SEA: A Striping-Based Energy-Aware Strategy for Data Placement in RAID-Structured Storage Systems

Tao Xie, *Member*, *IEEE*

Abstract—Many real-world applications need to frequently access data stored on large-scale parallel disk storage systems. On one hand, prompt responses to access requests are essential for these applications. On the other hand, however, with an explosive increase of data volume and the emerging of faster disks with higher power requirements, energy consumption of disk-based storage systems has become a salient issue. To achieve energy-conservation and prompt responses simultaneously, in this paper we propose a novel energy-aware strategy, called striping-based energy-aware (SEA), which can be integrated into data placement in RAID-structured storage systems to noticeably save energy while providing quick responses. Next, to illustrate the effectiveness of SEA, we implement two SEA-powered striping-based data placement algorithms, SEA0 and SEA5, by incorporating the SEA strategy into RAID-0 and RAID-5, respectively. Extensive experimental results demonstrate that compared with traditional non-stripping data placement algorithms, our algorithms significantly improve performance and save energy. Further, compared with an existing stripping-based data placement scheme, the two SEA-powered strategies noticeably reduce energy consumption with only a little performance degradation.

Index Terms—Allocation strategies, energy-aware systems, file organization, secondary storage.

1 Introduction

Many real-world applications intensively read data stored in large-scale parallel disk storage systems like Redundant Arrays of Inexpensive Disks (RAID) [5]. To guarantee the quality of service demanded by users, prompt responses to read requests are essential for these applications. For example, a Video-on-Demand (VOD) server has to quickly respond access requests from multiple users so as to provide them with continuous glitch-free video [13], [35]. Similarly, a data-intensive Web server application that publishes significant amounts of data stored in a back-end database must answer users' inquiries instantly before they lose patience [4], [27]. It is obvious that the performance of these read-intensive applications largely depends on the performance of underlying parallel disk storage systems. More precisely, reducing the mean response time of parallel disk storage systems is a must for these applications.

There is a wide variety of ways of reducing the mean response time or improving the system throughput for parallel I/O systems [1], [13], [19], [24], [36], [38]. Data placement, or file assignment, allocating of all of the data onto a disk array before they are accessed, is one such avenue that can significantly affect the overall performance of a parallel I/O system [1], [24], [36]. In order to fully exploit the capacities of a parallel disk storage system, data placement algorithms for parallel disk systems have been extensively investigated in the literature [1], [3], [8], [11], [13], [24], [29], [36]. Generally, these algorithms place data onto a parallel disk array so that a special cost function or performance metrics can be optimized. While common cost functions include communication costs, storage costs, and queuing costs, popular performance metrics are mean response time and overall system throughput [12]. It is well known that finding the optimal solution for a cost function or a performance metric in the context of data placement on multiple disks is an NP-complete problem [12]. Thus, heuristic algorithms became practical solutions.

Although performance objectives such as the mean response time are always the subjects of the research on parallel disk storage systems, energy consumption of disk-based storage systems has become a salient issue that not only raises the costs but also inversely affects our environment [33], [38]. According to a recent industry report [31], storage devices contribute to around 27 percent of the total energy consumed by a data center. This problem will become much more severe with an explosive increase in data volume and the emergence of faster disks with higher power requirements. Therefore, energy conservation and prompt response need to be achieved simultaneously through intelligent data placement. Unfortunately, traditional data placement algorithms such as Greedy [15], Sort Partition (SP) [24], Hybrid Partition (HP) [24], and PVFS [23] for parallel disk systems only pursue minimized mean response times and normally ignore energy conservation.

In this paper, we address the problem of energy-saving yet quick-response data placement in a parallel disk storage system where data accesses exhibit Poisson arrival rates and

• *The author is with the Department of Computer Science, San Diego State University, 5500 Campanile Drive, GMCS 544, San Diego, CA 92182. E-mail: xie@cs.sdsu.edu.*

Manuscript received 6 Apr. 2007; revised 23 Oct. 2007; accepted 24 Jan. 2008; published online 1 Feb. 2008.
Recommended for acceptance by J. Antonio.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2007-04-0121.
Digital Object Identifier no. 10.1109/TC.2008.27.

fixed service times. Each data can be viewed as a file, which will be assigned onto an array of disks in a striping manner. We propose a novel energy-aware strategy, called *s*triping-based *e*nergy-*a*ware (SEA), which can be integrated into data placement for RAID-structured storage systems to optimize the mean response time and the overall energy consumption simultaneously. The basic idea of SEA is to place popular data onto a subset of the disks in the array and assign unpopular data onto the rest of disks. The rationale behind this idea is that the distribution of Web page requests generally follows a Zipf-like distribution [24], where the relative probability of a request for the ith most popular page is proportional to $1/i^{\alpha}$, with α typically varying between 0 and 1 [2], [27]. Based on this workload characteristic, we first divide all data into two categories—popular and unpopular—based on their popularity weights [27]. Several previous studies [10], [14] claimed that the request frequency and the file size are inversely correlated, i.e., the most popular files are typically small in size, while the large files are relatively unpopular. Next, we separate disks in a disk array into two zones: the *hot disk zone* and the *cold disk zone*. Disks in the hot disk zone are called *hot disks* with popular data, whereas disks in the cold disk zone are named *cold disks* with unpopular data. The ratio between the hot disk number and the cold disk number in a disk array is decided by the load percentages of popular data and unpopular data in the whole data set. As a result, the overall load balancing between two zones can be achieved, which improves the interrequest parallelism. Furthermore, we employ multispeed disks in the disk array to save energy. Specifically, hot disks are always running in a higher speed mode with more energy consumption, while cold disks are continuously operating in a lower speed mode with less energy dissipation. Although real multi-speed (more than two speeds) hard disks are not widely available in the market yet [28], a few simple variations of multispeed disks, such as a two-speed Hitachi Deskstar 7K400 hard drive, have recently been produced [18]. In addition, we believe that real multispeed disks will be manufactured in the not-so-distant future because they provide more opportunities for storage systems to adapt to a wide spectrum of workloads. For simplicity, in this study, we only utilize two-speed hard disks. The most significant difference between the use of multispeed disks in our study and in traditional energy-saving techniques [4], [17], [30], [38] is that, in our study, once a disk was configured as a hot disk or a cold disk, its operation characteristics such as transfer (read) speed and energy consumption rate were fixed and it could not be dynamically switched to the other mode during the process of serving requests. The advantages of statically setting the operation mode for multispeed disks are obvious. First, mode transition is very expensive in both energy consumption and response times because it incurs a high overhead in the form of transition time and transition energy [30]. Second, frequent mode transitions adversely affect disks' reliability and their lifetime [13], [38] Finally, to provide quick responses and fault tolerance, we combine SEA with RAID structures so that each file is partitioned into multiple same size stripes, which are then allocated across an array of disks. This way, all disks in the same zone can simultaneously serve a request which targets on a file allocated in the zone. The consequence is that the response time for the request can be dramatically decreased due to an enhanced intrarequest parallelism. For example, we integrate SEA with RAID-5 (striping with parity) to create SEA5, a RAID-based energy-aware data placement algorithm which can generate data placement schemes that not only save energy and provide prompt responses but also improve fault tolerance by employing the parity data. Similarly, SEA0, a peer of SEA5, was generated by combining SEA with RAID-0 (stripping without parity).

Extensive experimental results demonstrate that, compared with three conventional nonpartitioned data placement algorithms, namely, Greedy [15], SP [24], and HP [24], SEA0 and SEA5 reduce the mean response time, on average, at least 45.8 percent and 39.3 percent while saving energy, on average, not less than 9.8 percent (96,657.1 J) and 7.9 percent (77,771.6 J), respectively. Compared with a well-known stripping-based file assignment scheme, PVFS [23], SEA0 and SEA5 save energy by up to 36.2 percent (376,063 J) and 38.6 percent (352,567 J) while increasing the mean response time, on average, by 0.018 and 0.024 s, respectively. The rest of this paper is organized as follows: In Section 2, we discuss related work and motivation. In Section 3, we build the system model and energy consumption model. Section 4 presents the SEA strategy and introduces four existing algorithms. In addition, the time complexity of SEA0 is proven in Section 4. In Section 5, we evaluate the performance of our algorithms based on synthetic benchmarks. Section 6 concludes this paper with summary and future directions.

2 Related Work and Motivation

Very recently, energy saving for parallel disk storage systems has begun to draw much attention from the research community [4], [17], [19], [30], [32], [39]. Results from this research have demonstrated that the average idle slot between disk access requests in high-end computing workload is too small to justify the cost caused by disk spin-up or spin-down [4], [17]. Thus, traditional energy-aware disk scheduling algorithms that utilize disk idle times are not feasible in I/O-intensive applications. Consequently, some other means of saving energy for parallel disk storage systems running I/O-intensive applications must be discovered. To this end, several pioneer researchers proposed a diversity of techniques to save energy for parallel disk storage systems under server workloads. A multispeed parallel disk system that can modulate disk speed dynamically was proposed by Gurumurthi et al. [17]. In [19], data replication was used to dynamically place copies of data in free blocks according to the disk access patterns. Data prefetching was employed by Son et al. to save disk energy by making it energy aware. Zhu et al. devised an offline energy-aware cache replacement algorithm which minimizes the underlying disk energy consumption [38].

