PARAID: The Gear-Shifting Power-Aware RAID

Charles Weddle, Mathew Oldham, Jin Qian, An-I Andy Wang, Florida State University
Peter Reiher, University of California, Los Angeles
Geoff Kuenning, Harvey Mudd College

Abstract

Server-class computers cannot consume power without bound, since increased energy consumption translates into increased heat dissipation, greater cooling requirements, reduced computational density, and higher operating costs. For a typical data center, storage alone accounts for 27% of the energy consumption, making storage an important target for energy reduction. Unfortunately, conventional server-class RAIDs cannot easily reduce power because loads are balanced such that they require the use of all the disks in the array for even light loads.

This paper introduces the *gear-shifting* Power-Aware RAID (PARAID), which reduces energy in server-class computing while retaining performance and reliability. The design of PARAID uses a skewed striping pattern to adapt to the system load by varying the number of powered disks. By powering off disks during periods of light load, PARAID can significantly reduce power consumption, but by matching the number of powered disks to the system load, PARAID can meet the performance demands. Based on our 4-gear PARAID prototype, PARAID consumes 19% less power than a conventional RAID-5 device, while achieving the same performance.

1 Introduction

The disk remains a significant source of power usage. In web servers, disks typically accounts for 24% of the power usage; in proxy servers, 77% [4, 13]. Storage devices can account for as much as 27% of the electricity cost in a typical data center [29]. The energy spent to operate servers in a data center has a cascading effect on other operating costs. Greater energy consumption leads to more heat dissipation, which in turn leads to greater cooling requirements [18]. The combined effect also limits the density of computer racks. The lower density of computers leads to more space requirements, thus higher operating costs.

Approaches to reducing the energy consumption in disks have been explored, but most are achieved by degrading performance significantly. Popular approaches involve trading off performance directly, such as reducing the rotational speed of the disk [3, 4, 13, 22]. Not until recently have new approaches started to emerge to achieve both goals [6, 19, 31].

Data centers that use large amounts of energy tend to rely on RAID to store much of their data, so improving the energy efficiency of RAID devices is promising to reduce the energy use of such installations. Achieving power savings on commodity server-class disks is challenging, because the performance and RAID reliability must be retained in order for a solution to be an acceptable alternative. Conventional RAID balances the load across all disks in the array for maximized disk parallelism and performance [20]. To reduce power, a server cannot simply rely on caching and powering off disks during idle times because such opportunities are not as frequent on servers. Also, the load balancing inherent in RAIDs means all disks are kept spinning even when the server load is light. To be able to reduce power consumption, we need to create opportunities to switch the power state of individual disks. However, server-class disks are not designed for frequent power cycles, which reduce life expectancy significantly.

We have designed, implemented, and measured the *gear-shifting* Power-Aware RAID (PARAID), which balances power against the performance and reliability requirements of server-class RAID devices. To our knowledge, PARAID is the first energy-efficient RAID to be prototyped and measured. PARAID introduces a skewed striping pattern, which allows RAID devices to use just enough disks to meet the system load. PARAID can vary the number of power-on disks by gear-shifting sets of disks, giving PARAID the opportunity to reduce power consumption. Compared to a conventional RAID, PARAID can reduce the power consumption by an average of 19%, while maintaining comparable performance and reliability.

In addition to the power savings obtained by PARAID, the process of creating a real energy measurement framework produced some useful insights into the general problem of measuring energy consumption and savings. These are also discussed in this paper.

2 Observations

Over-provisioned resources under RAID: Conventional RAID is designed to maximize peak performance. The balanced load allows a RAID device

to maximize disk parallelism and performance. This uniformity makes data management simple and allows all disks to be accessed in the same way. Its built-in load balancing also ensures that no disk becomes a bottleneck.

However, this uniform striping design is not favorable in the context of energy savings. Load balancing created by a uniform striping pattern provides significantly fewer opportunities to power off disks because all disks in the array need to be powered to serve a file. Therefore, even if a RAID receives relatively light loads, all disks have to remain powered, even though fewer disks could adequately handle the load.

Cyclic fluctuating load: Many system loads display daily cyclic fluctuations [5]. For example, on a typical day, academic web traffic displays activity as a bell curve with a crest in the afternoon, reflecting students' schedules. Figure 1 shows the trace data used in the evaluation of PARAID. The activity shows an example of daily cyclic fluctuating load. Depending on the types of traffic, different systems may exhibit different cyclic patterns, with varying ranges of light to heavy loads over the course of a day [14].

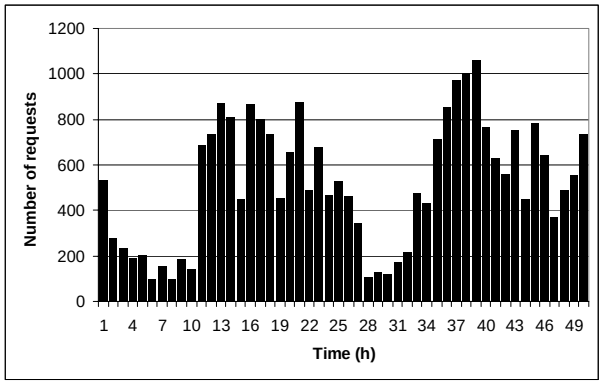


Figure 1: FSU web server activity for 50 hours over September 19 through September 21, 2004.

Therefore, we can exploit these patterns by varying the number of powered disks, while still meeting performance needs and minimizing the number of power switches. A few strategically placed power cycles can achieve significant power savings. Also, fewer power cycles mean that the life expectancy of disks will not be affected significantly.

Unused storage space: Increasingly, storage capacity is outgrowing demand, and not all the storage space is used. Jim Gray recently reported that disks at Microsoft are only 30% full on average [9]. Researchers are increasingly looking for creative ways to consume the unused storage. For example, research at Princeton explores trading off capacity for performance [28]. The Elephant file system explores the possibility of storing every version of file updates [25].

Also, many companies purchase storage with performance as the top criterion. Therefore, they may need many disks for parallelism to aggregate bandwidth, while the associated space is left largely unused. Additionally, administrators tend to purchase more space in advance to avoid frequent upgrades. Unused storage can then be used opportunistically for data block replication to help reduce power consumption.

Performance versus energy optimizations: Performance benefits are important only when a system is under heavy load, and may not result in an immediate monetary return to an organization. On the other hand, energy savings are available at once. For example, the electricity costs saved could be invested in additional computing capabilities. Also, unlike performance, which is essentially purchased in chunks as new machines are acquired, monetary savings can be invested immediately and compounded over the lifetime of the computers. Therefore, if a server usually operates below its peak load, optimizing energy efficiency offers attractive benefits.

3 Power-Aware RAID

The design of PARAID trades capacity for energy savings via a skewed striping pattern. Since servers are purchased for their peak performance, PARAID is designed to match that performance under the peak load. Under light loads, PARAID provisions disk parallelism as needed. Finally, PARAID exploits cyclic daily workload behavior to power-switch disks in a sparing and effective manner, to minimize the effect on the life expectancy of disks.

3.1 Skewed Striping for Energy Savings

PARAID exploits unused storage to replicate and stripe data blocks in a skewed fashion, so that disks can be organized into and behave like hierarchical overlapping sets of RAIDs. Each set contains a different number of disks, and is capable of serving all requests via either its data blocks or replicated blocks. Each set is analogous to a gear in automobiles, since different numbers of disks offer different levels of parallelism and aggregate disk bandwidth.

