. Dropsho. Profi le-based dynamic voltage and frequency scaling for a multiple clock domain microprocessor. *ACM SIGARCH Computer Architecture News*, 31(2):14–27, 2003.

[26] A. Merchant and P. S. Yu. Analytic modeling and comparisons of striping strategies for replicated disk arrays. *IEEE Transactions on Computers*, 44(3):419–33, 1995.

[27] B. Moore. Take the data center power and cooling challenge. Energy User News, August 27 2002.

[28] F. Moore. More power needed. Energy User News, November 25 2002.

[29] K. Okada, N.Kojima, and K. Yamashita. A novel drive architecture of HDD: multimode hard disc drive. In *Proc. of the International Conference on Consumer Electronics*, 2000.

[30] A. E. Papathanasiou and M. L. Scott. Energy effi cient prefetching and caching. In *Proc. of the USENIX Annual Technical Conference*, June 2004.

[31] D. Patterson, G. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks (RAID). In *Proc. of the ACM International Conference on Management of Data*, 1988.

[32] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In *Proc. of the 18th International Conference on Supercomputing*, June 2004.

[33] P. Ranganathan. The power management challenge: Getting the next 100x. Keynote presentation at the 2nd workshop on optimizations for DSPs and Embedded Systems, March 12 2004.

[34] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Proc. of the USENIX Winter Technical Conference*, 1993.

[35] S. Savage and J. Wilkes. AFRAID – a frequently redundant array of independent disks. In *USENIX Annual Technical Conference*, pages 27–39, 1996.

[36] E. Shriver, A. Merchant, and J. Wilkes. An analytic behavior model for disk drives with readahead caches and request reordering. In *Proc. of the International Conference on Measurement and Modeling of Computer Systems*, 1998.

[37] S. W. Son, G. Chen, and M. Kandemir. Disk layout optimization for reducing energy consumption. In *Proc. of the 19th International Conference on Supercomputing*, June 2005.

[38] G. Weikum, P. Zabback, and P. Scheuermann. Dynamic fi le allocation in disk arrays. In *Proc. of the ACM International Conference on Management of Data*, 1991.

[39] A. Weissel, B. Beutel, and F. Bellosa. Cooperative I/O: a novel I/O semantics for energy-aware applications. In *Proc. of the Fifth USENIX Symposium on Operating Systems Design and Implementation*, Dec. 2002.

[40] J. Wilkes, R. A. Golding, C. Staelin, and T. Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, 1996.

[41] E. Wusterenhoff. Service level agreement in the data center. http://www.sun.com/blueprints/0402/sla.pdf, Apr. 2002.

[42] H. Yada, H. Ishioka, T. Yamakoshi, Y. Onuki, Y. Shimano, M. Uchida, H. Kanno, and N. Hayashi. Head positioning servo and data channel for HDDs with multiple spindle speeds. *IEEE Transactions on Magnetics*, 36(5):2213–2215, Sept. 2000.

[43] Y. Zhou, A. Bilas, S. Jagannathan, C. Dubnicki, J. F. Philbin, and K. Li. Experiences with VI communication for database storage. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 257–268, May 2002.

[44] Q. Zhu and Y. Zhou. Power aware storage cache management. *IEEE Transactions on Computers*, 54(5):587–602, May 2005.