Heat-Based Dynamic Data Caching: A Load Balancing Strategy for Energy-Efficient Parallel Storage Systems with Buffer Disks

Ziliang Zong[1], Xiao Qin[2†], Xiaojun Ruan[2], and Mais Nijim[3]
[1]Mathematics and Computer Science Department, South Dakota School of Mines, Rapid City, SD, USA
[2]Department of Computer Science and Software Engineering, Auburn University, Auburn, AL, USA
[3] Department of Electrical Engineering and Computer Science, Texas A&M – Kingsville, Kingsville, TX, USA
ziliang.zong@sdsmt.edu *{xqin, xzr0001}@auburn.edu* mais.nijim@tamuk.edu

Abstract

Performance improvement and energy conservation are two conflicting objectives in large scale parallel storage systems. In this paper, we propose a novel solution to achieve the twin objectives of maximizing performance and minimizing energy consumption of parallel storage systems. Specifically, a buffer-disk based architecture (BUD for short) is designed to conserve energy. A heat-based dynamic data caching strategy is developed to improve performance. The BUD architecture strives to allocate as many requests as possible to buffer disks, thereby keeping a large number of idle data disks in low-power states. This can provide significant opportunities for energy conservation while making buffer disks a potential performance bottleneck. The heat-based data caching strategy aims to achieve good load balancing in buffer disks and alleviate overall performance degradation caused by unbalanced workload. Our experimental results have shown that the proposed BUD framework and dynamic data caching strategy are able to conserve energy by 84.4% for small reads and 78.8% for large reads with slightly degraded response time.

1. Introduction

We are now in an era of information explosion. Billions of data sets are generated from knowledge and information every day, which need to be stored in some form of digital media. Historically, tape libraries are preferred over disk arrays for massive data storage, in large part due to the capacity and cost differential between tapes and disks. Over the last decade the original tape systems have been gradually replaced by parallel disk systems because of the continuous expansion of disk capacity and continuous drop of disk price.

Modern parallel storage systems are able to provide higher performance at the cost of enormous energy consumption. For example, a typical robotic tape system provided by StorageTek would have an aggregate bandwidth of 1200MB/s [1] while a modern disk array could easily provide a peak bandwidth of 2,880,000MB/s. However, reading and storing 1,000TB of information would cost $9,400 to power the tape library system vs. $91,500(almost ten times) to power the disk array [7]. The gap is likely to increase when faster disks with higher power consumption rates appear and are widely deployed. A recent industry report showed that storage devices account for almost 27% of the total energy in a data center [3]. Even worse, this fraction tends to increase as storage requirements are rising by 60% annually [2]. As a result, new technologies for energy-efficient parallel storage systems are highly desirable.

In this paper, we introduce a buffer disk based architecture *(BUD for short)* to build energy efficient parallel storage systems. The basic idea of BUD is to conserve energy by serving most of the requests using a small number of buffer disks and turning as many data disks as possible to the power-saving mode. Nevertheless, one potential problem of the BUD architecture is that the limited number of buffer disks may easily become the performance bottleneck. Worst case access patterns can direct all requests to a single buffer disk, resulting in arbitrarily large delays even for very small arrival rates. Therefore, effective load balancing is critical to improve the performance.

The contributions of this study are two-fold. First, we designed the energy-efficient parallel storage system architecture (BUD) with buffer disks. Second, we developed a heat-based dynamic data-caching algorithm to achieve load balancing in BUD.

2. Related Work

Several techniques proposed to conserve energy in storage systems include dynamic power management [4], power-aware cache management [6], power-aware prefetching strategy [8], software-directed power management scheme [9], and multi-speed settings

[†] The work reported in this paper was supported by the US National Science Foundation under Grants No. under Grants No. CNS-0915762 (CSR), CCF-0845257 (CAREER), CNS-0757778 (CSR), CCF-0742187 (CPA), CNS-0917137 (CSR), CNS-0831502 (CyberTrust), CNS-0855251(CRI), OCI-0753305 (CI-TEAM), DUE-837341 (CCLI), and DUE-0830831 (SFS) No. 2005-04-070.

strategy [13]. However, none of these techniques have addressed the energy conservation and performance issues of parallel storage systems with buffer disks.

