Optimizing Power Consumption in Large Scale Storage Systems

Lakshmi Ganesh, Hakim Weatherspoon, Mahesh Balakrishnan, Ken Birman
Computer Science Department, Cornell University
{lakshmi, hweather, mahesh, ken}@cs.cornell.edu

Abstract

Data centers are the backend for a large number of services that we take for granted today. A significant fraction of the total cost of ownership of these large-scale storage systems is the cost of keeping hundreds of thousands of disks spinning. We present a simple idea that allows the storage system to turn off a large fraction of its disks, without incurring unacceptable performance penalties. Of particular appeal is the fact that our solution is not application-specific, and offers power-savings for a very generic data center model. In this paper, we describe our solution, identify the parameters that determine its cost-benefit tradeoffs, and present a simulator that allows us to explore this parameter space. We also present some initial simulation results that add weight to our claim that our solution represents a new power-saving opportunity for large-scale storage systems.

1 Introduction

The declining costs of commodity disk drives has made online data storage a way of life. So much so that companies like Google and Yahoo host hundreds of thousands of servers for storage. However, there is a catch: a hundred thousand servers consume a lot of power! Not only does this translate to many millions of dollars annually on electricity bills, the heat produced by so much computing power can be searing. An article in The New York Times describes one of Google's data centers: ".. a computing center as big as two football fields, with twin cooling plants protruding four stories into the sky"[9]. Conclusion: Power conservation is an important concern for big server clusters. Since disks account for a significant fraction of the energy consumed[6], several approaches for disk power management have been proposed and studied. We will examine some of these here. But first let us lay out some of the groundwork.

Any disk power management scheme essentially attempts to exploit one fact: disks can be run in high-power mode, or low-power mode, with a corresponding performance tradeoff. In the limit, a disk can be shut off so that it consumes no power. Given a large cluster of disks, only a fraction of them is accessed at any time, so that the rest could potentially be switched to a low-power mode. However, since mode transitions consume time and power, disk management schemes have to walk the tightrope of finding the right balance between power consumption and performance.

The solution space explored thus far in the literature can be divided as follows: (1) Hardware-based solutions, (2) Disk Management solutions, and (3) Caching solutions. Each of these solutions proposes a new system of some kind; *hardware-based solutions* propose novel storage hierarchies to strike the right balance between performance and power consumption; *disk management solutions* interject a new 'disk management layer' on top of the file system, which controls disk configuration and data layout to achieve power-optimal disk access patterns; *caching solutions* devise new power-aware caching algorithms that allow large fractions of the storage system to remain idle for longer periods of time, allowing them to be switched to lower power modes.

The principal contribution of this paper is to argue that there is a fourth niche as yet unexplored: (4) File System solutions. We do not present a new system; instead, we take an idea that has been around for well over a decade now - the Log-Structured File System (LFS) [13] and argue that technological evolution has given it a new relevance today as a natural power-saving opportunity for large-scale storage systems. The key insight is that, where other solutions attempt to predict disk access to determine which disks to power down, the LFS automatically provides a perfect prediction mechanism, simply by virtue of the fact that all write-accesses go to the log head. Section 3 explains and expands on this idea.

1.1 Idea Overview

To see why LFS is a natural solution to the problem of disk power management, consider some of the challenges involved:

- Short Idle Periods: Server systems typically are not idle long enough to make it worthwhile to incur the time+power expense of switching the disk to a low-power mode, and switching it back when it is accessed. This is a notable point of difference between server systems and typical mobile device scenarios (like laptops), which makes it hard to translate the solutions devised for mobile devices to server systems. As we shall see, LFS localizes write-access to a small subset of disks; this feature, when combined with a cache that absorbs read-accesses, results in long disk idle periods.

- Low Predictability of Idle Periods: Previous studies [7] have shown that there exists low correlation between a given idle period's duration and the duration of previous idle periods. This variability makes it difficult to devise effective predictive mechanisms for disk idle times. The LFS neatly circumvents this problem by predetermining which disk is written to at all times.

- Performance Constraints: Server systems are often constrained by Service Level Agreements to guarantee a certain level of performance, so that finding a solution that provides acceptable performance to only a fraction of the incoming requests (albeit a large fraction) may often not be sufficient. As we shall show, the LFS provides an application-independent solution that allows the system to perform consistently across a wide range of datasets.

- The law of large numbers: Large scale server systems process incredibly large request loads. Directing these to a small fraction of the total number of disks (the fraction that is in 'high-power mode') can significantly raise the probability of error and failure. The fact that the disks used in these contexts are typically low-end with relatively weak reliability guarantees, exacerbates this problem. As we shall see, our solution alleviates this problem by making sure that the live subset of disks is not constant.

