LogStore: Toward Energy-Proportional Storage Servers

Wei Zheng Ana Paula Centeno Frederic Chong[†] Ricardo Bianchini
Rutgers University [†]University of California at Santa Barbara
{wzheng,anapaula,ricardob}@cs.rutgers.edu chong@cs.ucsb.edu

ABSTRACT

Storage servers consume significant amounts of energy and are highly non-energy-proportional. Fortunately, researchers have proposed disks that adjust their rotation speeds based on utilization. However, these disks either pose significant engineering challenges (by having too many speeds) or adjust behavior only at a coarse grain (by having just two speeds). In this paper, we propose LogStore, a storage system that enables two-speed disks to achieve substantially increased energy proportionality and, consequently, lower energy consumption. Our evaluation with real I/O workloads shows that LogStore can decrease energy consumption significantly, even for very tightly provisioned servers, with a negligible impact on aggregate throughput.

Categories and Subject Descriptors

D.4 [**Operating systems**]: Storage management

Keywords

Energy conservation, multi-speed disks

1. INTRODUCTION

Many enterprises use storage servers, each of which with many high-performance disks and RAID redundancy, behind their database and/or file servers. These servers provide fast and reliable access to data, but also consume significant amounts of energy. For example, small-sized storage servers from multiple vendors provide 16-24 drive bays. When using fast SCSI disks, each of these servers can consume in excess of 400W just from the drives. Larger storage servers can have hundreds or even thousands of disks, which typically represent more than 50% of the total system power. For example, the EMC Symmetrix VMAX [7], an entry-level storage server, is rated at more than 8KW with disks representing 60% of this total power.

High peak power is not a problem in itself, as peak I/O loads occur rarely. The worse problem is that storage servers are highly non-energy-proportional, i.e. they consume a large fraction of their peak power even at low utilization. Recent years have seen advances in processor energy proportionality [11], as well as proposals for improving memory

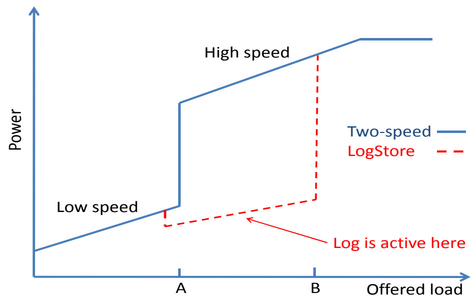

Figure 1: Main idea of LogStore.

energy proportionality [4, 6]. Even if storage servers take full advantage of these innovations, they will still exhibit limited proportionality due to their use of conventional disks.

The reason is that the energy consumption of conventional disks is dominated by the energy required to keep them spinning [9]. In fact, disk energy is a quadratic function of the rotation speed [9]. Based on these observations, researchers have proposed disks that can operate in one of two [2] or more speeds [9]. Under lighter loads, these disks can dynamically change speed and serve requests at lower speed. A few years earlier, Sony had produced a two-speed disk that required a re-start to change speeds [12]. Since these early works, some vendors started producing two-speed disks that can dynamically change speeds, e.g. [10]. However, these disks only perform accesses at the high speed; the low speed is used solely to conserve energy during periods of idleness. Since servers rarely exhibit idle periods that are long enough to justify speed changes, existing two-speed disks are useless for storage servers.

We argue that disks capable of serving accesses at multiple speeds are the key to disk energy proportionality, in the same way that processors capable of Dynamic Voltage and Frequency Scaling (DVFS) are more energy proportional. Our preference is for two-speed disks, since implementing more speeds poses engineering challenges, e.g. adjusting sensing mechanisms to the many distances between the disk head and the magnetic medium, and high manufacturing costs. However, two-speed disks allow only coarse proportionality.

Our goal is then to create *software* to improve the energy proportionality of two-speed disks that can serve accesses in both speeds, and thereby reduce energy consumption. Our storage system, called LogStore, increases energy proportionality by storing data into a (disk-based) log data structure at the utilization boundary between the two speeds. The improved write performance and lower disk utilization resulting from the log enable disks to stay in low speed longer. Figure 1 illustrates the main idea, showing disk power consumption as a function of the disk request rate (utilization). The full curve represents a sample two-speed disk, whereas the dashed curve represents the same two-speed disk when managed by LogStore. As we can see, LogStore allows the disk to serve a higher request rate still in low speed. Under LogStore, the speed transition occurs at offered load B, rather than offered load A. LogStore seeks to delay the transition without affecting throughput. However, the delayed transition means that more requests are served in low speed, which may increase the average response time.