Comparing with the energy-efficient techniques for the parallel disk storage systems mentioned above, data placement shows its unique advantages. First, to save disk energy, it does not need to modify applications. Next, no extra hardware such as cache is necessary. Last, the

overhead of data placement strategy is relatively low and it is easy to implement. Attracted by these advantages, a research group led by Son has proposed an array of energy-aware disk layout algorithms very recently [21], [33], [35]. Based on our knowledge, their studies are the only results of energy-aware data placement for parallel disk storage systems reported in the literature so far. The central idea in [21] is to decide the most appropriate set of disks to store a given disk-resident array so that other disks can run at a low speed. A profile-driven approach for determining disk layouts of array data was proposed in [33] to minimize the energy consumption without increasing overall execution cycles excessively.

The aforementioned existing energy-aware disk layout algorithms, however, have some limitations. First, they are only dedicated to array-based scientific applications. Still, there are many other types of disk I/O-intensive applications, such as VOD [13], [33], Web applications [4], [25], and transaction processing [38], where energy conservation and quick response need to be realized simultaneously through data placement. Therefore, a more general energy-response efficiency data placement scheme that can be applied to a wide range of disk I/O-intensive applications is needed. Furthermore, to apply their algorithms, one has to modify the compiler to make it become aware of disk layout information. This requirement prevents them from being readily used because it incurs an extra burden for system software programmers. Besides, to better exploit existing energy-saving capabilities, their disk layout algorithms need to be combined with application code restructuring to increase the length of idle periods. This strategy demands modifications of an application's code and thus brings users additional overhead. As a result, the need for a new energy-response efficiency data placement strategy that bridges the gap between the existing algorithms and the open problems is greatly felt. In this paper, we are proposing a heuristic energy-aware strategy SEA which can be incorporated with RAID structures to generate energy-aware data placement algorithms like SEA0 and SEA5. Compared with existing data placement algorithms, SEA0 and SEA5 noticeably reduce energy consumption while providing quick responses for disk I/O-intensive applications running on parallel architectures. Our schemes are orthogonal to existing disk layout strategies. First, there is no need to modify any software using our methods. Second, our schemes are not dedicated to some particular applications. Thus, they are more general in the sense that they can be applied in multiple application domains. Without loss of generality, we assume that each data is viewed as an independent file and it is allocated in a striping manner across an array of disks. Communication delays between any pair of disks are ignored because they are identical and negligibly small [24]. In addition, disk access (read) to each file is modeled as a Poisson process with a mean access rate λi. The reason is fourfold. First, the Poisson distribution models random events with a constant average rate from a large number of independent sources, which generally matches request arrival patterns of data center workloads [38]. Second, many types of real-world requests of large installation systems naturally obey the Poisson distribution

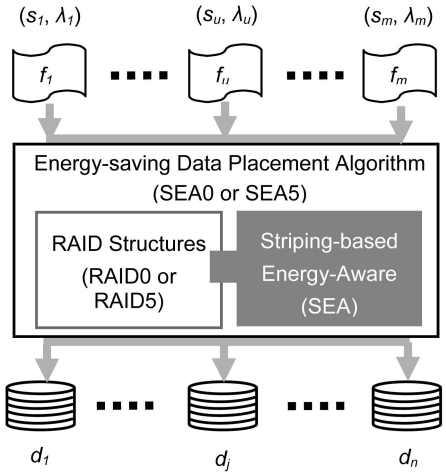


Fig. 1. Striping-based energy-aware data placement framework.

[9]. Third, the Poisson distribution has been well recognized as a standard statistic method in modeling disk access patterns [16] and has been widely used in modeling the performance of disk systems [16], [17], [24], [38]. Last, the three baseline algorithms, Greedy, SP, and HP, employed the Poisson distribution [24]. Also, we assume that each request accesses an entire file, which is a typical scenario for Web, proxy, ftp, and e-mail server workloads [24], [30]. Thus, for nonpartitioned file assignment, the service time svi for each file is fixed [24] and the positioning time (seek time plus rotation latency) is ignored. This assumption is valid because, when the basic unit of data access is an entire file on one disk, seek time and rotation latency are negligible in comparison with data transfer time. For stripping-based file assignment, however, the positioning time has to be considered as it is nontrivial compared with the data transfer time. Therefore, we include the positioning time when calculating the mean response times for SEA0, SEA5, and PVFS (see (5)).

3 Mathematical Models

We describe in this section mathematical models which were built to represent a disk array-based storage system, workload characteristics, and an energy consumption model.

3.1 System Model

Fig. 1 depicts a SEA data placement framework which integrates SEA with RAID structures to form energy-aware data placement algorithms like SEA0 and SEA5. Data placement algorithms such as Greedy, SP, HP, SEA0, and SEA5 allocate a set of data (hereafter files) onto a group of two-speed disks so that the mean response time can be minimized. The set of files is represented as $F = \{f_1, \ldots, f_u, f_v, \ldots, f_m\}$, which is further categorized into a popular file set $F_h = \{f_1, \ldots, f_h, \ldots, f_u\}$ and an unpopular file set $F_c = \{f_v, \ldots, f_c, \ldots, f_m\}$ ($F = F_h \cup F_c$ and $F_h \cap F_c = \oslash$). Since each file will be allocated onto a set of disks in a striping manner, let sp denote the size of a stripe (in megabytes) and it is assumed to be a constant in the system. A file f_i ($f_i \in F$) is modeled as a set of rational

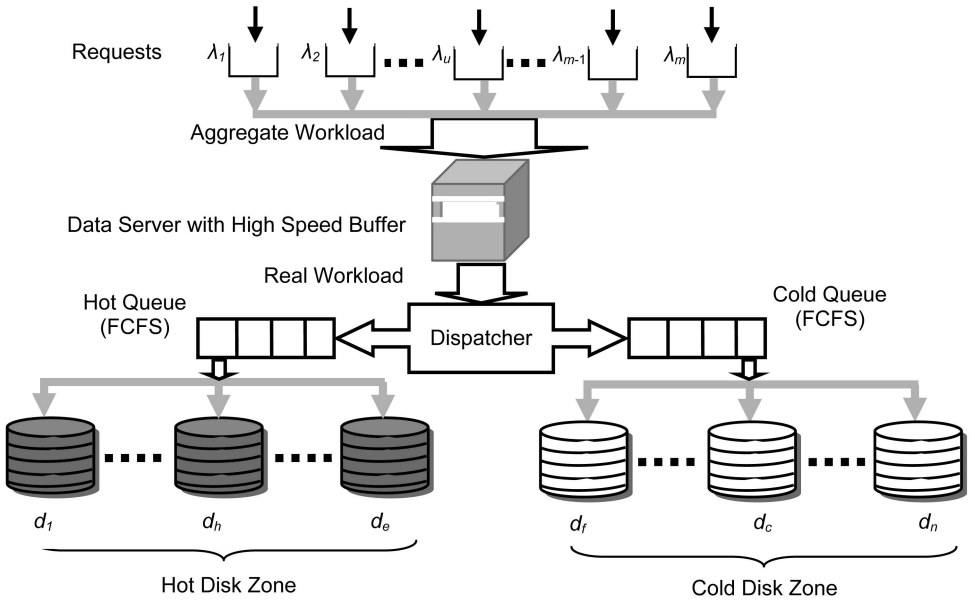


Fig. 2. Request scheduling architecture.

parameters, e.g., $f_i = (s_i, \lambda_i)$, where s_i and λ_i are the file's size in (in megabytes) and its access rate. In this paper, requests to a file f_u are modeled as a Poisson process with a mean access rate λ_i. Also, we assume that each access to file f_i is a sequential read of the entire file, which is a typical scenario in most file systems or WWW servers [22]. Besides, we assume that the distribution of file access requests is a Zipf-like distribution with a skew parameter $\theta = log_{\frac{A}{100}}/log_{\frac{B}{100}}$, where A percent of all accesses were directed to B percent of files [24] (see Fig. 4a). The value of $A : B$ is called *skew degree* (SD) in this paper and $\alpha = 1 - \theta$ (see Section 1 for α). In addition, the file access frequency and the file size are inversely correlated (see Fig. 4b). The number of popular files in F is defined as $|F_h| = (1 - \theta) * m$. Similarly, the number of unpopular files is $|F_c| = \theta * m$. Thus, the ratio between the number of popular files and the number of unpopular files in F is defined as η:

$$\eta = \frac{1-\theta}{\theta}. \tag{1}$$

In this study, each disk can be configured to run in either a high-speed mode or a low-speed mode. Note that, once a disk has been set to one mode, it cannot be dynamically switched to another mode during the process of serving requests. It can be transferred to another mode by the system administrator, however, after the current request set has been completed and before a new workload arrives if necessary. A disk array storage system consists of a linked group $D = \{d_1, \ldots, d_e, d_f, \ldots, d_n\}$ of n independent two-speed disk drives, which can be divided into a hot disk zone $D_h = \{d_1, \ldots, d_h, \ldots, d_e\}$ and a cold disk zone $D_c = \{d_f, \ldots, d_c, \ldots, d_n\}$ ($D = D_h \cup D_c$ and $D_h \cap D_c = \oslash$). Disks in the hot zone are all configured to their high-speed modes, which always run in the high transfer rate t^h (Mbytes/second) with the high active energy consumption rate p^h (J/Mbyte) and the high idle energy consumption rate i^h (J/s). Similarly, disks in the cold zone are set to their low-speed modes, which continuously operate in the low transfer rate t^l (Mbytes/s) with the low active energy consumption rate p^l (J/Mbyte) and the low idle energy consumption rate i^l (J/s). In the system, a hot disk d_h ($d_h \in D_h$) is modeled as a tuple $d_h = (c, t^h, p^h, i^h)$, where c is the capacity of d_h in gigabytes. Similarly, a cold disk d_c ($d_c \in D_c$) is modeled as a tuple $d_c = (c, t^l, p^l, i^l)$, where c is the capacity of d_c in gigabytes. Since we only consider homogeneous disks, all disks have the same capacity c. We assume that disks are always large enough to accommodate files to be assigned on them. Each popular file $f_h \in F_h$ is partitioned into multiple units, with the size of each unit being equal to sp. All units of f_h will be allocated across the hot disks in a RAID-0 (striping without parity) or a RAID-5 (striping with parity) fashion. Similarly, each unpopular file $f_c \in F_c$ is also partitioned into multiple-sized sp units and then allocated across the cold disks in a RAID-0 or a RAID-5 manner. Let sv_i be the expected service time of file f_i ($f_i \in F$). It can be computed by

$$sv_i = \begin{cases} s_i/t^h, & \text{if } f_i \text{ is popuplar} \\ s_i/t^l, & \text{if } f_i \text{ is unpopular.} \end{cases} \tag{2}$$