The replicated blocks are soft states, in the sense that they can be reproduced. Thus, as the need for storage capacity arises, replicated blocks can be reclaimed by reducing the number of gears. Unlike memory caches, PARAID soft states can persist across reboots.

Figure 1 shows an example of replicated data blocks persisting in soft states in the unused disk regions. By

organizing disks into gears, PARAID can operate in different modes. When operating in gear 1, with disks 1 and 2 powered, disks 3 and 4 can be powered off. As the load increases, PARAID upshifts into the second gear by powering up the third disk.

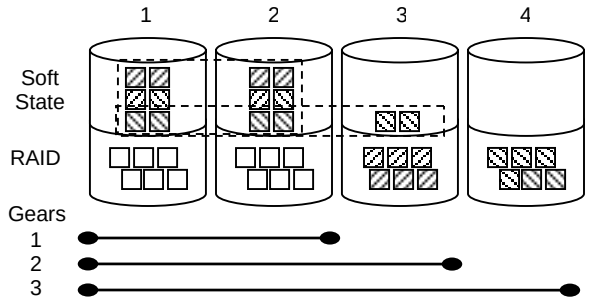


Figure 2: Skewed striping of replicated blocks in soft state, creating three gears over a four-disk RAID.

By adjusting the number of gears and the number of disks in each gear, PARAID provisions disk parallelism and bandwidth so as to follow the fluctuating performance demand curve closely through the day. By creating opportunities to turn off disk drives, PARAID conserves power.

While more gears can match the performance demand curve more closely, the number of gears is constrained by the unused storage available and the need for update propagation when switching gears. To minimize overhead, the gear configuration also needs to consider the number of gears and gear switches.

3.2 Preserving Peak Performance

PARAID matches the peak performance of conventional RAIDs by preserving the original disk layouts when operating at the highest gear. This constraint also allows PARAID to introduce minimal disturbances to the data path when the highest gear is in use.

In low gears, since PARAID offers less parallelism, the total bandwidth offered is less than that of a conventional RAID. Fortunately, the number of requests affected by this performance degradation is significantly less compared to peak hours. Also, as bandwidth demand increases, PARAID will up-shift the gear to increase disk parallelism.

However, PARAID also has the ability to improve performance in low-gear settings for two reasons. (1) Latency suffers as the number of disks increases in a RAID, since the probability that one of the disks will wait for a full rotation increases. Therefore, by decreasing the number of disks being used in parallel, PARAID can shave off a fraction of average rotational latency, which is fairly significant for small file accesses. (2) By reducing the number of active disks during light traffic periods, the average disk queue length increases, resulting in more opportunities to reduce seek time; again, this effect is more pronounced for small accesses Therefore, the performance of PARAID at lower gears largely depends on the average request size.

3.3 Retaining Reliability

To retain the reliability offered by conventional RAID, PARAID must be able to tolerate disk failures. To accomplish this goal, PARAID needs to supply the data redundancy provided by conventional RAIDs and address the reduced life expectancy of server-class disks due to power cycles.

PARAID is designed to be a device layer sitting between an arbitrary RAID device and its physical devices. Therefore, PARAID inherits the level of data redundancy, striping granularity, and disk layout for the highest gear provided by that RAID. PARAID only performs soft-state replication of blocks from the conventional RAID device. For example, a PARAID device composed with a RAID level-5 device would still be able to rebuild a lost disk in the event of disk failure. Section 4.4 has more details on failure recovery.

Because it relies on power-cycling disks to save energy, PARAID must also address a new reliability concern. Power-cycling reduces the MTTF of a disk, which is designed for an expected number of cycles during its lifetime. For example, the disks used in this work have a 20,000-power-cycle rating [8]. Every time a disk is power-cycled, it comes closer to eventual failure.

PARAID manages the power cycling of the disks by inducing a bimodal distribution of busy and idle disks. The busier disks stay powered on, and the more idle disks often stay off, leaving a set of middle-range disks that are power-cycled more frequently. PARAID can then prolong the MTTF of a PARAID device as a whole by rotating the gear-membership role of the disks and balancing their current number of power cycles.

In addition, PARAID sets rate limits on the power cycles for disks. By rationing power cycles, PARAID can operate with an eye to targeted life expectancy. For example, if the disks have a five-year life expectancy due to the system upgrade policy, and the disks are expected to tolerate 20,000 cycles, then each disk in the array cannot be power cycled more than 10 times a day. Once any of the disks has reached the rationed numbers of power cycles for a given period, PARAID can operate at the highest gear without energy savings. The power-saving mode resumes at the next rationing period.

4 PARAID Components

PARAID has four major components—block handler, monitor, reliability manager, and disk manager (Figure 2)—responsible for handling block I/O and replication, gear shifting, update propagation, and reliability.

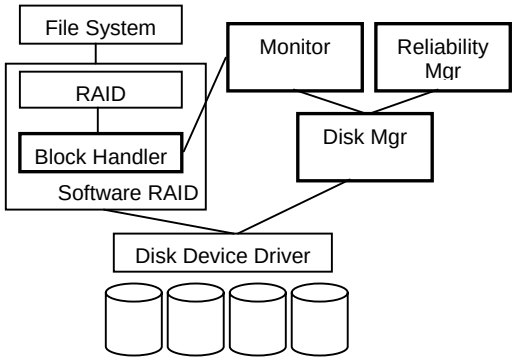


Figure 3: Logical association of PARAID system components.

4.1 Handling Block I/O

PARAID is a new device layer sitting between the conventional software RAID multi-device driver and the disk device driver. PARAID transparently remaps and forwards RAID requests.

If a disk request is sent to a powered disk, the disk simply replies to the request. If a block read request is sent to a powered-off disk, then the PARAID layer will remap the request to read from a replicated block stored on a powered disk. If a block write request is sent to a powered-off disk, then PARAID will write to a replicated block stored on a powered disk.

In the current design, PARAID delegates RAID regions for the purpose of storing replicated data for individual gears. If gear 1 has 3 drives, and gear 2 has 5 drives, the content of drives 4 and 5 are replicated and striped across the dedicated region on the first three drives in a round-robin fashion, so that drives 4 and 5 can be powered off when shifting down to gear 1.

Delegated RAID regions prompt the question of whether this disk layout will degrade performance due to longer seek distances. Our performance results show the contrary, since fewer drives used in parallel actually have longer disk queues that amortize the cost of individual disk seeks.

4.2 Update Propagation

When disks are powered off, no requests are sent to them. As soon as a powered-off disk misses a write request, it no longer contains the most up-to-date data for all data blocks, so it needs to synchronize the stale data either at the time when it is powered on or right before the stale information is accessed. Full synchronization requires that all stale data be updated with current data. Depending on the total size of the stale data, this process could take a long time. The on-demand approach only updates stale data when it is accessed. The on-demand approach allows the gear shift to take place much more quickly, but the full synchronization approach provides better data consistency.

To be able to synchronize a disk, outstanding write requests to powered-off disks are captured by the disk manager. In the case of full synchronization, when a powered-off disk is switched to a powered-on state, the disk manager reissues a list of outstanding write requests to the disk that is to be synchronized. Sometimes this process involves rereading the data from a replicated copy already stored on a powered disk before reissuing the write.

In the case of on-demand synchronization, the PARAID block I/O handler uses a dirty-block list. If a dirty block being accessed is not cached, PARAID will retrieve the block from the original gear and return it to the requestor. PARAID will then write that block to the target gear disks, effectively piggybacking the synchronization step at access time, and sometimes avoiding the rereading step.