LogStore replays the log against the original disk blocks when utilization is low. Since a disk cannot be accessed during a speed transition, storage systems with two-speed disks require data redundancy. When a transition is necessary, all accesses to the transitioning disk are directed to the disk(s) containing the redundant data. We designed LogStore for RAID 1, since it enables the highest energy savings. Nevertheless, LogStore can be easily adapted to any RAID level that uses block mirroring (e.g., RAID 1+0, RAID 0+1). We leave other RAID levels as future work.

We built a prototype of LogStore in user-space. To mimic two-speed disks that can serve accesses in both speeds, we also built a two-speed disk emulator (via delay injection) into the Linux kernel. We compare LogStore with RAID 1 implemented using: (1) conventional one-speed disks; (2) two-speed disks without LogStore; and (3) two-speed disks plus Diverted Accesses (DIV) [13]. Our results for real write-intensive workloads show that LogStore can decrease energy consumption by roughly 35%, compared to conventional disks, when the storage system is very tightly provisioned for the workload. Compared to two-speed disks without and with DIV, these savings are roughly 20% and 27%, respectively. Under more realistic provisionings, the LogStore energy savings can reach 81%, 57%, and 73%, respectively. Our results for these workloads also show that LogStore has a negligible impact on aggregate throughput. However, as expected, it does degrade response time somewhat (up to 2.4ms). For read-only workloads, LogStore behaves as well as the best of its competitors, still achieving the highest energy savings, having no impact on aggregate throughput, and minimally increasing response times. Thus, LogStore behaves well for throughput-oriented workloads, regardless of their write intensity.

2. BACKGROUND AND RELATED WORK

Two-speed disks. We study two-speed disks that are similar in functionality to those proposed in [2], but with contemporary seek performance. As in that study, the disks can serve requests from both speeds and speed transitions are based on utilization. However, we assume transitions triggered by the kernel, rather than the disk controller. For the kernel to select speeds, it queries the disks about their utilizations. We measure utilization as the disk busy time as a fraction of the wall-clock time. As suggested in [2], we take two measures to improve stability and reduce the impact of extraneous load spikes: filtering and smoothing of utilization observations. We filter the first observation that is more than twice as high/low as the previous one. After this first disregarded spike/valley, the filter takes new high/low observations into account. Using those observations that pass the filter, we smooth them using an exponentially weighted moving average with alpha equal to 0.875 [2].

Disks with two or more speeds continue to draw attention, e.g. [8, 22, 23]. However, a common concern is reliability. We argue that this concern is unfounded for two reasons: (1) some commercial server disks already switch to a lower speed when they become idle (e.g., [10]); and (2) commercial server disks are capable of many hundreds of thousands of stop/start cycles (which impose a heavier burden than speed transitions). For example, many Hitachi disks (e.g., [10]) are rated at 600,000 cycles. Since each disk is typically replaced after 3 years of use, the system would have to change speeds more often than once every 158 seconds on average to exceed the maximum number of cycles during the disk's lifetime.

Although commercial two-speed disks do not serve accesses in both speeds, there is no fundamental reason why disks that can do so cannot be manufactured [9]; the barrier is in achieving a high benefit-to-cost ratio. LogStore demonstrates that system-level techniques can significantly increase the potential benefits of such disks, creating a greater incentive for manufacturers to produce them.

Storage energy conservation. Many works have proposed techniques for reducing disk energy consumption, e.g. [2, 3, 9, 13, 17, 19, 20, 21, 24]. Zhu *et al.* [24] have exploited two-speed disks in enterprise storage servers as we do. However, they changed speeds only at the granularity of hours, relying on data migration for finer-grained adjustments. Our system relies on more frequent speed transitions and avoids data reorganization altogether due to their potentially high energy and performance overheads. Pinheiro *et al.* [13], Weddel *et al.* [20], and Yao and Wang [21] studied how to leverage redundancy to conserve energy using conventional disks. Unfortunately, the use of conventional disks limits the energy proportionality and savings that can be achieved in storage servers. Verma *et al.* [19] used selective data replication to improve proportionality. However, replication can be a significant overhead when workloads have large working sets and/or change frequently.

