Building JACK: Developing Metrics for Use in Multi-Objective Optimal Data Allocation Strategies

Technical Report UCSC-SSRC-14-01
January 2014

Christina R. Strong crstrong@cs.ucsc.edu
Ahmed Amer aamer@scu.edu
Darrell D.E. Long darrell@cs.ucsc.edu

Storage Systems Research Center
Baskin School of Engineering
University of California, Santa Cruz
Santa Cruz, CA 95064
http://www.ssrc.ucsc.edu/

Building JACK: Developing Metrics for Use in Multi-Objective Optimal Data Allocation Strategies

Christina R. Strong
University of California, Santa Cruz

Ahmed Amer
Santa Clara University

Darrell D.E. Long
University of California, Santa Cruz

Abstract

Current file systems optimize for a single objective, often to the detriment of other objectives. This can result in an unbalanced system that does not reflect the needs of the system or its users. We suggest addressing the data allocation problem as a multi-objective optimization problem, using our general data placement framework, JACK. This provides us with a means of finding an optimal data allocation, tailored to the requirements of individual systems, requiring at most the specification of the relative importance of competing performance goals.

For such an approach to be feasible, we need to define meaningful, yet observable, metrics for such a multi-objective optimization problem. We introduce the objectives we intend to optimize for and example metrics which can be used to gauge them. We provide a detailed description of how each metric was developed, using a data set provided by Los Alamos National Laboratory for evaluation. We also explain how the metrics enable the development of JACK, with the intent that others can make use of our metrics and develop their own metrics for additional performance goals.

1 Introduction

Current file systems are focused on optimizing for a single objective, with, e.g., one system claiming to have the fastest response time and another boasting of the most energy savings. Some systems attempt to optimize for two similar objectives, such as load balancing and system responsiveness. The problem arises when a system that has been optimized for energy savings, for example, is expected to have excellent load balancing. Not only are these two objectives in direct conflict with each other, but it effectively forces a custom data allocation scheme for that particular set of requirements and that particular system to be produced.

We are developing a general approach to address such problems that would be simpler to administer and deploy for different applications. Rather than developing custom solutions for specific domains and forcing these ad hoc applications onto systems that may not be suited for them, we are choosing to address data placement as a multi-objective optimization problem from the start. We have chosen to use energy savings, load balancing, and system responsiveness as our initial objectives, in order to show that multi-objective optimization is useful when considering both complimentary and contrasting objectives. The advantage of optimizing for multiple objectives is that rather than one single optimal answer, a set of optimal answers is obtained. Thus, we have the flexibility to chose an answer that reflects the requirements of the system and users, yet is still optimal.

To this end we are developing JACK, which stands for *Joining, Aggregating, and Collocating Knowledge*. The multi-objective nature of JACK allows the allocation problem to be tailored to any environment by specifying the objective of specific environments and users, rather than awkwardly adapting traditional single-objective schemes. Such adaptation is both imprecise and demands a mastery of both the allocation scheme and the specific environment, whereas an implementation of JACK would demand at most an understanding of the target system's performance goals.

In order to develop JACK, some groundwork is necessary: not only must we identify the objectives we intend to optimize for, it is also necessary to define the metrics that will be used to measure optimality for each objective. Our goal is to demonstrate how such metrics for JACK can be defined. We present a set of three example performance objectives, and demonstrate how we designed and refined metrics for these objectives. It is our intent that our metrics be used for other work that requires the measurement of these objectives and as a source of inspiration for the development of metrics for other objectives.

