A File Allocation Strategy for Energy-Efficient Disk Storage Systems

E. Otoo #1, A. Pinar #2, D. Rotem #3, S. C. Tsao #4

#*Lawrence Berkeley National Laboratory*
University of California
Berkeley, CA 94720
1ejo@hpcrd.lbl.gov
2apinar@lbl.gov
3rotem@hpcrd.lbl.gov
4weafon@lbl.gov

***Abstract*—Exponential data growth is a reality for most enterprise and scientific data centers. Improvements in price/performance and storage densities of disks have made it both easy and affordable to maintain most of the data in large disk storage farms. The provisioning of disk storage farms however, is at the expense of high energy consumption due to the large number of spinning disks. The power for spinning the disks and the associated cooling costs is a significant fraction of the total power consumption of a typical data center. Given the trend of rising global fuel and energy prices and the high rate of data growth, the challenge is to implement appropriate configurations of large scale disk storage systems that meet performance requirements for information retrieval across data centers. We present part of the solution to this challenge with an energy efficient file allocation strategy on a large scale disk storage system. Given performance characteristics of the disks, and a profile of the workload in terms of frequencies of file requests and their sizes, the basic idea is to allocate files to disks such that the disks can be configured into two sets of active (constantly spinning), and passive (capable of being spun up or down) disk pools. The goal is to minimize the number of active disks subject to I/O performance constraints. We present an algorithm for solving this problem with guaranteed bounds from the optimal solution. Our algorithm runs in $O(n)$ time where n is the number of files allocated. It uses a mapping of our file allocation problem to a generalization of the bin packing problem known as 2-dimensional vector packing. Detailed simulation results are also provided.**

Keywords and Index Terms: Disk Storage System, Performance, Energy conservation modeling, file allocation strategy.

I. 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. About 26% of the energy consumption [1] at data centers is attributed to disk storage systems. 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 more energy efficient. The problem is equally significant in high performance scientific computing centers, such as NERSC [2], that manage large scale scientific observational and experimental data, that are accessed by collaborating scientists around the world. Not only is long term retention of vital information essential, but meeting I/O response time requirements both in reading and writing long running simulation results require support of large arrays of disk storage as a staging storage for the short term. Subsequently, data may be offloaded onto tape storage for long term archiving. Figure 2, which we discuss in some detail later, illustrates a typical multi-tier storage hierarchy of a data center. Currently most typical data centers maintain all the disks of their disk storage systems continuously spinning.

The large number of spinning drives has created an emerging and growing energy usage concern at data centers. The problem posed then is what is an appropriate configuration of mass storage system, comprised of multi-tier storage systems of disks and tapes, that is not only energy efficient, but meets the data access response time requirements of applications. Since tape-storage uses only a minuscule fraction of the total energy, the focus is on techniques to minimize the energy usage of the disk storage. 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 [3] and hybrid disks (i.e., disks with integrated flash memory caches) [4]. At the system level, a number of of integrated storage solutions such as MAID [5], PARAID [6], Pergamum [7] and SEA [8] 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 after which they automatically spin down into a standby mode. When a read or write I/O is targeted to it, the disk spins-up again to service the I/O but at the expense of a longer response time. A spun up disk stays spun up until it becomes idle again for a duration of the time-out period (see Figure 4). The different strategies proposed vary primarily in the details of:

- which disks should be allowed to spin-down or spin-up and which should be continuously spinning.
- how file access workloads are characterized and used to direct data distributions and file placements over the array of disk storage.
- the media storage for data caching to support data accesses to the disks and whether it is distributed over the disks or centralized.
- whether the method or heuristic for reconfiguring disk into partitions of different tiers is done dynamically as on-line or an off-line optimization algorithm.

Our experience with the workload of *read accesses* to the data resources at one of the national high performance computing centers, NERSC [2], shows that there is some skewness to file access frequencies that follows Zipf-like distributions. That is at periodic intervals there a large number of file access requests that are directed to a small number of files. Similar access patterns have also been reported from Web server workloads [9].

We present a new system level solution for energy savings in the use of large scale disk storage systems at data centers. The basic ideas involve utilizing the concepts of a massive array of idle disks (MAID) [5] and popular data concentration (PDC) [10]. The array of disks is partitioned into two groups of an *active-tier* and *passive-tier*. The active-tier consists of a group of continuously spinning disks while the passive-tier consists of disks that are spun-down after some period of inactivity and spun-up when accessed but at a cost of a longer latency. By using the active-tier as an on-line disk storage and stirring frequently accessed data to be concentrated on the active-tier, accesses to the passive disks are minimized. The objective of our approach is to realize considerable cost savings by keeping the vast majority of the disks spun down while still providing reasonable response time for file requests.

A. Proposed Approach

The core of this work is how to exploit file allocation to improve energy efficiency. The rich literature on file allocation strategies for disk systems has primarily focused on minimizing response time by utilizing all disks. When power consumption is not a concern, it makes sense to use all resources for gains in performance. Power efficiency on the other hand, requires avoiding overprovisioning of resources, and carefully choosing the minimum set of resources necessary to meet a specified performance goal.