In 2002, Colarelli and Grunwald presented the “Massive Arrays of Idle Disks” or MAID architecture [7], which is similar to our BUD architecture. In MAID, it is challenging to deal with a mapping structure of active drives and passive drives, i.e. which buffer disk should be chosen as the candidate to cache the data whenever there is a data miss. Moreover, the load balancing issue, which very likely could lead to performance penalty, is not completely solved.

Another framework similar to MAID and BUD, called Popular Data Concentration (PDC), was proposed by Pinheiro and Bianchini in 2004 [17]. The basic idea of PDC is to migrate data across disks according to frequency of accesses, or popularity. The goal is to store popular and unpopular data on different disks. The disks that store unpopular data are mostly idle, so that they can be transitioned to a low-power mode. PDC is an efficient static offline algorithm. However, it is impossible in some cases for the system to predict which data is popular and which is not. This is especially true for the evolving workload, in which some data is popular at a particular period but becomes unpopular later.

In contrast to both MAID and PDC, our heat-based algorithm is implemented to control data caching in addition to data mapping between data disks and buffer disks in the BUD architecture. The heat-based algorithm was first proposed by Scheuermann *et al.* in 1998 [16]. Our algorithm is fundamentally different from theirs in that the heat of disks is applied in the data partitioning stage in their algorithm, whereas ours quantifies the heat of buffer disks in the data caching stage. Their approach focused on how to partition data to improve throughput, while our focus is to judiciously cache data to achieve load balancing.

3. Buffer-Disk Architecture

The buffer disk architecture [19] (see Fig.1) consists of four major components: a RAM buffer, *m* buffer disks, *n* data disks, and a buffer-disk controller. The buffer disks temporarily cache the requests for the data disks. Data disks remain in the low power state unless a read request misses in the buffer disk or the write log for a specific data disk grows too large. The buffer-disk controller is at the heart of BUD. The controller contains the energy related algorithms, data partitioning algorithms, data movement/placement strategies, and prefetching strategies. Our ultimate objective in this research is to conserve more energy without adversely affecting the performance of disk systems. More specifically, the controller strives to move the frequently accessed data from data disks into buffer disks. This allows as many data disks as possible to switch to low-power modes. The rationale behind this strategy relies on the fact that only a small percentage of the data is frequently accessed in a wide variety of data-intensive applications [14]. To achieve this goal, we proposed the heat-based data caching algorithm to dynamically balance the workload among buffer disks. This algorithm aids in avoiding the potential “traffic jam” caused by overloaded buffer disks. Please note that all our solutions and experimental results illustrated in the following sections are primarily based on read requests.

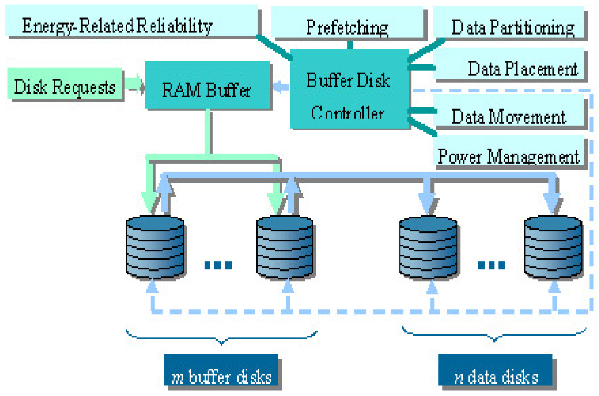

Fig. 1 The buffer disk architecture

4. Heat-Based Load Balancing Algorithm

To conserve energy, BUD makes an effort to run most data disks in the low power state, thereby directing most traffic to a limited number of buffer disks. This can potentially make the buffer disks overloaded and become the performance bottleneck. Load balancing is one of the solutions for the inherent shortcoming of the BUD architecture. Basically, there are three types of load balancing strategies called non-random load balancing, random load balancing, and redundancy load balancing. Sequential mapping belongs to non-random load balancing, because the buffer disks have fixed mapping relationships with specific data disks (e.g. buffer disk 1 can only cache the data from data disk 1, 2 and 3). The round-robin mapping is a typical random load balancing strategy by allocating data to each buffer disk with equal portions and in the round robin order. Redundancy load balancing strategies for storage systems include EERAID [10], eRAID [11], and RIMAC [12]. In this section, we present a heat-based load balancing strategy, which is also a random load balancing strategy. The primary objective of our strategy is to minimize the overall response time of disk requests by keeping all buffer disks equally loaded.