Since the combination of λ_i and sv_i accurately gives the load of f_i, we define the load h_i of f_i as follows [24]:

$$h_i = \lambda_i \cdot sv_i. \tag{3}$$

Fig. 2 depicts the subsequent file access requests scheduling process after the data placement process completes. All requests from multiple users form the aggregate workload, which is then directed to a high-performance data server with high-speed buffers. Since the majority portion of the aggregate workload can be successfully served by the data server, only a very small part of system workload will eventually turn out to be the real disk physical read accesses. This is because the miss rate of data buffers normally is lower than 10 percent in many real applications like OLTP [26]. Therefore, the real workload that the disk storage system indeed needs to serve is equal

to (the aggregate workload) $*$ (miss rate). The real workload is further divided by the *Request Dispatcher* into two groups: hot requests and cold requests. All hot requests are then directed into the *Hot Queue*, whereas all cold requests are delivered into the *Cold Queue.* Hot disks in Fig. 2 are represented as black disks and cold disks are denoted as the white disks. The ratio between the number of hot disks and the number of cold disks is defined as γ, which is decided by the ratio between the total load of popular files and the total load of unpopular files as follows:

$$\gamma = \frac{\sum_{i=1, f_i \in F_h}^{(1-\theta)*m} h_i}{\sum_{j=1, f_j \in F_c}^{\theta*m} h_j}. \tag{4}$$

Please note that we categorize a majority of files into the popular file group (see (1)). The reason is threefold. First, the sizes of popular files are much smaller than that of unpopular files (see Fig. 4b). Thus, requests on popular files are all small accesses in nature. Second, popular files are stored in the hot disk zone and unpopular files are stored in the cold disk zone. Hot disks are much faster than cold disks. This makes the expected service times of requests on popular files even shorter than that of requests on unpopular files (see (2)). Third, although popular files have a relatively large value of λ, the magnitude of the average λ of all popular files is much smaller than the magnitude of the average size of unpopular files. As a result, the total load of popular files is much smaller than that of unpopular files, which starves the hot disk zone. To balance the load between the two disk zones and thus obtain good performance in the mean response time, we enlarge the scope of popular file set so that more files can be in the hot disk zone.

In Fig. 2, we can see that the First-Come-First-Serve (FCFS) scheduling heuristic is used in both *Hot Queue* and *Cold Queue* to schedule arrival requests. Suppose there are x total requests in a request set which visits a file set that has been allocated on a disk array. The request set is designated as $R = \{r_1, \ldots, r_k, \ldots, r_x\}$, which can be separated into a hot request set $R_h = \{r_b, \ldots, r_h, \ldots, r_o\}$ and a cold request set $R_c = \{r_p, \ldots, r_c, \ldots, r_s\} (R = R_h \cup R_c, R_h \cap R_c = \oslash)$. Each request is modeled as $r_k = (fid_k, a_k)$, where fid_k is the file identifier targeted by the request and a_k is the request's arrival time. For each arrival request, the FCFS scheduler uses the allocation scheme X generated in data placement stage (see Fig. 1) to find the disks on which the target file of the request resides. In fact, the request workload is an m-class workload, with each class of requests having its fixed λ_i.

To obtain the response time of a request r_k, two important parameters, the earliest start time and the latest finish time of r_k, must be computed. We denote the earliest start time and the latest finish time of r_k by $est(r_k)$ and $lft(r_k)$, respectively. In what follows, we present derivations leading to the final expressions for these two parameters. Since each file is distributed across multiple disks in a striping manner, we need to compute the start time and the finish time for each stripe of the file that request r_k is targeting on. Suppose r_k is visiting file f_i, which was distributed on a disk set $\{d_a, \ldots, d_g, \ldots, d_w\}$ ($a \le g \le w$, $1 \le a, g, w \le e$, or $f \le a, g, w \le n$). The stripe set of f_i is represented as $\{s_i^1, \ldots, s_i^k, \ldots, s_i^z\}$, where $z = \lceil s_i/sp \rceil$. Also, a disk d_g has its own local queue Q_g in the set $\{Q_a, \ldots, Q_g, \ldots, Q_w\}$. There are three cases when r_k arrives on disk d_g. First, d_g is idle and Q_g is empty. Second, d_g is busy, but Q_g is empty. Third, d_g is busy and Q_g is not empty. Thus, the start time of transferring a strip s_i^k on disk d_g is

$$st_g^k(r_k) = \begin{cases} SK + RT + a_k, & \text{if } d_g \text{ is idle and } Q_g \text{ is empty,} \\ SK + RT + a_k + r_g, & \text{if } d_g \text{ is busy and } Q_g \text{ is empty,} \\ SK + RT + a_k + r_g & \\ + \sum_{r_p \in Q_g, a_p \le a_k} t_{fid_p}, & \text{otherwise,} \end{cases} \tag{5}$$

where SK denotes the average seek time, RT means the average rotation latency, r_g represents the remaining service time of a request currently running on d_g, and $\sum_{r_p \in Q_g, a_p \le a_k} t_{fid_p}$ is the overall service time of requests in Q_g whose arrival times are earlier than that of r_k. For simplicity, we assume that all disks are rotationally synchronized with each other and disks in the set $\{d_a, \ldots, d_g, \ldots, d_w\}$ start at the same cylinder [6]. This assumption is valid and has been used in studies in finding optimal strip unit size for RAID-based disk arrays [6], [7]. Therefore, the positioning times of each disk in set $\{d_a, \ldots, d_g, \ldots, d_w\}$ are identical and can be approximated by a constant value $(SK + RT)$. Consequently, $ft_g^k(r_k)$, the finish time of transferring the strip s_i^k on disk d_g, can be calculated by

$$ft_g^k(r_k) = st_g^k(r_k) + ts_i, \tag{6}$$

where ts_i is the service time of the stripe s_i^k on disk d_g, and it can be computed by

$$ts_i = \begin{cases} sp/t^h, & \text{if } d_g \text{ is hot} \\ sp/t^l, & \text{if } d_g \text{ is cold.} \end{cases} \tag{7}$$

As a result, the earliest start time of request r_k can be obtained by

$$est(r_k) = \min\left\{st_g^1(r_k), \ldots, st_g^k(r_k), \ldots, st_g^z(r_k)\right\}. \tag{8}$$

Consequently, the latest finish time of r_k is

$$lft(r_k) = \max\left\{ft_g^1(r_k), \ldots, ft_g^k(r_k), \ldots, ft_g^z(r_k)\right\}. \tag{9}$$

Hence, the service time of the request r_k is

$$str(r_k) = lft(r_k) - est(r_k). \tag{10}$$

The response time of the request r_k can be calculated by

$$rt(r_k) = lft(r_k) - a_k. \tag{11}$$

Therefore, the slowdown of r_k is

$$sd(r_k) = \frac{rt(r_k)}{str(r_k)}. \quad (12)$$

Thus, the mean response time of the request set R is expressed as follows:

$$mrt(R) = \sum_{k=1}^{x} rt(r_k) \Big/ x, \quad (13)$$

where x is the total number of requests in R. Similarly, the mean slowdown of the request set R can be obtained by the following expression:

$$msd(R) = \sum_{k=1}^{x} sd(r_k) \Big/ x. \quad (14)$$

3.2 Energy Consumption Model

For a request r_h in the hot request set R_h, assume that it accesses a popular file f_h in the popular file set F_h, which is allocated in the hot disk zone. The energy consumed by r_h can be written as follows:

$$e_h^{active} = s_h * p^h. \quad (15)$$

The aggregate service time of r_h provided by a set of hot disks where file f_h were allocated can be computed as follows:

$$at_h^{active} = \frac{s_h}{t^h}. \quad (16)$$

Note that the aggregate service time of r_h is defined as the summation of service times contributed by hot disks that accommodate a portion of f_h. It is equivalent to the service time of r_h when file f_h is stored entirely on a single hot disk. Thus, the energy consumption of the whole hot request set can be derived by

$$e_{R_h}^{active} = \sum_{h=1}^{|R_h|} e_h^{active}. \quad (17)$$

Similarly, the total aggregate service time imposed by the whole hot request set R_h in the hot disk zone is

$$at_{R_h}^{active} = \sum_{h=1}^{|R_h|} at_h^{active}. \quad (18)$$

In addition, we define rft_k as the finish time of request r_k. Then, we obtain the analytical formula for the energy consumed by the hot disks when they are idle:

$$e_{hot}^{idle} = i^h * \left(|D_h| * \max_{k=1}^{x}(rft_k) - at_{R_h}^{active} \right). \quad (19)$$