One implication is that the disk manager needs to track the stale block locations for synchronization. This list of dirty blocks is stored in memory for fast access during on-demand synchronization as well as on disk in case of system failure.

Another implication is that when downshifting, the on-demand approach is not applicable, since PARAID needs to finish the propagation before powering off drives.

A failed disk can stop the gear-shifting process. Disks can also fail in the middle of synchronization. However, the list of outstanding writes is maintained throughout the disk failure and recovery process. Once the failed disk recovers, the synchronization can continue from where it left off.

Whether to use on-demand or full synchronization for upshifting is configurable. On-demand synchronization will allow PARAID to be more responsive to sudden bursts of requests. This also means tracking additional writes while the disks are not synchronized. The full-synchronization approach may be preferable if there are few gear shifts and the workload is dominated by reads, effectively keeping the number of blocks to be synchronized small. The full synchronization method is

also available for manual maintenance of the PARAID device. For example, an administrator would need to have a consistent system state before pulling out a hard disk.

4.3 Asymmetric Gear-Shifting Policies

The disk manager performs shifts between gears. The PARAID monitor decides when a shift is needed, and the disk manager then performs the actual power cycles.

Switching to a higher gear is aggressive, so that the PARAID device can respond quickly to a sharp increase in workload. Downshifting gears needs to be done conservatively, so that wild swings in system activity will not (1) mislead the PARAID device into a gear that cannot handle the requests, or (2) cause rapid oscillations between gears.

To decide when PARAID should upshift, the monitor needs to know whether the current gear has reached a predetermined utilization threshold, in number of requests per time interval. The threshold is configurable, and is set to 80% for the Web servers in our experiments. The intent is that within the time it takes to spin up the disk and propagate updates, the utilization threshold would not reach 100%. The use of an online algorithm to automatically set thresholds will be future work,

To track the system load, the monitor keeps a moving average of utilization for each disk. The purpose of averaging is to filter out short bursts of requests that are frequently seen in real-world workloads. Interestingly, we were unable to check the disk busy status directly, since this probe would spin up a powered-down disk. Instead, once a second, each disk is checked to see if any access has occurred. If so, the disk is marked as active for that second. If any disk in an active gear has reached the utilization threshold, then the monitor will make a request to the disk manager to up-shift.

To decide when to downshift, the PARAID monitor needs to know the utilization trends as well as the average disk utilization. The intent is to avoid frequent gear switches due to wild workload fluctuations. A downward trend is detected with multiple threshold averages over different time windows. Monotonically decreasing averages indicate dropping utilization. Having identified this trend, the PARAID monitor then makes sure that the next lower gear with fewer disks can handle the current workload. If both conditions are met, a downshift is performed (Figure 4).

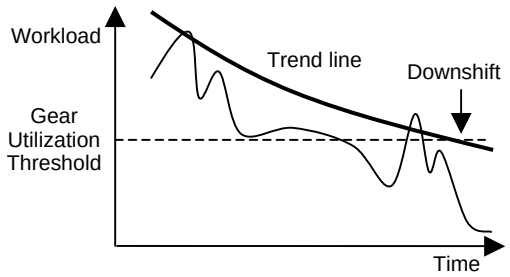


Figure 4: The downward trend in workload activity triggers a downshift in gears. Monitoring trends avoids downshifting during volatile workloads.

The effectiveness of an energy-saving gear configuration depends on the shape of the workload curve and the peak-to-trough ratio across the course of a day. Suboptimal gear configurations typically reveal themselves through lack of use.

The gear-shift triggering condition can significantly affect the performance, energy-savings, and reliability of PARAID. If the condition is too sensitive to traffic bursts, PARAID will be likely to operate at the highest gear at all times, resulting in little energy savings. Also, with too many power cycles, the life expectancy of PARAID suffers.

4.4 Reliability

The reliability manager rations power cycles and exchanges the roles of gear memberships to prolong the life expectancy of the entire PARAID, as mentioned in Section 3.3. The reliability manager is also responsible for recovering a PARAID device upon disk failure.

Although PARAID can inherit the reliability properties of RAID levels, the data and parity blocks of N disks cannot be striped across on N – 1 disks to achieve the same level of redundancy. If we simply assigned the Nth block to one of the still-powered disks, it would be possible for a single drive to lose both a data block and parity block from the same stripe, while the block stored on the powered-off disk might be out of date.

In the current design, when operating in lower gears, the underlying RAID data blocks are no longer updated in-place. Updates are written to alternative locations on disks and propagated back to the original layout when shifting to the highest gear. With this model, the data loss due to a crashed drive is bounded by the frequency of using the highest gear. PARAID can also force updates to propagate once a day. Therefore, in the case of a single drive failure, PARAID can experience a one-day loss of updates.

This problem can be prevented with additional gear-centric parities. Basically, blocks from disk N can be replicated and striped to N – 2 disks, with an additional parity block computed for these N – 2 blocks, which is stored on disk N – 1. This design avoids rereading blocks not involved in the actual replication. We plan to implement such an approach in a future system.

5 Implementation

The PARAID prototype was built on Linux 2.6.5. Linux was chosen for its open source and because it contains a software RAID module. The block I/O handler, monitor, disk manager, and reliability manager are built as kernel modules. A PARAID User Administration Tool runs in user space to help manage the PARAID devices. For the reliability manager, we have not implemented drive rotation and gear-centric parities. However, our gear-shifting policies and the characteristics of daily work cycles have limited the number of disk power cycles quite well. Since the content stored on the lowest gear is the most vulnerable, we keep those drives always powered. Additionally, PARAID currently relies on the underlying RAID levels to provide data recovery, so we can recover the information from the last time we shifted to the highest gear.

The Linux software RAID is implemented as the `md` (multiple device) device driver module, which builds RAIDs from multiple disks. For the PARAID block handler implementation, we changed the `md` device driver to make it PARAID-aware. The data path of the `md` device driver is intercepted by the PARAID device layer, so that requests from conventional RAID are redirected to the block queues of PARAID, which remaps and forwards requests to individual disk queues.

During initialization, the PARAID-aware `md` module starts a daemon that provides the heartbeat to the PARAID device and calls the monitor at regular intervals, so that it can decide when to gear-shift. The disk manager controls the power status of disks through the disk device I/O control interface.

To synchronize the content of a powered-off disk before bringing it back into operation, the disk manager keeps a per-disk red-black tree of references to outstanding blocks that need to be updated. This record is updated whenever a write request is made to a clean block on a powered-off disk, and the upkeep of this data structure is not CPU-intensive. For the current implementation, the disk manager performs a full synchronization after bringing back powered-off disks, by iterating through the tree for each disk and reissuing all outstanding writes. For each block that needs to be synchronized, the disk manager will first read in the data block from disks in the original gear, and then write the block to the disks being brought back online. Once the synchronization is complete, the gear-shifting manager switches to the new gear by enabling it to serve requests with the newly powered disks.

Note that during synchronization, PARAID still serves requests from the old gear until the target gear has been fully synchronized. During synchronization, new writes are temporarily written to both old and new gears. The old gear still serves all subsequent reads to the newly written data, while the new gear will skip propagation of the original overwritten dirty block. This conservative switching assures that no block dependency is violated through the ordering of updates. In the future, we will also explore the use of back pointers [1] to allow the new gear to be used while propagating the updates.

