Analysis of Trade-Off Between Power Saving and Response Time in Disk Storage Systems

E. Otoo, D. Rotem, S. C. Tsao
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA 94720
{ejotoo, d_rotem, weafon}@lbl.gov

Abstract

It is anticipated that in the near future disk storage systems will surpass application servers and will become the primary consumer of power in the data centers. Shutting down of inactive disks is one of the more widespread solutions to save power consumption of disk systems. This solution involves spinning down or completely shutting off disks that exhibit long periods of inactivity and placing them in standby mode. A file request from a disk in standby mode will incur an I/O cost penalty as it takes time to spin up the disk before it can serve the file. In this paper, we address the problem of designing and implementing file allocation strategies on disk storage that save energy while meeting performance requirements of file retrievals. We present an algorithm for solving this problem with guaranteed bounds from the optimal solution. Our algorithm runs in $O(n \log n)$ time where n is the number of files allocated. Detailed simulation results and experiments with real life workloads are also presented.

1. Introduction

Enterprises, research institutions and governmental agencies now provide on-line or near-line access to massively large data resources. The declining cost of commodity disk storage has now made such data resources very affordable for large data centers. However, maintaining these data resources over hundreds and thousands of spinning disks comes at a considerable expense of power usage. A recent paper states that about 26% of the energy consumption at data centers is attributed to disk storage systems [7]. This percentage of disk storage power consumption will continue to increase, as faster and higher capacity disks are deployed with increasing energy costs and also as data intensive applications demand reliable on-line access to data resources. It has become necessary to employ strategies to make disk system usage more energy efficient independent of manufacturers efforts. The problem is equally significant in high performance scientific computing centers, such as NERSC [10], that manage large scale data resources that are accessed by collaborating scientists around the world.

The solution is being addressed at two levels: *physical device* and *systems* level. At the physical device level, disk manufacturers are developing new energy efficient disks [14] and hybrid disks (i.e., disks with integrated flash memory caches). At the system level, a number of integrated storage solutions such as MAID [4], PARAID [16], PERGAMUM [15] and SEA [17] have emerged all of which are based on the general principle of spinning down and spinning up disks. Disks configured either as RAID sets or as independent disks, are configured with idle time-out periods, also called *idleness threshold*, after which they are automatically spun down into a standby mode. A read or write I/O request targeted to a standby disk causes the disk to spin-up again in order to service it. This of course comes at the expense of a longer

response time. A spun up disk stays spun up until it becomes idle again for duration of the time-out period (see Figure 1).

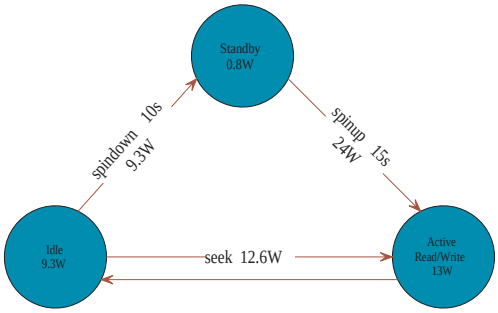

Figure 1. Power consumption of the different disk modes and transition times

1.1. Proposed Approach

Most existing research work in storage power management assumes that the contents of the disks are static and power saving can be realized during lengthy idle periods of the disks. Our approach is different (but complementary) in that we also allow reorganization of disk contents either dynamically or at periodic reorganization points in order to create more opportunities for power saving.

To understand the reasoning behind this approach, let us consider a case, where the file accesses are uniformly distributed among all the disks. There will be very little power saving in this case, since there are relatively short idle periods on each disk which are not sufficient to justify powering the disk down and then up again. Now, consider another case, where the disks are split into two groups, with the great majority of the accesses being made to the first group of files with high frequency of accesses. The infrequent file accesses go to the second group of disks with less frequently accessed files. This creates longer idle periods on the second group with an opportunity to shut down these disks for power saving. The more disks we have in the second group, the more power we can save. The trade-off however, is that the frequent accesses to the disks in the first group, may result in longer response times. Thus, we have to load these disks maximally, but without exceeding some guaranteed acceptable response times. As shown in Section 5, our strategy improves power savings also with the presence of a cache in front of the disk system (or individual disk caches).

Our experience with the workload of *read accesses* to the data resources at NERSC [10], one of the national high performance computing centers, shows that the access frequencies of files follow Zipf-like distributions. That is, at periodic intervals there are a large number of file access requests that are directed to a small number of files. We propose to exploit these observations for energy efficiency, using a strategy to transform these file access patterns to disk access patterns through file allocation strategies.