The rest of this paper is organized as follows: Section 2 describes some of the solutions explored in the first three quadrants mentioned above. Section 3 presents and analyzes our solution, while Section 4 discusses our evaluation methodology and results. We conclude in Section 5.

2 Related Work

Hardware-based Solutions
The concept of a memory hierarchy arose as a result of the natural tradeoff between memory speed and memory cost. Carrera et. al. point out in [1] that there exists a similar tradeoff between performance and power-consumption among high-performance disks and low-performance disks such as laptop disks. They explore the possibility of setting up a disk hierarchy by using high- and low-performance disks in conjunction with each other. In a related vein, Gurumurthi et. al.[8] propose Dynamic Rotations Per Minute (DRPM) technology, whereby disks can be run at multiple speeds depending on whether power or performance takes precedence. DRPM, however, poses a significant engineering challenge whose feasibility is far from obvious.

Another approach is proposed by Colarelli et. al. in [2], using massive arrays of inexpensive disks (MAID). They propose the use of a small number of *cache* disks in addition to the MAID disks. The data in these cache disks is updated to reflect the workload that is currently being accessed. The MAID disks can then be powered down, and need only be spun up when a cache miss occurs, upon which their contents are copied onto the cache disks. This approach has several of the weaknesses that memory caches suffer, only on a larger scale. If the cache disks are insufficient to store the entire working set of the current workload, then 'thrashing' results, with considerable latency penalties. Further, the cache disks represent a significant added cost in themselves.

Disk Management Solutions
Pinheiro and Bianchini [11] suggest that if data is laid out on disks according to frequency of access, with the most popular files being located in one set of disks, and the least popular ones in another, then the latter set of disks could be powered down to conserve energy. Their scheme is called Popular Data Concentration (PDC) and they implement and evaluate a prototype file server called Nomad FS, which runs on top of the file system and monitors data layout on disks. Their findings are that if the low-access disks are powered down, this results in a considerable performance hit; they suggest instead that they be run at low speed. While their idea is sound, it is not clear whether this scheme would adapt to different workloads.

Son et. al. propose another data layout management scheme to optimize disk access patterns [14]. Their approach uses finer-grained control over data layout on disk, tuning it on a per-application basis. Applications are instrumented and then profiled to obtain array access sequences, which their system then uses to determine optimal disk layouts by computing optimal stripe factor, stripe size, start disk etc. Again, the wisdom of marrying the disk layout to the application seems questionable.

Hibernator, proposed by Zhu et. al [6], combines a number of ideas. It assumes multispeed disks, and computes online the optimal speed that each disk should run at. To minimize speed transition overheads, disks maintain their speeds for a fixed (long) period of time - they call this the coarse-grained approach. Hibernator includes a file server that sits on top of the file system and manipulates data layout to put the most-accessed data on the highest speed disks. The authors address the issue of performance guarantees by stipulating that if performance drops below some threshold, then all disks are spun up to their highest speed.

Caching Solutions
Zhu et. al [5] observe that the storage cache management policy is pivotal in determining the sequence of requests that access disks. Hence, cache management policies could be tailored to change the average idle time between disk requests, thus providing more opportunities for reducing disk energy consumption. Further, cache policies that are aware of the underlying disk management schemes (eg. which disks are running at which speeds, say) can make more intelligent replacement decisions. The authors present both offline and online power-aware cache replacement algorithms to optimize read accesses. They also show through experiments the somewhat obvious fact that for write accesses, write-back policies offer more opportunities to save power than write-through policies.

3 A New Solution

We shall now argue that there remains an unexplored quadrant in this solution space. Caches are used to minimize accesses to disk. Good caching algorithms practically eliminate read accesses to disk. However, write accesses (whether synchronous or not) must still eventually access the disk. Thus, assuming perfect caching, disk access will be write-bound. Putting a disk management layer on top of the file-system to optimize data layout for writes is only halfway to the solution. To take this idea to its logical conclusion, it is necessary to rethink the file system itself. In the context of write-access optimization, a very natural candidate is the log-structured file system [13]. We now give a brief overview of the log-structured file system before describing the power-saving opportunity it represents.

3.1 Log-Structured File System

The Log-Structured File System (LFS) was motivated by a need to optimize the latency of write-accesses. Writing a block of data to a Seagate Barracuda disk costs about 11.5ms in seek time and 0.025ms/KB in transmission time. The key observation here is that seek time is a large and *constant* term in latency computation. To eliminate this term, the LFS replaces write operations by append operations. Secondary storage is treated as a large append-only log and writes always go to the log head. Seek time is thus eliminated, and write latency becomes purely a function of the disk bandwidth.