2 Background

It is necessary to understand the problem we are addressing before we can develop a set of measurable metrics; we therefore begin by observing the difference between a single objective optimization problem and a multi-objective optimization problem. A single objective optimization problem can be expressed as in Equation 1. Here, $f(x)$ refers to the objective function to be minimized over x. In a single objective optimization problem, you have a single objective function that you're minimizing (or maximizing). For example, consider optimizing a system for energy savings. We would formulate a function that represented the system's energy consumption, and attempt to minimize it as much as possible. While this is beneficial when that single objective is all you care about, if there are other objectives that are important, Equation 1 has no way to take those into account.

$$\underset{x}{\text{minimize}} \quad f(x) \tag{1}$$

The solution to this is to use a multi-objective optimization problem. The standard form of a multi-objective optimization problem is shown in Equation 2. This is extremely similar to Equation 1; in fact only one thing has changed. Instead of trying to optimize a single objective function $f(x)$, we now have a vector of objective functions, $F_i(x)$, that we are attempting to minimize.

$$\underset{x}{\text{minimize}} \quad \mathbf{F}(x) = [F_1(x), F_2(x), \ldots F_k(x)] \tag{2}$$

Multi-objective optimization problems occur when there are multiple objectives that need to be optimized, with no clear indication of a preferred objective relative to the others. In a multi- objective optimization, often there is no single global solution. This is especially true when objectives conflict with each other: consider two objectives that are inversely proportional to each other. For our purposes this is extremely beneficial, as it allows us to pick a solution that best fits the current system and users.

2.1 Example Optimization Objectives

Before we can begin to develop metrics it is also necessary to know what performance objectives we are using for optimization. The three objectives we are initially focusing on are load balancing, system responsiveness, and energy savings. Optimizing for load balancing and system responsiveness has been important since the early days of distributed systems [19]. Load balancing addresses the need to keep popular data evenly spread across the system, in order to not overload any device. It is closely related to system responsiveness, which is concerned with how quickly the system responds to a request. Energy savings is particularly an issue in backup systems, where disks can be spun down in order to save power and money [4, 3, 16], but is also beginning to be addressed in primary systems where spinning disks down is not always an option.

Very few optimization techniques go beyond optimizing for one objective; those that do typically consider no more than two, and in a very specific context. The work mentioned below all have features that make them unique in their specific single optimization problem. However, since we are looking at these objectives as part of a multi-objective optimization problem, the nature of our work differs from that below.

Load Balancing One of the more common methods to achieve load balancing is through replication of the popular data. A good example of this technique is CRUSH [20], which is a pseudo-random data distribution algorithm. CRUSH distributes object replicas across a storage cluster by mapping an input value to a list of devices on which to store object replicas. Distribution is pseudo-random in that there is no apparent correlation between resulting output from similar inputs or items stored on any device. Another example is MMPacking [10], a load and storage balancing method that uses a combination of replication and a weighted scheduling algorithm in order to achieve load balancing. The work done by Ma et al. [13] extends MMPacking, using best fit bin packing in order to find the optimal bin capacity in order to minimize the required disk capacity.

Load balancing is also used in the file allocation problem, as a way to minimize disk utilization. One popular algorithm is the Greedy algorithm, which originated from the Longest Processing Time algorithm (LPT) [9]. The LPT algorithm is a simple greedy algorithm designed for multiprocessor load balancing and can operate on- or off-line. At each step, LPT greedily assigns a process to the processor that has the least accumulated load. When LPT is running online, processes are assigned in the order of their arrival; in offline mode, processes are ordered by their load and assignment is done in decreasing load order. When applied to the file allocation problem, the LPT algorithm becomes the Greedy algorithm [12], where the load of each file is defined as the product of the file access rate and the access service time. Our metric for load balancing uses the same definition of load.

System Responsiveness System responsiveness refers to the amount of time it takes for the system to respond to a request, be it a query for data from the user or a simple *ls*. This is closely related to load balancing, as distributing the load often results in an increase of system

responsiveness. This is not always true, however, as one of the best load balancing algorithms, the Greedy algorithm [9], results in very poor system responsiveness according to the comparisons performed by Lee et al. [12]. Likewise, the best algorithm for average response time, Sort Partition, does very poorly in load balancing.

Lee et al. [12] present two different algorithms, Sort Partition (SP) and Hybrid Partition (HP), and evaluate against the Greedy algorithm using average response time as a performance metric. SP tries to optimize the response time by minimizing the service time variance at each disk, but is an offline algorithm that requires complete knowledge of the files. HP is the online version of SP, and attempts to reconcile minimizing the load variance across disks and minimizing service time variance at each disk by giving priority to minimizing service time variance when overall disk utilization is low. This is because when disk utilization is low, the load imbalance does not have a significant effect on the response time.