The basic idea of heat-based mapping is that blocks in data disks are mapped to buffer disks based on their heat, i.e. frequencies of accesses. Our goal is to make the accumulated heat of data blocks allocated to each buffer disk the same or close to this ideal situation.

To clearly describe the heat-based load balancing algorithm, we define the key parameters as follows.

Access Frequency: how many times a data block is repeatedly accessed within a specific time unit.

Heat weight: the ratio of requested data size and standard data size (e.g., 1MB).

Heat: the product of access frequency and heat weight.

Temperature: the accumulated heat of all data blocks residing in a buffer disk.

The heat is used to measure the popularity of a data block. The temperature indicates how busy a buffer disk is. To calculate the heat more accurately, we also consider the impact of block size. A large block size should have higher heat compared to a small block with the same access frequency. This is due to the fact that the system will spend more time to process I/O for the larger block. In other words, the larger blocks should have higher heat weight.

Since our algorithm is executed online, dynamic tracking of the heat of blocks is crucial. We implemented two strategies to dynamically track heat values. In the first strategy, the controller will take the first *k* requests of the request queue and run the heat calculation function. Once the *k* requests are captured, they will be removed from the main request queue in the memory. We call these *k* requests a snapshot request window, which will be the input of the heat-based load balancing algorithm. However, the snapshot window strategy is only suitable for bursty request patterns but not for sparse request patterns. When a sparse request pattern is encountered, it may take too long to collect a snapshot window of *k* requests. Therefore, we designed the second strategy called the observation time window strategy. In this strategy, the controller will serve the requests that arrive within a specific observation time (e.g. t seconds), no matter how many requests arrive.

Fig. 2 outlines the pseudo code of the heat-based load balancing algorithm. Note that the input parameter

```
1.  Input:  the request window ;          /*  request window  will be updated periodically */
2.  for each unique target block in the queue       /* each request has a target block to be accessed */
3.      AF = Access_Frequency_Calculation() ;             /* calculate the block access frequency*/
4.      HW = accessed block size/ standard block size;                   /*calculate the heat weight*/
5.      heat = AF * HW;                                                  /*calculate the heat */
6.  sort the data blocks based on heat and save them in Linklist_Block; /* first block has the highest heat */
7.  sort the buffer disk based on current temperature to a Linklist_Buffer ;/* first disk has lowest temperature*/
8.  pointer p_buffer = the first buffer disk in the Linklist_Buffer;
9.  pointer p_block = the first block in the Linklist_Block;
10. pointer t_buffer ;  /* t_buffer points to the buffer disk which have the copy of target block*/
11. for each block in the Linklist_Block
12.    if (p_block.found = = false)                  /* the target block cannot be found in buffer disks*/
13.        if (p_buffer. free = = true)        /* the candidate buffer disk has enough space*/
14.            wake up the corresponding data disk  and cache the data;
          /* The data blocks within the batch prefetching window will be copied to the buffer disk p_buffer;
15.           dispatch all requests accessing p_block to p_buffer;
16.           recalculate and update the information of block heat and buffer disk temperature ;
17.        else /* the first candidate buffer disk has no space*/
18.            if (p_buffer.next != empty)
                  p_buffer ++;                 /* seek another candidate buffer disk*/
19.               go to setp 12;
20.            else /* all buffer disks are already full*/
21.               reset p_buffer to the first buffer disk in the Linklist_Buffer;
22.               data_replace_function(p_buffer);  /* replace existing data blocks using LRU algorithm */
23.               dispatch all requests accessing p_block to p_buffer;
24.               recalculate and update the information of block heat and buffer disk temperature ;
25.    else /* p_block is found in one buffer disk t_buffer */
26.           dispatch all requests accessing p_block to t_buffer ;
27.           recalculate and update the information of block heat and buffer disk temperature ;
```