For the PARAID monitor, we currently use 10-, 60-, and 300-second time windows to compute moving averages of disk utilization. The choice of these time windows is somewhat arbitrary, but they work well for Web server workloads and can tolerate traffic bursts and dampen the rate of power cycles. Further investigation of the gear-shifting triggering conditions will be the subject of future work.

The `mkraid` tool, commonly used by Linux to configure RAIDs, had to be changed, so that it could handle making PARAID devices and insert entries to `/etc/raidtab`. Additional `raidtab` parameters had to be defined to be able to specify the gears.

Source File	Line Count
`paraid.c/.h`	1057
`paraid-dm.c/.h`	1047
`paraid-mon.c/.h`	610
`md.c/md_u.h/md_p.h/md_k.h`	409
`bio.h/ll_rw_blk.c`	12
`Raidtools (mkraid.c,parser.c,pdadm.c)`	358

Table 1: Line count for PARAID source files, Linux modifications, and Raidtools modifications

Table 1 lists the line counts for the PARAID source files as well as additions and modifications to the Linux and `Raidtools` source code. The PARAID logic is contained for the most part in the Linux Software RAID implementation. The ease of porting PARAID to future Linux kernel versions depends on future modifications to the Linux Software RAID. Because the logic for PARAID is fairly self-contained, it should be moderately portable to other software RAID implementations for other operating systems.

6 Web Trace Replay

Since the study of energy-efficiency approaches to RAIDs is relatively recent, most prior work on energy savings has been done analytically or via simulations.

Analytical methods provide us a fundamental understanding of systems with key variables. Simulation studies enable us to explore a vast parameter space to find broad understandings of system behaviors under a wide range of workload scenarios. However, we chose implementation and empirical measurements to see if we could overcome unforeseen physical obstacles and conceptual blind spots to bring us one step closer to a deployable prototype. When we designed, implemented, and evaluated PARAID empirically, we discovered why an empirical study is difficult for systems designed to save energy.

- Needless to say, prototyping PARAID was the first barrier, and the system had to be stable enough to withstand heavy benchmarking workloads.
- Commercial machines are not designed for energy measurements, and we had to rewire drives, power supplies and probes for power measurements.
- The conceptual behaviors of a system are far from close to its physical behaviors; therefore, we had to adjust our design along the way.
- Most benchmarks and workload generators measure the peak performance of a system at steady state, which is not applicable for measuring energy savings, where we need to capture daily workload fluctuations.
- For trace replays, since our physical system configuration was largely fixed, we had to try to match different trace environments with our physical environments in terms of the memory size, traffic volume, disk space consumption, and so on.
- Although a plethora of research trace replay tools is available, more sophisticated ones tend to involve kernel hooks and specific environments. Incompatibility of kernel versions prevented us from leveraging many research tools.
- Finally, since it cannot be easily automated and cheaply parallelized, measuring energy savings on a server was very time-consuming.

Taking these measurement challenges into consideration, we document our experimental settings to obtain our results. We demonstrate the power savings and the performance characteristics of PARAID via web trace replays. Although Web workloads are dominated by reads, they are still representative of a very large class of useful workloads. We used the PostMark benchmark [15] (Section 7) to demonstrate PARAID's performance characteristics in the presence of writes and under peak load. The PostMark benchmark also stresses the gear-shifting overhead.

6.1 Trace Replay Framework

The measurement framework consisted of a Windows XP client and a Linux 2.6.5 server. The client performed trace playback and lightweight gathering of measurement results, and the server hosted a web server running on a RAID storage device (Table 2). On the server, one disk was used for bootstrapping, and five disks were used to experiment with different RAIDs. The client and server computers were connected directly to each other by a CAT-6 crossover cable so that extraneous network traffic would not interfere with the experiments

	Server	Client
Processor	Intel Xeon 2.8 Ghz	Intel Pentium 4 2.8 Ghz
Memory	512 Mbytes	1 Gbytes
Network	Gigabit Ethernet	Gigabit Ethernet
Disks	Fujitsu MAP3367 36.7Gbytes 15k RPM SCSI Ultra 320 1 disk for booting 5 disks for RAID experiments	Seagate Barracuda ST3160023AS 160 Gbytes 7200 RPM SATA

Table 2: Server and client computer specifications.

To measure the power of the disks, the power measurement framework included an Agilent 34970A digital multimeter. Each disk probe was connected to the multimeter on a unique channel, and the multimeter sent averaged data to the client once per second per channel via a universal serial bus. Figure 4 shows the client and server computers and the multimeter in the measurement system.

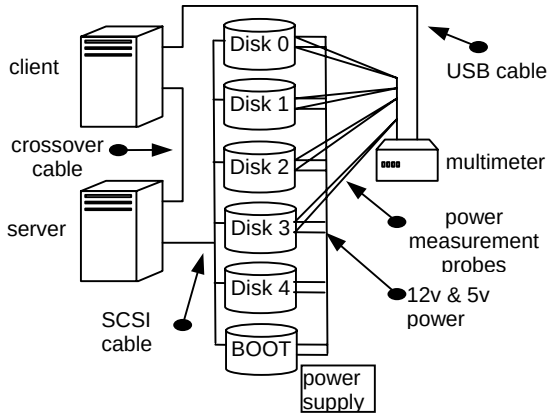


Figure 5: The measurement framework.

To measure the power of a disk, we inserted a 0.1-ohm resistor in series in the power-supply line, as shown in Figure 5. The multimeter measured the voltage drop across the resistor, V_r. The current I through the resistor—which is also the current used by the disk—can be calculated as V_r/R. Knowing the voltage drop across the disk, V_d, its power consumption is then V_d times I.

In the measurement system, we removed each disk from the server and introduced a resistor into its +12V and +5V power lines. The +12V line supplied power to the spindle motor; the +5V line provided power to the disk

electronics. The SCSI cable was connected directly to the motherboard, which allowed the cable to maintain the same performance as if the disks were connected to the SCSI hot swappable backplane in the server.

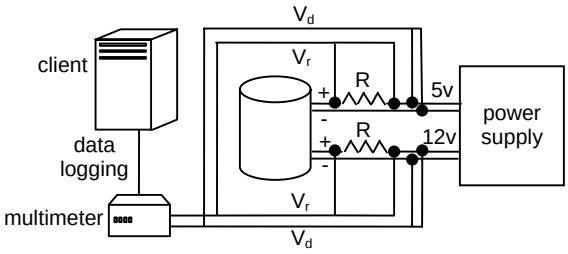


Figure 6: The resistor inserted in series between the power supply and the disk adapter.

On the client, the Agilent Multimeter software logged the data using Microsoft Excel XP. The multi-threaded trace driver, implemented in Java 1.5, was designed to replay web access log traces and collect performance numbers. The driver handled associated requests generated from the same IP address in separate threads, to emulate users clicking through web pages. The trace driver also collected server-side and end-to-end performance numbers.

The server hosted an Apache 2.0.52 web server on top of an ext2 file system operating over a RAID storage device that is described in Table 1.

6.2 Web Server Workload

Workload characteristics affect PARAID's ability to save energy. Under a constant high load, PARAID will not have opportunities to downshift and save energy. Under a constant light workload, trivial techniques like turning everything on and off can be used to save energy. In practice, workloads tend to show cyclic daily and weekly fluctuations. The chosen workload needs to capture these cyclic fluctuations to demonstrate PARAID's energy savings. In the trace we replayed, the number of accesses during peak hours can be 10x higher than that during light hours.