Not content with trading optimal load balancing for response time, and vice versa, Zhu et al. [23] propose two algorithms that optimize both response time and load balancing in parallel I/O systems. The first, Balanced Allocation with Sort (BAS), is an offline algorithm for static file assignment and needs full knowledge of the service times and access rates for all files. The second, Balanced Allocation with Sort for Batch (BASB), is an online algorithm, which uses information about the coming batch of files and the previously assigned files rather than needing complete knowledge like BAS. Files in a batch are sorted in descending order of their service times, with no correlation between service times in other batches, and then assigned to disks based on the average disk load.

Energy Savings Energy savings has been a concern with archival systems for some time: as systems add more storage capacity, the amount of power needed increases along with the cost of cooling. Since in some situations, disks and the power to cool them consume more energy than the rest of the system combined [2, 8], enabling more disks to remain idle is a big part of this research [4, 3, 16]. While HPC workloads are not ideal candidates for techniques which exploit idle disks [3], there has been work which indicates that significant idle periods exist in enterprise workloads [14].

Grouping data on disk to help improve response time is a well-researched topic [1, 11, 18]. Building off this work, Essary and Amer [7] present a theoretical framework that aims to reduce energy consumption as well as improve response time. This is accomplished using a predictive grouping algorithm, called OE ME which stands for *optimized expansion, maximized expectation*. OE ME uses first order successors to build groups, combining breadth first and depth first expansion strategies to create a balanced expansion to find successors.

The work done by Wildani et al. [21] extends this by providing a realistic prediction mechanism and semi-permanent groupings, which reduces the need for constant prediction. Wildani et al. show that by grouping data likely to be accessed within a short period of time together, the number of times disks have to spin up is reduced. This is supported by other research showing that data arrangement can have an impact on energy savings in single-disk systems [6, 17, 7].

3 Experimental Design

Having identified the objectives we are initially focusing on, we can begin to develop the framework necessary for JACK, our approach to multi-objective data allocation optimization. If you recall Equation 2, we need a set of objective functions in order to have a multi-objective optimization problem. These objective functions take the form of the metrics outlined in the following sections.

We have defined three different observable metrics, under specific constraints. It was imperative to us that the file (or object) management system be able to maintain control, so the first constraint was that the metrics had to be in units under the direct control of the management system. In addition, we required that the metrics be in units that can be observed or measured by the management system. Finally, the metric has to have some influence on the objective. The three metrics we've defined are similarity, popularity, and coverage.

3.1 Metric Definitions

Similarity influences system responsiveness, and depends on the probability that similar data will be accessed together. By accessed together, we mean accessed near each other within a certain time frame. While there are many ways to define similarity, such as comparing metadata, file or object content, or provenance, we are focusing on metadata for the purposes of this paper. Using other definitions of similarity will be done in future work.

Coverage influences energy savings, and measures the percentage of data on the disk that is "covered" by other disks in the system. This allows us to identify redundant devices: devices that are 100% covered by the rest of the system can be spun down with few repercussions. It is necessary to keep in mind that energy savings for reading is not the same as energy savings for writing. If you are trying to optimize for writing, the simplest energy savings technique is to place the data on the nearest active disk. When optimizing for reading, however, the data should be placed on the device with the most similar data. For the purposes of this paper, we are optimizing

Fields Used	Fields Unused
file permissions	unique identifier
file size (in bytes)	block size (in bytes)
user ID	path to file
group ID	
creation time	
modification time	

Table 1: Metadata fields in the Los Alamos National Laboratory data sets.

for reading; optimizing for writing, and optimizing for both reading and writing, is future work.

Popularity influences load balancing, and is our term for the more frequently used term "heat". It relates to the frequency or recency of access. Frequency of access refers to the number of times an object was accessed; recency of access refers to when the object was last accessed. A combination of recency and frequency is a better metric: this object was accessed x times in the past y days.