Still leveraging redundancy, Amur *et al.* [1] and Thereska *et al.* [18] focused on clustered storage systems. They achieve proportionality by intelligently placing data replicas across servers (their conventional disks). The number of disks and the amount of redundancy are typically larger in a clustered system than in a single storage server, enabling finer-grained energy proportionality and greater savings.

3. LOGSTORE

3.1 Design

The key observation behind LogStore is that sequential disk access leads to higher disk throughput, and lower response time and occupancy. Thus, if we could transform many random disk accesses into sequential ones near a transition to high speed, we would not need to effect the transition until a higher load is offered to the disk. A well-known way to achieve this transformation is by using an on-disk log for writes [14]. *LogStore uses one log per disk.* Unfortunately, a log may also make some disk reads more expensive, since sequential read accesses may become random when directed to the log. LogStore mitigates this problem in two ways: the log is only used when the disk is about to transition to high speed; and LogStore sits behind the file system buffer cache, allowing most reads to hit it. In our terminology, each block has a "regular location" selected by the file system and possibly a (temporary) "log location".

In more detail, LogStore implements three modes of operation: regular, log, and log-replay. In the regular and log-replay modes, LogStore writes data to their regular

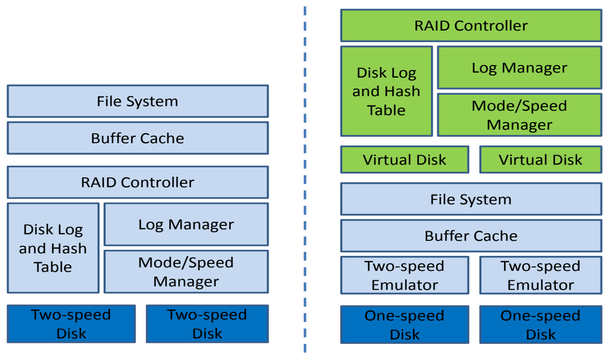

Figure 2: Design (left) and prototype (right). Green = user space; Light blue = kernel; Dark blue = hardware.

locations. In the log mode, it writes data to the end of the log. In the log-replay mode, LogStore replays the log against the blocks' regular locations. We detail these modes and the transitions between them below.

LogStore comprises five components: the on-disk log for each disk, an in-memory hash table for each log, a log manager, a software RAID controller, and software mode/speed manager. Figure 2(left) illustrates these components. As aforementioned, we design LogStore for RAID 1, but it can be easily adapted to handle other RAID levels that use block mirroring. In fact, the RAID controller is the only piece of LogStore that depends on the RAID level.

Data structures. LogStore maintains two data structures for each disk: the on-disk log and an in-memory hash table. The space for the log is statically allocated. When a disk is in the log mode, LogStore appends all block writes to the disk's log. In our experiments, we never need more than 3 GBytes for a log. (LogStore is not a log-structured file system; it does not implement log cleaning or compaction.) Since the log is just an optimization, our implementation limits each log to this size. Each log entry contains the address of the written block and the written data itself. This is enough information to reconstruct the associated hash table, when recovering from a server crash.

Each log has a hash table to track the location of the logged blocks. Each table entry has a field for the block's regular address, one for the log address, and a pointer to any other entry hashing to the same slot. The table is small. For example, each 3-GByte log requires only 15 MBytes (when the log is full) for its table, assuming 4-KByte blocks.

Log manager. It is responsible for accessing the log and replaying it against the blocks' regular locations.

In the regular and log-replay modes, this manager directs a block write to its regular location. If the hash table shows that the log includes this block, the manager removes the table entry and invalidates the log entry. The invalidation is needed for server crash recovery. A block read in these modes accesses the regular location, if the table does not have an entry for the block. If it does, the read is directed to the proper log entry. The modes differ in that this manager replays entries from the log against the regular locations in the log-replay mode, but not in the regular mode.

In the log mode, LogStore appends the disk writes to the log. If an entry for the same block exists in the hash table, the manager updates the table entry to point to the new log entry. If a table entry for this block does not exist, LogStore creates one. The manager handles reads in this mode in the same way mentioned above for the other modes.