Hence, the total energy consumed by the hot disk zone can be computed by

$$\begin{aligned} e_{hot} &= e_{R_h}^{active} + e_{hot}^{idle} \\ &= \sum_{h=1}^{|R_h|} e_h^{active} + i^h * \left(|D_h| * \max_{k=1}^{x}(rft_k) - at_{R_h}^{active} \right). \end{aligned} \quad (20)$$

Input: A disk array *D* with *n* 2-speed disks, a collection of *m* files in the set *F*, and the skew parameter *θ*
Output: A file allocation scheme *X* (*m*, *k*), where *k* = $\max_{i=1}^{m}\left(\frac{s_i}{sp}\right)$

1. Use **Eq. 1** to compute the number of popular files and the number of unpopular files in *F*
2. Use **Eq. 4** to compute γ, the ratio between the number of hot disks and the number of cold disks
3. Hot disk number *HD* = $\frac{\gamma * n}{\gamma + 1}$, cold disk number *CD* = *n* – *HD*, d_h=1, d_c=1
4. Configure *HD* of *n* disks to high speed mode and set *CD* of *n* disks to low speed mode
5. **for** each popular file $f_p \in F_h$ **do**
6. p = 1;
7. **for** each stripe of f_p **do**
8. $X(f_p, p) = d_h$
9. $p = p + 1$
10. **if** $d_h == HD$
11. d_h = 1
12. **else**
13. $d_h = d_h + 1$
14. **end if**
15. **end for**
16. **end for**
17. **for** each unpopular file $f_u \in F_c$ **do**
18. u = 1;
19. **for** each stripe of f_u **do**
20. $X(f_u, u) = d_c$
21. $u = u + 1$
22. **if** $d_c == CD$
23. d_c = 1
24. **else**
25. $d_c = d_c + 1$
26. **end if**
27. **end for**
28. **end for**

Fig. 3. The SEA strategy with RAID-0 combined.

Similarly, the total energy consumed by the cold disk zone can be obtained by

$$\begin{aligned} e_{cold} &= e_{R_c}^{active} + e_{cold}^{idle} \\ &= \sum_{c=1}^{|R_c|} e_c^{active} + i^l * \left(|D_c| * \max_{k=1}^{x}(rft_k) - at_{Rc}^{active} \right). \end{aligned} \quad (21)$$

Therefore, the total energy consumption for the whole storage system is

$$e_{total} = e_{hot} + e_{cold}. \quad (22)$$

4 The SEA Strategy

In this section, we first present a detailed description of the SEA strategy, which is integrated with RAID-0. Then, we prove the time complexity of SEA0, as well as providing a brief introduction of the four baseline algorithms Greedy, SP, HP, and PVFS.

4.1 Strategy Description

Fig. 3 outlines the SEA strategy with RAID-0 combined. Note that the input F has been sorted in ascending order in terms of popularity before it is fed into SEA. In other words,

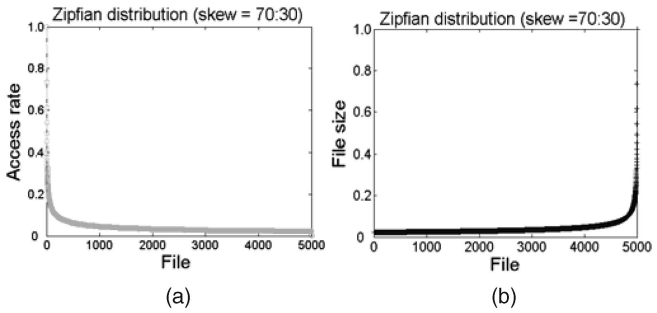


Fig. 4. (a) File access rate distribution. (b) File size distribution.

file f_1 is the most popular file with the smallest file size, whereas file f_m is the most unpopular one with the largest file size. First, SEA uses the skew parameter θ to derive the number of popular files and the number of unpopular files in F based on (1) (Step 1). Second, Step 2 calculates γ, the ratio between the number of hot disks and the number of cold disks, based on (4), which, in turn, results in the number of hot disks HD and the number of cold disks CD. Consequently, the HD of n disks are configured to their high-speed modes and the CD of n disks are set to their low-speed modes (Step 4). Next, SEA assigns all popular files onto the hot disk zone in a striping manner (Steps 5-16). Similarly, all unpopular files are allocated onto the cold disk zone in a striping fashion (Steps 17-28). Although the number of popular files defined by SEA is much larger than the number of unpopular files, SEA sets a minority portion of n disks as hot disks. The reason is threefold. First, hot disks are always running in their high-speed modes and therefore should accommodate more files than cold disks. Second, a relatively large number of cold disks provide more opportunities to save energy as cold disks operate in low-speed modes. Third, more cold disks lead to a higher intrarequest parallelism in the cold disk zone, which can compensate for the low running speed used by cold disks so that responses to cold requests can still be completed in a timely manner. Note that the SEA strategy described in Fig. 3 is actually the SEA0 data placement algorithm, which is a combination of SEA and RAID-0. To generate the SEA5 algorithm, one needs to slightly modify the algorithm so that the parity stripes are interleaved with the normal data stripes. The difference between SEA0 and SEA5, in terms of performance, mainly lies in the fact that SEA0 can effectively use one more disk than SEA5. However, in terms of fault tolerance and data reliability, SEA5 clearly outperforms SEA0 due to the use of parity data. In short, SEA0 offers low cost and maximum performance but provides no fault tolerance. Businesses use SEA0 mainly for applications requiring fast access to a large capacity of temporary disk storage such as video/audio postproduction, multimedia imaging, CAD, and data logging, where, in case of a disk failure, the data can be easily reloaded without impacting the business. On the contrary, SEA5 has the potential of being applied in server environments requiring fault tolerance. The RAID parity requires one disk drive per RAID set. Therefore, usable capacity for SEA5 will always be one disk drive fewer than the number of available disks for SEA0. Still, SEA5 can provide good read performance.

4.2 Time Complexity of SEA0

Before qualitatively comparing our scheme with the four existing algorithms, we demonstrate the worst-case time complexity of the SEA0 algorithm.

Theorem 1. *Given a parallel disk array* $D = \{d_1, \ldots, d_e, d_f, \ldots, d_n\}$ *of* n *independent two-speed disk drives and a collection of files represented by* $F = \{f_1, \ldots, f_u, f_v, \ldots, f_m\}$, *the worst-case time complexity of SEA0 is* $O((k+1)m)$, *where* m *is the number of files in* F, k *is the number of stripes of the largest file in* F, *and* $k = \max_{i=1}^{m}(\frac{s_i}{sp})$.

Proof. It takes $O(m)$ time to derive an appropriate value of γ based on (4) (see Step 2). Let k represent the number of stripes of the largest file in F. Therefore, $k = \max_{i=1}^{m}(\frac{s_i}{sp})$. The upper bound of time complexity for allocating a popular file f_p in the set F_h is k and we have $|F_h|$ number of popular files. Hence, the worst-case time complexity for allocating all popular files is $O(k|F_h|)$ (Steps 5-16). Similarly, the worst-case time complexity for allocating all unpopular files is $O(k|F_c|)$ (Steps 17-28). Since $|F_h| + |F_c| = m$, the worst-case time complexity for allocating all files in F is $O(km)$ (Steps 5-28). Other steps simply take $O(1)$ time. Thus, the worst-case time complexity of the SEA0 algorithm is $O(m) + O(km) = O((k+1)m)$. □

Theorem 1 indicates that the time complexity of the SEA0 algorithm is typically low. For example, in our experiments, the value of m is 5,000, the value of sp is 512 Kbytes, and k is in the range [864, 30,240], which should take less than thousands of microseconds to complete the SEA algorithm in modern processors. An implication of Theorem 1 is that SEA has the potential of being extended to be applied in real-world applications because of its low complexity.

