Workload-Adaptive Management of Energy-Smart Disk Storage Systems

Ekow Otoo, Doron Rotem, Shih-Chiang Tsao
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720
{ejotoo, d_rotem, weafon}@lbl.gov

Abstract

Recent studies have identified disk storage systems as one of the major consumers of power in data centers. Many disk power management (DPM) schemes were suggested where the power consumed by disks is reduced by spinning them down during long idle periods. Spinning the disks down and up results in additional energy and response time costs. For that reason, DPM schemes are effective only if the disks experience relatively long idle periods and the scheme does not introduce a severe response time penalty. In this paper we introduce a dynamic block exchange algorithm which switches data between disks based on the observed workload such that frequently accessed blocks end up residing on a few "hot" disks thus allowing the majority of disks to experience longer idle periods. We validate the effectiveness of the algorithm with trace-driven simulations showing power savings of up to 60% with very small response time penalties.

keywords: Disk storage, Energy Efficiency, File allocation, Scientific workload, Workload-Adaptive Management.

1 Introduction

High energy costs of data centers have become an emerging critical issue. Among the various components of a data center, spinning disks and their cooling are known to be one of the major consumers of energy with budget estimations of about $7–9 million per year in medium size data centers. This trend is expected to continue as both commercial and scientific organizations need to store huge volumes of data and on the other hand disk vendors are introducing faster spinning disks which have higher power requirements. [21].

Recognizing this trend, researchers and vendors developed the concept of MAID (massive array of idle disks) [2, 4] storage systems where only a small fraction of the disks in the storage array are spinning while the rest of the disks are kept in standby mode until some data residing on them is requested. Powering up a standby disk to serve a request incurs a penalty of both time (typically about 10-15 seconds) and energy. For that reason, careful optimizations must be performed to determine when to power off disks. More specifically, a decision during runtime must be made concerning the optimal threshold on the length of the idle period after which it is beneficial to power the disk off. The main factors that must be considered are the expected energy to spin up a powered-off (standby) disk as compared with the energy savings realized during the standby period. Another factor that must be weighed is the expected response time penalties incurred by requests made to standby disks.

It is well known that energy savings can be improved by extending the expected length of the disk idle period. Several methods have been used to extend the expected idle time of disks, these include using a memory or SSD based cache such that disks do not need to be powered on when reading blocks which are in

the cache. Although the use of a cache may reduce response time and extend the idle period of disks, it may not reduce much power if the idle period is not long enough. Our experiments and several recent research papers show that the presence of a cache by itself provides only small improvements to energy consumption. More details about these solutions are presented in Section 2. Thus, besides cache, we need other ways to further prolong the idle time of disks.

Another approach for extending the disk idle period is to exchange data between disks such that the most frequently accessed data ends up residing on a few of the disks which are kept almost always spinning (active state) while the rest of the disks can be powered off most of the time. In this paper we present a new dynamic algorithm for performing such data exchanges. The general idea is best explained in terms of temperatures of the disks and of Exchange Blocks (XB), where each XB may consist of several physical disk blocks. The temperature of a disk is determined by the arrival rate of requests to it. It becomes hotter with higher arrival rates. Similarly, an XB is hot if it is accessed frequently otherwise it is considered cool. Our exchange algorithm moves relatively cool XBs from hot disks and replaces them with hot XBs taken from cool disks. At the same time, to guarantee desirable response times, disks that are overloaded are cooled off by removing hot XBs from them to cooler disks.

Our main contributions are:

- We propose a new dynamic block exchange algorithm which may save up to 50% of energy consumption while satisfying response time constraints.
- We use a queuing model and measurements of observed workloads in order to re-distribute the workload among the disks by exchanging blocks such that a small fraction of the disks are kept spinning while the rest can be placed in standby mode.
- The algorithm can be used with off-the-shelf disk storage systems such as nested RAID or MAID with or without the presence of SSD caches.
- We developed efficient data structures to keep track of disk block maps to allow fast location of blocks that can be beneficially exchanged.
- We developed a simulation program (written in SimPy [5]) based on modern disk characteristics found in data centers rather than the commonly used DiskSim program which is based on older disks models. Results showing power savings of over 50% with small response tine penalties were obtained for workloads taken from two real life traces as well as a synthetic workload.

The rest of the paper is organized as follows. In Section 2 we survey some additional related work. In Section 3 we present the exchange algorithm and the computations that are needed to determine which disks will participate in block exchanges. In section 4 we describe the data structures used in locating blocks that can be exchanged. In Section 5 we present our experimental results. Finally, in Section 6 we present our conclusions and topics for future work.

2 Related Work

The area of energy efficient storage systems has received much attention lately from multiple communities such as storage hardware vendors, software designers, system architects and theoretical computer scientists. Hardware vendors are now offering energy-friendly alternatives to hard disk drives (HDD) generally referred to as SSD (Solid State Devices). These typically use non-volatile flash memory or battery backed RAM which offer great energy savings as there are no mechanical parts involved in storing or accessing data. Another recent hardware trend are hybrid-disks [18] that use a smaller SSD drive as a cache in front of the

HDD in order to allow reading of frequently accessed data and also buffering of writes without the need to power the HDD. Hybrid disks are produced by Samsung as well as other vendors.

Energy efficient storage systems including both hardware and software are offered by companies such as COPAN mainly targeting write-once/read-occasionally (WORO) data. Their solution is based on the MAID (Massive Array of Idle Disks) platform which guarantees that only about 25% of the disks in each enclosure are powered at any one time. Another energy efficient prototype storage system for this kind of data, called Pergamum, has been reported in [15]. It is based on a distributed network of disk-based storage appliances using the hybrid-disk approach. Pergamum uses a relatively small NVRAM attached to each node, called Tome, to allow storage of data signatures (used in disk recovery) and also metadata.

A different approach is taken by the experimental system Hibernator [21] which assumes the availability of multi-speed disks. The system divides the disks into tiers where disks in different tiers can spin at different speeds. The system dynamically assigns speeds to different tiers based on observed workloads while also automatically migrating data between the disks in order to save energy while satisfying response time constraints. A storage system called PARAID (Power-Aware RAID) presented in [19] introduces a skewed striping pattern that allows RAID devices to use just enough disks to meet the system load. The system "shifts gears" based on the observed workload by varying the number of powered-on disks to meet the response time constraints while conserving energy. The main difference between these systems and the block exchange algorithm in this paper is that both Hibernator and PARAID allow exchange of blocks only within a single RAID group whereas our algorithm is more powerful as we also allow exchanges of blocks between different RAID groups. Also, Hibernator assumes multispeed disks which are not currently available whereas in this paper we assume the disks can only be in either of two states (active or standby). Furthermore PARAID needs to reserve extra space on active disks for storing the replicated data from the standby disks. The idea of concentrating frequently accessed blocks on a few disks in this paper, has some similarity with the popular data concentration (PDC), of [10]. However, PDC exchanges popular "files", instead of "data blocks". Exchanging fixed-size blocks below the file system may be more efficient and portable than exchanging the variable-length files above the file system because of the response time and storage allocation issues.

