A File Assignment Strategy Independent of Workload Characteristic Assumptions

TAO XIE
San Diego State University
and
YAO SUN
Teradata Corporation

The problem of statically assigning nonpartitioned files in a parallel I/O system has been extensively investigated. A basic workload characteristic assumption of most existing solutions to the problem is that there exists a strong inverse correlation between file access frequency and file size. In other words, the most popular files are typically small in size, while the large files are relatively unpopular. Recent studies on the characteristics of Web proxy traces suggested, however, the correlation, if any, is so weak that it can be ignored. Hence, the following two questions arise naturally. First, can existing algorithms still perform well when the workload assumption does not hold? Second, if not, can one develop a new file assignment strategy that is immune to the workload assumption? To answer these questions, we first evaluate the performance of three well-known file assignment algorithms with and without the workload assumption, respectively. Next, we develop a novel static nonpartitioned file assignment strategy for parallel I/O systems, called static round-robin (SOR), which is immune to the workload assumption. Comprehensive experimental results show that SOR consistently improves the performance in terms of mean response time over the existing schemes.

Categories and Subject Descriptors: D.4.3 [**Operating Systems**]: File Systems Management—*File organization*; C.4 [**Performance of Systems**]: Design Studies

General Terms: Design, Performance

Additional Key Words and Phrases: File assignment, parallel I/O, load balancing, Zipfian distribution, workload characteristics

A preliminary version of the article was published in *Proceedings of the 36th International Conference on Parallel Processing* [2007].
The research was partially supported by the National Science Foundation under Grants CNS-0834466 and CCF-0742187.
Authors' addresses: T. Xie, Department of Computer Science, San Diego State University, San Diego, CA 92182; email: xie@cs.sdsu.edu; Y. Sun, Teradata Corporation, San Diego, CA 92127; email: calvin.sun@teradata.com.

DOI 10.1145/1629075.1629079 http://doi.acm.org/10.1145/1629075.1629079

ACM Reference Format:
Xie,T. and Sun, Y. 2009. A file assignment strategy independent of workload characteristic assumptions. ACM Trans. Storage 5, 3, Article 10 (November 2009), 24 pages.
DOI = 10.1145/1629075.1629079 http://doi.acm.org/10.1145/1629075.1629079

1. INTRODUCTION

Many real-world applications intensively read data stored in large-scale parallel I/O systems like RAID, Redundant Arrays of Inexpensive Disks [Chen et al. 1994]. To guarantee the quality of service demanded by end-users, prompt responses to read requests are essential for these applications. For example, a data-intensive Web server application that publishes significant amounts of data stored in a back-end database must answer end-users' queries instantly before they lose patience [Carrera et al. 2003; Merialdoet et al. 2003]. It is obvious that the performance of these read-intensive applications largely depends on the performance of underlying parallel I/O systems, where disk arrays serve arrival requests simultaneously. More precisely, reducing mean response time of parallel disk storage systems is a must for these applications.

There are a wide variety of ways to reduce the mean response time or to improve the system throughput for parallel I/O systems [Hsu et al. 2005; Huang et al. 2005; Lee et al. 2000; Tewari 1992]. File assignment, allocation of all the files onto disk arrays before they are accessed, is one of such avenues that can significantly affect the overall performance of a parallel I/O system [Lee et al. 2000; Tewari 1992]. In order to fully exploit the capacities of a parallel disk storage system, *file assignment problem* (FAP) for parallel disk systems have been extensively investigated in the literature [Dowdy and Foster 1982; Pattipati and Wolf 1990]. A generic FAP formulation can be summarized as follows. Given a set of M files and N disks, find the file-disk allocation that optimizes some cost functions or performance metrics. While common cost functions include communication costs, storage costs, and queuing costs, popular performance metrics are mean response time and overall system throughput [Dowdy and Foster 1982]. It is well known that finding the optimal solution for a cost function or a performance metric in the context of file assignment on multiple disks is an NP-complete problem [Dowdy and Foster 1982]. Thus, heuristic algorithms became practical solutions.