4.3 The Four Baseline Algorithms

In Section 5, we will compare the performance of SEA0 and SEA5 against three traditional nonpartitioned file assignment algorithms, namely, Greedy [15], SP [24], and HP [24], and one stripping-based data placement scheme PVFS [23]. The purpose of this section is to briefly introduce the four baseline algorithms, which are well-known data placement algorithms whose goal is to minimize the mean response time. The average disk load ρ can be obtained by the following equation:

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{m} h_i. \tag{23}$$

Note that the Greedy, SP, and HP algorithms assign nonpartitioned files onto a disk array. In other words, each file must be allocated entirely onto one disk. In contrast, PVFS essentially employs RAID-0 structure to partition each file across the disk array with uniform stripe size. In addition, since all four baseline algorithms only pursue a minimized mean response time, all disks in the disk array

TABLE 1
Main Characteristics of a Two-Speed Disk

Description	Value	Description	Value
Disk model	Seagate Cheetah ST39205LC	Standard interface	SCSI
Storage capacity	9.17 GBytes	Rotational speed	10000 rpm
Avg. seek time (*ST*)	5.4 msecs	Avg. rotation time (*RT*)	3 msecs
Transfer rate in high mode	31 Mbytes/second	Transfer rate in low mode	9.3 Mbytes/second
Idle power at high mode	5.26 Joules/second	Idle power at low mode	2.17 Joules/second
Active energy at high mode (8-KB read)	61 mJoules	Active energy at low mode (8-KB read)	43 mJoules

are configured to hot disks with high speed. The four algorithms are briefly described as follows:

1. *Greedy.* This is the most common heuristic for multiple disks load balancing. It can operate in either the online mode or the offline mode. Here, we only consider its offline mode because SEA is an offline energy-aware file assignment strategy. It first calculates the mean load of all files and then assigns a consecutive set of files whose total load is equal to the mean load onto each disk. Its goal is to generate a file assignment scheme such that the mean response time of the parallel I/O system can be minimized. The time complexity of Greedy is $O(m + mn)$, where m is the number of files and n is the number of disks.
2. *SP (Sort Partitions).* It first computes the average disk utilization using (23). Next, it sorts all files into a list I in descending order of their service times. Finally, it allocates each disk d_j the next contiguous segment of I until its load $load_j$ reaches the maximum allowed threshold ρ. The remaining files (if any) after one round allocation will be assigned to the last disk d_n. It improves the performance of the Greedy algorithm by minimizing the variances of service times at each disk. Its time complexity is $O(m + mn + m\lg m)$, where m is the number of files and n is the number of disks.
3. *HP (Hybrid Partition).* In case files arrive in batches which can be sorted prior to their assignment, HP attempts to simultaneously minimize the load variance across all disks, as well as the service time variance at each disk. For each batch, HP assigns files to disks in distinct allocation intervals. The algorithm selects, for each allocation interval l, a different disk d_k as the allocation target. It chooses the disk with the smallest accumulated load. During one allocation interval, a number of files are allocated to the target disk d_k until its load reaches a given threshold T_k. The complexity of the HP algorithm is $O((b + 1)n\lg n + m)$, where m is the number of files, n is the number of disks, and b is the number of batches.
4. *PVFS (Parallel Virtual File System*). PVFS is an open source implementation of a parallel file system developed specifically for cluster-based parallel computers [23]. In order to provide high-performance access to data stored on the file system by many clients, PVFS spreads data out across multiple cluster nodes. Basically, it mimics a RAID-0 style stripping to distribute each file onto multiple I/O nodes in a round-robin fashion. Its worst-case time complexity is $O(km)$, where m is the number of files and k is the number of stripes of the largest file in F. To make the comparisons between PVFS and our SEA0 and SEA5 fair, the strip size for all three stripping-based algorithms is set to 512 Kbytes.

5 Performance Evaluation

This section presents the results of a comprehensive experimental study by comparing the proposed two SEA-powered energy-aware data placement algorithms, namely, SEA0 and SEA5, with the four traditional approaches, Greedy, SP, HP, and PVFS. To the best of our knowledge, the SP algorithm is arguably the most effective among existing static nonpartitioned data placement algorithms, in part because it can result in a minimal mean response time [24]. In this section, we first outline the performance metrics that we used and explore the parameter space by discussing six critical parameters. Next, we introduce a synthetic workload generator that we built to investigate the impact of the parameters that we discuss in Section 5.1. Finally, in Sections 5.3-5.7, we analyze experimental results under the workload assumption that file access frequency is inversely correlated with file size [10], [14]. Some preliminary results in Sections 5.3 were presented in [37].

5.1 Simulator and Parameter Space

We have developed an execution-driven simulator that models an array of two-speed disks. We adopt the same strategy used in [30] to derive corresponding low speed mode disk statistics from parameters of a conventional Cheetah disk. The main characteristics of the two-speed disk are shown in Table 1. The performance metrics by which we evaluate system performance include:

Mean response time. Average response time of all file access requests submitted to the simulated parallel disk storage system (see (13)).

Energy consumption. Energy (in joules) consumed by the disk systems during the process of serving the entire request set (see (22)).

Mean slowdown. Ratio between the average request turnaround time and the average request service time (see (14)).

Two categories of parameters directly influence the data placement algorithms that we investigate: workload characteristics and disk drive characteristics. Among the large number of parameters that specify a workload, we identified six key characteristics:

1. *Number of files.* Since the total number of files to be assigned onto a parallel disk array directly determines the disk array's load, we set it to 5,000 so that each disk can accommodate around 312 files in case there are 16 disk drives in the array. The number of files per disk is a realistic mimic of the real-world situation.
2. *Request rate.* Each file access represents a sequential read of the entire file. Hence, the service time of a file access request is proportional to the file's size. We assume that each file has a fixed request arrival rate λ_i and the arrival interval times are exponentially distributed. The aggregate arrival rate of the entire system is defined as $\sum_{i=1}^{5,000} \lambda_i$. The value of the aggregate arrival rate represents the intensity of the total access requests submitted to the disk array, where 5,000 files have been assigned across.
3. *File popularity weight.* File popularity weight relates to the frequency with which file requests arrive at the parallel disk array system. Since the frequency of file access usually exhibits a Zipf-like distribution, we assume that the distribution of file access requests is a Zipf-like distribution with a skew parameter $\theta = \log\frac{A}{100} / \log\frac{B}{100}$, where A percent of all accesses were directed to B percent of files [24]. Fig. 4a shows a Zipf-like distribution of file access rate on the 5,000 files, with $A : B = 70 : 30$, assuming that file f_1 is the most popular file and $f_{5,000}$ is the most unpopular one. In our simulations, we tested three values of θ, with *skew degree* $A : B$ changing from $60 : 40$ to $80 : 20$.
4. *File size distribution.* The distribution of file sizes and the distribution of access rates across the files were inversely correlated with the same skew parameter, θ, as shown in Fig. 4b. The parameter *file size base* is defined as the smallest file size in the whole file set.
5. *Coverage of the system.* The file system coverage is defined as the percentage of the entire file repository that is actually accessed by the request workload. We set the coverage of the system to 100 percent in our simulations, which means that all files in the parallel disk array system are accessed at least once.
6. γ. The ratio between the number of hot disks and the number of cold disks in the disk set is calculated by (4). In fact, γ is a critical parameter which directly influences the mean response time and energy consumption. On one hand, a large value of γ implies a longer mean response time with more energy consumption. On the other hand, a low value of γ indicates a shorter mean response time with less energy consumption. γ must be adjusted so that a good balance between quick response and energy saving can be achieved. Section 5.7 shows that a fine-tuned γ results in quick mean responses and low energy consumption.

Table 2 summarizes the configuration parameters of a simulated parallel disk array system used in our experiments and characteristics of the synthetic workload. All synthetic workloads used from Sections 5.3 to 5.7 were created by our trace generator. Although the number of disks, aggregate access rate, and size of files are synthetically generated, we examined the impacts of these important parameters on system performance by controlling the parameters.

TABLE 2
Characteristics of System Parameters

Parameter	Value (Fixed) – (Varied)
Number of files	(5000)
File request rate	Each file has a fixed Poisson request arrival rate λ_i
File popularity weight (Zipf-like)	The popularity weight of file f_i is $p_i = c/rank_i^{1-\theta}$, where $c = 1/H_N^{(1-\theta)}$.
File access distribution (Zipf-like)	Skew degree (A:B =70:30) – (60:40, 70:30, 80:20)
File size distribution (Inverse Zipf-like)	file size base (1) – (15, 20, 25, 30, 35, 40) Mbyte
File size distribution (Uniform)	(1,10) Mbyte
Coverage of the file system	(100%) – each file is at least accessed once
Number of batches (for HP)	(4) – each batch has 1250 files
Number of disks	(16) – (12, 14, 16, 18, 20, 22, 24, 26)
γ	(5:11) – (3:13, 4:12, 5:11, 6:10, 7:9, 8:8, 9:7, 10:6)
Aggregate access rate $\sum_{i=1}^{5000} \lambda_i$	(35) – (25~45) (1/second)
Simulation duration	(1000) seconds

5.2 Synthetic Workload Generator

To study the impact of the parameters that we discussed in Section 5.1, we built a workload generator that can produce file traces and request traces with user-specified request rate, popularity weight, file sizes, the number of files, coverage, skew degree, and the number of requests.