Several interesting theoretical results have been published during the last decade in the area of Dynamic Power Management (DPM) for disk systems. Many of these results are reviewed in [7]. Most of this work assumes a single disk and attempts to find an optimal idle waiting period (also called idleness threshold time) after which a disk should be moved to a state which consumes less power. Requests can only be served when the disk is at the highest power state (active state) and there is a penalty associated with moving from a lower power state to the active state. The problem is that of devising dynamic on-line algorithms for selecting optimal idleness threshold times, based on observed idle periods between request arrivals, to transition the disk from one power state to another. The most common case has only two states namely, active state (full power) and standby (sleep) state. The quality of these algorithms is measured by their competitive ratio which compares their power consumption to that of an optimal offline algorithm. It has been shown that in the two state case a competitive ratio of 2 is the best possible. Another type of theoretical work uses a probabilistic model checking tool PRISM used to explore DPM using probabilistic models.

Other algorithmic approaches to conserve energy include power-aware caching policies where replacement is based on minimizing energy consumption rather that minimizing cache misses. Several caching algorithms that use dynamic programming are presented in [22, 23]. Yet another method to save power in disk systems uses a compiler-driven approach targeted at scientific applications that use arrays and execute on parallel architectures [14]. This is done by exposing disk layout information to the compiler and deriving optimal disk access patterns in terms of the order in which parallel disks are accessed.

Another area of active research is that of devising new benchmarks for measuring energy efficiency of database servers, and storage subsystems. Recently the Transaction Processing Performance Council (TPC) has formed a working group to look into adding energy efficiency metrics to all its benchmarks. In [11] this

is done by extending benchmarks such as TPC-C with power measurement features. In [12] an external sort benchmark, for evaluating the energy efficiency of a wide range of computer systems is presented.

3 The Block Exchange Algorithm

3.1 The Disk Storage Configuration

Our storage architecture is illustrated in Figure 1. It consists of an array of conventional disk storage configured into identical RAID Groups (RDG). A RAID (Redundant Array of Inexpensive Disks), is a form of storage system in data centers that provides high performance and fault tolerance at a relatively low cost. Each RDG in our systems can be configured as RAID-0, RAID1, RAID-4 or RAID-5. We will assume, from now on, that each RDG is configured as a RAID-5 unit. A RAID Group uses a Solid State Drive (SSD) as a large cache to stage data read from and written into an RDG. Using an SSD for each RDG is attractive since it is fast, durable, noiseless and energy efficient and currently is also available in large capacities of the order $64, 128, 256$ and 512 Gbytes [16]. An I/O sub-system, complete with I/O nodes, maintains a large number of RDGs. Clients interact with the I/O subsystems by writing and reading files either as a parallel file system or as Storage Area Network (SAN), with the provision that each block of a file must be entirely contained in an RDG unit. A good conceptual view of our systems is that of nested RAID-5+0 [8] (see Figure 6, except that the RAID-0 controller maintains modules that provide equivalent functionality of MAID (Massive array of Idle Disks) and has a strip-width of 1. It allows for migrating RDG segments (as explained subsequently) of disk blocks from one RDG to another.

Files in each RDG unit are striped according to the configured RAID level of the RDGs. One primary objective of the system is to concentrate active files in a small enough number of active RDG units without compromising the aggregate I/O bandwidth that satisfies the required response time of data accesses. Two approaches to achieving this is by dynamically migrating either entire files or RDG segments from an RDG with less I/O requests to others with high I/O activity but with just enough load on the active RDGs that the overall bandwidth is sufficient to meet the response time requirement. In this manner the less active RDGs can be idle long enough for the entire RDG unit to be spun down.

The SSD caches may prolong the idle times of RDG units and extend the shutdown periods of an RDG unit. We call the set of disk blocks (also termed disk chunks) in an RDG that form a stripe an RDG segment. In this paper the unit of exchange is called an XB (exchange block) which which consists of one or more RDG segments.

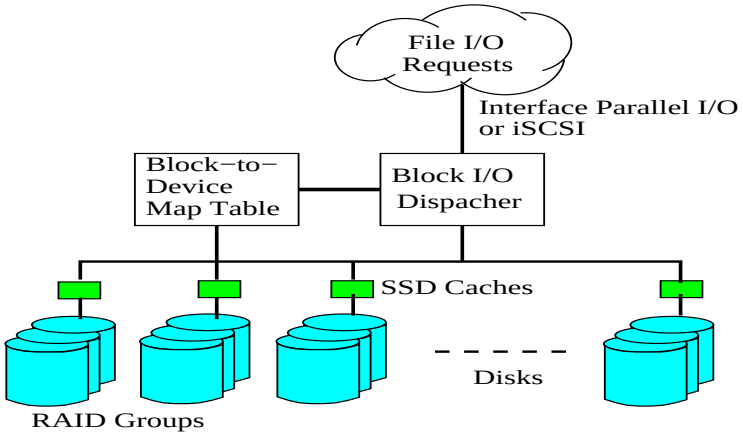

Figure 1: Overview of the energy smart disk storage

3.2 Arrival and Departure Rate

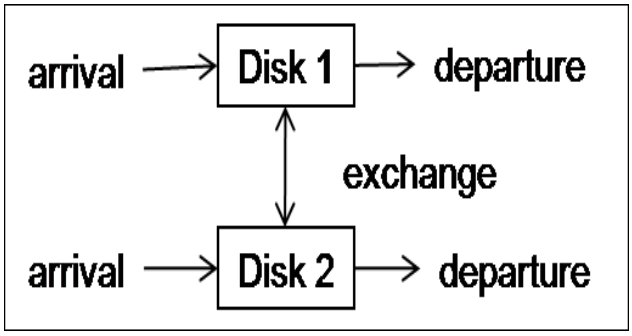

Figure 2: Arrivals, Departures and Exchanges for Disks

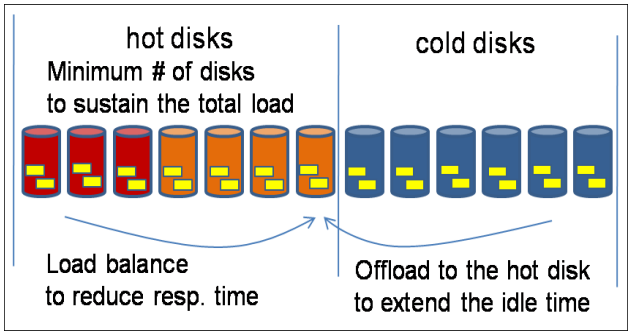

Figure 3: Two ways of XB exchanges

Before introducing the exchange system, we first define the arrival and departure rate for a disk. As shown in Figure 2, the arrival rate for a disk means the arrival rate of block requests received by a disk but excluding the requests introduced by the data exchange. We divide the timeline into epochs of T seconds and predict the mean arrival rate R for the next epoch as a weighted average of our predicted arrival rate R^{old} and the measured arrival rate R^{meas} for the previous epoch. The rate R is computed by the following equation,

$$R = w \cdot R^{old} + (1-w) \cdot R^{meas},$$

where the constant w represents the weight of the old predicted arrival rate. As shown later, when R exceeds a given threshold the arrival rate of the disk is too high and its workload should be off-loaded to other disks. In computing R, we intentionally ignore the portion of the arrival rate caused by block exchanges in order to prevent off-loading a disk whose load is caused only by exchanges but not by real data requests. We now turn to describing the procedure for predicting departure rates from the disk. The predicted departure rate D includes serviced requests due to either original data requests as well as requests caused by exchanges. To ensure proper update of D under heavy workload, we update D after every K requests are serviced, instead of the fixed time epochs used for computing R. We use the following equation to compute D,

$$D = w \cdot D^{old} + (1-w) \cdot D^{meas},$$

where similar to the computation of R, D^{old} is our previous prediction and D^{meas} is the average measured departure rate over the latest K requests, i.e., $D^{meas} = K/q$ where q is the number of seconds it took to service the last K requests. By such a definition, the departure rate of a *hot* disk reflects whether the disk is still suitable to share more workload from other disks immediately, particularly when the disk becomes busy. Including the exchange workload in the computation of D can stop a disk from a short term overloading due to exchange. In this work, we set w to 0.875, T to 10 sec, and K to 128 based on our experience with several real workload traces.