The main source of power dissipation in a disk system is spinning of the disks. One way to improve power efficiency would be to spin down disks if we knew they would not be accessed in the near future, thus it is important to be able to predict the file access patterns. It has been shown that using the access pattern of the most recent time slot to predict the access pattern of the next time slot, can grant significant power savings. Our goal in this work is to increase these savings by clever file allocation. Consider a case, where the file accesses are uniformly distributed among all the disks. There will be very little of power saving in this case, since there would be very little time between two accesses on the same disk. Now, consider another case, where the disks are split into two groups, with great majority of the accesses being to the first group and with infrequent accesses to the second group, creating 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 the frequent accesses to the disks in the first group, which may result in longer response times. Thus, we have to load these disks maximally, but not go beyond an upper bound to guarantee acceptable response times.

One key observation behind our approach is that most of the accesses target only a small group of files. Moreover, we know that the access frequency of a file is inversely proportional to its size. That is smaller files are accessed more frequently. We propose to exploit these observations for energy efficiency, using a strategy to transform these file access patterns to disk access patterns through a clever file allocation.

We face the following combinatorial problem: how do we allocate files to disks so that the number of inactive disks are maximized, and the average response time for the active disks is below a specified threshold? We attack this problem in two phases. First we truncate the list of files so that files with very low access frequencies are placed on the inactive disks. The infrequent accesses to these files allows spinning down these disks for power efficiency. Then we try to pack the remaining files onto as few disks as possible while respecting the disk capacity and response time constraints (see Figure 1. To define this problem formally, we first introduce our notation.

We start with a set of n files, where let s_i and p_i denote, respectively, the size of the ith file and the fraction of accesses to this file to all accesses in a unit time. We define the load of a file as $l_i = p_i s_i R$, where R is the file access for the system. With this definition, the load corresponds to how much the disk will spend accessing only this file in a specified period. 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 a disk is below L. In our experiments, we define L as a percentage of the maximum disk transfer rate. Formally, we define our problem as follows.

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 sets $D_1,D_2,\ldots,D_k$, so that each tuple is assigned to a set D_i, and

$$\sum_{(s_i,l_i)\in D_i} s_i \le S \quad \text{and} \quad \sum_{(s_i,l_i)\in D_i} l_i \le L \quad \text{for } i = 1,\ldots k$$

This problem has been studied in the literature as the *2-dimensional vector packing problem* (2DVPP). In Section III, we will discuss this problem in more detail, and describe a linear-time approximation algorithm. Our algorithm improves on the algorithm in [11], by describing an efficient data structure that cuts down the complexity to $O(n)$, as opposed to the $O(n^2)$ complexity of the original algorithm.

After applying our file allocation algorithm, we have three groups of disks. In addition to the active and passive disks, we will have empty disks, as a sign of success of our packing algorithm, as illustrated in Fig. 1. We assume the empty disks will be used for writing new files by the applications.

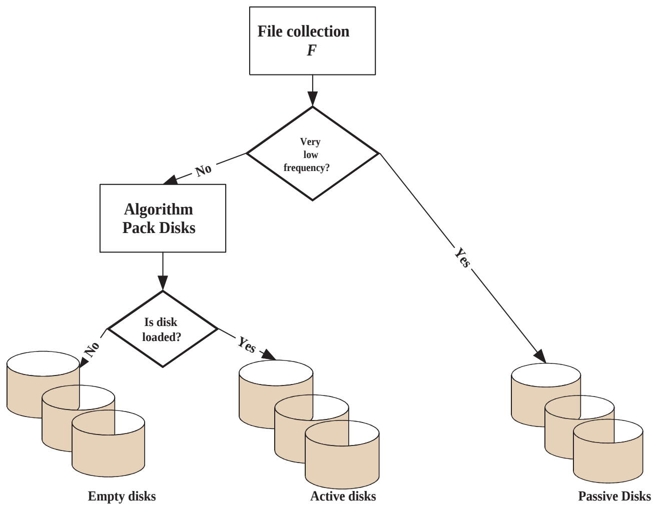

Fig. 1. File Allocation Strategy

B. Main Contributions

The main contributions of this paper are:

- We demonstrated that file allocation strategies can help to significantly reduce energy consumption of disk dystems. These strategies can be used in conjuction with other techniques on the physical device level introduced in the literature.
- We mapped 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 improved the running time of a 2DVPP algorithm with best known bounds from optimality such that it now runs in $O(n)$ rather than $O(n^2)$ time [11] where n is the number of files.
- We developed techniques for quantifying and controlling the tradeoffs involved in energy conservation vs. response time of disk based file systems. This is done by varying the allowable utilization of each disk (also called *load*) in terms of the fraction of its maximal transfer rate.
- Using extensive simulations with realistic workloads and accurate disk characteristics, we calculated the energy savings and response times achieved by using our file allocation techniques as compared with random file allocation. We demonstrated that our techniques achieve significant energy savings over a wide range of workload parame-

ters values with minimal response time degradation. Although we characterize our solution as an off-line solution, it can be applied in a semi-dynamic manner by accumulating access statistics for periodic reorganization of the file allocations. Another use of the solution presented is for computing the percentage of disks that must be maintained on-line to meet data access response time given a limited budget constraint.