The workload generator takes the number of files (*FILE_NUM*), the aggregate access rate (*TOTAL_ACCESS_RATE*), and the skew degree (*SKEW_DEGREE*) as input parameters and then computes popularity weight for each file based on Zipf-like distribution with the skew parameter $\theta = \log\frac{A}{100} / \log\frac{B}{100}$. Suppose p_i is the popularity weight of file f_i. In a Zipf-like distribution, $p_i = c/rank_i^{1-\theta}$, where $rank_i$ is the rank of the file from 1 to the total number of files (FILE_NUM) and $c = 1/H_N^{(1-\theta)}$. $H_N^{(1-\theta)}$ is the Nth harmonic number of order $(1-\theta)$ and it is defined as $\sum_{k=1}^{N} \frac{1}{k^{(1-\theta)}}$. A file's popularity weight multiplied by the aggregate access rate is equal to its access rate. The same skew degree value was used to produce file sizes when the file size obeys a Zipf-like distribution (Sections 5.3-5.7). Therefore, each file has the following attributes: popularity weight, access rate, and file size in Zipf-like distribution. To generate a request trace, for each file, the workload generator first computes the number of requests targeted on the file during the simulation duration (1,000 s) according to each file's access rate. Since we set the *coverage of the file system* to 100 percent, each file has at least one

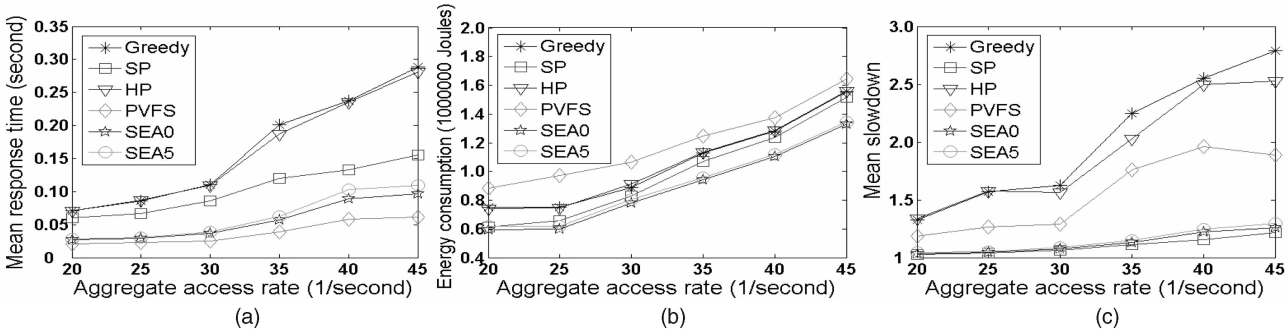


Fig. 5. Impact of aggregate access rate.

request. The interarrival times of access to file f_i were exponentially distributed with a fixed mean $1/\lambda_i$. Hence, for each file, the generator creates a request list. Next, the generator mixes all of the file's request list into one request queue and sorts it in ascending order in terms of arrival time. As a result, there are two features for each request: the identifier of the file that it targets on and its arrival time. The request trace was used to drive the simulated parallel disk array, with all files having been assigned on.

5.3 Impact of Aggregate Access Rate

The goal of this experiment is to compare the proposed SEA0 and SEA5 algorithms against the four well-known file assignment schemes and to understand the sensitivity of the six heuristics to the aggregate access rate in a parallel disk storage system, where an array of two-speed disk drives serve requests simultaneously. The aggregate access rate varies from 20 (1 J/s) to 45 (1 J/s). The file sizes were generated according to a Zipf-like distribution with skew degree $70:30$ and the *file size base* is set to 1 Mbyte.

Fig. 5 shows the simulation results for the six algorithms on a parallel disk array with 16 disk drives, where five of them are hot disks and 11 of them are cold disks. We observe in Fig. 5a that SEA0 and SEA5 consistently outperform the three nonpartitioned approaches in terms of the mean response time. This is because they employ a striping-based data placement scheme where intrarequest parallelism is very high. SP takes the third place in the mean response time metric, which is consistent with our expectation because it is one of the best existing nonpartitioned data placement heuristics, which is superior to Greedy and HP [24]. Compared with the SP algorithm, SEA0 and SEA5 can reduce the mean response time, on average, by 45.8 percent and 39.3 percent while saving energy, on average, by not less than 9.8 percent (96,657.1 J) and 7.9 percent (77,771.6 J), respectively (see Fig. 5b). PVFS [23] is in first place in terms of the mean response time. On average, it lowers the mean response time to 0.018 and 0.024 s when compared with SEA0 and SEA5, respectively. This is because it employs the same stripping manner that SEA0 does. More importantly, to pursue quick responses, it uses only high-speed disks in the simulated system. However, in terms of energy consumption, PVFS is the worst algorithm among the six approaches. The reason behind this is that PVFS uses the entire disk array to serve every single request and, thus, each disk has an interleaved high speed idle-high speed active reading mode sequence until the finish time of the last request in the whole request set R. In contrast, for the three traditional nonpartitioned algorithms, an individual disk stops consuming energy immediately after the disk finishes serving the last request targeting on it. In other words, the disk does not need to keep running until the finish time of the last request in the entire request set R. Thus, the three nonpartitioned algorithms consume less energy than PVFS does. Compared with PVFS, SEA0 and SEA5 save energy by up to 36.2 percent (376,063 J) and 38.6 percent (352,567 J), respectively. We argue that saving energy by more than 35 percent (i.e., more than 350,000 J per simulation duration time), at the price of a small performance degradation (i.e., less than 0.025 second), is absolutely worthwhile. An interesting observation is that, in terms of the mean slowdown, SP and SEA0 deliver a similar performance (Fig. 5c). The reason is that the average service time of each request is relatively higher in SP because no intrarequest parallelism exists. Although we only test a relatively light physical read workload (in the range [20, 45] 1 J/s), the actual system workload can be 10 times heavier (in the range [200, 450] 1 J/s) because of the very low miss rates (less than 10 percent) provided by the high-speed buffers on the data server (see Fig. 2). The implication is that both SEA0 and SEA5 can be applied in applications where the system workload is heavy. One example of such applications is Online Transaction Processing (OLTP).

5.4 Scalability

This experiment is intended to investigate the scalability of the six algorithms. We scale the number of disks in the system from 12 to 32. The aggregate access rate is configured to 35 (1 J/s). The skew degree is still set to $70:30$. Fig. 6 plots the performance of the six algorithms as functions of the number of disks. The results show that SEA0 and SEA5 exhibit good scalability.

Fig. 6 shows that all six algorithms deliver better performance in the mean response time and the mean slowdown when the number of disks increases. This is because each disk has few files to be assigned on when the system is scaled up. With more disks available, it is easy to understand that the total energy consumption will be increased (Fig. 6b). The SEA0 and SEA5 algorithms almost tie with PVFS in the mean response time when the system has more than 20 disks (Fig. 6a). Meanwhile, SEA0 and SEA5 can save more energy compared with all four baseline algorithms when the system has more disks (Fig. 6b). The implication of this observation is that SEA is suitable for a

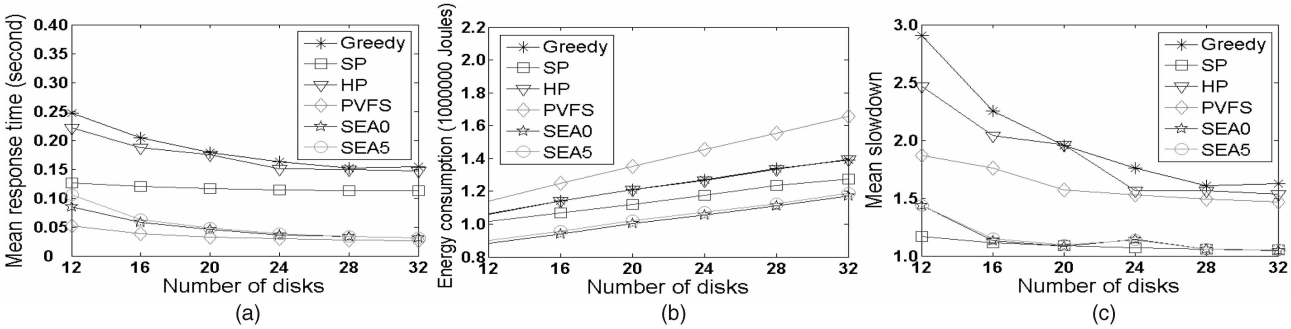


Fig. 6. Impact of the number of disks.

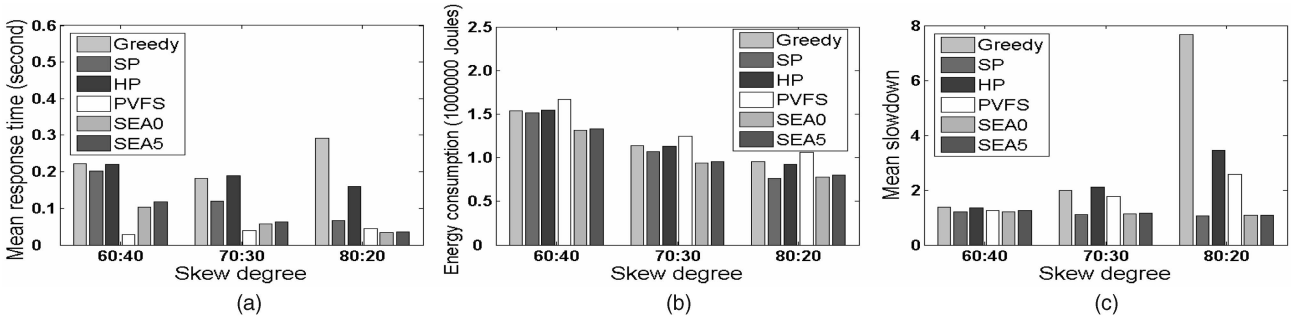


Fig. 7. Impact of skew degree.