3.3 Hot and Cool Disks

We maintain a sorted list L of the n disks in decreasing order of their D (departure rate) values. Each time the value of D changes for any disk, the disk will be moved to its appropriate position in L according to the sorting order. As shown in Figure 3, the system divides L into two groups. The first m disks in L are named *hot* disks while the other $n-m$ disks are *cool* disks. We now describe the general ideas behind our exchange algorithm, more details are given in Section 3.5 .

In the following we refer to the unit of data exchanged between disks by the algorithm as XB (exchange block), it may consist of one or several RAID stripes. We use two types of block exchanges. The first type of exchange is performed after a block is accessed on a cool disk. The algorithm interchanges the XB containing this block with an XB that was not accessed recently residing on a *hot* disk starting the search for the first such XB with the disk at position m in L followed by position $m-1$ etc. The logic behind this type of exchange is to "lower" the temperature of *cool* disks to further reduce their request arrival rate and thus extend their idle time periods. The second type of exchange is used to avoid overheating of *hot* disks that may cause long queuing delays and excessive response times. This exchange is performed after accessing a block on a *hot* disk at position j in L where $1 < j < m$ if it is determined that the disk is overloaded. In this case, the XB containing the accessed block will be interchanged with an XB that was not accessed recently in "cooler" disks still in the *hot* disk group. The search for such an XB starts from disks at position $m, m-1, \ldots, j+1$ until a first such XB is found.

The number, m, of *hot* disks is dynamically determined based on the arrival rate of user requests. If the arrival rate is high, then more disks would be included in this group. These *hot* disks are never shut down in order to serve the bulk of arriving requests efficiently. On the other hand, the *cool* disks are supposed to have long idle time periods allowing greater power saving. The calculation of m is shown in the following subsection.

3.4 Sustainable Rate of Disk

As explained above, the goal of the block exchange system is to dynamically balance the load of the *hot* disks. By moving out the frequently-accessed XBs from an overloaded disk, the arrival rate of the disk would be reduced and the response time of requests can be efficiently shortened. However, the penalty associated with this exchange is that it can potentially cause more disks to move to an active state in order to serve the user requests. Let us denote by t the constraint on expected response time acceptable to users. We now show how to calculate the maximum arrival rate that is sustainable by a hard disk while still satisfying t. In the remainder of this paper, we will call this rate the *sustainable rate* of a disk and denote it by $S(R)$.

In [9] we analyzed the energy savings and response time trade-offs, and presented an analytical model to estimate the power cost and response time of a disk under different arrival rates and service times of requests. We now describe how $S(R)$ is computed. From [9], we know that the expected response time, $E[J]$ in in the *hot* disks is

$$E[J] = \frac{\rho}{1-\rho}\frac{E[S^2]}{2E[S]} + E[S].$$

where ρ is the load, i.e., $\rho = R * E[S]$, where R is the arrival rate, and $E[S]$ is the expected service time of requests. In our case, based on the disk characteristics given in Figure 10 and the real life workload of [17] , we get E[S]=0.022 and E[S^2]=5.24E-4. Figure 4 plots the relationship between the expected response time, $E[J]$ and the arrival rate $S[R]$ for a single *hot* disk as calculated from the above expression. From the figure, we can see that, to achieve a response time constraint $t = 0.05$, the sustainable rate of a disk $S(R)$ should be set to 30 requests per second assuming block exchange operations are executed.

Then, after determining $S(R)$, we can calculate the number m of *hot* disks as follows,

$$m = \left\lceil \frac{\sum_{i=1}^{n} R_i}{S(R)} \right\rceil$$

where R_i is the arrival rate of the i^{th} disk.

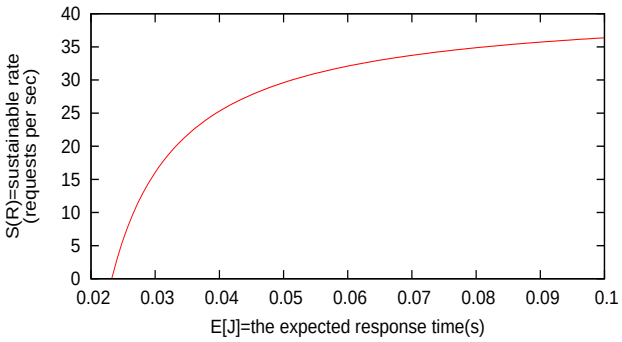

Figure 4: Relationship between expected response time and sustainable rate of a disk

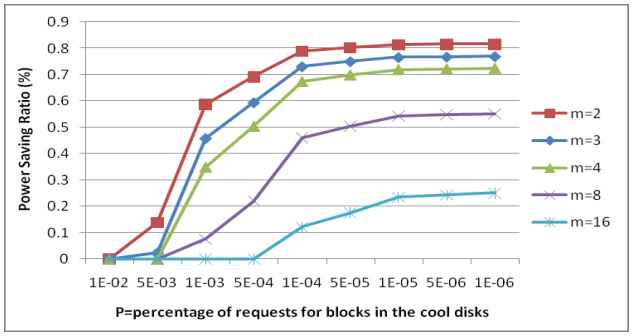

Figure 5: The potential power saving ratio of EXG-BLK under different request arrival rates

Also, based on our analysis model, we can estimate the potential power saving by our block exchange algorithm called BLKEXG. For example, in Figure 5 we plot the power saving ratio as a function of P the percentage of requests arrivals (including exchanges) to the cool disks for different values of m (hot disks).

Figure 5 shows the normalized power saving of BLKEXG, compared to the NPS (no power saving) case over different P and m, respectively. In this figure we assume the system consists of $n = 24$ disks with $t = 0.08$, from which we derive $S(R)$ to be 35 based on Figure 4.