C. Organization of the Paper

The remainder of the paper is organized as follows. In the next section we present a background and related work on energy saving approaches in large scale disk-storage based data centers. We give our heuristic algorithm in Section III and provide an analysis of the runtime and the quality performance of the proposed algorithm. A discussion of the our simulation environment is given in Section IV. Section V presents our experimental results and we conclude in Section VI where we also discuss directions for future work.

II. System Environment and Related Work

A. System Architecture and Motivation

Tape storage has long been the standard for mid-to-long-term archival of enterprise's data. Its slow I/O rates and lengthy access times, coupled with the revolving cost of transferring data to new tape media, have prompted the need for research into alternative approaches for mass storage systems. The need for on-line and near-line accesses to ever increasing growth of an institution's data resources has further motivated the demand for an alternative approach. One solution proposed is that of a disk-based mass storage system either as a stand-alone system or complementary to a robotic-tape system. The ideas on implementing disk-based mass storage systems are discussed in some details in [5], [7]. The problem is that disk-based mass storage systems involve massively large number of disks that expend a considerable amount of energy for spinning and cooling.

High performance scientific computing centers, run by government laboratories, generate and manage massively large datasets that are retained on mass storage systems. These centers, like their data center counterpart in private industries, are projected to be predominately disk-based and may continue to include robotic tapes for long term archives. The motivation for this work is the need to reduce the energy consumption of large data center with particular focus on mass storage support for scientific data. Figure 2 illustrates a typical storage configuration of a data center that motivated this work.

The disk-storage is used to augment long term archival tape library storage. The group of passive disks can be perceived as a second level tier termed the *passive-tier* and sits between a first tier termed the *active-tier* and a third tier of a tape system. The disk arrays are configured into units of RAID5-sets called *storage units*, and are spun up or down in such units. A storage unit is designated as a member of either the active-tier or the passive-tier. The active-tier serves effectively as the on-line disk storage. The passive-tier forms a long term storage, but may also be used as a disk cache if a tape-based library is rather used for long term storage. Unit of data accesses is a file. Each file is allocated in its entirety to one RAID set. We do not consider file partitioning or replication. The use of a RAID set is to ensure reliability. To simplify subsequent discussions, we consider each RAID set as a single disk. This does not restrict the generality of the configuration. We consider homogeneous disk storage system with each disk having the same performance characteristics

The services of the multi-tier storage is as follows. For read requests, the file is serviced from the active-tier if a copy of the file already resides there. It is serviced from the passive disk if the file resides on a disk in the passive-tier. In this case the storage unit needs to be powered up at the cost of additional latency before servicing the request. Otherwise it is serviced from tape storage by first caching the file directly onto an active disk before being transferred to the user. Files to be written are first directed written to reserved disks in the passive-tier that are spun-up and stay active until all writes are completed. The rationale is that most scientific applications write massive amounts of data from simulation and then conduct analysis on the data after a number of days. File in the passive-tier would be subsequently migrated to tape after a sufficiently long period of inactivity. By storing frequently accessed files on active storage and infrequently accessed ones on passive storage, we expect considerable energy savings.

B. 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) [5], popular data concentration [10] and energy-aware caching [12]. Similar to the work in [10], 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.

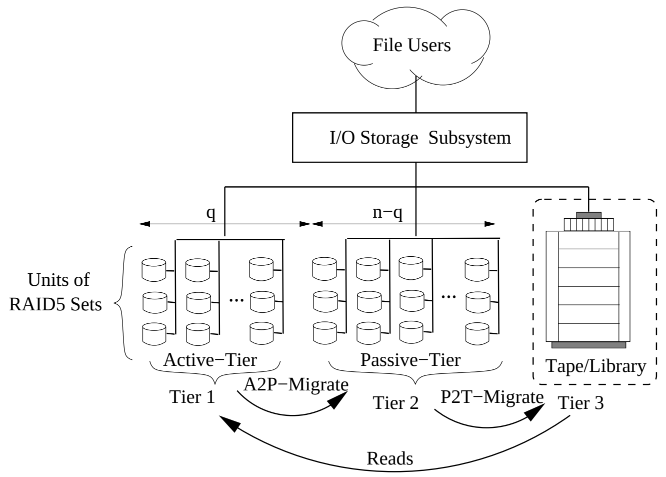

Fig. 2. Typical Configuration of Large Disk Storage Usage at a Data Center

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.

Portable computers, have for some time now, implemented power reduction techniques to achieve longer battery lives. e.g., spinning down the disk during long periods of inactivity. This idea has only been recently explored in different contexts for conserving energy in large scale computing [10], [13], [14]. Colarelli and Grunwald [5] 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 [13], [15]. 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 [14]. Other energy conservation techniques proposed are addressed in [7], [10], [16], [17], [17]–[19].