Fig. 2 Heat-based load balancing algorithm

can be either a snapshot window or an observation time window. This algorithm will periodically collect the requests waiting in the queue, analyze the target block of each read request, and calculate the heat of each unique block. If the target block cannot be found in the buffer disk, the controller initiates a data miss command. This in turn will wake up the corresponding data disk in order to copy the block to the buffer disk that has the lowest temperature. In a special case, the selected buffer disk may not have free space to store a new data block. The controller will seek the next buffer disk with a temperature that is higher than the initial buffer disk selected, but still lower than any other buffer disks. In the worst case, no candidate buffer disk will be found because all buffer disks are full. A data replacement function using the Least Recently Used (LRU) algorithm will be executed to evict some existing data blocks. If the target block has already been cached in one of the buffer disks, then that buffer disk will serve the corresponding request. Once the algorithm has made the decision how to dispatch these requests, the block heat and buffer disk temperature need to be recalculated and updated accordingly.

5. Performance Evaluation

To simulate the BUD architecture, we implemented a simulator, called BUD_Sim, using the Java language. In BUD_Sim, we calculate the seek time as a non-linear function (Table 1) of the seek distance using the seek-time-versus-distance curve presented in [15]. In addition, we implemented a load generator, which can generate synthetic workloads according to specified parameter distributions, analyze and filter real traces and feed manipulated traces as the input to BUD_Sim.

Table 1: Seek time calculation

Seek distance	Seek time (ms)
< 616 cylinders	$3.45 + 0.597\sqrt{d}$
≥ 616 cylinders	$10.8 + 0.012\,d$

In BUD_Sim, we simulate the buffer disks using IBM 36Z15 Ultrastar, and data disks using IBM 73LZX Ultrastar. Table 2 illustrates the detailed parameters of these two types of disks, which are from IBM manuals and power measurements published in [18]. In Table 3, we summarize the important parameters that have been used in our simulation.

5.1 Evaluation of energy consumption

This set of experimental results aims at evaluating the energy efficiency of the buffer disk based parallel storage systems. To fairly compare the results, we generated and executed a large number of requests and simulated both large reads (average data size is 64MB) and small reads (average data size is 64KB). Fig. 3 and Fig. 4 plot the total energy consumption of NO-buffer and Heat-BUD when running 2000, 5000, 10000, and 20000 large read requests and small read requests, respectively.

Table 2: Hardware characteristics of disks

Parameters	IBM 36Z15 Ultrastar **(high perf.)**	IBM 73LZX Ultrastar **(low perf.)**
Standard interface	SCSI	SCSI
Number of platters	4	2
Rotations per minute	15000	10000
Average seek time	3.4 ms	4.9 ms
Average rotation time	2 ms	3 ms
Transfer rate	55 MB/sec	53 MB/sec
Power (active)	13.5 W	9.5 W
Power (idle)	10.2 W	6.0 W
Power (sleep)	2.5W	1.4W
Energy (spin down)	13.0 J	10.0 J
Time (spin down)	1.5 s	1.7 s
Energy (spin up)	135.0 J	97.9 J
Time (spin up)	10.9 s	10.1 s

Table 3: Important parameters

Parameters	**Range/Value**
# of requests:	{2000,5000,10000,20000}
# of buffer disks	3
# of data disks	30
data block size	{64KB, 1MB, 4MB, 64MB}
average interval (light load trace)	2.5s
average interval (heavy load trace)	0.5s

There are three important observations here. First, the BUD can significantly conserve energy compared with No-Buffer parallel storage systems. Second, the more requests BUD serves, the more potential power savings is revealed. For example, BUD outperforms No-Buffer in terms of energy conservation by 75.83%, 77.89%, 80.18% and 81.16% for 2000, 5000, 10000, and 20000 large reads, respectively. This is expected because more requests lead to more opportunities for BUD to keep the data disks in the low power state. Third, BUD performs better for small reads (average 84.4% improvement) than large reads (average 78.77% improvement). The reason is that BUD consumes more energy when moving large data blocks to buffer disks.