In this paper we focus on read requests, in case the access sequence includes write requests we propose to follow one of the energy-friendly approaches given in [15], i.e., write files into an already spinning disk if sufficient space is found on it or write it into any other disk (using best-fit or first-fit policy) where sufficient space can be found. The written file may be re-allocated to a better location later during a reorganization process.

The main contributions of this paper are:

- The use of file allocation strategies to significantly reduce energy consumption of disk systems. These strategies can be used in conjunction with other techniques in the literature.
- We present a mapping of the problem of file allocation on disks with maximal power conservation and response time constraints to a generalized bin packing problem called *two-dimensional vector packing*

problem (2DVPP). This mapping allowed us to use algorithms that solve 2DVPP with provable bounds from the optimum.

- We give an algorithm that improves on the running time of a 2DVPP algorithm [3] (that has the best known bounds from optimality), from $O(n^2)$ to $O(n \log n)$
- The use of extensive simulations with both simulated and realistic workloads and accurate disk characteristics, to calculate the energy savings and response times achieved by using our file allocation techniques as compared with random file allocation. We demonstrate that our techniques achieve significant energy savings over a wide range of workload parameters values with minimal response time degradation.

Although our solution may be characterized as an off-line solution, it can be applied in a semi-dynamic manner by accumulating access statistics over periodic intervals and performing reorganization of file allocations. Another use of the solution presented is for computing the percentage of disks that must be maintained on-line to meet file access response time under budget constraints.

The remainder of the paper is organized as follows. In the next section we present the background and related work on energy saving approaches in large scale disk-storage based data centers. We present our heuristic algorithm in Section 3 where a variant for a special case observed in our generated workload based on real life logs is also given. A discussion of our simulation environment is given in Section 4. Section 5 presents our experimental results and we conclude in Section 6 where we also discuss directions for future work.

2. Related Work

The general techniques being advocated in this work are based on some energy conservation techniques used in computing. In particular, we employ three principles: massive array of idle disks (MAID) [4], popular data concentration [11] and energy-aware caching [20]. Similar to the work in [11], our approach is to concentrate short term and frequent data accesses on a fraction of the disk arrays, while the rest are, for most times, set in standby mode due to their long periods of inactivity. Modern disks provide multiple power modes: active, idle, and standby modes and most operating system can be configured for the power management of these disks.

Conserving energy in large scale computing has been recently explored in [6, 11]. Colarelli and Grunwald [4] proposed *MAID* for near-line access to data in a massively large disk storage environment. They show, using simulation studies, that a MAID system is a viable alternative and capable of considerable energy savings over constantly spinning disks. A related system was implemented and commercialized by COPAN systems [5, 6]. This system, which is intended for a general data center, is not focused on scientific applications and is not adaptively reconfigurable based on workloads. Further, the disks are remotely accessible via NFS mounts. Our approach uses iSCSI protocol for remote accesses which provides a better I/O bandwidth than NFS.

The theory of Dynamic Power Management of disks has also drawn a lot of attention recently from the theoretical computer science community (see [8] for an extensive overview of this work). Most of this work considers a single disk only 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. More specifically, the problem discussed in these research works is based on the assumption that the disk can be transitioned among n power consumption states where the i^{th} state consumes less power than the j^{th} state for $i < j$. The disk can serve file requests only when it is in the highest power state (the n^{th} state) which is also called the active state. The system must pay a penalty β_i if a request arrives when the disk is in the i^{th} state, the penalty is proportional to the power needed to spin up from state i to the active state n. The penalty is decreasing with

the state number, i.e., $\beta_j < \beta_i$, for $j > i$, and $\beta_n = 0$. The problem is that of devising online algorithms for selecting optimal threshold times, based on 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 that can see the entire request sequence in advance before selecting state transition times. For a two state system there is a tight bound of 2 for the competitive ratio of any deterministic algorithm. Response time penalty is not considered in this work.

Another theoretical work which also deals with the affects of power management policy on the latency of a single disk is described in [13]. Our work is complementary to that work as we consider multiple disks rather than just one disk and attempts to allocate files among the disks to improve the total power consumption of the storage system subject to response time constraints. Other energy conservation techniques proposed are addressed in [2, 9, 11, 15, 16, 19].