We chose a Web server workload, captured within the Computer Science Department at Florida State University. The web server has 512 Mbytes of RAM, two 1.8 GHz Intel P4 CPUs, and two 120-GByte disks in a RAID-0 configuration. The activity was captured from August 2004 to November 2004. The file system contained approximately 47 Gbytes of data, 44,000 directories, and 518,000 files. A snapshot of the file system was recreated from the last day of the trace to account for the files not referenced by the web trace and associated disk space usage. Although the trace playback might reference files that were deleted or moved, web server content tends to persist once created [2]. Also, the replay did not include dynamic file content, which accounts for relatively few references.

To protect privacy, the actual file blocks stored on the web server were refilled with random bits. Also, file names were encrypted via an SHA-1 algorithm [23], to anonymize files that were not referenced or not meant to be available to the public (e.g. homework_solutions.txt).

We chose a 50-hour trace starting from September 19, 2004. The cyclic workload pattern is characteristic throughout the two-month long trace (Figure 1). The duration included 26,000 requests, with 1038 Mbytes of data. Due to the multi-threaded nature, the trace was replayed five times. The data is presented at the 90% confidence level.

6.3 Web Trace Replay Experimental Settings

PARAID was compared with a RAID-5 device. The PARAID device used five disks with four gears: gear N contains disks 0 to N. Both client and server were cleanly rebooted before each experiment, and PARAID was configured to start with the lowest gear, with content in different gears pre-populated. The client replayed pre-encrypted trace log entries to the server. Due to the hardware mismatch and light trace workload, the collected trace was accelerated at different speeds to illustrate the range of possible savings with different levels of workloads. The presented data included 32x, 64x, 128x, and 256x speedup factors. Timing dependent on human interactions, such as the time between user mouse clicks on links, was not accelerated.

6.4 Power Savings

Figure 7 compares PARAID and RAID-5 in terms of the amount of power consumed over time, with different speedup factors. In these graphs, each data point is averaged over 30 minutes.

Since PARAID started with the lowest gear, the first finding was that turning off 3 out of 5 drives did not achieve anywhere near 60% energy savings. Powering off a disk only stopped it from spinning its platter and therefore, only the 12V line was shut off.

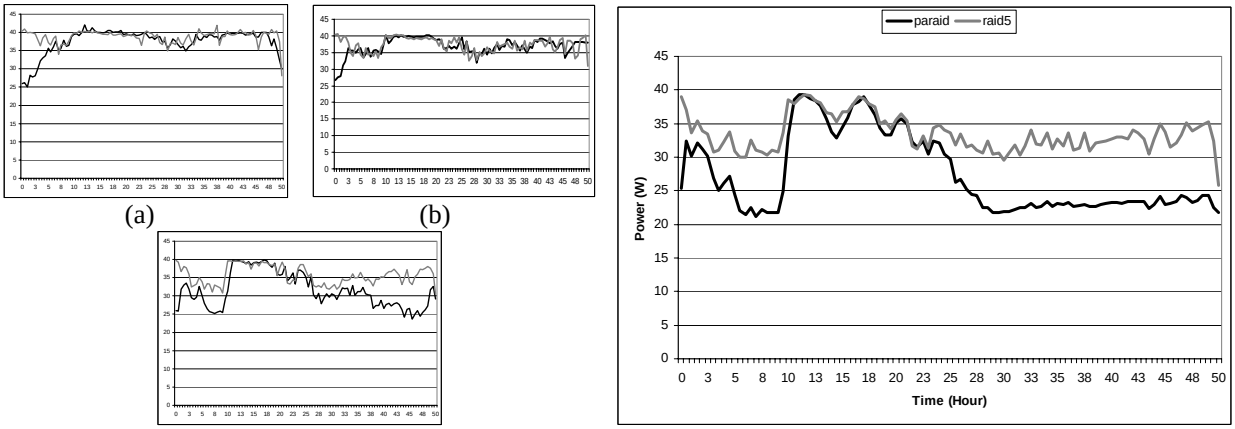


Figure 7: Power graphs over 256x (a), 128x (b), 64x(c), and 32x (d) speedup factors.

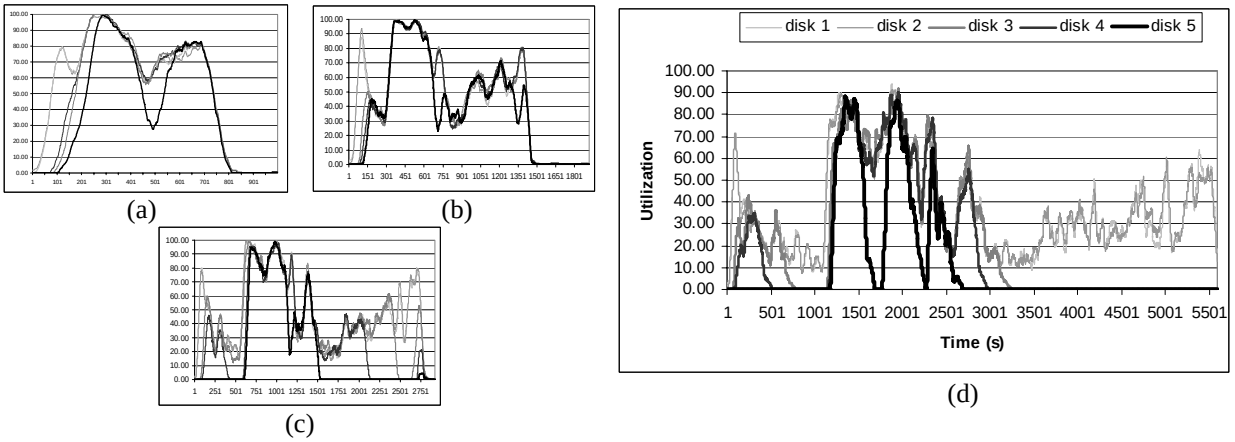


Figure 8: Disk utilization graphs over 256x(a), 128x(b), 64x(c), and 32x(d) speed-up factors.

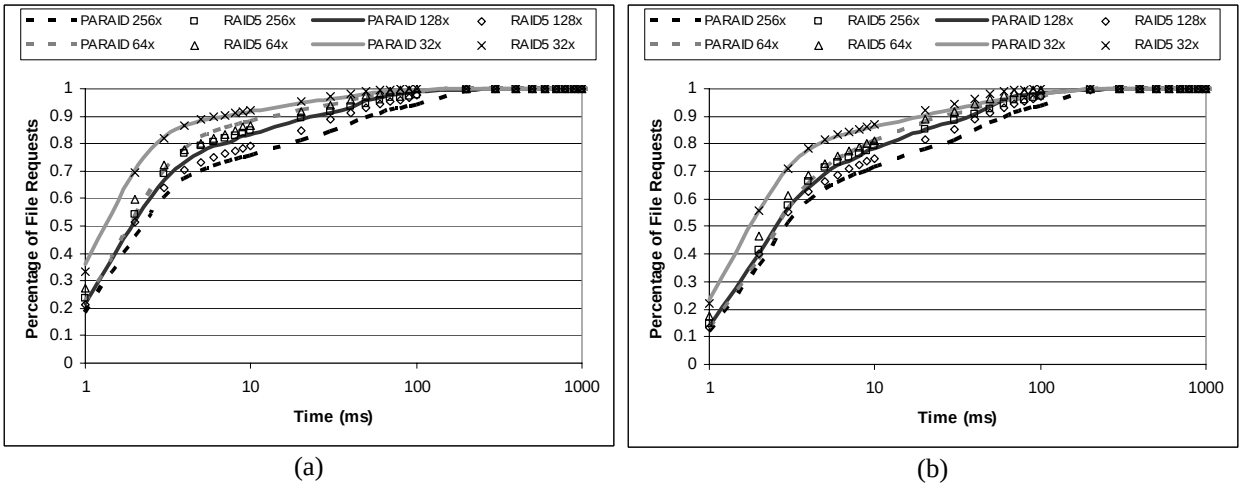


Figure 9: Peak-hour latency (a) and total completion time (b) for PARAID and RAID devices over 256x, 128x, 64x, and 32x speed-up factors.