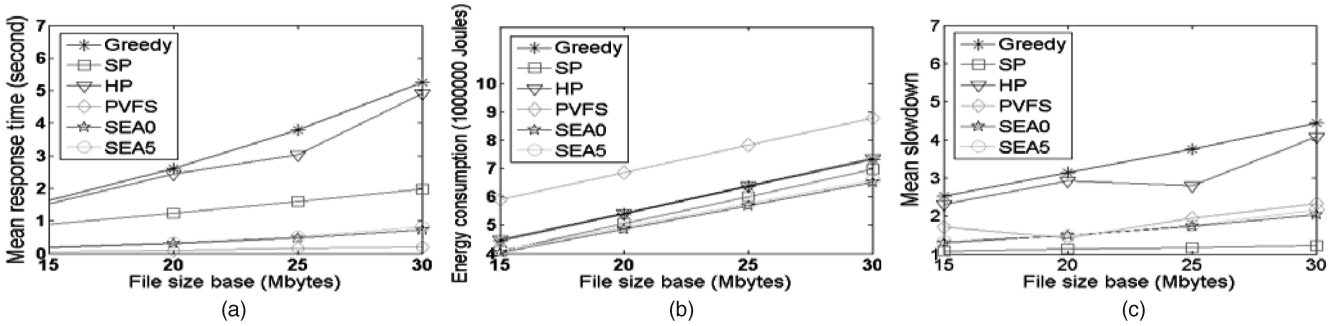


Fig. 8. Impact of file size.

parallel I/O system where the number of disks is adequate for a heavy workload. In addition, SEA0 and SEA5 obviously perform better in the mean slowdown compared with PVFS (Fig. 6c). The rationale behind this is that PVFS provides each request with a shorter service time due to a higher level intrarequest parallelism while offering almost the same response time. A shorter service time combined with a similar response time leads to a larger slowdown value (see (12)).

5.5 Sensitivity to Skew Parameter θ

To verify the performance impact of the skew parameter θ, we evaluate the performance as functions of skew degree. When the skew degree increases from $60:40$ to $80:20$, the improvement of our SEA algorithms in both the mean response time and energy consumption compared with the best nonpartitioned algorithm SP becomes less pronounced. This is because, when the skew degree becomes high, more requests will concentrate on a small portion of popular files with small sizes. In this case, the advantage of the intrarequest parallelism employed by SEA0 and SEA5 becomes less salient. In other words, SP can also quickly complete most of the requests, without using intrarequest parallelism in this scenario. We observe in Fig. 7 that SEA0 and SEA5 achieve the best balance between the mean response time improvement and energy saving when the skew degree is $70:30$.

5.6 Impact of File Size

In this section, we examine the performance impact of file size when the parameter *file size base* varies from 15 to 30 Mbytes. Note that *file size base* is the size of the smallest file in a file set F and the sizes of all other files can be generated based on the inverse Zipf-like distribution shown in Fig. 4b. Obviously, when the size of files is enlarged, the mean response time and energy consumption correspondingly increase (Figs. 8a and 8b). Another purpose of this experiment is to examine the maximal possible file size in the system when the mean response time is controlled within a realistic range, e.g., $3 \sim 5$ seconds, which is acceptable for some real-time systems [20]. When the *file size base* is set to 30 Mbytes, the size of files changes in the range [35, 13,360] Mbytes. Still, SEA0 and SEA5 perform much better than the three nonpartitioned algorithms in

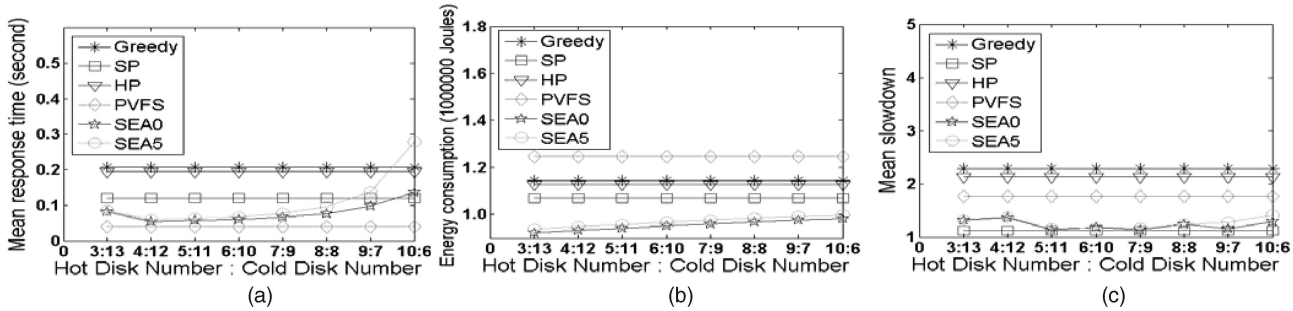


Fig. 9. Impact of γ.

both the mean response time and energy conservation. Although PVFS can deliver a shorter mean response time, it achieves this slim performance improvement, but at the cost of significant energy consumption (Fig. 8b). The results from this test demonstrate that SEA0 and SEA5 can also be applied in *decision support systems* (DSSs).

5.7 Impact of γ

To measure the impacts of parameter γ, the ratio between the number of hot disks, and the number of cold disks in a disk array, we evaluate the six algorithms while changing γ from $3:13$ to $10:6$. The performance patterns plotted in Fig. 9 are similar to those reported in previous sections except that the four baseline algorithms are kept constant in all three figures. This is because there is no parameter γ in these three algorithms and they all set disks to their high-speed modes. When γ is configured to $10:6$, both SEA0 and SEA5 are beaten by SP and PVFS in terms of the mean response time. The rationale behind this phenomenon is that the limited number of cold disks makes the mean response times for large but unpopular files dramatically enlarged. Meanwhile, the energy saving becomes less significant when the number of hot disks dominates the entire disk array. The best choice for γ is $4:12$, where SEA0 and SEA5 can achieve their lowest mean response times while noticeably saving energy. The best value of γ $(4:12)$ obtained from our experiments above is very close to the value calculated by (4) $(5:11)$, which verifies its effectiveness.

One important observation from Fig. 9 is that only when the number of hot disks is not larger than the number of cold disks can our algorithms achieve a better mean response time than the three nonpartitioned baseline algorithms. In other words, a better performance is achieved by using a minority of disks to serve a majority of files, which is counterintuitive at first glance. Nevertheless, the disk partition method (see (4)) is correct because it assigns disks to the two disk zones based on the load percentage. Note that, although the number of popular files is in the majority, the total load of the popular files is not necessarily dominant as the total load of a file set is decided by the average file size and the disk transfer rate as well (see Section 3.1).

6 Conclusions

In this paper, we have addressed the issue of allocating striped files onto a RAID-structured disk storage system where the file access requests exhibit Poisson arrival rates and fixed service times. To provide quick response and energy conservation simultaneously, a new energy-saving strategy, called SEA, is developed to generate optimized file allocation schemes. SEA0 and SEA5, two SEA-powered RAID-based data placement algorithms, are also implemented to evaluate the effectiveness and practicality of SEA. Comprehensive experimental results show that both SEA0 and SEA5 consistently improve the performance of parallel disk storage systems in terms of the mean response time and save energy over three well-known nonpartitioned data placement algorithms. Compared with SP, one of the best existing nonpartitioned file assignment algorithms, SEA0 and SEA5 decrease the mean response time by averages of 45.8 percent and 39.3 percent while saving energy, on average, by not less than 9.8 percent (96,657.1 J) and 7.9 percent (77,771.6 J), respectively. Compared with a widely used stripping-based file assignment scheme PVFS [23], SEA0 and SEA5 save energy by up to 36.2 percent (376,063 J) and 38.6 percent (352,567 J) while only increasing the mean response time, on average, by 0.018 and 0.024 s, respectively. More importantly, SEA5 also offers fault tolerance by utilizing the parity data, whereas all four existing algorithms provide no fault tolerance at all. Multispeed disks have been adopted by some traditional energy-saving schemes like Multispeed [4], DRPM [13], PDC [30], and Hibernator [38]. Typically, these techniques dynamically switch disks from one-speed mode to another to better serve disk access requests and save energy. There are two inherent drawbacks of these approaches. First, disk speed mode transitions bring extra overhead in terms of transition time and transition energy [30], which is against their original goals. Second, frequent disk speed mode transitions are detrimental to the lifetime of hard disks [4]. SEA0 and SEA5 avoid these two shortcomings by statically configuring all disks to one of the multiple modes before they start to serve requests based on workload characteristics. Furthermore, there is no speed mode transition during the process of serving the requests. In case the workload pattern changes, the system administrator can reconfigure all disks periodically so that the whole disk array can serve it better. In summary, compared with traditional nonpartitioned file assignment algorithms, the SEA strategy realizes energy saving but not at the cost of performance degradation and disk reliability. Rather, it delivers much shorter mean response times. Compared with existing stripping-based non-energy-aware data placement schemes like PVFS, the SEA strategy substantially

saves energy while only marginally degrades system performance. Besides, the SEA5 algorithm can provide fault tolerance because of the RAID structures upon which it relies.

Future studies in this research can be performed in the following directions: First, we will extend our scheme to a fully dynamic environment, where file access characteristics are not known in advance and may vary over time. As a result, a dynamic energy-saving data placement strategy that can predict request access patterns based on workload history is mandatory. Fortunately, data center workload tends to be highly self-similar, which makes future workload prediction possible. Two feasible solutions for dynamically redistributing files across a disk array to adapt to access pattern changes are file migration and file replication. File migration, however, incurs a relatively heavy overhead. In case the size of popular files is small, we may consider using a file replication approach to avoid file transfer overhead. We believe that a combination of the two techniques provides us with a good way of solving the file redistribution problem. Second, we intend to develop an energy-saving data placement scheme for write-dominated workload. The SEA strategy in its current form only considers read-dominated workload. For write-dominated workload, the SEA5 algorithm needs to frequently update the parity data for each write access, which obviously affects system performance and energy saving. One possible method to alleviate this burden is to utilize large size cache so that data updating only occurs periodically.