5.2 Evaluation of load balancing

In this section, we evaluate the load balancing ability of the heat-based algorithm. Recall that the temperature of a buffer disk clearly indicates how busy it is. In order to demonstrate how the dynamic load balancing works, we plot the temperature tracking trace of the initial stage and intermediate stage in Fig. 5 and

Fig. 6. At the initial stage, the three buffer disks are not load balanced. Buffer disk 2 is the busiest disk and buffer disk 1 is lightly loaded. As a result, the heat-based algorithm will keep allocating requests to buffer disk 1. We observe that the temperature of buffer disk 1 keeps growing and it catches buffer disk 3 first. After that, the temperatures of buffer disk 1 and 3 cross-rise for a while and then they catch buffer disk 2. At this point, the system is load balanced for the first time. Fig. 6 shows that the entire system is perfectly load balanced in the intermediate stage because the temperatures of three buffer disks rise in turns.

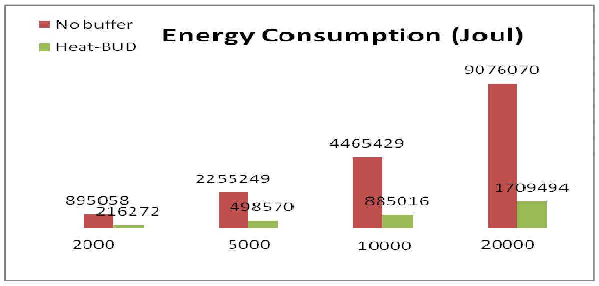

Fig. 3 Energy consumption for large reads

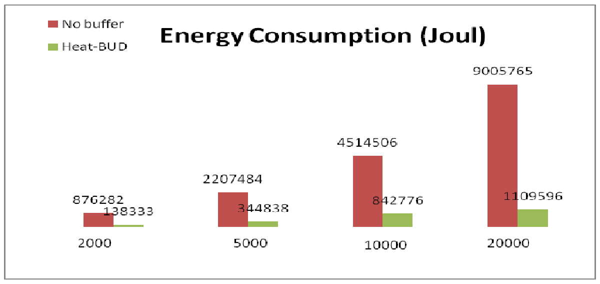

Fig. 4 Energy consumption for small reads

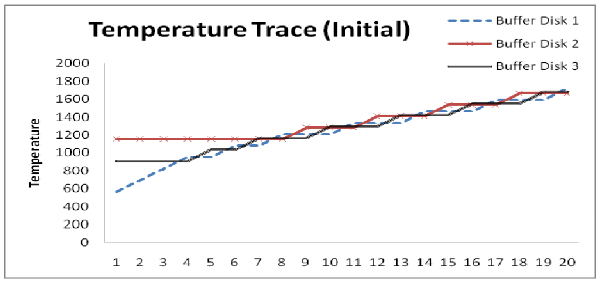

Fig. 5 Temperatures in initial stage

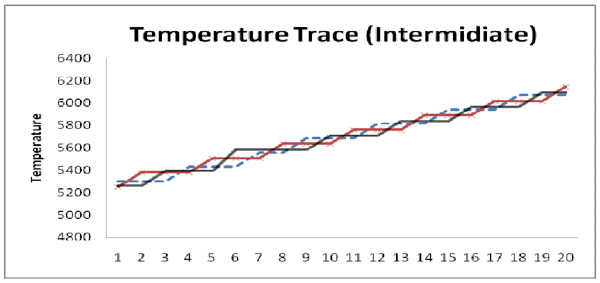

Fig. 6 Temperatures in intermediate stage

To compare the load balancing efficiency of sequential mapping, round robin mapping, and heat-based mapping, we tested 2500 requests with average data size of 4MB using these three mapping strategies. The simulation results depicted in Fig.7 prove that our heat-based algorithm outperforms the other two algorithms in terms of achieving load balancing.