3. Optimal File Allocation Algorithm

This section deals with the combinatorial problem of allocating files to disks so that the minimum number of disks are used and the response time for file requests is below a specified threshold. To define this problem formally, we first introduce some notations. We start with a set of n files. For the i^{th} file let s_i denote its size and let p_i denote the fraction of accesses to it relative to the total file accesses in a unit of time. We observe that the amount of time a disk will spend on servicing requests for a given file is strongly correlated with the frequency of accessing the file as well as its size. For simplicity, we therefore define the load of the i^{th} file, l_i as $l_i = Rp_i\mu_i$ where μ_i is the service time for the file which is a function of its size, i.e., $\mu_i = f(s_i)$ and R is the rate at which requests arrive in the system. Any function, $f(s_i)$ can be used in the proposed algorithms. With this definition, the load corresponds to the fraction of the disk service time spent on servicing the i^{th} file. We use S to denote the total storage capacity of a disk that we are allowed to use, and L to denote the load capacity of a disk. We assume the response time constraint is satisfied, if the cumulative loads of files on any disk are below L. In our experiments, we define L as a percentage of the maximum disk transfer rate. Formally, we define our problem as follows.

Definition 1. *Given a list of tuples $((s_1,l_1),(s_2,l_2),\ldots(s_n,l_n)$, and bounds S and L, find a minimum number of sets $D_1,D_2,\ldots,D_k$, so that each tuple is assigned to a set D_i and $\sum_{(s_i,l_i)\in D_i} s_i \leq S$ and $\sum_{(s_i,l_i)\in D_i} l_i \leq L$ for $i = 1,\ldots,k$.*

This problem has been studied in the literature as the *2-dimensional vector packing problem* (2DVPP). It is known to be NP-Complete and several approximation algorithms for it are known [3]. We will now describe an approximation algorithm, called *Pack_Disks*, which improves on the algorithm in [3], by introducing an efficient data structure that cuts down the time complexity from $O(n^2)$ to $O(n\log n)$. Our file allocation algorithm can be applied periodically based on specified reorganization intervals or triggered automatically whenever the energy consumption of the system exceeds some specified threshold or the response time exceeds the guranteed upper bound.

3.1. The Algorithm

The input to the algorithm is a collection F of n elements corresponding to the files to be allocated. Each element (s_i, l_i) corresponds to a file f_i with size s_i and load l_i. For simplicity, we will normalize the constraints based on the disk capacity S and load L so they are both equal to 1 and the s_i's and l_i's represent fractions of S and L respectively, so they are all within the range $[0,1]$, i.e., $s_i = (\text{size of } f_i)/S)$ and $l_i = (\text{ load of } f_i)/L$. We also assume that all s_i's and l_i's are bounded by some small constant $0 < \rho < 1$. We will later prove that the number of disks loaded by the algorithm is within a factor of $(1/(1-\rho)$ of the optimum.

Our algorithm uses a *max heap* data structure which is a full binary tree with keys on the nodes. In a *max heap*, the following property is always maintained: if node y is a child of node x , then $key(x) \geq key(y)$. In the running time analysis we use the fact that a *max heap* can be created in $O(n)$ time and maintained after an insertion or removal of an element in $O(\log n)$ time (See [1] for more details) . This implies that the key of the the root node is the maximum of the set of keys. Before running the algorithm we will construct two heaps $\vec{S}$ and $\vec{L}$. The heaps $\vec{S}$ and $\vec{L}$ are constructed from F as follows. Let $ST(F)$ contain all elements from F where $s_i \geq l_i$ (also called size-intensive elements) and $LD(F)$ all the other elements (also called load-intensive elements), i.e., $ST(F) = (s_i, l_i) : s_i \geq l_i$ and $LD(F) = (s_i, l_i) : l_i > s_i$. For each element in $ST(F)$, we compute the value $\vec{s}_i = s_i - l_i$ and construct a heap $\vec{S}$ with $\vec{s}_i$'s as keys. Similarly we compute the value $\vec{l}_i = l_i - s_i$ for each element of $LD(F)$ and construct a heap $\vec{L}$ with $\vec{l}_i$'s as keys. We keep with every element of each list its original index in the set F. The algorithm given below then partitions the elements of $\vec{S}$ and $\vec{L}$ into subsets D_i. This in turn induces an allocation of the files represented by F to disks where the original indices of the elements allocated to a subset D_i correspond to the files allocated to the i^{th} disk. For that reason we will use the terms subset or disk D_i interchangeably. For a set $X$where $X \subseteq F$ we denote by $S(X)$ the total storage required by X and by $L(X)$ the total load of X, i.e., $S(X) = \sum_{(s_i,l_i)\in X} s_i$; and $L(X) = \sum_{(s_i,l_i)\in X} l_i$.