3.5 Thresholds to Start Block Exchanges

There are three thresholds, LowTH, TargetTH, and HighTH, applied to each disk to determine whether to proceed with a block exchange. Since our goal is to keep the arrival rate of each *hot* disk close to the $S(R)$, sustainable rate of a disk, we set TargetTH to $S(R)$. Then, as described in Procedure Recv(), to achieve a load balance among the *hot* disks, whenever a request arrives for blocks in a *hot* disk, we check whether the arrival rate of the disk is higher than the high threshold HighTH. If it is, then the disk will be marked as a *hotspot* disk. Next, for each of the requests arriving to the disk, after serving the request, the ExgBlk function is called to locate the XB which contains the requested block. Then, the XB is exchanged with an un-accessed XB starting the search from disk currently ranked m in L (*hot* disk with the lowest departure rate) and proceeding to $m-1$ up to 1. In the search, *hotspot* disks will be skipped due to their heavy loads. Also, if the currently accessed disk is ranked j, then the search is stopped after the disk ranked $j+1$ is searched as continuing the search will only encounter disks with ranks smaller than than j which must have higher D (departure rates) and thus are more busy.

The procedure to locate the un-accessed block in a selected disk is called GetUnaccessedXB() and is described in Section 4 and uses a tree data structure. It returns -1 if all blocks on the disk are accessed.

Such an exchange will proceed whenever a request arrives to the hotspot disk until the arrival rate of the disk is lower or equal to the TargetTH. Then, the *hotspot* mark will be removed from the disk. The following functions are used in the algorithm, for lack of space we simply explain here what each of them does.

(1) GetMappedDevID(): Get the current ID of the device to be accessed by a request.

(2) GetMappedBlockID(): Get the address of a block on a device.

(3) ForwardToDev(): Forward a request to a device.

(4) GetCorrespondingXB(): Locate the XB (exchange block containing a given block).

(5) GetDevIDWithRank(): Get the ID of a device with a given rank in L.

(6) Exchange(): Perform an exchange between two XBs.

```
Procedure Recv(Req,HotSpot)
begin
    Input: Req: Received Request
    Output: HotSpot: a bit vector where each bit corresponds to a storage device
    id_dev ← GetMappedDevID(req) ;
    id_blk ← GetMappedBlockID(req) ;
    ForwardToDev(id_dev,id_blk) ;
    if (isHot(id_dev) then
        if R(id_dev)≥ HighTH then
            HotSpot[id_dev]← 1
        else
            if R(id_dev) < TargetTH then
                HotSpot[id_dev] ← 0
        if HotSpot[id_dev]=1 then
            ExgBlk(id_dev,id_blk)
    else
        if R(id_dev) < LowTH then
            ExgBlk(id_dev,id_blk)
end
```

```
Function ExgBlk(id_dev,id_blk)
begin
    Input: id_dev: device id, id_blk: block id
    id_XB ← GetCorrespondingXB(id_blk) ;
    for i ← m downto 1 do
        nid_dev ← GetDevIDWithRank(i);
        if HotSpot[nid_dev] = 1 then continue ;
        if nid_dev = id_dev then break ;
        nid_XB ← GetUnaccessedXB(nid_dev) ;
        if nid_XB < 0 then continue ;
        Exchange(id_dev,id_XB,nid_dev,nid_XB) ;
        break ;
end
```

Let us now turn to the procedure for extending the idle time of the *cool* disks which is similar to the above procedure. Whenever a request arrives for blocks in a *cool* disk, we examine whether the arrival rate of the disk is lower than the low threshold LowTH. If it is, after serving the request, we call ExgBlk to locate the XB which covers the requested block and then exchange the XB with an un-accessed XB located at the *hot* disk starting the search with the disk with the lowest departure rate.

Note that we exchange blocks with *hot* disk based on measurements of departure rate (the lowest departure rate), instead of the lowest arrival rate. The reason is that a disk with low arrival rate may be busy serving the queued requests. Also, since the exchange itself also adds loading to the selected *hot* disk, by picking the *hot* disk with the lowest departure rate, we can further avoid the selected disk from short-term overloading due to the exchange operations. Also note that due to the dynamic sorting of the disks based on their departure rates, the blocks are exchanged with different *hot* disks at different times thus leading to load balancing among the *hot* disks.

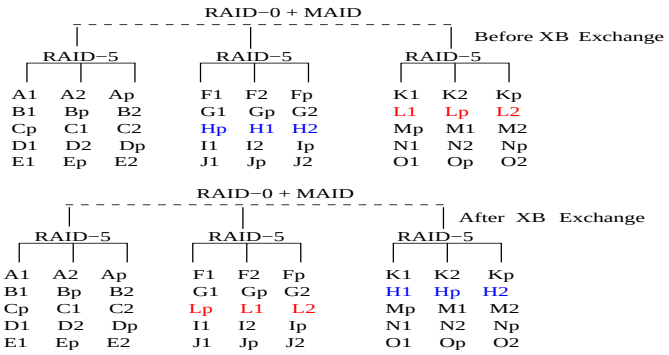

Figure 6: Illustration of XB exchanges in a RAID-0 + MAID configurations

4 Data Structures and File Chunking

4.1 Selection of Non frequently Accessed Exchange Blocks

As described in Section 3.3, we need to pick a non-frequently accessed XB from the coolest *hot* disk for the exchange. Thus, we need a method to quickly find such an XB on a disk. To simplify the problem, let's first assume we want to distinguish between the XBs accessed at least once from the ones who have never been accessed called un-accessed XBs. The naïve way to satisfy this goal is to allocate a bit vector (initialized to all 0's)which we call XB-Access-Vector where each bit in the vector corresponds to one XB on the disk. Once an XB is accessed, we turn its corresponding bit to 1 if it is 0. Then, to find an un-accessed XB, one can sequentially search the XB-Access-Vector until encountering a 0 bit. However, this method needs N-bits of space and O(N) search time, where N is the number of XBs in one disk. To reduce the search time, we propose a binary tree structure as shown in Figure 7 which we call double-bit tree (DBT).

Each internal node of the DBT consists of two bits called the left-bit and the right-bit. The leaves of the DBT are formed from the entries of the XB-Access-Vector, i.e each bit corresponds to an XB and it is set to 0 or 1 according to whether the corresponding XB was accessed or not respectively. The left-bit (right-bit) of an internal node in a DBT is set to 0 if at least one of the leaves in its left (right) sub-tree are 0 otherwise it is set to 1. To find an un-accessed XB, one starts searching from the root of the DBT, if both left and right bits at the root are 1 all XBs at the leaves have been accessed and the search stops. Otherwise we search recursively in the left sub-tree of the root if the left-bit=0 or search in the right sub-tree if left-bit=1. For reference purposes we call this DBT search procedure GetUnaccessedXB. It is easy to see that GetUnaccessedXB operates in $O(logN)$ steps. Updating the DBT once an un-accessed XB gets accessed is done by changing the bit at the leaf corresponding to that XB to 1 and updating the internal nodes on the path from the leaf to the root as necessary. The update procedure simply changes the left-bit (right-bit) of a parent node to 1 if both the left and right bits of its left (right) child are 1. This can also be done in $O(logN)$ steps.