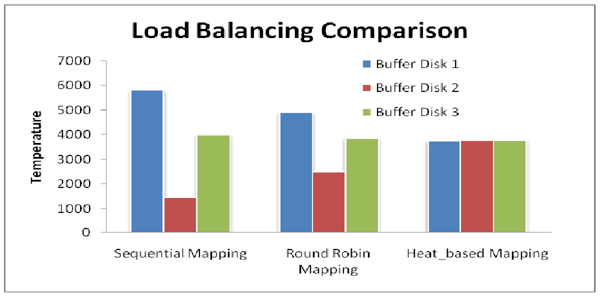

Fig. 7 Load balancing comparison

5.3 Evaluation of response time

To evaluate the average response time of the BUD architecture, we simulated 25000 disk requests for large reads (64MB) and small reads (64KB), which are illustrated in Figs. 8-11 respectively. For each experiment, BUD first processes 20000 requests to complete the caching stage. Next, we let BUD handle 5000 more disk requests to see whether or not the system can leverage the response time delay. We plot the trend line in each figure (the black line inside) to better analyze the response time changes. The trend line is plotted by calculating the average response time of every 100 disk requests.

Figs. 8 and 10 verify our prediction of the response time delay in the early caching stage. For example, Fig. 8 reveals that the response time delay rises up to 140s. However, we are very delighted to witness the performance improvement when more and more popular data blocks are cached in the buffer disks (Fig. 9 and Fig. 11). After the initial caching stage, the average response time is very close to the performance of an optimized No-Buffer parallel storage system. For example, the average response time of BUD is 1.219s for large reads and 0.01s for small reads (as shown in Table 4).

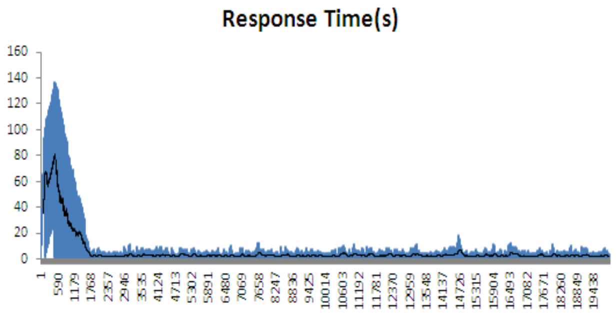

Fig. 8 Response time trace before caching (64MB)

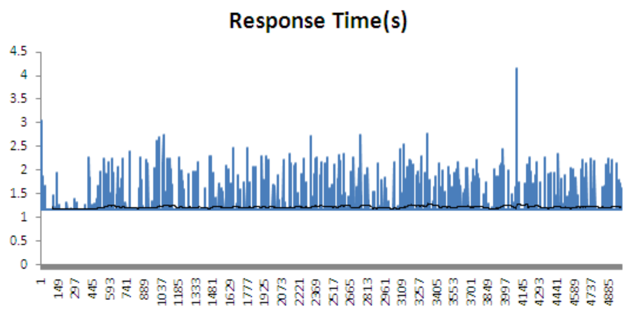

Fig. 9 Response time trace after caching (64MB)

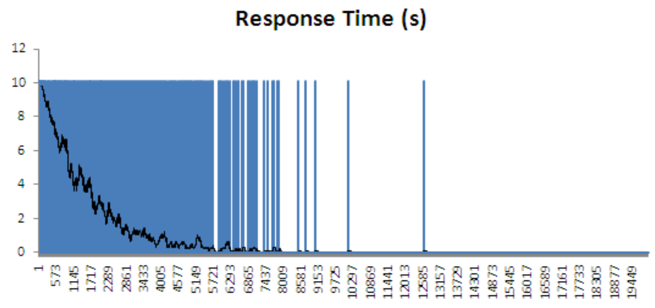

Fig.10 Response time trace before caching (64KB)

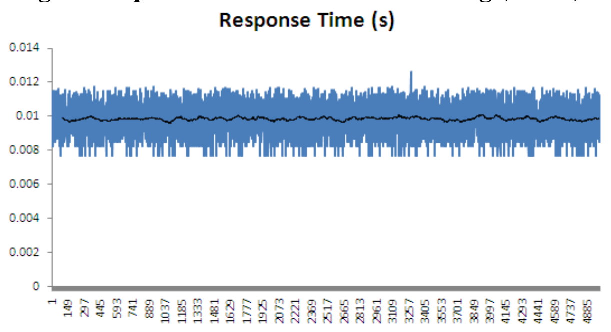

Fig. 11 Response time trace after caching (64KB)

Table 4: Average response time comparison

	Average Response Time
before caching (64MB):	5.614s
after caching (64MB):	1.219s
before caching (64KB):	0.767s
after caching (64KB):	0.01s
NO-Buffer(64MB)	1.216s
NO-Buffer(64KB)	0.01s

References

[1] StorageTek Corp. 9310 tape silo information, http://www.storagetek.com/products/tape/9310/2001 .