A subset D_i is *s-complete* if $1 \geq S(D_i) \geq (1-\rho)$ and it is *l-complete* if $1 \geq L(D_i) \geq (1-\rho)$. It is called *complete* if it is both *s-complete* and *l-complete*. Intuitively, packing all disks such that they are either *complete*, *s-complete* or *l-complete* guarantees that our algorithm will not use more disks than a factor of $1/(1-\rho)$ from the optimal algorithm.

The algorithm assigns elements to one disk at a time, called the current disk. The current disk is packed with elements (one at a time) until it is determined that it cannot take anymore elements or it has enough elements to guarantee that the specified bounds from optimality will be satisfied. The current disk is then closed and a new empty disk becomes the current disk. The main idea of the algorithm is to maximize the number of elements packed into a disk by selecting the next packed element to be either storage-sensitive or load-sensitive based on the the current state of the packed disk. More specifically, the next element assigned to a disk which is dominated by storage-intensive elements (i.e., $S(D_i) \geq L(D_i)$) comes from the load-intensive heap and vice versa. The next few lemmas and theorem prove that the algorithm terminates within $O(n \log n)$ steps with the number of subsets D_i created bounded from the optimum. The main improvement we made to the algorithm of [3] is to better organize the items added to a set D_i in order to avoid searching for an element that needs to be removed from it and placed back in the heaps $\vec{S}$ or $\vec{L}$. This is done by separating the items added to a subset D_i into two lists, namely, *s-list[i]* and *l-list[i]*, based on whether an item's origin is from $\vec{S}$ or $\vec{L}$. As proven below, this allows us to find an appropriate element to be removed from D_i in $O(1)$ time rather than $O(n)$ time required by the algorithm presented in [3]. For lack of space, we state the necessary lemmas and only include here proofs of the lemmas that show that the algorithm terminates in $O(n \log n)$ time and packs the disks within the specified bounds from optimality after our modifications.

Lemma 1. *If $S(D_i) \geq L(D_i)$ and $S(D_i) + s_j > 1$ (lines 5 and 7 of the algorithm), then the last element $\vec{s}_k$ in s-list[i] satisfies the condition $S(D_i) - L(D_i) \leq \vec{s}_k$ (line 8).*

Lemma 2. *If $L(D_i) \geq S(D_i)$ and $L(D_i) + l_j > 1$ (lines 12 and 14 of the algorithm), then the last element $\vec{l}_k$ in l-list[i] satisfies the condition $L(D_i) - S(D_i) \leq \vec{l}_k$ (line 15).*

Lemma 3. *After removing $\vec{s}_k$ and adding $\vec{l}_j$ to D_i (lines 10 and 11), the disk D_i is complete.*

Lemma 4. *After removing $\vec{l}_k$ and adding $\vec{s}_j$ to D_i (lines 17 and 18), the disk D_i is complete.*

Lemma 5. *After exiting the while loop (line 22), all disks except the last one are complete, and at most one of the heaps $\vec{S}$ or $\vec{L}$ is non-empty.*

Lemma 6. *After performing Pack_Remaining$_S$ (or Pack_Remaining$_L$) all disks, except possibly the last one, are either s-complete or l-complete.*

Lemma 7. *Given a set F of n elements, Algorithm Pack_Disks requires $O(n\log n)$ steps.*

Proof. As mentioned before, the formation of the heaps $\vec{S}$ and $\vec{L}$ can be done in $O(n)$ steps For each element, the algorithm removes it from a heap and packs it into a disk exactly once, except under the condition of line 7 or line 14 where an element is removed from the currently packed disk and placed back in one of the heaps $\vec{S}$ or $\vec{L}$ respectively. However by Lemmas 3 and 4, whenever this event happens, the current disk becomes complete and the packing of a new disk is started. Since the algorithm never uses more than n disks, the total number of element removals is at most n. Maintaining the heap structure after removal of the largest element or insertion of an element can be done at a cost of $O(\log n)$. Thus the running time of the algorithm is $O(n\log n)$ as claimed. □

Function `Pack_Remaining_S`

```
1 begin
2     while S⃗ ≠ ∅ do
3         remove next element s⃗_j from list S⃗ ;
4         if S(D_i) + s_j > 1 then
              /* start loading a new disk                    */
5             i ← i + 1;   D_i ← ∅ ;
6             s-list[i] ← ∅; l-list[i] ← ∅
7         insert s⃗_j at the end of the s-list[i]
8 end
```

Function `Pack_Remaining_L`

1 (Same as Pack_Remaining_S() with $\vec{S}$ and $\vec{s}_j$ replaced with $\vec{L}$ and $\vec{l}_j$ respectively)

For completeness we include a proof of the bound on the number of packed disks used by the algorithm, it is similar to the one found in [3].