4.2 Block Exchange Information

We can now build a slightly more complex data structure that monitors the access frequency of XBs across a time window of s time units and decides whether an XB is frequently accessed during the time window. To do this we maintain a set of XB-Access-Vectors (XAB) called $XAB(T_i)$ for the last s time units $T_1, T_2, ..., T_s$. We then compute an additional vector called Total-XB-Access-Vector where each of its entries is a function g (based on some weighted average) of the corresponding entries in $XAB(T_i)$ for $i = 1, 2, .., s$. The DBT is then built using the Total-XB-Access Vector entries as its leaves. An example of this is given in Figure 8 for $s = 3$. In the formula below we give more weight (W_i) to more recent accesses than ones that occurred in the past but at the same time our function also gives some fixed weight to the frequency of accesses independent

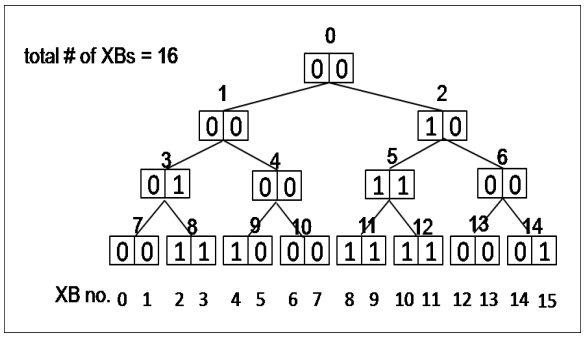

Figure 7: The Tree Structure to locate an un-accessed-accessed Block

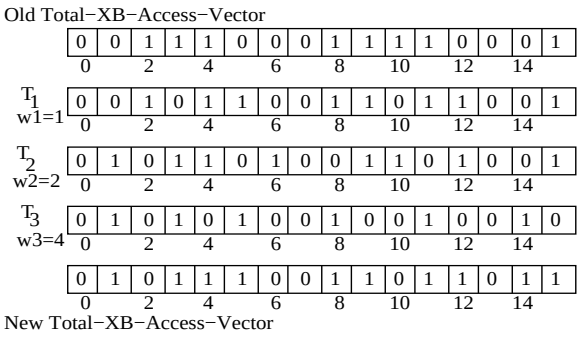

Figure 8: Information retained to track frequently accessed XBs

of when they occurred. In this case the function $g(j)$ which determines the value of the j^{th} bit in the New Total-XB-Access-Vector is set as follows

$$\exp(j) = \sum_{i=1}^{3} XAB(T_i)[j](W_i + 1)$$

and

$$g(j) = \begin{cases} 1 & \exp(j) \geq 5 \\ 0 & \text{otherwise} \end{cases}$$

5 Experimental Results

5.1 Simulation

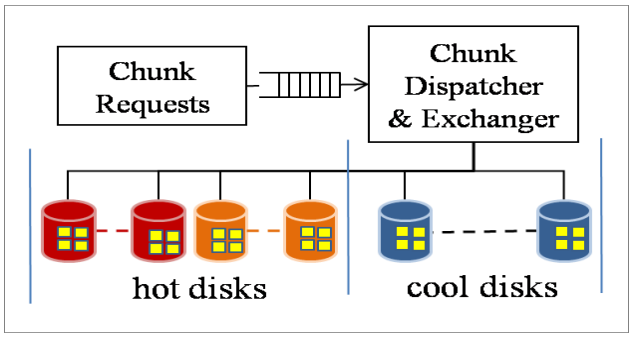

Figure 9: Configuration of disks used in the simulation

We developed a simulation model to examine the block exchange system proposed in Section 3. The simulation environment was developed and tested using SimPy [5], as illustrated in Figure 9. The environment consists of a workload generator, a block dispatcher, and a group of hard disks. Figure 10 shows the characteristics of the hard disk used in the simulation. With the specifications taken from [13] and [20] we built our own hard disk simulation modules. A hard disk is spun down and set into standby mode (see Figure 11) after it has been idle for a fixed period which is called idleness threshold [10], [3]. We do not use the recently revised DiskSim simulator [1], that is commonly used in the literature for our simulations because the number of events needed to handle a file request is highly correlated with file sizes making DiskSim too slow for a realistic data center simulation that involves disks, each of the order of 500 GBytes and tens of thousands of files requiring terabytes/petabytes of total data storage.

The workload generator supports two different ways to produce block requests. First, the generator can follow a Poisson process to produce requests at a rate R to access disks. Based on the statistics collected

Figure 10: The characteristics of the hard disk

Description	***Value***
Disk model	Seagate ST3500630AS
Standard interface	SATA
Rotational speed	7200 rpm
Avg. seek time	8.5 msecs
Avg. rotation time	4.16 msecs
Disk size	500GB
Disk load (Transfer rate)	72 MBytes/sec
Idle power	9.3 Watts
Standby power	0.8 Watts
Active power	13 Watts
Seek power	12.6 Watts
Spin up power	24 Watts
Spin down power	9.3 Watts
Spin up time	15 secs
Spin down time	10 secs

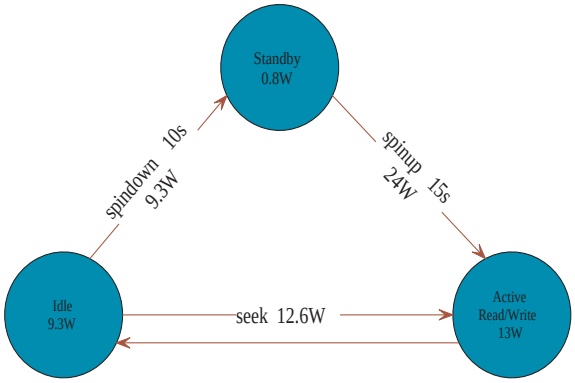

Figure 11: Power usage modes of a disk drive

from the real workload [17], we set the number of disks to 24 and the data size required by each request to 8, 16, 24, or 32KB with even probabilities. Then, the arrival requests to disks are generated based on a Zipf distribution whose cumulative density function $F(x)$ is given by

$$F(x) = P(x < i) = (i/N)^{\theta} \; \forall 1 \leq i \leq N.$$

where N=24 and its skew parameter θ is set to log 0.8/log 0.2, which means 80% of all requests would go to 20% of disks. Similarly, we divided the space of a disk into 500 segments. Then, the Zipf distribution is used again to determine the frequency of requests for blocks in each segment, where N is set to 500.

Besides producing workload based on probability model, the workload generator can produce requests based on a log of block access to a storage system. Two realistic workload logs are used in our experimental results. The first log is recorded on a storage system operating a financial application [17]. There are 4099352 write requests and 1235632 read requests for the blocks distributed in 24 hard disks. The mean rate of arrival requests is 123.5/s. The second log is recorded on a storage system that supports a web search engine. There are 4579809 read requests invoked within 4.5 hours and 99% of requests are concentrated on three disks. The mean arrival rate is 297.487/s. Finally, once a request is generated by the generator, the block dispatcher forwards it into its corresponding hard disk.

In the experiments, the sustainable rate of a disk was set to 35 requests per second. The high threshold (HighTH in Section 3) is 1.3 times of 35, or 45 while the low threshold (LowTH) is half of 35, i.e. 17.5. Besides, to compare the effects of the block exchange algorithm on power saving and response time, we also examined the effects of cache and DPM with fixed idleness thresholds. To measure effect of caching, each disk was allocated a LRU cache. Multiple cache sizes were tested in the experiment where the page size in the cache is fixed to 16K. Dynamic power management (DPM) algorithms have been proposed [7] to determine on-line when the disk should be transitioned to a lower power dissipation state while experiencing an idle period. Analytical solutions to this on-line problem have been evaluated in terms of their competitive ratio. This ratio is used to compare the energy cost of an on-line DPM algorithm to the energy cost of an optimal offline solution which knows the arrival sequence of disk access requests in advance. It is well known [6] that for a two state system where the disk can be in either standby or in idle mode there is a tight bound of 2 for the competitive ratio of any deterministic algorithm. This ratio is achieved by setting the idleness threshold, T_t, to b/(P_t-P_s) where b is the energy penalty (in joules) for having to serve a request

while the disk is in standby mode, (i.e., spinning the disk down and then spinning it up in order to serve a request). P_t and P_s are the rates of energy consumption of the disk (in watts) in the idle mode and in the standby mode, respectively. We call this value the competitive idleness threshold, which value herein is 53s, computed based on the characteristics of hard disks shown in Figure 10.