3.2 Metric Development

In order to develop metrics for each of these objectives, we worked with data sets from the Los Alamos National Laboratory (LANL). These data sets are static anonymized metadata snapshots, with a subset of metadata fields shown in Table 1. We focused on the anon-lnfs-fs4 data set, which is 163, 267 files over 12 nodes. For the purposes of these experiments, we looked at assigning files to nodes both sequentially and in a round robin manner. Sequential assignment simply assigned files in the order they appeared in the data set until a node filled; round robin distributed them across nodes.

3.2.1 Similarity

Similarity is measured using Shannon Entropy, as seen in Equation 3. What this tells us is that given a specific metadata attribute, we can calculate the probability of each value occurring on a node. A low entropy means that there are many similar values, a high entropy means that there are many diverse values. Since we're currently looking only at static metadata, entropy works well as a metric, as shown by Parker-Wood et al. [15].

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i) \tag{3}$$

As you can see in Figure 1, similarity is dependent on the assignment strategy. The reason the entropy values in both assignment strategies are fairly large is due to the nature of the LANL data set. Despite the large values, it is still possible to see the difference between the two strategies.

When data is assigned sequentially, files in a directory are assigned to the same node, resulting in many of the metadata fields having the same value and lower entropy. When data is assigned in a round robin fashion, files in a directory are spread across the nodes, resulting in the metadata fields having diverse values on each node and higher entropy. This supports the findings of Parker-Wood et al. [15] that entropy is a good metric for comparing the similarity of files on a node.

3.2.2 Coverage

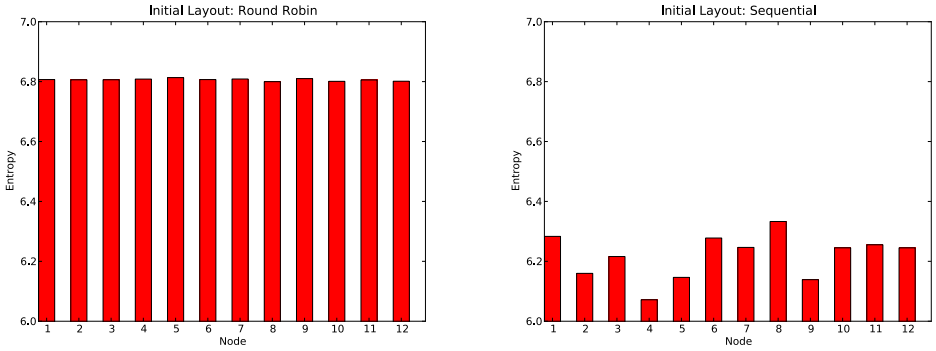
The original design of the coverage metric was to compare the number of different values (on average) for each metadata attribute on each node. The first problem that arose was that this technique resulted in as many values as devices, when a single value was needed. To resolve this, we took the average of the values. The problem with that, however, was that if two nodes covered each other very well, but had little in common with the rest of the system, the average was highly skewed.

Our next approach was to count the number of other nodes that covered the current node, but a simple count doesn't tell you which node is covered by which. Since the goal is to identify which nodes can be down with little to no loss of data, this was not a valid metric either. We needed a metric that rewarded skew, since that was a good indicator that there were two highly similar nodes, and we needed to know which node covered which. So we developed the current metric for coverage, which is the percent of the node that is covered by the rest of the system. Nodes that have the exact same percentage as the current node are the ones that cover it.

As with similarity, coverage is highly dependent upon the assignment strategy, as can be seen in Figure 2. When all the files in one folder are stored on a single node, the coverage is poor as seen in Figure 2(b). However, when the files of a folder are spread evenly across the nodes, coverage increases by close to 65% (Figure 2(a)). This drastic difference, due to the assignment strategy, mimics what we would expect, and supports our coverage metric.