Theorem 1. *Let the minimum number of disks needed to pack F by any algorithm be denoted by C^* and let the number of disks used by Algorithm Pack_Disks be C^{PD} then*

$$C^{PD} \leq \frac{C^*}{1-\rho} + 1$$

Proof. Clearly $C^* \geq \max(\sum_{(s_i,l_i)\in F} s_i, \sum_{(s_i,l_i)\in F} l_i)$, as $\sum_{(s_i,l_i)\in F} s_i$; and $\sum_{(s_i,l_i)\in F} l_i$ are lower bounds on the number of disks required to satisfy the total size and load requirements respectively.

On the other hand, by Lemmas 5 and 6, the algorithm *Pack_Disks* packs all subsets D_i (except possibly for the last one) such that exactly one of the following 3 cases occurs:

1. all subsets $D_i's$ are *complete*,
2. all subsets $D_i's$ are *s-complete*, one or more are not *l-compete*,
3. all subsets D_i's are *l-complete*, one or more are not *s-complete*.

Under case 1, the theorem follows directly. Under case 2,

$$C^{PD} \leq 1 + \frac{1}{1-\rho} \sum_{(s_i,l_i)\in F} s_i \leq 1 + \frac{1}{1-\rho} C^*.$$

An analogous argument also works under case 3 thus proving our bound. □

3.2. A Variant of the Algorithm

The advantage of *Pack_Disks* may be diminished in some special cases, which were observed in our real life workload log, when many users request a batch of files of similar sizes all at once. As Pack_Disks tends to pack many "same size" files on the same disk it may cause long response time delays for such batched requests. In fact, this case degrades the effect of all the algorithms which tend to pack similar-size files into one disk. To avoid the long response time caused by this case, we introduce some randomization in the packing by modifying the Pack_Disks algorithm to partition the disks into groups and then pack files *group-by-group*, instead of disk at a time. The disks within a single group are packed in a round-robin manner. We call this variant of the algorithm Pack_Disks_v, where v denotes the number of disks in one group.

4. The Simulation

We developed a simulation model to examine the tradeoffs between power saving and request response time. Our simulation environment was developed using SimPy (a simulation framework in Python). The environment consists of a workload generator, a file dispatcher, and a group of hard disks. The workload generator produces file requests based on the configuration parameters given in Table 1. We followed the request patterns used in [17] for generating file sizes and access frequencies using Zipf-like distributions. In this simulation, we assumed that a file has an inverse relation between its access frequency p_i and its size s_i, i.e., the access frequencies of the files follow a Zipf-like distribution while the distribution of their sizes follows inverse Zipf-like distribution. Interestingly, in Section 5, where we analyzed real life work logs we found that this assumption doesn't always hold. Assuming the arrival rate of requests follows a Poisson distribution with expected value *R*, the access rate r_i for the file f_i is p_i*R. In this simulation we assumed that a file request always asks for the whole file. Then the disk load contributed by the file f_i is $l_i = r_i * s_i$. Note that in case we would like to model requests for parts of the file, only the value of s_i can be adjusted accordingly. Once a request is generated, the file dispatcher forwards it to the corresponding disk based on the file-to-disk mapping table, which is built using *Pack_Disks*, our file allocation algorithm. In addition, for the purpose of comparison of power consumption and response times, we also generated a mapping table that randomly maps files among all disks. The mapping time in the dispatcher is ignored since it is negligible when compared with the access time of the big files. Table 2 shows the characteristics of the disk drive used in the simulation. Using the specifications in [14, 18], we built our own disk drive simulation modules. To save energy, the hard disk would be spun down and go into standby mode (Figure 1) after it has been idle for a fixed period which is called *idleness threshold* [11, 12]. Similar to [11, 12], we set the *idleness threshold* to be equal to the time that the disk has to be in the standby mode in order to save the same amount of power that will be consumed by spinning it down to standby mode and subsequently spinning it up to the active mode.

5. Experimental Results

In the following discussion, we examine the behavior of the Pack_Disks algorithm under varying levels of disk load constraints, *L*. The value of *L* is expressed as a fraction of the maximum transfer rate of the disk ($72MB/s$). The results for the group-version of Pack_Disks is ignored here, due to space limitation, but are

similar to those of Pack_Disks because the bursty-arrival phenomenon mentioned does not exist under the Poisson arrival.