Power was still needed for the 5V line that powered the electronics, so that it could listen for a power-up command and passed commands along the daisy-chained SCSI cable. Therefore, the power never dropped to zero for the PARAID cases, even when all of the disks were powered off. In fact, a disk with the spindle motor powered off still consumed about three watts of power for the electronics, which is noticeably higher than the 1.0W to 2.5W extracted from various datasheets and used in many simulations [4, 10, 13, 22, 29, 30, 31]. Although these numbers may be hard-drive- and vendor-specific, they do show that these variations in physical characteristics can change the expected results drastically.

The second surprise is that the cyclic patterns observed in the web log (Figure 1) have a poor correlation with the energy consumption at the disk-drive level (Figure 7). Although this finding reveals more about the nature of caching than the energy benefits of PARAID, it does suggest the value of further investigations into the relationship between server-level activities and after-cache device-level activities.

To make sure that the daily cyclic patterns are not completely lost below the memory cache, we examined the disk usage level over time (Figure 8). We found that the cyclic effect is still present, but PARAID had successfully consolidated the loads to the lowest gear, so that the remaining disks could be powered off to save power. The disk utilization graph also reveals that if the entire RAID is power-switched as a whole, the opportunity to save power is quite limited.

Table 3 lists the energy saved versus the speed-up factor. When the trace was played back at 128x and 256x the normal speed, PARAID was overloaded and had to power on all disks over most of the time. As the speed of the trace was slowed down, PARAID found opportunities to save power. PARAID was able to use 12% less power than RAID 5 at 64x speedup. At 32x speedup, PARAID was able to use 19% less power. The power savings were calculated as the area between the RAID 5 and PARAID power curves, divided by the area under the RAID 5 power curve.

Speed-up	Power Savings	Variance(+/-)
256x	3.3%	0.42%
128x	1.2%	0.57%
64x	12%	0.96%
32x	19%	0.22%

Table 3: The percent energy saved versus the speed-up factor.

6.6 Performance

For performance, Figure 9 shows the CDFs of per-request latency and completion time respectively, in milliseconds. The per-request latency measures the time from the last byte of the request sent from the client to the first byte of data received at the client. The completion time measures the time between sending the first byte of a request from the client to receiving the last byte at the client end. Throughput was a less meaningful metric, since the usability of a Web server largely depends on the responsiveness of user requests [16].

As expected, increased trace playback speed lengthens latencies and completion times. However, the differences between RAID-5 and PARAID are nearly identical at a 32x speed-up factor. At 256x speed, RAID 5 had 85% of the file requests served with latencies less than 10 ms, compared to 76% for PARAID. RAID 5 had 79% of the file requests served with a completion time less than 10 ms, compared to 71% for PARAID.

7 PostMark Benchmark

The PostMark benchmark is a popular ISP synthetic benchmark, which is used to stress the peak performance of a storage device for its read- and write-intensive activity [15]. Running PostMark with PARAID starting at the lowest gear can be indicative of overhead and latency of gear-shifts during a request burst. The PostMark Benchmark was run directly on the server.

In Figure 10, we present PostMark results comparing the elapsed times of RAID 5, PARAID starting with the highest gear, and PARAID starting with the lowest gear under three different configurations. Each configuration was measured five times. PARAID propagated updates synchronously during gear shifts.

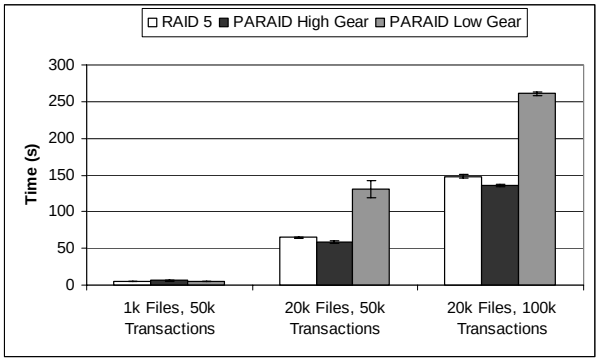
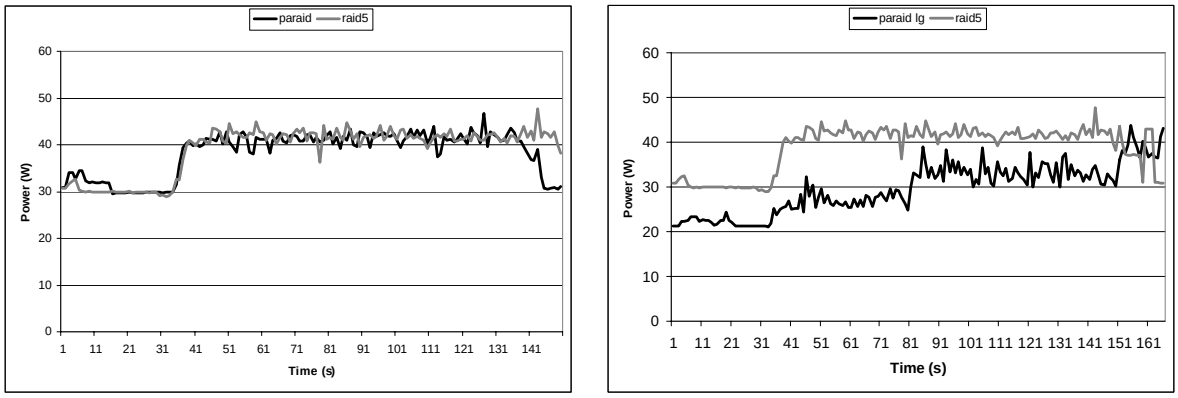


Figure 10: Postmark results for a RAID 5 device compared to a PARAID device starting in the highest gear and starting in the lowest gear.

Under different PostMark configurations, PARAID starting with the highest gear demonstrates performance similar to RAID 5, which is reflective of how we have preserved the layout of underlying RAID and



(a) (b)

Figure 11: The power consumption for the Postmark Benchmark experiments at 20k files - 100k transactions for PARAID starting in a high gear (a) and PARAID starting in a low gear (b).

introduced minimal disturbances to the `md` data path. PARAID may actually perform slightly better, partly due to its intentional caching of some dirty data waiting to be propagated to lower gears. Figure 11(a) shows that the energy consumption of PARAID and RAID-5 is also comparable in these configurations. Note that during the first 30 seconds of the PostMark benchmark, the buffered-write interval of ext2 prevented accesses to be landed on the disk.

Figure 10 also compares the performance of RAID-5 with PARAID starting in the lowest gear. It demonstrates the current up-shift policy that prevents PARAID from being responsive to bursts. The slowdown factor is about two, since it took about two minutes for gears to up-shift incrementally (spin-up disk, propagate updates, and determine whether to up-shift further) (Figure 11b). The most responsive approach is obviously jumping from gear 1 to gear 4. However, this would cause too many gear shifts throughout a day. Fortunately, from what we observed, daily web workloads tend to cause few gear shifts. Thus, this overhead is not noticeable. As future work, we will explore online algorithms to improve the responsiveness to burst loads while minimizing the number of gear shifts.