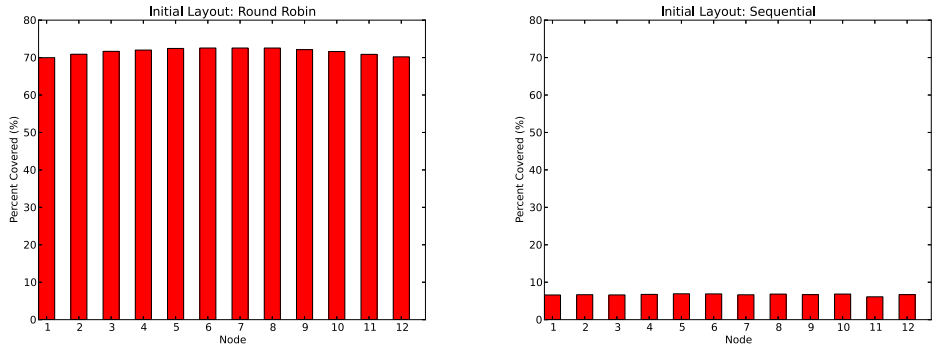
3.2.3 Popularity

Originally, we intended to look only at recency, since we were using static metadata and had no measure of frequency. However, since using both recency and frequency is a better metric, we were going to try to estimate popularity by assuming that a more recent access meant more frequent access. However, having developed two solid metrics, we decided to instead see if we could use



(a) Round Robin Assignment (b) Sequential Assignment

Figure 1: Similarity: Measured by entropy. Y-axis begins at 6 to show variation.



(a) Round Robin Assignment (b) Sequential Assignment

Figure 2: Coverage: Measured by the percent of node covered by other nodes.

the definition for "heat" [5, 12, 22, 23].

What we developed was an estimate for popularity, based on the definition of heat seen in Equation 4 where you're looking at the access rate and the expected service time. Disk accesses to a file can be modeled as a Poisson process with mean access rate of λ. Service time of a file is assumed to be fixed as s_i.

$$h_i = \lambda_i \times s_i \tag{4}$$

We started by looking at the access rate λ_i, which is the popularity weight times the aggregate access rate. The aggregate access rate can be seen in Equation 5, where N is the number of files. We found that the aggregate access rate was fixed between 100 [5] and 200 [22], and varied from 20 to 240 or 1000 [22, 23]. For these experiments we present results for a rate of 200. Evaluation of varied rates is not considered for this work.

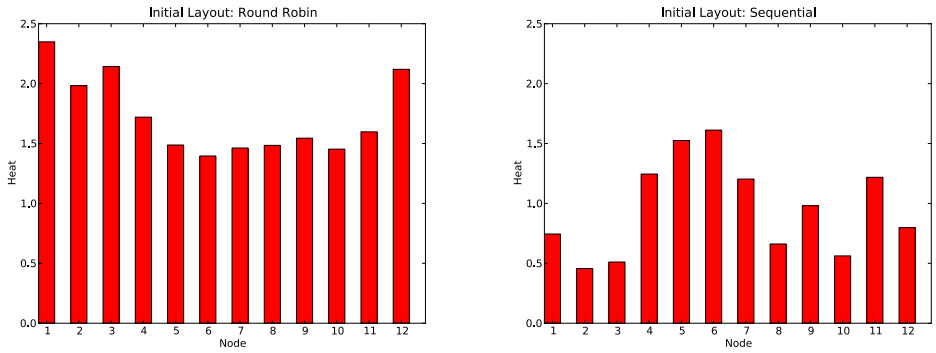
$$\sum_{i=1}^{N} \lambda_i \tag{5}$$

The other half of the access rate is the popularity weight, which gives us an idea of the frequency of requests for that file. Equation 6 shows us a definition for the popularity rate of a file, p_i, given a Zipf-like file distribution. Here, rank refers to the rank of the file (starting at 1), and $c = \frac{1}{H_N^{1-\theta}}$. The denominator of c is the Nth harmonic number of order $1-\theta$, which means that c can also be written as seen in Equation 7. A common theme here is $1-\theta$, but we have no definition for θ.

$$p_i = \frac{c}{\text{rank}_i^{1-\theta}} \tag{6}$$

$$c = \frac{1}{\sum_{k=1}^{N} \frac{1}{k^{1-\theta}}} \tag{7}$$

Most of the current work assumes that the file access request rate distribution is Zipfian with a skew of $\theta = \frac{\log \text{A}/100}{\log \text{B}/100}$, where A% of accesses are directed at B% of files. Given the Zipfian nature, generally the parameters are set as A$=$ 70 and B$=$ 30. However, since we