In the log-replay mode, the log manager starts replaying the log from its beginning, using a separate thread, as soon as the mode/speed manager changes the mode to log-replay. For each block in the log, the thread checks the hash table. If it finds the block there, it writes the data to its regular location and removes the entry. Otherwise, it skips the log entry. The thread replays requests when disk traffic is low to avoid degrading foreground performance.

Software RAID controller. It forms the top part of LogStore. Our design for RAID 1 with two-speed disks allows the disks of each mirrored pair to spin at different speeds. The controller allows only one speed transition at a time for each pair. Write accesses are directed to both disks in the pair. For disks with non-volatile controller caches, it may be possible to buffer the writes in the caches, regardless of the disks' speeds. Whether this approach would work depends on the write rate and the size of the caches. As we discuss later, our prototype uses a different implementation that is more generally applicable.

The controller is oblivious to the modes of operation, so the mirrored disks can be in different modes as well. *This means that the pair of disks are mirrors in terms of content, but not in terms of the exact location of the latest version of each block.* For example, a write to a block may be written to its regular location on one disk and to the log of the other.

Before deciding on the destination disk for each read, the controller predicts what the utilizations would be in case both mirrored disks were at low speed. The “Balance” threshold defines the utilization starting at which at least one disk must be in high speed. If the sum of the utilizations is lower than this threshold (both disks can be at low speed), the controller evenly distributes the reads across the disks. If not (one disk at least must be in high speed), it concentrates the reads on one disk, only using the other when the first disk is nearly saturated.

LogStore uses the fraction of disk busy time as a measure of utilization. It assumes that the disks provide a standard interface (e.g., SMART [16]) by which the host can query the disk controller about the average busy time, and the average seek, rotation, and data transfer times. Using these data (and assuming the near-future will resemble the near-past), LogStore can predict what the utilization would be at low speed using the following equations: $T_{high} = T_{seek} + T_{rotation} + T_{transfer}$, $T_{low} = T_{seek} + \frac{S_{high}}{S_{low}} \times (T_{rotation} + T_{transfer})$, and $U_{low} = \frac{T_{high}}{T_{low}} \times U_{high}$, where T_{high} and T_{low} are the average access times at high and low speed, S_{high} and S_{low} are the high and low rotational speeds, and U_{high} and U_{low} are the high and low utilizations, respectively. Given U_{high}, we can compute U_{low} and vice-versa.

Mode/speed manager. It is responsible for setting the mode of operation and the rotational speed of each disk. It also divides the execution into epochs.

This manager uses a thread to select each disk's mode based on the policy in Figure 3. It considers 4 utilization thresholds set by the user: MaxRegular, MinRegular, MaxReplay, and MinLog. MaxRegular specifies the maximum utilization before we transition to the log mode. MinRegular specifies the regular-mode utilization up to which we switch to the log-replay mode. MaxReplay specifies the utilization up to which we replay the log. MinLog specifies the minimum utilization before we switch to the regular mode.

```
 1. At the beginning of each epoch:
 2.   Determine disk util in the last epoch
 3.   If curr_speed == low && curr_mode == regular
 4.      If util >= MaxRegular
 5.         Change mode to log and exit
 6.      If util <= MinRegular && log != NULL
 7.         Change mode to log-replay and exit
 8.   If curr_speed == low && curr_mode == log &&
            predicted util in regular mode <= MinLog
 9.      Change mode to regular and exit
10.   If curr_speed == low && curr_mode == log-replay
11.      If log == NULL || util >= MaxReplay
12.         Change mode to regular and exit
13.   If curr_speed == high && curr_mode == log-replay
            && log == NULL
14.      Change mode to regular and exit
```

Figure 3: Policy for mode selection.

```
1. At the beginning of every epoch and more than 3
        epochs since the last speed transition:
2.   Determine disk util in the last epoch
3.   If curr_speed == low && util >= GoUp
4.      Change mode to log-replay
5.      Transition to high speed and exit
6.   If curr_speed == high && predicted util at
           low speed, log mode <= GoDown
7.      Change mode to log
8.      Transition to low speed
```

Figure 4: Policy for speed selection.

This manager may need to predict the utilization of a disk if it were to transition from the log mode to the regular mode (line 8). We predict this utilization by using the formulas above (replacing *high* for *log* subscripts), and the seek and rotational time statistics from before the disk went into the log mode to compute T_{low}.