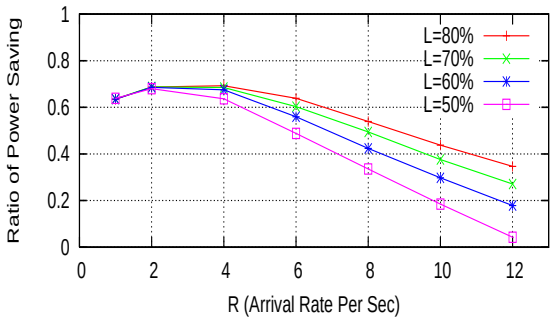

Figure 2. The ratio of power saving v.s. the arrival rate of file access

As shown in Figure 2, when the expected arrival rate of file requests, R, is less than 4 per second, over 60% of power consumption can be saved by using the Pack_Disks algorithm, compared to random placement of files. However, the ratio of power saving, as shown in Figure 5, may decrease along with increasing R, since more active disks are necessary to support the increasing load contributed by these files accesses.

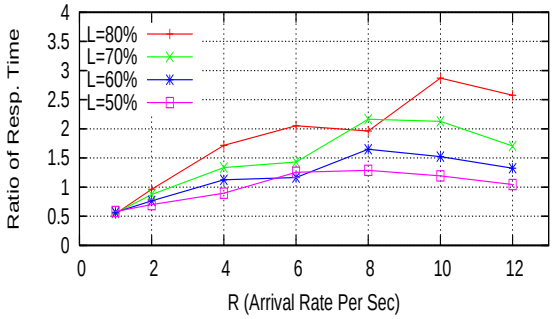

Figure 3. The ratio of the Pack_Disks algorithm to the random allocation in terms of response time

Figure 3 shows the response-time ratio of the Pack_Disks algorithm to random allocation for different L's. The response time in Pack_Disks is 0.5∼2.5 times of that under the random allocation. Figure 4 shows the trade-offs between power cost and access response time for Pack_Disks algorithm while varying L, the constraint on the disk load and setting R at 6. As expected, increasing L can allow us to store files in fewer disks and therefore save more power. This is done at the expense of longer request queues for each of the active disks resulting in longer response times.

5.1. Generated Workload from NERSC Traces

To further demonstrate the effect of Pack_Disks, we collected real life workload logs from NERSC [10] and then used these in the workload generator. NERSC manages large scale scientific observational and experimental data, which are accessed by collaborating scientists around the world. File requests arriving in the center were logged for 30 days (between May 31 and June 29, 2008). There are 88,631 distinct files involved in the 115,832 read requests. The average arrival rate (per second) of the requests is 0.044683. The mean size of files accessed by these requests is 544 MB, which incurred about 7.56 sec of service time when

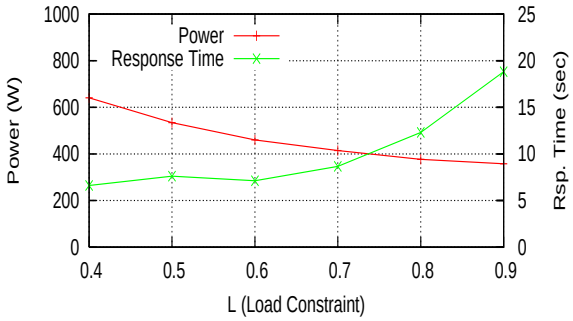

Figure 4. Power cost and response time for different values of L

the disk transmission rate is 72MBps. The minimum space required for storing all the requested files is 95 disks. Next we classified the 88,631 files into 80 bins by their size.

We then computed the proportion of the number of files in each bin compared with the total number of files. The computation shows that the distribution of file sizes is closely related to a Zipf distribution because the proportion decreases almost linearly in the log-log scale. Besides, in this workload, no significant relationship can be observed between the file size and its access frequency.