5.2 Power Saving Results for Financial Workload

Figure 12 shows the power saving ratios of DPM, CACHE+DPM, EXG, and EXG+CACHE under the financial application workload, which ratios are normalized with the power cost of 24 disks which are always spinning. As shown in the figure, for this workload, simply using DPM can save 10% of power only. Then, even using DPM and 256MB LRU cache can only save 20% of power. The reason is that the mean arrival rate for each disk in the workload, 123.5/24 or 5.15/s, is too high for DPM to save power. Although by using 256MB LRU cache, a 68% of hit ratio is achieved and the arrival rate of each disk can be effectively reduced to 1.65/s, which however is not low enough to save power by DPM. According to [9] DPM can only save power if the arrival rate of a disk is smaller than 0.029/s. As shown in Figure 13, increasing the cache size is useless since it cannot increase the hit ratio and thus cannot further reduce the arrival rate of disks or extend the idle time of disks to save power.

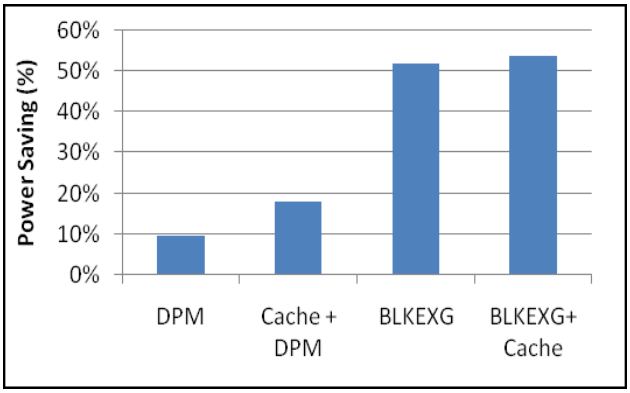

Figure 12: Power saving ratio of DPM, DPM+Cache, BLKEXG and BLKEXG+Cache

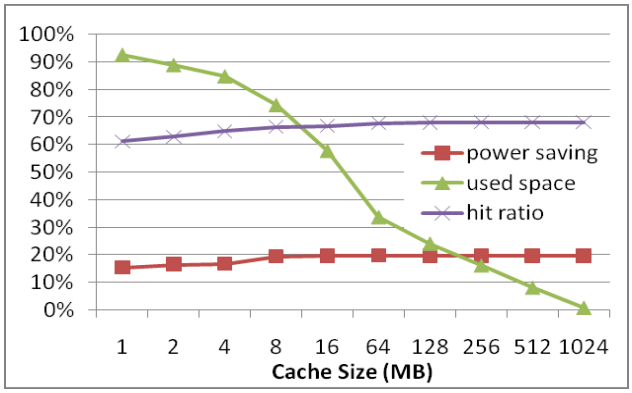

Figure 13: The hit ratio, power saving ratio and % of used space for different cache sizes

Finally, compared to the of power saving of using cache and DPM which is at most 20%, the BLKEXG system can save 50% power cost. Fig 14 further displays the power cost for each disk. Obviously, by concentrating frequently accessed blocks into the *hot* disks from the *cool* disks, only three disks are necessary to be active(always spinning) to serve requests. The other 21 disks can have longer idle periods to save power and thus their mean power cost is only 4W. On the other hand, by using a large cache to reduce the arrival rate of requests, although 10 disks may have long idle periods which are sufficient to produce beneficial power savings from DPM, the other 14 disks are still too busy to save any power. Figure 15 shows the mean idle period of disks affected under these power saving mechanisms. Using this figure, we can find that BLKEXG can increase the length of the idle period by a factor of almost 100 times of that under DPM and 10 times of Cache + DPM, by concentrating blocks into the *hot* disks. The BLKEXG curve shows that the ”hottest” three disks are chosen to always spin as they have short idle times due to the concentration of frequently accessed blocks on them, this however makes the other 21 disks have long idle periods sufficient to produce power savings. Figure 16 displays the frequency of exchange produced by BLKEXG. The mean number of exchanged blocks is 0.5 per second, which obviously is not high and thus does not cost much power.

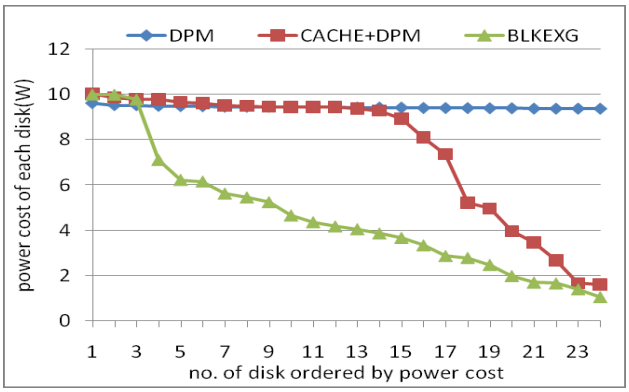

Figure 14: The power cost of each disk under DPM, DPM+Cache and BLKEXG

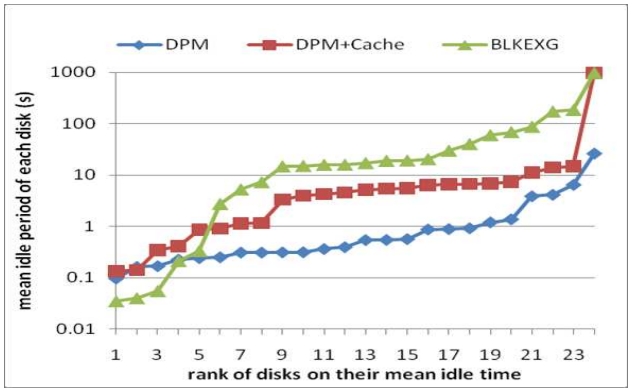

Figure 15: The mean idle period of each disk under DPM, DPM+Cache and BLKEXG

5.3 Response Time Results for Financial Workload

Saving power often implies an increase in response time. Therefore, it is important to measure the response time penalty due to the power saving policy. Figure 17 shows the response times of requests using DPM, cache + DPM, BLKEXG, and BLKEXG + cache. Again, while DPM and cache only save 10∼20% of power, they seriously impact the performance, i.e. increase the response time by a factor of more than 20 of that of disks without power saving (NPS), i.e. from 0.013s to 0.40 and 0.25, respectively. On the other hand, using BLKEXG + cache only increases the response time to 0.036 while saving 50% of power cost. Obviously, deploying the block exchange system with cache is a preferable option.