How do reads in the LFS work? In the same way as in conventional file systems! Reads require random-access, and hence do not avoid seek-latency. However, the assumption is that with good caching mechanisms, reads will be a small fraction of disk accesses.

As can be imagined, space reclamation is a tricky problem in log structured file systems. However, excellent solutions have been proposed to solve it, and one such is of interest to us: the disk is divided into large log segments. Once a log segment gets filled, a new log segment is allocated and the log head moves to the new segment. When some threshold of a segment gets invalidated, its valid data is moved to another segment (replacing that segment's invalid data), and it is then added to the pool of free log segments. Over time, this process results in a natural division of allocated segments into stable (ie.. consisting almost entirely of data that is rarely invalidated/modified), and volatile ones (which need to be constantly 'cleaned'). We will see how this feature can be used to save power.

3.2 LFS: A Power-Saving Opportunity

The disk-management policies described in the related works section essentially attack the problem by trying to predict in advance which disk any given access will go to. They optimize the data layout on disks to ensure that accesses are localized to some fraction of the disks, so that only these need be powered up. However, these are all probabilistic models - a new access has some probability of not fitting this model and needing to access a powered-down disk. Further, in such schemes, disk layout becomes tied to particular applications; two applications that have completely different access patterns might require different data layouts on disk leading to conflicts that reduce possible power-savings.

Since all writes in an LFS are to the log head, we know in advance which disk they will access. This gives us the perfect prediction mechanism, at least for write-accesses. Besides being deterministic, this prediction mechanism is also application-independent. Thus, if most accesses to disks were writes, we could power down every disk but the one that the log head resides on. This, however, is an ideal case scenario. Our view is that, with a good caching algorithm (the power-aware caching algorithms described in the related works section are good candidates), reads to disk can be minimized, and only a small fraction of the disks need be powered on in order to serve all writes as well as reads.

However, what about the performance and power costs of log cleaning? Matthews et. al present some optimizations in [4] to hide the performance penalty of log cleaning even when the workload allows little idle time. The power costs of log cleaning are a little more tricky to justify; however, this is where the natural division of segments into stable and volatile ones that the log cleaning process results in (as described above) can help. After a significant fraction of segments on a disk have been classified as stable, volatile, or free, we power the disk on and copy the stable segments to a 'stable' disk, volatile segments to a 'volatile' disk (disk is kept on), and the entire disk is freed for reuse. This is similar to the log cleaning scheme described in [10], which uses a 'hidden' structure embedded in the log to track segment utilization. Cleaning an entire disk amortizes the cost of powering it on.

4 Evaluation

4.1 Methodology

We have proposed the use of LFS in lieu of conventional file systems in data-center scenarios to achieve power conservation. For this idea to be accepted, two questions need to be answered in the affirmative: (1) *Does this new scheme result in significant power savings?*, and (2) *Does this new scheme provide comparable performance to existing schemes?* Further, the answers to these questions must be largely application-independent, and must apply to a generic data center model. To address these questions, we present a simulator - Logsim. Logsim consists of less than a thousand lines of Java code and is a single-threaded, discrete event-based simulator of a log-structured file system. Given a trace of read and write requests, Logsim returns the observed access latencies, disk utilization, cache-hit ratio, disk-mode transitions etc., for the chosen set of configuration parameters. We use real-world traces for our simulations from a web-server that serves images from a database[12]. Table 1 describes the characteristics of a sample trace. While a true evaluation of the feasibility and efficacy of our solution can only be achieved through an actual implementation, simulation provides an elegant way to identify and explore some of the cost-benefit tradeoffs in a scaled-down version of our system.

Number of accesses	476884
Number of files touched	23125
Number of bytes touched	4.22GB
Average number of bytes/access	8.8 KB

Table 1: Sample Trace Characteristics.

The mechanism we simulate is as follows: All (non-empty) disks are assumed to begin in the 'on' state, and an access count (an exponentiated average) is maintained for each disk. The user specifies the maximum percentage (m) of disks that are kept powered on. Periodically (200ms, in our experiments), a 'disk check' process scans the access count for each disk and powers down all but the most-accessed top $m\%$ of the disks, as well as any disk which does not have at least t access count. t is 0 in our experiments. If a cache-miss results in an access to a powered-down disk, then this disk is spun up (to remain powered on until the next 'disk check'), and there is a corresponding latency penalty. Judicious choice of m and t minimizes the probability of this occurrence.