[2] Fred Moore, "More Power Needed," Energy User News, Nov. 2002.

[3] "Power, heat, and sledgehammer," White paper, Maximum Institution Inc., http://www.max-t.com/downloads/whitepapers/SledgehammerPowerHeat20411.pdf, 2002.

[4] F. Douglis, P. Krishnan, and B. Marsh, "Thwarting the Power-Hunger Disk," *Proc. Winter USENIX Conf.*, pp.292-306, 1994.

[5] K. Li, R. Kumpf, P. Horton, and T. E. Anderson, "A Quantitative Analysis of Disk Drive Power Management in Portable Computers," *Proc. Winter USENIX Conf.*, pp.279-292, 1994.

[6] Q. Zhu, F. M. David, C. F. Devaaraj, Z. Li, Y. Zhou, and P. Cao, "Reducing Energy Consumption of Disk Storage Using Power-Aware Cache Management," *Proc. High-Performance Computer Framework*, 2004.

[7] D. Colarelli and D. Grunwald. "Massive Arrays of Idle Disks for Storage Archives," *Proc. 15th High Performance Networking and Computing Conf.*, Nov. 2002.

[8] S.W. Son and M. Kandemir, "Energy-aware data prefetching for multi-speed disks," *Proc. ACM Int'l Conference on Computing Frontiers*, Ischia, Italy, May 2006.

[9] S.W. Son, M. Kandemir, and A. Choudhary, "Software-directed disk power management for scientific applications," *Proc. Int'l Symp. Parallel and Distributed Processing*, Apr. 2005.

[10] D. Li and J. Wang, "EERAID: Energy-Efficient Redundant and Inexpensive Disk Array," *Proc. 11th ACM SIGOPS European Workshop*, Sep. 2004.

[11] J. Wang, H.J. Zhu, D. Li, "eRAID: Conserving Energy in Conventional Disk-Based RAID System," *IEEE Transactions on Computers*, vol. 57, no. 3, pp. 359-374, Mar. 2008.

[12] X. Yao and J. Wang, "RIMAC: A Redundancy-based Hierarchical I/O Architecture for Energy-Efficient Storage Systems," *Proc. 1st ACM EuroSys Conf.*, Apr. 2006.

[13] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Fanke, "DRPM: Dynamic Speed Control for Power Management in Server Class Disks," *Proc. Int'l Symp. of Computer Framework*, pp. 169-179, Jun. 2003.

[14] T.T. Kwan, R.E. McGrath, and D.A Reed, "NCSA's World Wide Web Server: Design and Performance," *IEEE Computer*, vol. 28, no. 11, pp. 68 – 74, Nov. 1995.

[15] C. Ruemmler, J. Wilkes, "An introduction to Disk Drive Modeling," *IEEE Transactions on Computers*, vol. 27, no. 3, pp. 17 – 28, 1994.

[16] P. Scheuermann, G. Weikum, P. Zabback, "Data partitioning and load balancing in parallel disk systems," I*nt'l Journal on Very Large Data Bases*, vol. 7, issue 1, pp. 48-66, Feb.1998

[17] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. 18th Int'l Conference on Supercomputing (ICS'04),* Jun. 2004.

[18] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers", *Proc. 17th Int'l Conf. on Supercomputing (ICS'03),* Jun. 2003.

[19] Z. L. Zong, M.E. Briggs, N.W. Connor, and X. Qin, "An Energy-Efficient Framework for Large-Scale Parallel Storage Systems," *Proc. IEEE Int'l Parallel & Distributed Processing Symp. Workshop*, Mar. 2007.