III. Optimal File Allocation on Algorithm

In this section we present an algorithm called *Pack_Disks* which is an improved version of the algorithm given in [11]. As a result of these improvements, our algorithm runs in $O(n)$ time as opposed to the $O(n^2)$ time requirements of the algorithm presented in [11]. 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 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.,

$$\begin{aligned} s_i &= \frac{\text{size of file}\, f_i}{S} \\ l_i &= \frac{\text{load of file }\, f_i}{L}. \end{aligned}$$

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 $\frac{1}{1-\rho}$ of the optimum. Since for most applications ρ is much smaller than 0.5, the algorithm of [11] is more attractive for our purposes than the generic one given in [20] which guarantees a factor of 2 from optimal for any value of ρ but runs in $O(nlogn)$ time.

Before running the algorithm we will construct two lists $\vec{S}$ and $\vec{L}$. The lists $\vec{S}$ 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 will compute the value $\vec{s_i} = s_i - l_i$ and construct a list $\vec{S}$ from the values $\vec{s_i}$. Similarly we will compute the value $\vec{l_i} = l_i - s_i$ for each element of $LD(F)$ and construct a list $\vec{L}$ from the values $\vec{l_i}$. We will keep with every element of each list its original index in the set F. The algorithm given below will partition 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 corresponds 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*.

The next few lemmas and theorem prove that the algorithm terminates within $O(n)$ steps with the number of subsets D_i created bounded from the optimum. The main improvement we made to the algorithm of [11] 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 lists $\vec{S}$ 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 its origin. 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 [11]. We only include here proofs of the lemmas that show that the algorithm terminates in $O(n)$ time and packs the disks correctly after our modifications.

Algorithm 1: Algorithm Pack_Disks

```
Input: A set of $n$ elements
       $F = \{(s_1,l_1),(s_1,l_2),\ldots,(s_n,l_n)\}$, two
       lists $\vec{S}$ and $\vec{L}$
Output: Partition of $F$ into subsets
       $D_1, D_2, \ldots, D_q$
1  begin
       /* start loading first disk */
2      $i \leftarrow 1$;   $D_i \leftarrow \emptyset$ ;
3      s-list[i] $\leftarrow \emptyset$; l-list[i] $\leftarrow \emptyset$ ;
4      while $((S(D_i) \geq L(D_i)$ and $\vec{L} \neq \emptyset)$ or
       $(S(D_i) < L(D_i)$and $\vec{S} \neq \emptyset))$ do
5          if $(S(D_i) \geq L(D_i))$ then
6              remove an element $\vec{l}_j$ from the list $\vec{L}$;
7              if $S(D_i) + s_j > 1$ then
8                  let the element, $\vec{s}_k$, be the last
                   element added to the s-list[i] ;
                   /* we will prove that
                      $(S(D_i) - L(D_i) \leq \vec{s}_k)$ */
9                  add $\vec{s}_k$ back to the list $\vec{S}$ ;
10                 remove $\vec{s}_k$ from s-list[i] ;
11             insert $\vec{l}_j$ at the end of l-list[i];
12         else
13             remove an element $\vec{s}_j$ from the list $\vec{S}$;
14             if $L(D_i) + l_j > 1$ then
15                 let the element, $\vec{l}_k$, be the last
                   element added to the l-list[i] ;
                   /* we will prove that
                      $(L(D_i) - S(D_i) \leq \vec{l}_k)$ */
16                 add $\vec{l}_k$ back to the list $\vec{L}$ ;
17                 remove $\vec{l}_k$ from l-list[i] ;
18             insert $\vec{s}_j$ at the end of s-list[i] ;
19         if $D_i$ is complete then
               /* start new disk */
20             $i \leftarrow i+1$;   $D_i \leftarrow \emptyset$ ;
21             s-list[i] $\leftarrow \emptyset$; l-list[i] $\leftarrow \emptyset$ ;
22     if $(\vec{S} \neq \emptyset)$ then Pack_Remaining_S ;
23     if $(\vec{L} \neq \emptyset)$ then  Pack_Remaining_L ;
24 end
```

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).*

Proof. Let m be the number of elements currently assigned to D_i. Clearly $m > 0$ and its *s-list[i]* is not empty. If m=1, the only element in D_i must be in its *s-list[i]* and it trivially satisfies the above condition, so it can be chosen as $\vec{s}_k$(line 8).

If m >1, consider the last element $\vec{s}_k$ added to the *s-list[i]* of D_i, and let s_k and l_k be its corresponding *s-value* and *l-value*, i.e., $\vec{s}_k = s_k - l_k$. Let D'_i represent those elements assigned to D_i just before the element $\vec{s}_k$ was added and $\bar{D}_i$ the elements added to D_i after $\vec{s}_k$. By the nature of the algorithm $L(D'_i) > S(D'_i)$ ($\vec{s}_k$ was added from $\vec{S}$, line 13) and since all elements of $\bar{D}_i$ were added from $\vec{L}$ (they are in the list *l-list[i]*) we have $L(\bar{D}_i) > S(\bar{D}_i)$.