If the manager does not change the mode, it uses the policy in Figure 4 to select disk speeds. To allow utilization to settle after a speed transition, it does not change speeds again for a few epochs. The GoUp and GoDown thresholds are also pre-defined by the user. They specify the utilizations at which the disk should be transitioned from low to high speed and from high to low speed, respectively.

3.2 Implementation

For simplicity, we implemented the LogStore prototype at the user-level on top of Linux, as depicted in Figure 2(right). Each disk is implemented as a large file, called "Virtual Disk", with 4KByte blocks. The functionality of the prototype components is as described above, except for the software RAID controller.

On a block read access, our prototype RAID controller uses a new system call we added to Linux to find out if the corresponding "mirror block" (i.e., the same block at its regular location on the other disk) is in the buffer cache. If it is, the controller performs the read to the other disk. If it is not in the buffer cache, the controller issues the request for the accessed block. This implementation achieves the same behavior as an in-kernel design (left side of Figure 2) would have. Unfortunately, the same cannot be done for block writes. Writes have to be performed to the two disks, causing two copies of the block to reside in the buffer cache. In addition, writes to the logs also take up space in the cache. Thus, our prototype caches less useful data than an in-kernel design would, i.e. the LogStore performance and energy results would be better for the latter system.

The fact that one of the disks can be transitioning speeds when a write arrives also complicates our RAID controller. Our prototype does not assume that disks have large non-volatile caches. Instead, the controller records each write in main memory while the write has not completed at both disks. In addition, the controller also flags a write as complete to the file system as soon as it is performed at either of the disks. If a disk is transitioning speeds when a write arrives, the controller performs the write at the other disk. When the transition completes, it performs the postponed write from the memory record. The prototype does not guarantee that all written data can be recovered after a server crash: blocks with pending writes at the time of the crash may be lost. Except for writes during speed transitions, this is the same semantics that existing software RAID controllers provide [5]. To deal with each transition, the prototype could perform two actions: complete all pending writes at both disks before starting the transition; and set (reset) a persistent flag before (after) the disk starts (ends) a transition. Because transitions are infrequent, these actions would have negligible overhead.

Parameter	Low speed 3K rpm	High speed 10K rpm
P_{idle}	0.5 W	6 W
P_{access}	2.5 W	9 W
T_{change}	7 sec	7 sec
E_{up}	123 J	–
E_{down}	–	17 J

Table 1: Defaults for the two-speed disks. P_{idle} = power when idle; P_{access} = power when accessing; T_{change} = time for speed transition; E_{up} = energy to transition to high speed; E_{down} = energy to transition to low speed.

We emulate the disks we model in the kernel using conventional (one-speed) disks. The disks operate at 15K rpm, whereas we model a 10K rpm high speed and a 3K rpm low speed. Our emulation uses careful delay injection between the disk scheduler and the disk driver. Accesses are not allowed to complete until the proper time has elapsed. For accurate timing, we use a dedicated kernel thread that monitors time using performance counters and sleeps using the kernel's precision sleep functionality.

4. EVALUATION

4.1 Methodology

We use a dual-core Xeon server with 2 GBytes of memory, one 10K rpm disk (for the kernel), and two 15K rpm Maxtor Atlas II disks. On this server, we run a customized Linux kernel, version 2.6.18, with the ext2 file system. The kernel implements our system call for checking the buffer cache and our two-speed disk emulator (Section 3.2). We mount the file system with the "noatime" option to eliminate metadata updates that would unnecessarily move the disk arm.

We emulate two mirrored two-speed disks using the Maxtor disks. The speeds are 3K and 10K rpm. We list the disks' default parameters in Table 1. The 10K rpm values were measured by Carrera [2]. From those values, we extrapolated the 3K rpm values based on [9]. The seek time is that of the Maxtor disk itself, whereas the rotational and transfer times are inversely proportional to the speed. Note that the difference in average rotational time (1/2 of a rotation) between the two speeds we emulate is 7ms. We validated that the emulator achieves accurate timing by comparing its response times with those the emulated disks should achieve according to our performance models. In all experiments, the average difference between these times is less than 0.3%. The emulator computes the disks' energy consumption based on their behavior over time.