Typically, heuristic file assignment algorithms fall into two camps: static and dynamic. Most static file assignment algorithms require complete knowledge about the workload statistics such as service times and access rates of all the files. Dynamic file assignment algorithms, on the other hand, generate file-disk allocation schemes online to adapt to varying workload patterns without a prior knowledge of the files to be assigned in the future. While stripping-based file assignment strategies like RAID-0 and RAID-5 are common for file systems with large files [Xie 2008], nonpartitioned file assignment algorithms are suitable for Web, proxy, and email server applications [Merialdo et al. 2003; Nishikawa et al. 1998; Qiu et al. 2001] where files are small in nature. In this article, we address the problem of statically assigning nonpartitioned files in a

parallel disk system where file accesses exhibit Poisson arrival rates and fixed service times.

Several previous studies [Cunha et al. 1995; Glassman 1994] show that the distribution of Web page requests generally follows a Zipf distribution [Lee et al. 2000] where the relative probability of a request for the i'th most popular page is proportional to 1/i. Moreover, they claim that the request frequency and the file size are inversely correlated, that is, the most popular files are typically small in size, while the large files are relatively unpopular. Based on these workload characteristic study results, many existing static file assignment algorithms, such as Greedy [Graham 1969], SP [Lee et al. 2000], and HP [Lee et al. 2000], were developed to reduce parallel I/O systems' mean response time. Experimental results either from prototype implementation or synthetic simulations demonstrate that they work well when the workload characteristics clearly exhibit. Some recent investigations on the characteristics of Web proxy traces, however, discovered that the correlation between access frequency and file size, if any, is so weak that it can be ignored [Breslau et al. 1999; Nishikawa and Hosokawa 1998]. In other words, at least in some real applications, the correlation assumption does not hold. Therefore, it is necessary to reexamine the existing static file assignment approaches to verify whether they are still efficient when the correlation does not exist. More importantly, it is indispensable to design and implement a new file assignment strategy, which can deliver good mean response times no matter the correlation assumption holds or not.

To achieve these two goals, we first measure the performance of three well-known algorithms, namely, Greedy [Graham 1969], SP, and HP [Lee et al. 2000], when the correlation assumption holds. Then we compare it with the performance when the correlation assumption does not hold. Next, we develop a novel static file assignment strategy, called static round-robin (SOR), which aims at minimizing mean response time under different workload conditions no matter the correlation assumption is valid or not. The basic idea of SOR is to assign all files sorted in their size onto an array of disks in a round-robin fashion. Further, we evaluate the performance of SOR together with Greedy, SP, and HP when the correlation assumption holds. Experimental results manifest that SOR consistently performs best, while SP delivers a comparable performance. Finally, we examine the four algorithms under the situation where the file access frequency is independent of the file size. Again, SOR outperforms the three traditional algorithms in all the tested cases, whereas Greedy, SP, and HP increase their mean response times on average by 2.44, 2.48, and 2.45 times, respectively. In summary, SOR demonstrates its strength and effectiveness under various workload conditions.

The rest of the article is organized as follows. In the next section we discuss the related work and motivation. In Section 3, we formulate the problem and present the SOR strategy as well as the three existing algorithms. In Section 4 we evaluate performance of our algorithms using both synthetic benchmarks and real-world traces. Section 5 concludes the article with summary and future directions.