In our experiment, we let the random placement algorithm pack files into 96 disks similar to the number of disks used by Pack_Disks. Our goal was to examine whether Pack_Disks still saves power even when it uses the same number of disks as the random placement. The simulation was run for 720 simulation hours, and all of the 115,832 requests are regenerated based on the time in the real life workload data. For conveniently observing power savings, we normalized the power cost of both algorithms by taking the power cost as a fraction of the power cost incurred by spinning N disks without any power-saving mechanism. As shown in

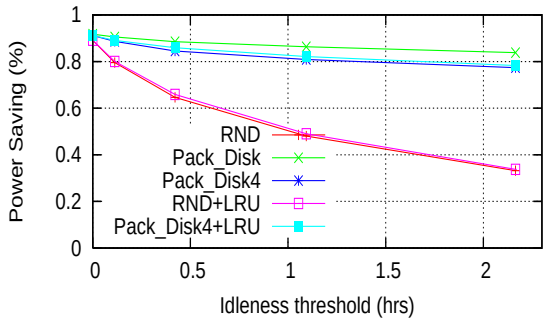

Figure 5. Power Savings under different Idleness Threshold

Figure 5, both Pack_Disks and Pack_Disks_4 can save on the average about 85% of the power consumption. This is much more than that of random placement, which varies from 30% to 90%, under different values of idleness threshold. In fact, saving power even when a long idleness threshold, e.g. 2 hours, is given would be an important feature, because it implies the low frequently spinning down and up, which can prevent the mean-time-to-failure of disks from dramatically decreasing. Figure 6 shows the response time of both algorithms. Although Pack_Disks_4 saves much more power, the requests under Pack_Disks_4 still exhibit response times which are very similar or better to of that random replacement. Figure 6 also reveals that idleness threshold larger than 0.5 hours is necessary for random placement to guarantee that the response time will be within 10 seconds.

Figures 5 and 6 also plot the effects of random placement and Pack_Disk_4 when a 16GB LRU cache is used to cache the frequently accessed files. Unfortunately, for such a workload, the LRU cache does not have much help, where the average hit ratio is only 5.6%. In addition, to observe the effect of Pack_Disk_v,

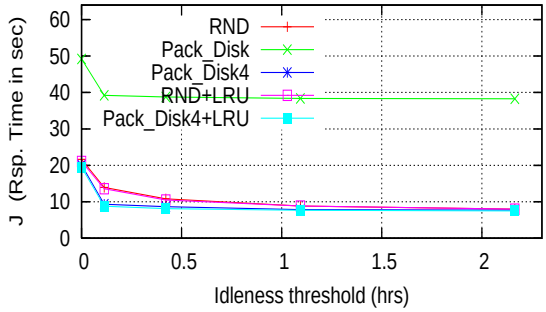

Figure 6. Response times under different Idleness Threshold

with different values of *v*, we measured the response time and power saving ratio of Pack_Disk_v when v is changed from 1 to 8, and Pack_Disk_1 is equal to Pack_Disk. The idleness threshold is set at 0.5 hour. The results reveal 4 is the ideal number of disks to be packed concurrently, because packing disks more than 4 in one time no more reduces response time but degrades the capability of power saving.

6. Conclusions and Future Work

In this paper we demonstrated the importance file allocation strategies for power conservation on disk systems. We showed that careful packing of the files on disks results in a smaller number of spinning disks leading to energy savings of up to a factor of 4 with modest increases in response time. The results of this paper can also be used as a tool for obtaining reliable estimates on the size of a disk farm needed to support a given workload of requests while satisfying constraints on I/O response times. The simulation showed that power saving decreases with arrival rates and increases with higher allowable constraints on disk loads.

In the future we plan to work on improvement of the file allocation algorithm as well as improved modeling of the system in terms of additional workloads as well as more detailed modeling of the disk storage system. More details about planned future work is given below. As a result of our extensive simulation we discovered that further improvements to the response time can be made by restricting the types of files that are allocated to the same disk. For example, we noted that large files that introduce long response time delays, residing on the same disk with small and frequently accessed files lead to the formation of long queues of requests for the latter files waiting for completion of servicing the large file. Additional work also needs to be done to make dynamic decisions about migrating files between disks if it is discovered that the frequency of retrieval of a file deviates significantly from the initial estimates used as an input to the file allocation algorithm. We also plan to investigate our techniques with more real life workloads that include various mixes of read and write requests. In addition, we will include a cache as we believe that cache size and replacement policies may also affect the trade-off between power consumption and response time.

7. *

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] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. The Design and Analysis of Computer Algorithms. Addison - Wesley, Reading, Mass., 1974.