From the fact that $D_i = D_i' \cup \vec{s}_k \cup \bar{D}_i$ we have

$$S(D_i) = S(D_i') + s_k + S(\bar{D}_i)$$

and

$$L(D_i) = L(D_i') + l_k + L(\bar{D}_i),$$

and

$$S(D_i) - L(D_i) = (S(\bar{D}_i) + S(D_i')) - (L(\bar{D}_i) + L(D_i')) + (s_k - l_k).$$

Since

$$(S(\bar{D}_i) + S(D_i')) - (L(\bar{D}_i) + L(D_i')) < 0$$

it follows that $S(D_i) - L(D_i) \leq s_k - l_k$ as claimed.

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).*

Proof. Omitted, uses the same arguments as Lemma 1.

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.*

Proof. Proved in [11]

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.*

Proof. Omitted, uses the same arguments as Lemma 2.

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

Proof. Proved in [11]

Lemma 6. *After performing Pack_Remaining_S (or Pack_Remaining_L) all disks except possibly the last one are s-complete (or l-complete).*

Proof. Proved in [11]

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

Proof. The algorithm packs each element of F exactly once into a disk, except under the condition of line 7 or line 14 where an element is removed from the currently packed disk and placed back in the lists $\vec{S}$ 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 and thus the running time of the algorithm is $O(n)$ as claimed.

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 [11].

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 $C^* \geq \max\{\sum_{(s_i,l_i)\in F} s_i, \sum_{(s_i,l_i)\in F} l_i\}$. 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-complete*
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.

Function `Pack_Remaining_S`

```
1 begin
2   while $\vec{S} \neq \emptyset$ do
3     remove next element $\vec{s}_j$ from list $\vec{S}$ ;
4     if $S(D_i) + s_j > 1$ then
          /* start loading a new disk */
5       $i \leftarrow i+1$;   $D_i \leftarrow \emptyset$ ;
6       s-list[i] $\leftarrow \emptyset$; l-list[i] $\leftarrow \emptyset$
7     insert $\vec{s}_j$ at the end of the s-list[i]
8 end
```

Function `Pack_Remaining_L`

```
1 begin
2     while $\vec{L} \neq \emptyset$ do
3         remove next element $\vec{l_j}$ from list $\vec{L}$ ;
4         if $(L(D_i) + l_j > 1)$ then /* start loading new disk */
5             $i \leftarrow i+1$;   $D_i \leftarrow \emptyset$ ;
6             s-list[i] $\leftarrow \emptyset$; l-list[i] $\leftarrow \emptyset$ ;
7         insert $\vec{l_j}$ at the end of the l-list[i]
8 end
```

IV. The Simulation

We developed a simulation model to examine the tradeoffs between power saving and request response time. Our simulation environment, developed using SimPy [21], is shown in Figure 3. 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 I. We followed the request patterns used in [8] for generating file sizes and access frequencies using a Zipf-like distributions [22]. A file has an inverse relation between its access frequency p_i and its size s_i [9], [23], i.e., the access frequencies of the files follow a Zipf-like distribution while the distribution of their sizes follow inverse Zipf-like distribution. 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. Assuming that a file request always asks for the whole file, then the disk load contributed by the file f_i is l_i where $l_i = r_i * s_i$.

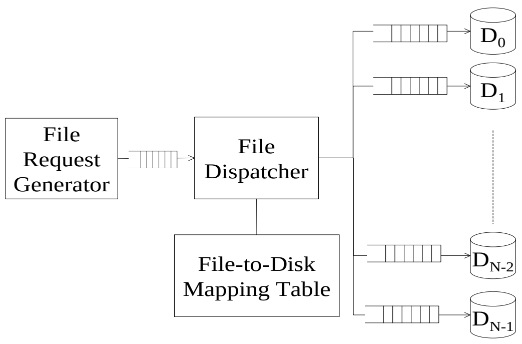

Fig. 3. The Components of the simulation

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 time, 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 II shows the characteristics of hard disk used in the simulation. Using the specifications in [24] and [3] we built our own hard disk simulation modules, instead of revising DiskSim [25], to simulate the performance and the energy cost of disks, because the latest version of DiskSim, released at May, 2008, still provides only old and small disk models, e.g. 1998's 8GBytes disks. Also, although DiskSim is a discrete-time simulator, the number of events needed to handle a file request is highly correlated with its size. Thus, DiskSim was found to be too slow for realistic data center environments on the scale we are interested in, i.e. $\sim$ 500GBytes per disk with tens of thousands of files and multiple terabytes of total data storage. Finally, to save energy, the hard disk would be spun down and go into standby mode (Figure 4) after it has been idle for a fixed period which is called *idleness threshold* [10], [26]. Similar to [10], [26], 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.