4.2 Results

To save power, we must turn off some percentage of disks in the storage system. However, there are two opposing forces at play here. A large number of powered-on disks results in good performance (low latency), but also low power savings. On the other hand, decreasing the number of powered-on disks incurs two possible penalties: increased latencies, and increased mode-transitions. Mode-transitions consume power and thus counter the potential savings achieved by powered-down disks. To find the optimal percentage of disks to be powered down, we ran a set of simulations on Logsim and varied the number of disks that we kept powered up from none (except the log-head disk), to all, in steps of 20%. Thus, out of a total of 66 disks, 1, 13, 26, 39, 52, and 66 disks were kept powered up, respectively. For each run, we examine both its performance (in terms of observed access latencies), as well as its power-consumption. Fig. 1, 2 and 3 show the results of these simulations.

The performance of our system depends heavily on its cache configuration. Since cache optimization is

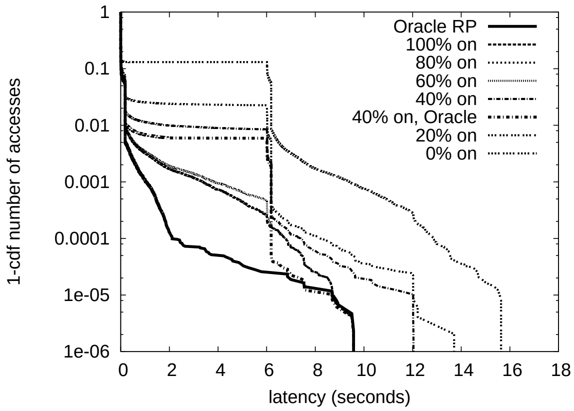

Figure 1: CCDF: Effect of increasing percentage of powered-up disks on performance.

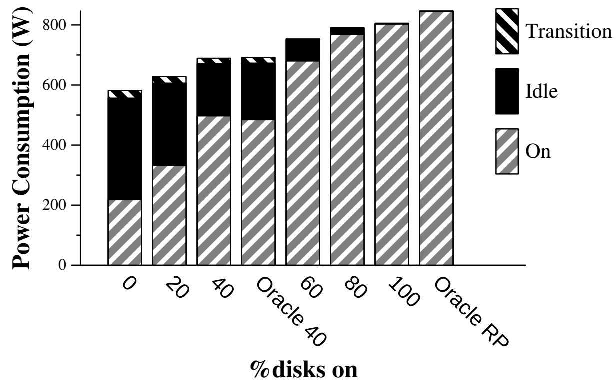

Figure 2: Effect of increasing percentage of powered-up disks on power consumption

an orthogonal issue that comprises an entire field of research in itself, it is important to isolate its effect on performance. To achieve this, we implemented an 'ideal cache' algorithm, which we term the *oracle*. Experiments using the oracle approximate the best performance we could achieve since an oracle has future knowledge and is able to replace items accessed furthest in the future [3]. In fig. 1, 2, 3, the data points that use this algorithm are annotated with the word 'Oracle'.

Finally, we also wish to compare our system against conventional (not log-structured) file systems. As an approximation of such a system, we implemented a 'random placement' (RP) algorithm, which maps each block to a random disk. All disks are kept powered up, and ideal caching (oracle) is assumed. This data point is labeled 'Oracle RP' in our graphs.

Having set the context, let us examine our results. Fig. 1 shows the relation between performance (per-access latency) and the number of disks that are powered on. If we imagine a line at y=.001 (ie.. 99.9% of the accesses live above this line) 60% disks ON is the third best configuration, next only to the Oracle RP and 100% disks ON configurations. Further, the performance degradation in going from 100% disks ON to 60% disks ON is negligibly small. The principal take-away is that, for the system under test, the optimal configuration is to have 60% of the disks powered on. In other words, 40% of the disks can be spun down while still maintaining performance comparable to that of a conventional file system.

Fig. 2 shows an estimate of the actual power savings achieved by our solution. The height of each bar is the average power consumed while processing the trace. Further, each bar shows the break-up of power consumed by powered-up disks (On), powered-down disks (Idle), and mode-transitions (Transition). We assume the following disk specifications: Avg. operating power = 12.8 W, Avg. idle power = 7.2 W, Avg. mode transition power = 13.2 W, Avg time for transition = 6s. We see that turning off 40% of the disks results in 12% power savings (as compared to 32% with all the disks off), while maintaining acceptable performance.

Finally, fig. 3 shows how much time the disks spend in on/off/transition states. The height of each bar is the cumulative time spent by each disk in each of these three states. When 0% disks are on, the run takes 7253 cumulative hours; we omit this bar from our graph for clearer presentation. We see that, both the total duration of the experiment, as well as the number of mode-transitions, increase as the percentage of disks that is powered on is decreased. However, as in fig. 1, we see that keeping 60% disks on strikes an acceptable balance.