(a) Round Robin Assignment

(b) Sequential Assignment

Figure 3: Popularity: Measured by the average heat of files on each node.

have a static data set, we don't actually know our access request rate distribution. That being said, the denominator of c is incredibly similar to the denominator of the Zipf distribution (Equation 8). Here, s is the value of the exponent characterizing the distribution, N is the number of elements, and k is the rank.

$$\frac{\frac{1}{k^s}}{\sum_{n=1}^{N} \frac{1}{n^s}} \tag{8}$$

The value of the exponent characterizing the distribution can be calculated, as s is the slope of the log-log graph of the data set. However, calculating s still requires knowing the file access request rates. So instead we make the common assumption that file sizes are inversely correlated with file access request rates. This allows us to say that the file size distribution is an inverse Zipf distribution with the same skew of $theta$. Since we have the file size distribution, as long as it follows an inverse Zipf distribution, the file access request rate can be calculated. We found that our file size distribution does indeed follow an inverse Zipf distribution, allowing us to calculate file access request rates.

However, if you recall Equation 4, the file access request rate is only one half of the equation. The other half is service time, which is comprised of seek time, rotational time, and transfer time. Seek time and rotational time are negligible compared to the transfer time, which is heavily influenced by the size of the file. We use an estimated service time metric equivalent to the size of the file in Mbytes. Thus we are able to calculate popularity based on a definition of "heat" used previously, using the static metadata available to us.

Since file popularity does not change regardless of where the file is placed, it is calculated before placement. Since we are dealing with simply static metadata to determine popularity and are using the size of the file in megabytes, the resulting values are extremely small. As a result, once the average heat of the files on each node is calculated, we have multiplied by 10^{13} for readability in Figure 3. As you can see, the results consistently vary depending on the assignment strategy as with our previous two metrics.

4 Conclusions and Future Work

We have introduced the concept of addressing the data allocation problem as a multi-objective optimization problem, and explained how multi-objective optimizations give us the flexibility to find solutions that are applicable to the system and its users. We identified load balancing, system responsiveness, and energy savings as the objectives we are initially addressing, and defined a set of observable metrics to measure these objectives: popularity, similarity, and coverage. These metrics provide us with a foundation to construct JACK, a data allocation strategy phrased as a multi-objective optimization problem. We intend to explore the usefulness of these metrics in both full optimization problems as well as in approximation algorithms.

It is also a goal of the authors that the metrics described in this paper will be adopted by others needing a way to measure these objectives. When designing these metrics, we purposefully used only elements controlled by the file or object management system. This was to ensure that they would be useful regardless of the amount of information known about the underlying devices and components of the system.

References

[1] AMER, A., LONG, D., PÂRIS, J., AND BURNS, R. File access prediction with adjustable accuracy. *Proceedings of the International Performance Conference on Computers and Communication (IPCCC '02)* (2002).

[2] BARROSO, L. A., AND HOLZLE, U. The case for energy-proportional computing. *IEEE Computer 40*, December (2007), 33–37.

[3] CARRERA, E. V., PINHEIRO, E., AND BIANCHINI, R. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing* (New York, NY, USA, 2003), ICS '03, ACM, pp. 86–97.

[4] COLARELLI, D., AND GRUNWALD, D. Massive arrays of idle disks for storage archives. In *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing* (Los Alamitos, CA, USA, 2002), Supercomputing '02, IEEE Computer Society Press, pp. 1–11.

[5] DONG, B., LI, X., XIAO, L., AND RUAN, L. A file assignment strategy for parallel i/o system with minimum i/o contention probability. In *Grid and Distributed Computing*, T.-h. Kim, H. Adeli, H.-s. Cho, O. Gervasi, S. S. Yau, B.-H. Kang, and J. G. Villalba, Eds., vol. 261 of *Communications in Computer and Information Science*. Springer Berlin Heidelberg, 2011, pp. 445–454.