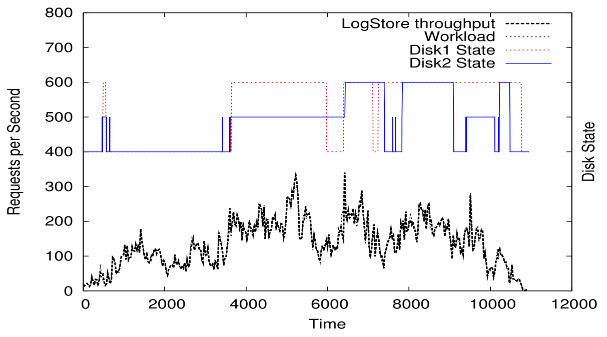

Figure 5: Financial load (4x) and disk state over time.

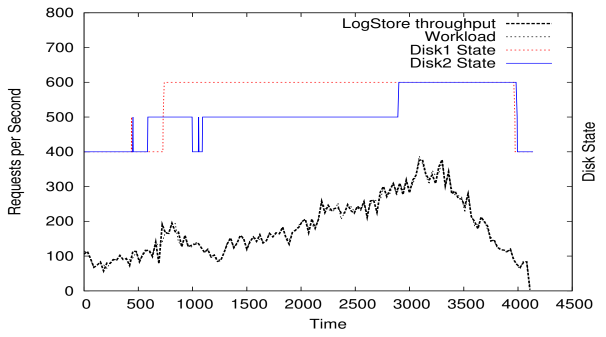

Figure 6: Proxy load (21x) and disk state over time.

Name	Num requests	Avg access size (bytes)	Write %	Write % after cache
Financial	1484205	2914	68%	93%
Proxy	750758	8510	43%	93%
Search	1544375	15218	0%	0%

Table 2: Trace characteristics.

We experiment with the real-world workloads described in Table 2. The first (Financial) is an OLTP trace from a financial institution [15]. The trace is write-intensive and dominated by small requests. The second (Proxy) represents a Web proxy and is also somewhat write-intensive [2]. The vast majority (93%) of the disk accesses in these workloads is writes. The third (Search) is a read-only Web search trace [15]. Since the disk traffic in these traces is light, our trace replayer accelerates (shortens the time between accesses) each of them by a user-defined factor to achieve a desired disk utilization. The maximum factor we use leads to a peak utilization of 90%. We also explore lower factors. Previous papers have also accelerated traces, e.g. [2, 20].

In our experiments, we set the LogStore thresholds to 60% (Balance), 50% (GoUp), 30% (GoDown), 50% (MaxRegular), 30% (MinRegular), 50% (MaxReplay), and 30% (MinLog). The epoch length is 3 seconds. We study the impact of these default values below.

For comparison, we study two energy conservation techniques: Two-speed disks with RAID 1 but not LogStore; and Diverted Accesses with two-speed disks and RAID 1. Diverted Accesses or simply DIV [13] concentrates all read accesses on a single disk and only uses the other if the first disk approaches saturation. The disk that receives most reads typically stays in high speed, whereas the other typically stays in low speed. We compare against DIV because it represents techniques (e.g., [1, 18, 20, 21]) that leverage redundancy to conserve energy. As a baseline, we study a RAID 1 system with conventional disks spinning at 10K rpm. All systems rely on the same code base and have the same reliability and availability characteristics.

4.2 Experimental Results

Figure 5 shows the offered load and LogStore throughput (Y-axis on the left), and disk states (Y-axis on the right) over time for the Financial (4x acceleration) trace. Each point in the graph corresponds to 30 seconds. We represent low speed as the lowest disk state, low speed in the log mode as the middle state, and high speed as the highest state. Figure 6 shows the same data for the Proxy (21x) trace.

We can see that, in both cases, one of the disks stays in the log mode for an extended period. The two-speed disk without LogStore transitions to high speed at those points and consumes substantially more energy. Specifically, in LogStore the disks operate in low speed 58% and 48% of the time in these Financial and Proxy experiments, respectively. In contrast, the same statistics are only 38% and 27%, respectively, when using two-speed disks without LogStore.

We can also see that the offered load and LogStore throughput curves are nearly indistinguishable, showing that LogStore has a negligible impact on throughput at this granularity. At substantially shorter granularities (not shown due to space constraints), the impact of queueing during speed transitions becomes more pronounced in both LogStore and two-speed disks without LogStore.