5 Conclusion

In this paper, we point out a new opportunity for saving power in large-scale storage systems. The idea is elegant in its simplicity: log structured file systems write only to the log head; as a result, if read accesses are served by the cache, then write accesses touch only the log head disk, potentially allowing us to power down all the other disks. Existing solutions like disk management solutions and caching solutions are typically application-specific; our solution, on the other hand, is applicable to any cacheable dataset. Since existing solutions are typically layered on top of the file-system, they could be used in conjunction with our solution to take advantage of application-specific optimizations.

We also provide some initial simulation results that

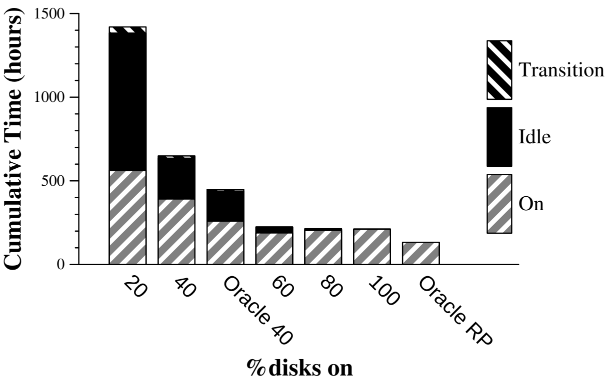

Figure 3: Effect of increasing percentage of powered-up disks on trace running time.

validate our claim that power-savings are possible using a log-structured file system. While simulations cannot provide conclusive evidence for the feasibility of a system, they are an effective means to identify promising solutions. Our principal contribution in this paper is in having shown a new fit for an old idea; we believe that the log-structured file system shows promise as a power-saving opportunity for large-scale storage systems.

In future work, several questions still remain to be addressed. Our evaluation has been limited by the difficulty of obtaining real filesystem traces from commercial data centers; we are actively looking for more recent traces to test our solution against. We are also working on a more thorough study of the efficacy of the log cleaning approach we outline here. Finally, we believe the LFS provides an interesting substratum to build more elaborate solutions on, and we are working on some promising options that we hope to share with the community soon.

Acknowledgments

This work was partially funded by Intel Corporation and the National Science Foundation. Special thanks to Saikat Guha for his input in the simulator design. We also wish to thank our anonymous reviewers for their valuable feedback.

References

[1] E. Carrera, E. Pinheiro, and R. Bianchini. Conserving disk energy in network servers. In *ICS '03: Proceedings of the 17th International Conference on Supercomputing*, June 2003.

[2] D. Colarelli, D. Grunwald, and M. Neufeld. The Case for Massive Arrays of Idle Disks (MAID). In *The 2002 Conference on File and Storage Technologies*, 2002.

[3] Peter J. Denning. The working set model for program behavior. *Commun. ACM*, 11(5):323–333, 1968.

[4] J. Matthews et. al. Improving the performance of log-structured file systems with adaptive methods. In *SOSP '97*, October 1997.

[5] Q. Zhu et. al. Reducing energy consumption of disk storage using power-aware cache management. *HPCA*, 00:118, 2004.

[6] Q. Zhu et. al. Hibernator: helping disk arrays sleep through the winter. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, 2005.

[7] S. Gurumurthi et. al. Interplay of energy and performance for disk arrays running transaction processing workloads. In *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2003.

[8] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Reducing disk power consumption in servers with drpm. In *IEEE Computer*, Dec 2003.

[9] J. Markoff and S. Hansell. Hiding in Plain Sight, Google Seeks More Power. In *The New York Times, Online Ed., June 14*, 2006.

[10] Oracle. Berkeley db java edition architecture. An Oracle White Paper, September 2006. http://www.oracle.com/database/berkeley-db/je/index.html.

[11] Eduardo Pinheiro and Ricardo Bianchini. Energy conservation techniques for disk array-based servers. In *ICS '04: Proceedings of the 18th annual international conference on Supercomputing*, 2004.

[12] D. Roselli and T. Anderson. Characteristics of file system workloads. In *UCB/CSD Dissertation*, 1998.

[13] Mendel Rosenblum and John K. Ousterhout. The design and implementation of a log-structured file system. *ACM Transactions on Computer Systems*, 10(1):26–52, 1992.

[14] S. W. Son, G. Chen, and M. Kandemir. Disk layout optimization for reducing energy consumption. In *ICS '05: Proceedings of the 19th annual international conference on Supercomputing*, 2005.