[6] DOUGLIS, F., KRISHNAN, P., AND MARSH, B. Thwarting the power-hungry disk. In *Proceedings of the USENIX Winter 1994 Technical Conference on USENIX Winter 1994 Technical Conference* (Berkeley, CA, USA, 1994), WTEC'94, USENIX Association, pp. 23–23.

[7] ESSARY, D., AND AMER, A. Predictive data grouping: Defining the bounds of energy and latency reduction through predictive data grouping and replication. *Transactions on Storage 4*, 1 (May 2008), 2:1–2:23.

[8] GANESH, L., WEATHERSPOON, H., BALAKRISHNAN, M., AND BIRMAN, K. Optimizing power consumption in large scale storage systems. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems* (Berkeley, CA, USA, 2007), HOTOS'07, USENIX Association, pp. 9:1–9:6.

[9] GRAHAM, R. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics 17*, 2 (1969), 416–429.

[10] KENYON, C. Best-fit bin-packing with random order. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 1996), SODA '96, Society for Industrial and Applied Mathematics, pp. 359–364.

[11] KROEGER, T., AND LONG, D. D. E. Design and implementation of a predictive file prefetching algorithm. *Proceedings of the 2001 Annual USENIX Technical Conference* (January 2001), 105118.

[12] LEE, L.-W., SCHEUERMANN, P., AND VINGRALEK, R. File assignment in parallel i/o systems with minimal variance of service time. *IEEE Transactions on Computers 49*, 2 (February 2000), 127–140.

[13] MA, Y.-C., CHIU, J.-C., CHEN, T.-F., AND CHUNG, C.-P. Variable-size data item placement for load and storage balancing. *Journal of Systems and Software 66*, 2 (May 2003), 157–166.

[14] NARAYANAN, D., DONNELLY, A., AND ROWSTRON, A. Write off-loading: Practical power management for enterprise storage. *ACM Transactions on Storage (TOS) 4*, 3 (November 2008), 10:1–10:23.

[15] PARKER-WOOD, A., STRONG, C., MILLER, E., AND LONG, D. Security aware partitioning for efficient file system search. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (May 2010), pp. 1–14.

[16] PINHEIRO, E., AND BIANCHINI, R. Energy conservation techniques for disk array-based servers. In *Proceedings of the 18th Annual International Conference on Supercomputing* (New York, NY, USA, 2004), ICS '04, ACM, pp. 68–78.

[17] RYBCZYNSKI, J. P., LONG, D. D. E., AND AMER, A. Adapting predictions and workloads for power management. In *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation* (Washington, DC, USA, 2006), MASCOTS '06, IEEE Computer Society, pp. 3–12.

[18] SELTZER, M., BOSTIC, K., MCKUSICK, M. K., AND STAELIN, C. An implementation of a log-structured file system for unix. In *Proceedings of the USENIX Winter 1993 Conference* (Berkeley, CA, USA, 1993), USENIX'93, USENIX Association, pp. 3–3.

[19] WAH, B. File placement on distributed computer systems. *Computer 17* (1984), 23–32.

[20] WEIL, S. A., BRANDT, S. A., MILLER, E. L., AND MALTZAHN, C. Crush: controlled, scalable, decentralized placement of replicated data. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), SC '06, ACM.

[21] WILDANI, A., AND MILLER, E. Semantic data placement for power management in archival storage. In *Proceedings of the 5th Annual Petascale Data Storage Workshop (PDSW10)* (November 2010), pp. 1–5.

[22] XIE, T., AND SUN, Y. A file assignment strategy independent of workload characteristic assumptions. *ACM Transactions on Storage (TOS) 5*, 3 (November 2009), 10:1–10:24.

[23] ZHU, Y., YU, Y., WANG, W. Y., TAN, S. S., AND LOW, T. C. A balanced allocation strategy for file assignment in parallel i/o systems. In *Proceedings of the 2010 IEEE Fifth International Conference on Networking, Architecture, and Storage* (Washington, DC, USA, 2010), NAS '10, IEEE Computer Society, pp. 257–266.