[2] T. Bisson, S. A. Brandt, and D. D. E. Long. A hybrid disk-aware spin-down algorithm with I/O subsystem support. In Proc. Perf., Comput., and Com. Conf., (IPCCC'07), pages 236–245, New Orleans, LA, May 2007.
[3] S. Y. Chang, H.-C. Hwang, and S. Park. A two-dimensional vector packing model for the efficient use of coil cassettes. Comput. Oper. Res., 32(8):2051–2058, 2005.
[4] D. Colarelli and D. 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.
[5] A. Guha. A new approach to disk-based mass storage systems. In 12th NASA Goddard - 21st IEEE Conf. on Mass Storage Syst. and Tech., College Park, Maryland, 2004.
[6] A. Guha. Data archiving using enhanced maid (massive array of idle disks), May 15 – 18 2006.
[7] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke. Reducing disk power consumption in servers with drpm. Computer, 36(12):59–66, 2003.
[8] S. Irani, G. Singh, S. K. Shukla, and R. 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.
[9] D. Narayanan, A. Donnelly, and A. Rowstron. Write off-loading: Practical power management for enterprise storage. In Proc. 6th USENIX Conf. on File and Storage Technologies (FAST'2008), pages 253 – 267, San Jose, California, Feb. 2008.
[10] NERSC. National energy research scientific computing center.
[11] 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.
[12] E. Pinheiro, R. Bianchini, and C. Dubnicki. Exploiting redundancy to conserve energy in storage systems. SIGMETRICS Perform. Eval. Rev., 34(1):15–26, 2006.
[13] D. Ramanathan, S. Irani, and R. Gupta. Latency effects of system level power management algorithms. In ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design, pages 350–356, Piscataway, NJ, USA, 2000. IEEE Press.
[14] Seagate. Seagate Barracuda 7200.10 Serial ATA Product Manual. Seagate Technology, Scotts Valley, CA, Dec 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] 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.
[17] T. Xie. Sea: A striping-based energy-aware strategy for data placement in RAID-structured storage system. IEEE Transactions on Computers, 57(6):748–769, 2008.
[18] 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.
[19] 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.
[20] 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.

Algorithm 3: Algorithm Pack_Disks

Input: A set of n elements $F = \{(s_1, l_1), (s_1, l_2), \ldots, (s_n, l_n)\}$, two heaps $\vec{S}$ and $\vec{L}$
Output: Partition of F into subsets $D_1, D_2, \ldots, D_q$

```
1  begin
       /* start loading first disk                                          */
2      i ← 1;   D_i ← ∅ ;
3      s-list[i] ← ∅; l-list[i] ← ∅ ;
4      while ((S(D_i) ≥ L(D_i) and L⃗ ≠ ∅) or (S(D_i) < L(D_i) and S⃗ ≠ ∅)) do
5          if (S(D_i) ≥ L(D_i)) then
6              remove an element l⃗_j from the heap L⃗;
7              if S(D_i) + s_j > 1 then
8                  let the element, s⃗_k, be the last element added to the s-list[i] ;
                   /* we will prove that (S(D_i) − L(D_i) ≤ s⃗_k)            */
9                  add s⃗_k back to the list S⃗ ;
10                 remove s⃗_k from s-list[i] ;
11             insert l⃗_j at the end of l-list[i];
12         else
13             remove an element s⃗_j from the heap S⃗;
14             if L(D_i) + l_j > 1 then
15                 let the element, l⃗_k, be the last element added to the l-list[i] ;
                   /* we will prove that (L(D_i) − S(D_i) ≤ l⃗_k)            */
16                 add l⃗_k back to the heap L⃗ ;
17                 remove l⃗_k from l-list[i] ;
18             insert s⃗_j at the end of s-list[i] ;
19         if D_i is complete then
               /* start new disk                                            */
20             i ← i + 1;   D_i ← ∅ ;
21             s-list[i] ← ∅; l-list[i] ← ∅ ;
22     if (S⃗ ≠ ∅) then Pack_Remaining_S ;
23     if (L⃗ ≠ ∅) then Pack_Remaining_L ;
24 end
```

Table 1. System Parameters

Parameter	***Value***
n =Number of files	$n = 40000$
R = Expected request rate of files	Poisson arrival rate expected value, R per second ($1 \sim 12$)
p_i = Access frequency of a file	Zipf-like distribution. $p_i = c/\text{rank}_i^{1-\theta}$, where $c = 1 - H_n^{1-\theta}$, $\theta = \log 0.6/\log 0.4$, and $H_n^{1-\theta} = \Sigma_{k=1}^{n} \frac{1}{k^{1-\theta}}$
r_i = Access rate of a file	$r_i = p_i * R$
s_i = File size	Inverse Zipf-like distribution Minimum: 188MB, Maximum: 20 GB
l_i = Disk load contributed by a file	$l_i = r_i * s_i$
Number of disks	100
Simulated Time	4000 sec
Space requirement for all files	12.86 TB

Table 2. 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
Idleness threshold	53.3 secs