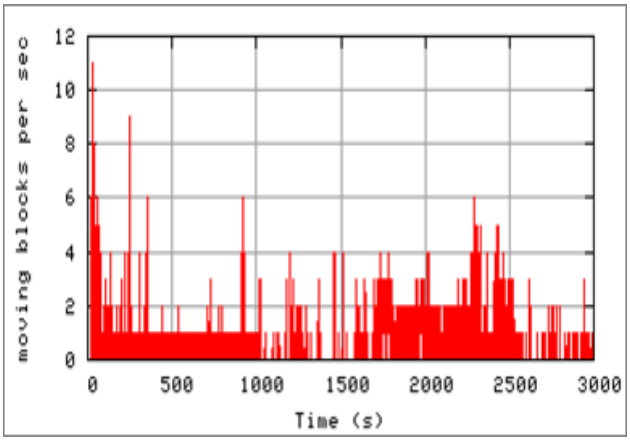

Figure 16: The frequency of block exchanges in BLKEXG

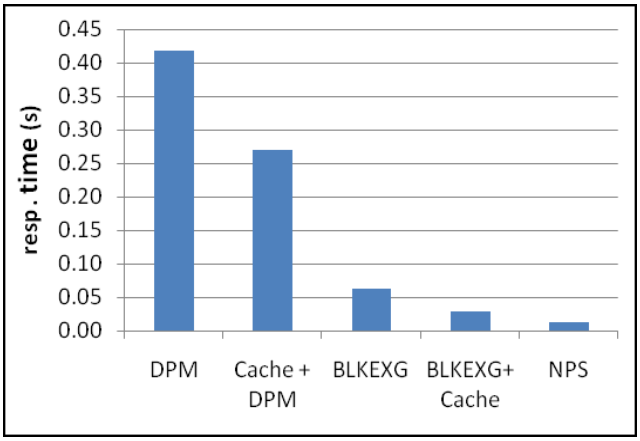

Figure 17: Comparison of response times for different power saving schemes

5.4 Power Savings and Response Time Results For the Search Engine Workload

We now examine the power saving and response time provided by BLKEXG under the search engine server workload log. This workload is more skewed than the first one. In the first log requests arrived for blocks unevenly distributed on all 24 disks, whereas arrival requests in the second log are highly concentrated for the blocks in three disks only. Thus, as shown in Figure 18, by simply enabling DPM on disks, 70% of power can be saved for this workload. However, the response times of requests under NPS and DPM are very high (19), because the three disks are actually overloaded. Even when a large 640MB cache is deployed for each disk, these disks are still overloaded. Besides, due to the high arrival rate and short idle periods, no

additional power can be saved by using the cache, as compared to DPM without cache.

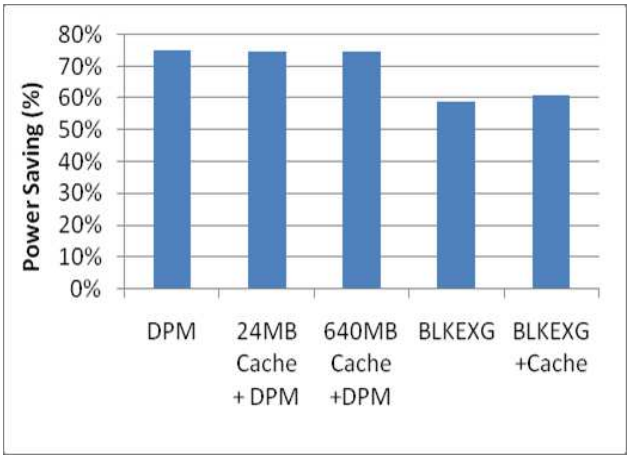

Figure 18: Comparison of power savings of search engine workload for different power saving schemes

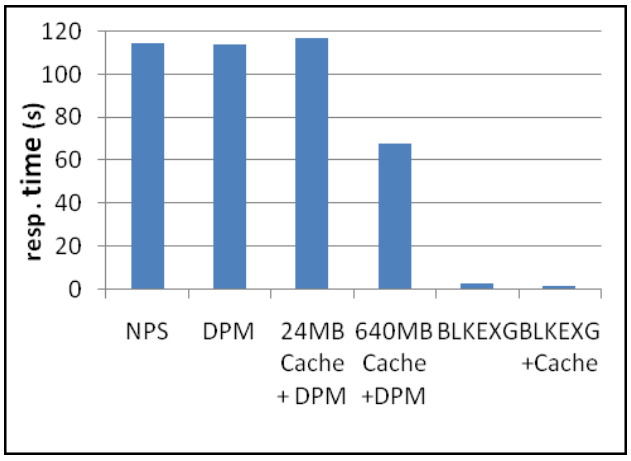

Figure 19: Comparison of response times of search engine workload for different power saving schemes

On the other hand, since the BLKEXG system estimates the actual number of *hot* disks necessary to serve the arrival workload and keeps on balancing the load among these disks, the response time under BLKEXG can be dramatically reduced to 0.12 sec while still saving about 60% of power, after an off-loading procedure works consequently on the three hotspot disks for 250 seconds. In fact, according to the workload in the second log, about 10 active disks are necessary to serve these requests, but under DPM only 6 disks are active, which leads DPM to save additional 10% of power but causes an unacceptable expected response time.

5.5 Results for Synthetic Workload

To further verify BLKEXG under different heavy levels of workloads, we used the synthetic workloads with different arrival rate of requests and measured the effect on power saving and response time of the BLKEXG system. We also, measured power savings and response times for NPS, DPM, CACHE+DPM and BLKEXG+CACHE for comparison.In all cases the cache size for each disk was set to 128MB. Figure 20 shows the power saving ratios of these policies under this workload, the ratios are normalized with the power cost of 24 disks which are always spinning. The results show that BLKEXG can save 75% of power while DPM or CACHE only provide about 15% of power saving on average. Also, the effect of cache on power saving is insignificant under these synthetic workloads.

Figure 21 shows the response time of requests provided by these power saving mechanisms. Similarly to the results under the real workloads, DPM and CACHE cause very long response times particularly when the arrival rate is low, because each request has a high probability of arrival during the periods where the disk is in standby mode and then has to wait a long time for the spinning up of the disk. However, since BLKEXG would concentrate frequently-accessed blocks into *hot* disks, most requests would be redirected to these disks and only a few requests which are directed to the *cool* disks may suffer such long response time delays. Therefore, the response time under BLKEXG is much shorter than DPM and CACHE.