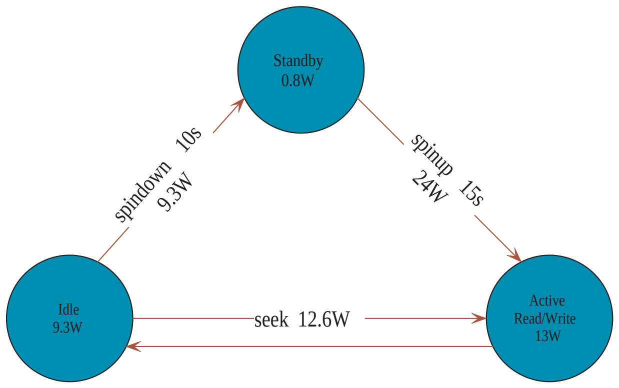

Fig. 4. Power consumption of the different disk modes and transition times

V. Experimental Results

In the following discussion, we examine the behavior of our 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).

TABLE I
CONFIGURATIONS OF THE 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 II
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

As shown in Figure 5 , when the expected arrival rate of file requests, R, is less than 4, over 60% of power consumption can be saved by using the Pack_Disks algorithm, compared to random placement of files. Figures 6 and 7 further show the plots of their exact power consumption when $L = 80$ and 60 respectively. 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. These are plotted in Figures 8 and Figure 9. In these figures we plot how many of our 100 disks have files loaded into them (active-disks) by the algorithm. These active disks can further be classified based on whether they are *complete* (achieving both size and load constraints), *l-complete* (achieving only load constraint), or *s-complete* (achieving only size constraint). As expected, with increasing rate of arrival, R, the number of active disks increases, the number of *s-complete* decreases and the number of *l-complete* increases.

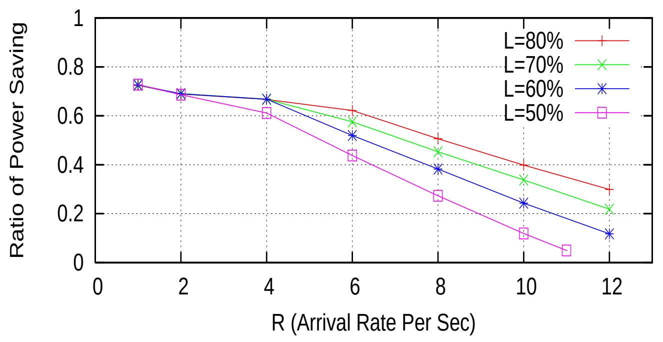

Fig. 5. The ratio of power saving v.s. the arrival rate of file access

However, as shown in Figure 10, although power consumption is saved by using our algorithm, the expected response time of a file access becomes longer as compared with random allocation. When $R < 4$, the response time of our algorithm increases since the total number of active disks does not increase with the arrival rate (shown in Figure 8). When $R >4$, the number of active disks increases and files are distributed among more disks. We found that few small files with large r_i's

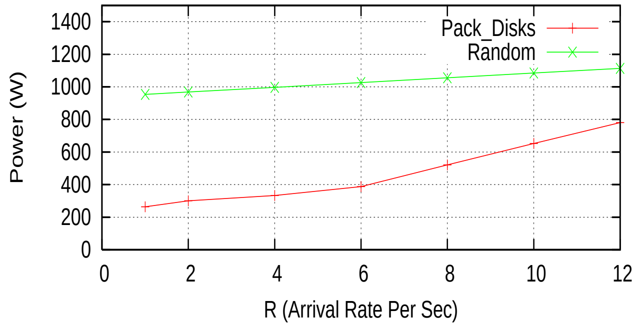

Fig. 6. Comparison of power costs between the Pack_Disks algorithm and the random allocation when L=80%

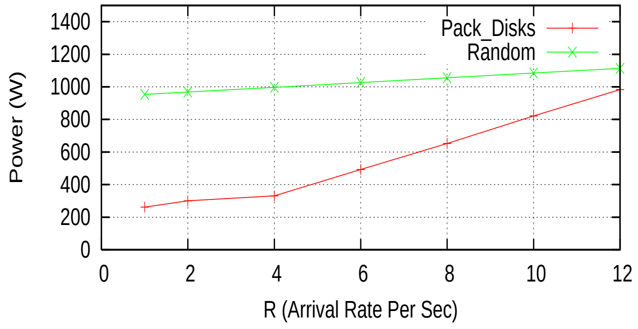

Fig. 7. Comparison of power costs between the Pack_Disks algorithm and the random allocation when L=60%