8 Related Work

Prior energy-reduction studies have been mostly in the area of mobile computing [7, 12]. Only recently have energy reductions been a concern in server-class computing. Various approaches range from the hardware level and the RAID level to the file system level and the server level.

Reducing power consumption in hard disks: Based on simulations, Carrera, et al. [4] suggested using hypothetical two-speed disks such that during periods of high intensity, the disk runs at maximum throughput, using the most power. During periods of lower intensity, the disk spins at a lower speed before possibly going into an idle state. The simulation reports disk energy savings between 15% to 22% for web servers and proxy servers, with throughput degradation of less than 5%.

Energy-efficient RAIDs: Hibernator [31] aims to reduce energy consumption in arrays of disks without degrading performance. Hibernator explores the possibility of using disks that can spin at variable speeds to achieve energy savings. According to demand, data blocks are placed at different tiers of disks spinning at different speeds. A novel disk block distribution scheme moves disk content among tiers to match disk speeds. When performance guarantees are violated, Hibernator spins disks at full speed to meet the demand. In simulation, Hibernator shows a 65% energy savings.

Colarelli et al. [6] introduced the idea of massive arrays of idle disks, the primary purpose of which is archival. A small set of cache disks are on to serve requests. Their simulation has reported comparable performance to traditional RAID, while using $1/15^{th}$ of the power. PARAID is designed with a very different mindset, where energy savings are achieved with fluctuating loads that exercise all drives daily.

Popular data concentration (PDC) [22] centers on the popularity, or the frequency, of file access. PDC puts the most popular data on the first disk, the second most popular on the second disk, and so on. Disks are powered off in PDC based on an idleness threshold. Without striping, PDC does not exploit disk parallelism.

With the absence of disk striping, the power-aware cache management policy (PA-LRU) [29] saves power by caching more data blocks from the less active disks. Lengthening the access interval for less active disks

allows them to be powered off for longer durations. Partition-based cache management policy (PB-LRU) [30] divides the cache into separate partitions for each disk. Each partition is managed separately by a replacement algorithm such as LRU. PB-LRU provides similar energy savings of 16% to that of PA-LRU.

Energy-aware storage systems: Nightingale et al. [19] suggest BlueFS, a distributed file system, which uses a flexible cache hierarchy that adaptively decides when and where to access data, based on the energy characteristics of each device. In measurements of an actual implementation, BlueFS achieved a 55% reduction in file system energy usage. When used in combination, PARAID can extend the energy benefits with BlueFS.

The *Conquest-2* file system [27] uses inexpensive persistent RAM to store small files to save energy consumed by disks. PARAID can be readily used as a counterpart to serve large files while conserving energy.

Saving power in server clusters: Chase, et al. [5] and Pinheiro, et al. [22] have developed methods for energy-conscious server switching to improve the efficiency of server clusters at low request loads. They have reported energy reductions of 29% to 43% for Webserver workloads.

The PARAID approach can be combined with the server paradigm, so that over-provisioned servers used to cushion highly bursty loads or pre-powered to anticipate load increases can turn off many PARAID drives. Since powering on disks is much faster than booting servers, PARAID pays a lighter latency penalty to respond to traffic bursts.

Further, in the case where traffic loads involve a mixture of reads and writes, disk switching in PARAID avoids data movement across machines and associated stress on the network infrastructure.

9 Ongoing Work

PARAID is an ongoing project. Our top priority is to understand PARAID under a wider range of workloads. We are currently measuring PARAID with a UCLA web server workload. Also, we are preparing to replay the `cello99` trace [11] and a financial trace [26] from the UMass Trace Repository.

Our next priority is to implement gear-centric parity schemes, so that single-drive failures can be recovered with minimal data loss. We will incorporate the S.M.A.R.T tools [24] to monitor the health of disk drives continuously, to make more informed decisions on rationing power cycles, and to rotate the gear-membership of disks.

Currently PARAID is not optimized in the sense that the selection of the number of gears, the number of disks in each gear, and gear-shifting policies are somewhat arbitrary. Since empirical measurement is not suitable for exploring a large parameter space, we are constructing a simulation for this purpose, and PARAID-validated simulation will give us much greater confidence in obtained results. At the same time, we are exploring analytical approaches to develop online algorithms with provable optimality.

Further, we will modify our disk synchronization scheme to explore the potential asynchrony of update propagation, to allow newly powered-on drives to serve requests immediately.

Finally, we plan to mirror a PARAID server to FSU's department server for live testing, and deploy PARAID in a real-world environment.

10 Lessons Learned

The concept of PARAID was born as a simple concept to mimic the gear-shifting analogy in vehicles to conserve fuel. However, turning this concept into a kernel component for practical deployment has been much more difficult than we anticipated.

First, design-to-fit matters. Our early design and prototype of PARAID involved cloning and modifying RAID-0. As a result, we had to bear the burden of inventing replication-based reliability mechanisms to match different RAID levels. However, our second-generation design now can largely inherit the RAID encoding scheme, which makes the evolution of new RAID levels independent of PARAID. Although the resulting energy savings and performance characteristics can be comparable, the architecture of PARAID can significantly affect its structural complexity, development time, and potential for practical deployment.

Second, measuring energy consumption is difficult because of data alignment problems and a lack of integration of tools. With continuous logging, aligning data sets is largely manual. For multi-threaded experiments and physical disks, the alignment of data sets near the end of the experiment is significantly poorer than at the beginning of the experiment. At the beginning, the results obtained from averages were not explainable, since unaligned square waves can be averaged into anything but squares.

Third, measuring systems under normal loads is harder than measuring systems under peak loads. We could not simply replay traces as quickly as possible, and we had to explore a range of speedup factors to see how

PARAID reacts to a different range of loads. Since we are interested in server loads with constant streams of requests, we cannot apply the trick of skipping idle periods [21], since such opportunities are relatively infrequent.

Fourth, modern systems are complex. As modern hardware and operating systems use more complex optimizations, our perception of system behaviors increasingly deviates from their actual behaviors. Memory caching can reduce disk activity, while file systems can increase the burstiness of RAID traffic arrivals due to delayed write-back policies. Disks are powered in spikes of current, making it difficult to compute power consumption with the areas under the spike. Disk drives can still consume a significant amount of power even when they are spun down.

Fifth, matching the trace environment to our benchmarking environment is difficult. If we use a memory size larger than that of the machine being traced, we may encounter very light disk activity. The opposite situation can saturate the disks and achieve no power savings. Cyclic workload patterns before the cache may poorly reflect the workload patterns after the cache. Additionally, traces might not be using RAIDs, some traces may be too old, and the RAID geometry might not match our experimental settings. The base system might have more than one CPU, which makes it difficult to judge whether a single modern CPU is powerful enough. Although the operating system research community is well aware of these problems, the solutions still seem to be achieved largely by trial and error.

11 Conclusion

PARAID is a file system designed to save energy for large computing installations that currently rely upon RAID systems to provide fast, reliable data storage. PARAID maintains the desirable characteristics of standard RAIDs, while decreasing their energy use by up to 19%. Since PARAID is not currently optimized, and since we measured only 5 drives (among which 2 are always powered), we believe that optimized PARAID with many disks can achieve significantly more energy savings. Since disk drives consume over 27% of the entire energy use of a major data center, the use of PARAID via a simple software update can reduce total electricity costs by 5%, an improvement worth pursuing in a large data center.