Table 3 details the disk energy (in KJoules) and average response time (in ms) of the Financial trace under each of the techniques we study. The table shows that LogStore conserves a substantial amount of energy compared to the one-speed results: the highest savings (81%) are for acceleration 2x, whereas the lowest (35%) for acceleration 6x. Regardless of the acceleration, LogStore does not affect the aggregate throughput. The average response time increases by 1.5ms at 2x acceleration and 0.6ms at 6x.

At 2x acceleration, LogStore consumes 57% and 73% less energy than two-speed disks without LogStore and DIV with two-speed disks, respectively. The response time losses are 0.7ms and 1.4ms again compared to these techniques. For the highest acceleration, LogStore consumes 18% and 27% less energy, while degrading response time by only 0.3ms and 0.6ms, respectively.

Interestingly, the intermediate acceleration (4x) leads to higher energy than the other accelerations for LogStore. With the higher load, the disks are more likely to transition to high speed. Whether this increases consumption depends on how long the disks stay in high speed (how much the greater acceleration shortens the execution).

Table 4 shows the same statistics for the Proxy trace. The LogStore energy savings are significant, 75% at 7x acceleration and 36% at 21x acceleration, compared to one-speed disks. Again, regardless of the acceleration, LogStore does not affect the aggregate throughput. The LogStore savings are also significant in comparison to two-speed disks without LogStore and DIV with two-speed disks. At the highest acceleration, LogStore still consumes 21% and 28% less energy than these techniques, respectively.

The results for these write-intensive workloads show that LogStore achieves its main goal: conserving energy while retaining aggregate throughput. LogStore degrades response time for three main reasons: (1) the additional buffer cache space taken by writes to the logs (this effect would not occur in an in-kernel implementation), which causes a higher read

Rate	LogStore		Two-speed		DIV		One-speed	
	E	RT	E	RT	E	RT	E	RT
2x	52.1	2.2	120.0	1.5	195.2	0.8	274.5	0.7
4x	76.3	1.7	99.5	1.3	115.9	1.0	142.9	1.0
6x	64.1	1.9	78.4	1.6	87.8	1.3	98.9	1.3

Table 3: Disk energy (KJoules) and average response time (ms) for the Financial trace.

miss ratio; (2) the additional time disks spend in low speed (during this time, each disk access takes 7ms longer); and (3) the improved write latency (and potentially worse read latency) when the logs are active. Reasons (1) and (2) are the most important in our experiments, but (1) is due to our user-level implementation and is not intrinsic to LogStore.

Under read-only workloads, LogStore cannot take advantage of disk logs. In these cases, LogStore behaves the same as the best competing technique. For example, for the Search trace at the highest acceleration, LogStore achieves 39% energy savings and an increase of 0.4ms in average response time, compared to the one-speed system. This slight performance degradation is due to the many requests that we serve in low speed. The two-speed system without LogStore achieves roughly the same energy savings and performance. DIV with two-speed disks consumes 10% more energy but does not degrade performance.

4.3 Sensitivity Analysis

We study the impact of many parameters on the behavior of LogStore on the Financial trace with acceleration 4x.

Lowering MaxRegular and/or MinLog tends to lower energy consumption but increase response times, since the disks stay in log mode longer. For example, for MaxRegular = 30% and MinLog = 15%, energy consumption decreases 28%, while the average response time becomes 2.3ms. Lowering MinRegular and/or MaxReplay has a smaller impact on energy and response time, since these parameters only affect the amount of time the system stays in log-replay mode. Lowering GoUp or GoDown tends to increase energy consumption while lowering response times, since the disks stay in high speed longer. These trends should guide users in selecting values for the LogStore parameters.

As one would expect, shorter transition times lead to lower energy. For a transition time of 3.5s, the LogStore energy savings become 57% (instead of 47%) compared to the one-speed system. Aggregate throughput is not affected. The increase in average response time becomes 0.8ms (instead of 0.7ms), again compared to the baseline.

Finally, the epoch length exhibits a tradeoff between performance and energy. Lengthening the epochs tends to decrease energy consumption while degrading response time. For example, with 5-second epochs, the LogStore energy savings become 64% and the increase in average response time becomes 1.3ms, compared to the one-speed system.