ACKNOWLEDGMENTS

The author thanks the anonymous reviewers, whose comments noticeably improved the quality of this paper. This work was supported by the US National Science Foundation Computing Processes and Artifacts (CISE-CCF) under Grant 0742187.

REFERENCES

[1] S. Akyürek and K. Salem, "Adaptive Block Rearrangement," *ACM Trans. Computer Systems,* vol. 13, no. 2, pp. 89-121, 1995.

[2] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web Caching and Zipf-Like Distributions: Evidence and Implications," *Proc. IEEE INFOCOM '99,* pp. 126-134, 1999.

[3] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, Distributed Data Placement Strategies for Storage Area Networks," *Proc. 12th Ann. ACM Symp. Parallel Algorithms and Architectures,* pp. 119-128, 2000.

[4] E.V. Carrera, E. Pinheiro, and R. Bianchini, "Conserving Disk Energy in Network Servers," *Proc. 17th Ann. Int'l Conf. Supercomputing,* pp. 86-97, 2003.

[5] P.M. Chen, E.K. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson, "RAID: High-Performance Reliable Secondary Storage," *ACM Computing Surveys,* vol. 26, no. 2, pp. 145-185, 1994.

[6] P.M. Chen and D.A. Patterson, "Maximizing Performance in a Striped Disk Array," *Proc. 17th Int'l Symp. Computer Architecture,* pp. 322-331, 1990.

[7] P.M. Chen and E.K. Lee, "Striping in a RAID Level 5 Disk Array," *ACM Sigmetrics Performance Evaluation Rev.,* vol. 23, no. 1, pp. 136-145, 1995.

[8] Y. Cho, M. Winslett, Y. Chen, and S.W. Kuo, "Parallel I/O Performance of Fine Grained Data Distributions," *Proc. Seventh Int'l Symp. High Performance Distributed Computing,* pp. 163-170, 1998.

[9] A.L. Couch, N. Wu, and H. Susanto, "Toward a Cost Model for System Administration," *Proc. Usenix 19th Conf. Large Installation System Administration,* pp. 125-141, 2005.

[10] C. Cunha, A. Bestavros, and M. Crovella, "Characteristics of WWW Client-Based Traces," Technical Report 1995-010, Boston Univ., 1995.

[11] C.H.Q. Ding and Y. He, "Data Organization and I/O in a Parallel Ocean Circulation Model," *Proc. 13th Ann. Int'l Conf. Supercomputing,* 1999.

[12] W. Dowdy and D. Foster, "Comparative Models of the File Assignment Problem," *ACM Computing Surveys,* vol. 14, no. 2, pp. 287-313, 1982.

[13] S. Ghandeharizadeh, S.H. Kim, and C. Shababi, "On Disk Scheduling and Data Placement for Video Servers," *Sigmetrics Performance Evaluation,* vol. 23, no. 1, pp. 37-46, 1995.

[14] S. Glassman, "A Caching Relay for the World Wide Web," *Proc. First Conf. World Wide Web,* pp. 165-173, 1994.

[15] R.L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Applied Math.,* vol. 7, no. 2, pp. 416-429, 1969.

[16] P. Greenawalt, "Modeling Power Management for Hard Disks," *Proc. Second Int'l Workshop Modeling, Analysis, and Simulation of Computer and Telecomm. Systems,* pp. 62-66, Jan. 1994.

[17] S. Gurumurthi, A. Sivasubramaniam, M. Kandemir, and H. Franke, "DRPM: Dynamic Speed Control for Power Management in Server Class Disks," *Proc. 30th Int'l Symp. Computer Architecture,* pp. 169-179, June 2003.

[18] "Hitachi Power & Acoustic Management: Quietly Cool," white paper, Hitachi Corp., Mar. 2004.

[19] H. Huang, W. Hung, and K.G. Shin, "FS2: Dynamic Data Replication in Free Disk Space for Improving Disk Performance and Energy Consumption," *Proc. 12th ACM Symp. Operating Systems Principles,* pp. 263-276, 2005.

[20] B. Inmon, "Information Management: Real-Time Decision Support Systems," *DM Rev. Magazine,* Aug. 2006.

[21] M. Kandemir, S.W. Son, and G. Chen, "An Evaluation of Code and Data Optimizations in the Context of Disk Power Reduction," *Proc. Int'l Symp. Low-Power Electronics and Design,* pp. 209-214, 2005.

[22] T. Kwan, R. Mcgrath, and D. Reed, "Ncsas World Wide Web Server Design and Performance," *Computer,* vol. 28, no. 11, pp. 67-74, Nov. 1995.

[23] R. Latham, N. miller, R. Ross, and P. Carns, "A Next-Generation Parallel File System for Linux Clusters: An Introduction to the Second Parallel Virtual File System," *Linux World Magazine,* pp. 56-59, Jan. 2004.

[24] L.W. Lee, P. Scheuermann, and R. Vingralek, "File Assignment in Parallel I/O Systems with Minimal Variance of Service Time," *IEEE Trans. Computers,* vol. 49, no. 2, pp. 127-140, Feb. 2000.

[25] P. Merialdo, P. Atzeni, and G. Mecca, "Design and Development of Data-Intensive Web Sites: The Araneus Approach," *ACM Trans. Internet Technology,* vol. 3, no. 1, pp. 49-92, 2003.

[26] M. Narris and J. Obal, "Performance Analysis of the Linux Buffer Cache while Running an Oracle OLTP Workload," Worcester Polytechnic Inst., Jan. 2002.

[27] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshida, and H. Tsuji, "Memory-Based Architecture for Distributed WWW Caching Proxy," *Proc. Seventh Int'l Conf. World Wide Web,* pp. 205-214, 1998.

[28] A.E. Papathanasiou and M.L. Scott, "Power-Efficient Server-Class Performance from Arrays of Laptop Disks," *Proc. Usenix Ann. Technical Conf. Work-in-Progress Presentation,* 2004.

[29] Z. Peterson, D.E. Long, and S.A. Brandt, "Data Placement Based on Seek Time Analysis of a MEMS-Based Storage Device," *Proc. Conf. File and Storage Technology Work-in-Progress Session,* Jan. 2002.

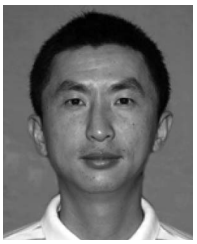
[30] E. Pinheiro and R. Bianchini, "Energy Conservation Techniques for Disk Array-Based Servers," *Proc. 18th Ann. Int'l Conf. Supercomputing,* pp. 68-78, June 2004.

[31] "Power, Heat, and Sledgehammer," white paper, Maximum Inst., www.max-t.com/downloads/whitepapers/SledgehammerPowerHeat20411.pdf, 2002.

[32] X. Ruan, X. Qin, M. Nijim, Z. Zong, and K. Bellam, "An Energy-Efficient Scheduling Algorithm Using Dynamic Voltage Scaling for Parallel Applications on Clusters," *Proc. 16th IEEE Int'l Conf. Computer Comm. and Networks,* pp. 735-740, Aug. 2007.

[33] N.J. Sarhan and C.R. Das, "Adaptive Block Rearrangement Algorithms for Video-on-Demand Servers," *Proc. 30th Int'l Conf. Parallel Processing,* pp. 452-459, 2001.

[34] S.W. Son, G. Chen, and M. Kandemir, "Disk Layout Optimization for Reducing Energy Consumption," *Proc. 19th Ann. Int'l Conf. Supercomputing,* pp. 274-283, 2005.
[35] S.W. Son, G. Chen, M. Kandemir, and A. Choudhary, "Exposing Disk Layout to Compiler for Reducing Energy Consumption of Parallel Disk-Based Systems," *Proc. 10th ACM Symp. Principles and Practice of Parallel Programming,* pp. 174-185, 2005.
[36] P. Triantafillou, S. Christodoulakis, and C. Georgiadis, "Optimal Data Placement on Disks: A Comprehensive Solution for Different Technologies," *IEEE Trans. Knowledge and Data Eng.,* vol. 12, no. 2, pp. 324-330, Feb./Mar. 2000.
[37] T. Xie and Y. Sun, "No More Energy-Performance Trade-Off: A New Data Placement Strategy for RAID-Structured Storage Systems," *Proc. 14th Ann. IEEE Int'l Conf. High-Performance Computing,* pp. 35-46, Dec. 2007.
[38] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton, and J. Wilkes, "Hibernator: Helping Disk Arrays Sleep through the Winter," *Proc. 12th ACM Symp. Operating Systems Principles,* pp. 177-190, 2005.
[39] Z. Zong, X. Qin, M. Nijim, X. Ruan, K. Bellam, and M. Alghamdi, "Energy-Efficient Scheduling for Parallel Applications Running on Heterogeneous Clusters," *Proc. 36th Int'l Conf. Parallel Processing,* 2007.



Tao Xie received the BSc and MSc degrees from Hefei University of Technology, Hefei, China, in 1991 and 2000, respectively, and the PhD degree in computer science from the New Mexico Institute of Mining and Technology in 2006. He is currently an assistant professor in the Department of Computer Science at San Diego State University, California. His research interests include storage systems, high-performance computing, cluster and grid computing, parallel and distributed systems, real-time/embedded systems, and information security. He is a member of the IEEE, the IEEE Computer Society, and Usenix.