A second important conclusion arises from the research described in this paper. Actual implementation and measurement of energy savings systems are vital, since many complex factors such as caching policies, memory pressure, buffered writes, file-system-specific disk layouts, disk arm rescheduling, and many physical characteristics of disk drives are difficult to fully capture and simultaneously validate using only simulation. Also, implementations need to address compatibility with legacy systems, the use of commodity hardware, and empirical evaluation techniques, all of which are necessary for practical deployments.

Unfortunately, our research also shows that there are considerable challenges to performing such experiments. We overcame several unforeseen difficulties in obtaining our test results, and had to invent techniques to do so. This experience suggests the value of developing standard methods of measuring the energy consumption of computer systems and their components under various conditions. We believe this is another fruitful area for study.

Acknowledgements

We would like to acknowledge Ted Baker, Kartik Gopalan, and Nancy Greenbaum for their early comments. We also thank Noriel Lu, Sean Toh, Daniel Beech, Carl Owenby, and Nicholas Wallen for their early contributions to the measurements of PARAID. Additionally, we thank Jason Flinn, Daniel Peek, Margo Seltzer, Daniel Ellard, Ningning Zhu, HP, and StorageTek (now Sun Microsystems) for providing accesses to various tools and traces. This work is sponsored by NSF CNS-0410896.

References

[1] M. Abd-El-Malek, W.V. Courtright II, C. Cranor, G.R. Ganger, J. Hendricks, A.J. Klosterman, M. Mesnier, M. Prasad, B. Salmon, R. R. Sambasivan, S. Sinnamohideen, J.D. Strunk, E. Thereska, M. Wachs, J.J. Wylie, *Proceedings of the 4th USENIX Conference on File and Storage Technology (FAST '05)*, San Francisco, CA. December, 2005.

[2] O. Brandman, J. Cho, H. Garcia-Molina, N. Shivakumar, Crawler-Friendly Web Servers, *SIGMETRICS Performance Evaluation Review*, 2005.

[3] P. Cao, E.W. Felten, K. Li, Implementation and Performance of Application-Controlled File Caching, *Proceedings of the 1st Operating Systems Design and Implementation Symposium*, 1994.

[4] E. Carrera, E. Pinheiro, R. Bianchini, Conserving Disk Energy in Network Servers, *Proceedings of the 17th Annucal ACM International Conference on Super Computers*, 2003.

[5] J. Chase, D. Anderson, P. Thakar, A. Vahdat, R. Doyal, Managing Energy and Server Resources in Hosting Centers, *Proceedings of the 18th ACM Symposium on Operating System Principles*, 2001.

[6] D. Colarelli, D. Grunwald, Massive Arrays of Idle Disks For Storage Archives, *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*, November 2002.

[7] F. Douglis, P. Krishnan, B. Bershad Adaptive Disk Spin-down Policies for Mobile Computers *Proceedings of the 2nd USENIX Symposium on Mobile and Location-Independent Computing, 1995.*

[8] Fujitsu, MAP Series Disk Drive, 2005.

http://www2.fcpa.fujitsu.com/sp_support/ext/enterprise/datasheets/map10krpm-datasheet.pdf

[9] J. Gray, Keynote Address Greetings from a Filesystem User, *the 4th USENIX Conference on File and Storage Technologies*, 2005.

[10] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, H. Franke, DRPM: Dynamic Speed Control for Power Management in Server Class Disks, *Proceedings of the International Symposium on Computer Architecture*, pages 169-179, June 2003.

[11] HP Labs, Tools and Traces, 2005.
http://www.hpl.hp.com/research/ssp/software/

[12] D.P. Helmbold, D.D.E. Long, B. Sherrod, A dynamic disk spin-down technique for mobile computing, *Proceedings of the 2nd Annual Internacional Conference on Mobile Computing and Networking (MobiCon'06), 1996.*

[13] H. Huang, P. Pillai, K.G. Shin, Design and Implementation of Power Aware Virtual Memory, *Proceedings of the 2003 USENIX Annual Technical Conference*, 2003.

[14] A. Iyengar, J. Challenger, D. Dias, P. Dantzig, High-performance Web Site Design Techniques, *IEEE Internet Computing*, 4(2):17–26, March 2000.

[15] J. Katcher, PostMark: A New File System Benchmark, Technical Report TR3022, Network Appliance Inc., October 1997

[16] S. Manley, M. Seltzer, M. Courage, A Self-Scaling and Self-Configuring Benchmark for Web Servers**,** *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, Madison, Wisconsin, 1998

[17] E. Miller, R. Katz, An analysis of file migration in a Unix supercomputing environments, *Proceedings of the 1993 USENIX Winter Technical Conference*, pages 421-433, 1993.

[18] J. Moore, J. Chase, P. Ranganathan, R. Sharma, Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers, *Proceedings of the 2005 USENIX Annual Technical Conference*, 2005.

[19] E.B. Nightingale, J. Flinn, Energy-Efficiency and Storage Flexibility in the Blue File System, *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, December 2005.

[20] D.A. Patterson, G. Gibson, RH Katz, A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD International Conference on Management of Data*, 1(3):109-116, June 1988.

[21] D. Peek, J. Flinn, Drive-Thru: Fast, Accurate Evaluation of Storage Power Management, *Proceedings of the 2005 USENIX Annual Technical Conference*, 2005.

[22] E. Pinheiro, R. Bianchini, Energy Conservation Techniques for Disk Array-Based Servers, *Proceedings of the 18th Annual ACM International Conference on Supercomputing (ICS'04)*, June 2004.

[23] RFC-3174 - US Secure Hash Algorithm 1, 2001. http://www.faqs.org/rfcs/rfc3174.html

[24] SANTools, Inc. 2005. http://www.santools.com/smartmon.html

[25] D.S. Santry, M.J. Feeley, N.C. Hutchinson, A.C. Veitch, R.W. Carton, J. Ofir, Deciding when to forget in the Elephant File System, *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, 1999.

[26] UMass Trace Repository, Storage Traces, 2005.
http://signl.cs.umass.edu/repository/walk.php?cat=Storage

[27] R. Xu, A. Wang, G. Kuenning, P. Reiher, G. Popek, Conquest: Combining Battery-Backed RAM and Threshold-Based Storage Scheme to Conserve Power, *Work in Progress Report, 19th Symposium on Operating Systems Principles (SOSP)*, October 2003.

[28] X. Yu, B. Gum, Y. Chen, R. Wang, K. Li, A. Krishnamurthy, T. Anderson, Trading Capacity for Performance in a Disk Array, *Proceedings of the 4th Symposium on Operating Systems Design and Implementation*, October 2000.

[29] Q. Zhu, F.M. David, C. Devaraj, Z. Li, Y. Zhou, P. Cao, Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management, *Proceedings of the 10th International Symposium on High Performance Computer Architecture*, February 2004.

[30] Q. Zhu, A. Shanker, Y. Zhou, PB-LRU: A Self-Tuning Power Aware Storage Cache Replacement Algorithm for Conserving Disk Energy, *Proceedings of the 18th Annual ACM International Conference on Supercomputing (ICS'04)*, June 2004.

[31] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, J. Wilkes, Hibernator: Helping Disk Arrays Sleep through the Winter, *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005.