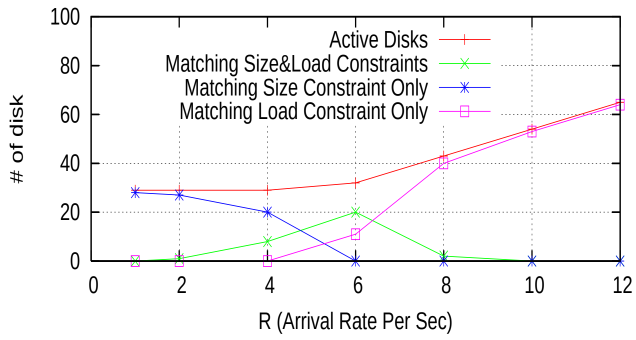

Fig. 8. Active disks and their space/load usage status when L=80%

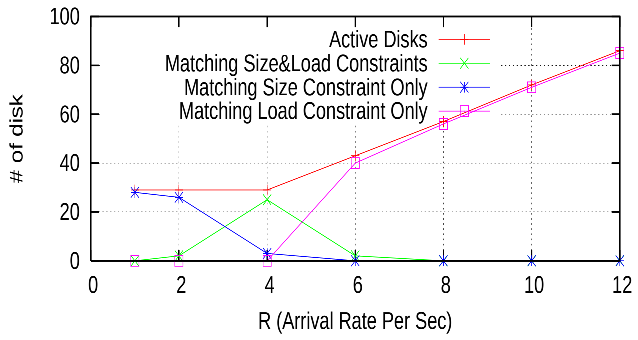

Fig. 9. Active disks and their space/load usage status when L=60%

are allocated along with huge files on the same disk, and thus the response time decreases. For example, we found that the smallest 118 files are placed on the first disk along with the biggest 97 files when R=4, but the number of the small files decreases to 1 when the R= 8. However, when R >8, all files are classified into the load-intensive type, which changes the trend on the ratio of the size-intensive files to the load-intensive ones for each disk. The smallest files are again put with the biggest files in the first several disks. Similar patterns were observed in Figure 11, where L= 60%. Finally, Figure 12 shows the response-time ratio of the Pack_Disks algorithm to the random allocation for different L's. The response time in Pack_Disks is 1.5∼2.5 times of that under the random allocation.

Figure 13 shows the tradeoffs between power cost and access response time for our 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.

VI. Conclusion and Future Work

In this paper we demonstrated the importance of smart 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.

Our current work is a first step in this important and challenging field of energy-efficient disk systems. 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.

A. *Algorithmic improvements*

As a result of our extensive simulation we discovered that further improvements to the response time can be

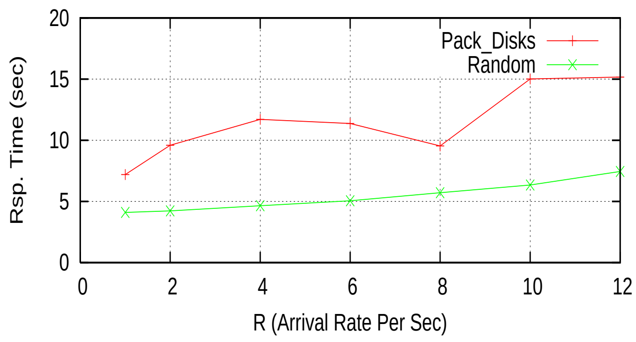

Fig. 10. Comparison of response times between our algorithm and random allocation when *L*= 80%

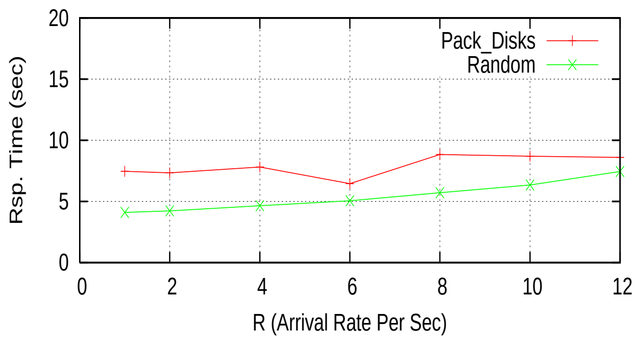

Fig. 11. Comparison of response times between our algorithm and random allocation when *L*= 60%

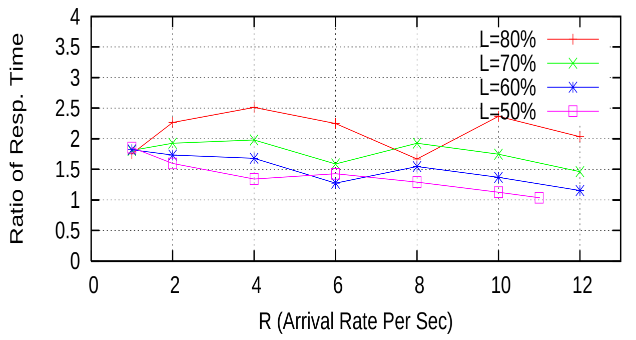

Fig. 12. The ratio of the Pack_Disks algorithm to the random allocation in terms of response time

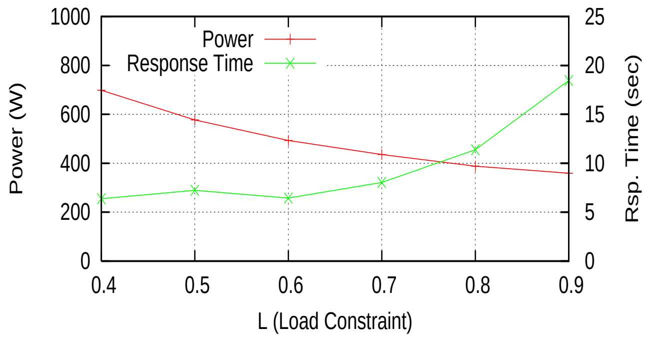

Fig. 13. Power cost and response time for different valuse of *L*

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.