6 Conclusion

In this paper we developed a response time sensitive adaptive algorithm for reducing energy consumption of disk storage systems. It operates by measuring arrival rates at the disk and then dynamically re-distributing

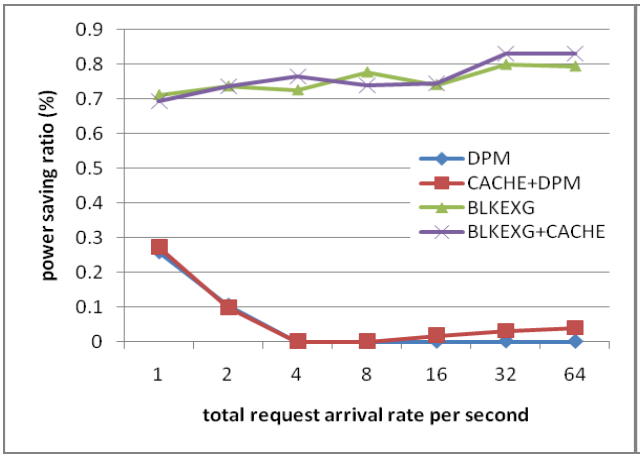

Figure 20: Power saving ratios of different schemes under varying request arrival rates

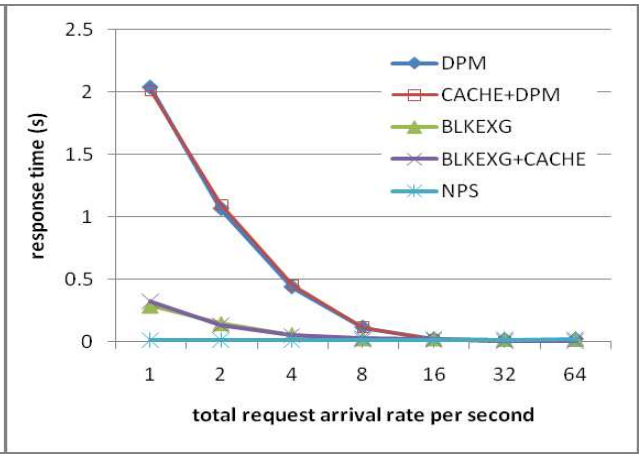

Figure 21: Response times of requests for different schemes under varying arrival rates

the workload by exchanging data between disks when necessary. We also developed the data structures needed to support the fast location of blocks which are candidates for such exchanges. We have shown that under real life workloads the algorithm leads to the concentration of the bulk of the disk access traffic on a small fraction of the available disks. This allows the remaining disks to experience longer idle periods which in turn makes the DPM (dynamic power management) procedures much more effective.

Extensive simulations with our block exchange algorithm showed that it is much more efficient than simply using known DPM procedures or just power-aware caches. It can save up to 50% of the energy consumption while still satisfying response time constraints. The algorithm does not need any specialized hardware such as multispeed disks and can be readily applied to existing disk storage configurations such as nested RAID or MAID systems found in many commercial and scientific data centers.

In the future, we plan to investigate our techniques on more real life workloads that include various mixes of read and write requests obtained from our scientific data center at NERSC. To further validate our results, we plan to test the algorithm on several disk storage configurations that will allow us to compare energy consumption measurements with the simulation results.

Acknowledgment

This work is supported by the Director, Office of Laboratory Policy and Infrastructure Management of the U. S. Department of Energy under Contract No. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing (NERSC), which is supported by the Office of Science of the U.S. Department of Energy.

References

[1] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger. The disksim simulation environment.

[2] Dennis Colarelli and Dirk Grunwald. Massive arrays of idle disks for storage archives. In Supercomputing'02: Proc. ACM/IEEE Conference on Supercomputing, pages 1 – 11, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.

[3] Carlos Cunha, Azer Bestavros, and Mark Crovella. Characteristics of www client-based traces. Technical report, Boston University, Boston, MA, USA, 1995.

[4] A. Guha. Data archiving using enhanced maid (massive array of idle disks), May 15 – 18 2006.

[5] SimPy: SimPy Simulation Package in Python. http://simpy.sourceforge.net/archive.htm.

[6] S. Irani, R. Gupta, and S. Shukla. Competitive analysis of dynamic power management strategies for systems with multiple power savings states. In DATE '02: Proceedings of the conference on Design, automation and test in Europe, page 117, Washington, DC, USA, 2002. IEEE Computer Society.

[7] Sandy Irani, Gaurav Singh, Sandeep K. Shukla, and Rajesh K. Gupta. An overview of the competitive and adversarial approaches to designing dynamic power management strategies. IEEE Trans. VLSI Syst., 13(12):1349–1361, 2005.

[8] Nested RAID Levels. http://www.absoluteastronomy.com/topics/nested_raid_levels.

[9] Ekow J. Otoo, Doron Rotem, and Shih-Chiang Tsao. Energy smart management of scientific data. In 21st Int'l. Conf. on Sc. and Stat. Database Mgmnt., New Orleans, Louisiana, USA, Jun. 2009. To Appear.

[10] E. Pinheiro and R. Bianchini. Energy conservation techniques for disk array-based servers. In Proc. Int'l. Conf. on Supercomputing (ICS'04), Saint-Malo, France, June 26 2004.

[11] Meikel Poess and Raghunath Othayoth Nambiar. Energy cost, the key challenge of today's data centers: a power consumption analysis of tpc-c results. Proc. VLDB Endow., 1(2):1229–1240, 2008.

[12] S. Rivoire, M. A. Shah, P. Ranganathan, and C. Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pages 365–376, New York, NY, USA, 2007. ACM.

[13] Seagate. Seagate Barracuda 7200.10 Serial ATA Product Manual. Seagate Technology, Scotts Valley, CA, Dec 2007.

[14] S. W. Son, G. Chen, O. Ozturk, M. Kandemir, and Alok Choudhary. Compiler-directed energy optimization for parallel disk based systems. IEEE Transactions on Parallel and Distributed Systems, 18(9):1241–1257, 2007.

[15] M. W. Storer, K. M. Greenan, and E. L. Miller. Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage. In Proc. 6th USENIX Conf. on File and Storage Technologies (FAST'2008), pages 1 – 16, San Jose, California, Feb. 2008.

[16] Solid State Drives From Toshiba. http://www.toshiba.com/taec/catalog/family.do - ?familyid=7&subfamilyid=900314.

[17] UMassTraceRepository. http://traces.cs.umass.edu/index.php/storage/storage.

[18] A. A. Wang, G. Kuenning, P. Reiher, and G. Popek. The conquest file system: Better performance through a' disk/persistent-ram hybrid design. ACM Trans. on Storage (TOS), 2(3):309 – 348, Oct. 2006.

[19] C. Weddle, M. Oldham, J. Qian, and A. Wang. PARAID: A gear shifting power-aware RAID. ACM Trans. on Storage (TOS), 3(3):28 – 26, Oct. 2007.

[20] J. Zedlewski, S. Sobti, N. Garg, F. Zheng, A. Krishnamurthy, and R. Wang. Modeling hard-disk power consumption. In FAST'03: Proc. 2nd USENIX Conf. on File and Storage Tech., pages 217–230, Berkeley, CA, USA, 2003. USENIX Association.

[21] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes. Hibernator: Helping disk arrays sleep through the winter. In SOSP'05: Proc. 20th ACM Symp. on Operating Syst. Principles, pages 177–190, NY, USA, 2005. ACM Press.

[22] Q. Zhu, F. M. David, C. F. Devaraj, Z. Li, Y. Zhou, and P. Cao. Reducing energy consumption of disk storage using power-aware cache management. In HPCA'04: Proc. of the 10th Int'l. Symp. on High Perform. Comp. Arch., page 118, Washington, DC, USA, 2004. IEEE Computer Society.

[23] Qingbo Zhu and Yuanyuan Zhou. Power-aware storage cache management. IEEE Trans. Comput., 54(5):587–602, 2005.