5. CONCLUSIONS

We proposed LogStore, a storage system that improves the energy proportionality of storage servers with two-speed disks. LogStore can conserve up to 81% and 57% energy, compared to one-speed and two-speed disks, respectively.

We conclude that expensive disks with many speeds are not necessary to achieve storage energy proportionality. Good results can be achieved with much simpler two-speed disks and careful software management.

Rate	LogStore		Two-speed		DIV		One-speed	
	E	RT	E	RT	E	RT	E	RT
7x	38.3	3.7	47.6	2.7	104.0	1.3	155.8	1.3
14x	54.3	2.2	55.9	2.2	66.2	1.7	82.0	1.7
21x	36.0	2.5	45.6	2.0	49.9	1.6	56.0	1.6

Table 4: Disk energy (KJoules) and average response time (ms) for the Proxy trace.

Acknowledgements: This research was partially supported by Google and NSF grant CCF-1017578.

6. REFERENCES

[1] H. Amur et al. Robust and Flexible Power-Proportional Storage. In *SOCC*, 2010.

[2] E. V. Carrera, E. Pinheiro, and R. Bianchini. Conserving Disk Energy in Network Servers. In *ICS*, 2003.

[3] D. Colarelli and D. Grunwald. Massive Arrays of Idle Disks For Storage Archives. In *SC*, 2002.

[4] H. David et al. Memory Power Management via Dynamic Voltage/Frequency Scaling. In *ICAC*, 2011.

[5] T. Denehy, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Journal-guided Resynchronization for Software RAID. In *FAST*, 2005.

[6] Q. Deng et al. MemScale: Active Low-Power Modes for Main Memory. In *ASPLOS*, 2011.

[7] EMC. Symmetrix VMAX Enterprise Storage. http://www.emc.com/storage/symmetrix/vmax.htm.

[8] R. Garg et al. Markov Model Based Disk Power Management for Data Intensive Workloads. In *CCGrid*, 2009.

[9] S. Gurumurthi et al. DRPM: Dynamic Speed Control for Power Management in Server Class Disks. In *ISCA*, 2003.

[10] Hitachi. Hitachi Ultrastar 7K3000. http://www.hitachigst.com/internal-drives/enterprise/ultrastar/ultrastar-7k3000.

[11] Intel. Intel Core i7 Processor Series, 2011. http://download.intel.com/design/processor/-datashts/320834.pdf.

[12] K. Okada, N. Kojima, and K. Yamashita. A Novel Drive Architecture of HDD: "Multimode Hard Disk Drive". In *ICCE*, 2000.

[13] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting Redundancy to Conserve Energy in Storage Systems. In *SIGMETRICS*, 2006.

[14] M. Rosenblum and J.K. Ousterhout. The Design and Implementation of a Log-Structured File System. *ACM TOCS*, 10(1), 1992.

[15] P. Shenoy et al. UMass Trace Repository, 2011.

[16] SMART. Self-Monitoring Analysis and Reporting Technology, 2011. http://smartlinux.sourceforge.net/smart/attributes.php.

[17] M. W. Storer et al. Pergamum: Replacing Tape With Energy-Efficient, Reliable, Disk-Based Archival Atorage. In *FAST*, 2008.

[18] E. Thereska, A. Donnelly, and D. Narayanan. Sierra: Practical Power-Proportionality for Data Center Storage. In *Eurosys*, 2011.

[19] A. Verma et al. SRCMap: Energy Proportional Storage Using Dynamic Consolidation. In *FAST*, 2010.

[20] C. Weddle et al. PARAID: A Gear-Shifting Power-Aware RAID. *ACM TOS*, 3(3), 2007.

[21] X. Yao and J. Wang. RIMAC: A Redundancy-based, Hierarchical I/O Architecture for Energy-Efficient Storage Systems. In *EuroSys*, 2006.

[22] S. Yin et al. Improving Energy Efficiency and Security for Disk Systems. In *HPCC*, 2010.

[23] Y. Zhang et al. Software-Directed Data Access Scheduling for Reducing Disk Energy Consumption. In *HPDC*, 2011.

[24] Q. Zhu et al. Hibernator: Helping Disk Arrays Sleep Through the Winter. In *SOSP*, 2005.