The algorithm *Pack_Disks* can be viewed as a template for many heuristics that use different policies for selecting items to be placed in a disk (lines 6 and 13) or re-allocated (line 8 and 15) . Such heuristics need to be analyzed and may lead to improved bounds from optimality or tailor the disk packing to satisfy some additional constraints.

B. System modeling

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.

Acknowledgments

This work was supported by the Director, Office of Science, 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. Special thanks go to Jason Hick and Wayne Hurlbert for providing us with summary logs of HPSS accesses over a months period, that use the HSI protocol.

References

[1] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, “Reducing disk power consumption in servers with drpm,” *Computer*, vol. 36, no. 12, pp. 59–66, 2003.

[2] NERSC, “National energy research scientific computing center.” [Online]. Available: http://www.nersc.gov/

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

[4] T. S. HM16HJ1, “Hybrid HDD.” [Online]. Available: http://www.samsung.com

[5] D. Colarelli and D. Grunwald, “Massive arrays of idle disks for storage archives,” in *Supercomputing’02: Proc. ACM/IEEE Conference on Supercomputing*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2002, pp. 1 – 11.

[6] C. Weddle, M. Oldham, J. Qian, and A. Wang, “PARAID: A gear shifting power-aware RAID,” *ACM Trans. on Storage (TOS)*, vol. 3, no. 3, pp. 28 – 26, Oct. 2007.

[7] 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)*, San Jose, California, Feb. 2008, pp. 1 – 16.

[8] T. Xie, “Sea: A striping-based energy-aware strategy for data placement in RAID-structured storage system,” *IEEE Transactions on Computers*, vol. 57, no. 6, pp. 748–769, 2008.

[9] C. Cunha, A. Bestavros, and M. Crovella, “Characteristics of www client-based traces,” Boston University, Boston, MA, USA, Tech. Rep., 1995.

[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] 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.*, vol. 32, no. 8, pp. 2051–2058, 2005.

[12] 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.* Washington, DC, USA: IEEE Computer Society, 2004, p. 118.

[13] A. Guha, “Data archiving using enhanced maid(massive array of idle disks),” College Park, Maryland, May 15 – 18 2006. [Online]. Available: http://romulus.gsfc.nasa.gov/msst/conf2006/index.html

[14] P. Radkov, L. Yin, P. Goyal, P. Sarkar, and P. Shenoy, “A performance comparison of NFS and iSCSI for IP-Networked storage,” in *Proc. of the 3rd USENIX Conference on File and Storage Technologies*, March 2004, pp. 101 – 114. [Online]. Available: citeseer.ist.psu.edu/radkov04performance.html

[15] 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. [Online]. Available: http://romulus.gsfc.nasa.gov/msst/conf2004/

[16] C. Weddle, M. Oldham, J. Qian, and A. Wang, “Paraid: A gear-shifting power-aware raid,” in *Proc. 5th USENIX Conf. on File and Storage Technologies (FAST’2005)*, San Jose, California, Feb. 2007.

[17] 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)*, San Jose, California, Feb. 2008, pp. 253 – 267.

[18] 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)*, New Orleans, LA, May 2007, pp. 236–245.

[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*. NY, USA: ACM Press, 2005, pp. 177–190.

[20] H. Kellerer and V. Kotov, “An approximation algorithm with absolute worst-case performance ratio 2 for two-dimensional vector packing,” *Operations Research Letters*, vol. 31, no. 1, pp. 35 – 41, Jan. 2003, approx2DVectorPack.pdf.

[21] S. S. S. P. in Python, “http://simpy.sourceforge.net/archive.htm.”

[22] L.-W. Lee, P. Scheuermann, and R. Vingralek, “File assignment in parallel I/O systems with minimal variance of service time,” *IEEE Transactions on Computers*, vol. 49, no. 2, pp. 127–140, Feb. 2000. [Online]. Available: http://www.computer.org/tc/tc2000/t0127abs.htm

[23] S. Glassman, “A caching relay for the world wide web,” in *Selected papers of the first conference on World-Wide Web*. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., 1994, pp. 165–173.

[24] 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.* Berkeley, CA, USA: USENIX Association, 2003, pp. 217–230.

[25] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger, “The disksim simulation environment.” [Online]. Available: http://www.pdl.cmu.edu/DiskSim/

[26] E. Pinheiro, R. Bianchini, and C. Dubnicki, “Exploiting redundancy to conserve energy in storage systems,” *SIGMETRICS Perform. Eval. Rev.*, vol. 34, no. 1, pp. 15–26, 2006.