2. RELATED WORK AND MOTIVATION

The *file assignment problem* (FAP) exists in a wide range of distributed systems including distributed file systems [Tewari 1992], distributed databases [Wolfson et al. 1997], video servers [Scheuermann et al. 1998], content distribution networks [Buchholz and Buchholz 2004] and the Grid [Foster 2004]. The first research work on FAP dates back to late 1960s [Chu 1969]. Since then FAP has been comprehensively investigated because the potential gain obtained by solving a FAP is significant [Dowdy and Foster 1982]. Typically, solutions to FAP fall into two categories: static and dynamic. Most static file assignment algorithms assume that access statistics are immutable, and hence the file assignment allocation scheme needs to be computed only once and can continuously work for a long time period [Chu 1969; Dowdy and Foster 1982; Kangasharju et al. 2002; Loukopoulos and Ahmad 2000; Qiu et al. 2001; Tang and Xu 2004]. Greedy, originated from *longest processing time* (LPT) algorithm proposed by Graham [1969] is one of the most well-known static file assignment heuristic algorithms. Dynamic file assignment algorithms [Qiu et al. 2001; Pattipati and Wolf 1990], on the other hand, update the file allocation scheme potentially upon every request. Obviously, they are effective when the files are relatively small in size, such as the case in Web proxy caching. However, in applications like distributed video servers [Scheuermann et al. 1998], since the files are of large size and they do not change in size, dynamic schemes become less useful.

With the advent of advances of distributed systems, new algorithms have been developed to solve problems such as data object replica placement [Karlsson and Karamanolis 2004; Loukopoulos et al. 2005], data management for large-scale storage systems [Alvarez et al. 2001; Lee et al. 2000; Weil et al. 2006], and automatic near-optimal storage system designs [Anderson et al. 2005]. Essentially, these problems are either directly derived from or closely related to the FAP problem. In recognition that minimizing the variance of service times at each disk is of the same importance as minimizing the utilization of each disk, Lee et al. [2000] proposed a static file assignment algorithm called sort partition (SP) and a semi-dynamic file assignment algorithm named hybrid partition (HP). Compared to the traditional Greedy algorithm, SP significantly improves the mean response time by taking the two minimizations into account simultaneously [Lee et al. 2000]. On the other hand, HP is a batch-based variant of SP, which can run in online mode. Based on our knowledge, SP is one of the best existing static file assignment algorithms so far.

Most of the new algorithms mentioned above, however, are based on some critical workload characteristic assumptions. Two most important ones are: the file access rate obeys a Zipfian distribution and the file access frequency is inversely correlated to the file size. These two workload assumptions were supported by several early studies on Web requests [Almeda et al. 1998; Cunha et al. 1995; Glassman 1994]. However, some recent research projects [Breslau et al. 1999; Nishikawa et al. 1998] conducted on real-world Web proxy traces suggested otherwise. They revealed that the distribution of file requests generally follows a Zipf-like distribution, where the relative probability of a request

for the i'th most popular page is proportional to $1/i^{\alpha}$, with α typically varying between 0 and 1, rather than a strict Zipfian distribution. In addition, the correlation between file access frequency and file size does not explicitly exist. Consequently, the foundation of the existing file assignment algorithms is potentially shaken by the new findings [Breslau et al. 1999; Nishikawa et al. 1998] at least in some applications. Therefore, it is necessary to reexamine the existing algorithms under the conditions where the two assumptions do not hold. To this end, we conducted a group of tests to evaluate three representative traditional algorithms, namely, Greedy, SP, and HP, under the situation where the file access frequency is independent of file size and the distribution of file sizes follows a random uniform distribution. Our preliminary results show that on average the mean response times of the three algorithms degrade to 2.44–2.48 times compared to the situations where the correlation assumption holds. Hence, it is mandatory to design and implement a new file assignment algorithm, which can deliver good mean response times no matter the workload assumptions hold or not. In other words, the need of a new file assignment algorithm that is immune to these workload assumptions is greatly felt.

In this paper, we are proposing SOR, a static heuristic file assignment strategy, which offers better mean response time compared to the three representative existing algorithms under a wide spectrum of workload conditions with or without the workload assumptions. Although we model a parallel I/O system as a set of homogeneous stand-alone disks in this article, our algorithm can be easily extended to RAID-structured storage systems. This is because the delays on the buses or controllers of the disks are negligible compared to the queuing delays on the disks, which are the dominant components in overall response times for many disk I/O-intensive applications due to heavy workloads. Similarly, we do not consider file partitioning in this work, and thus, each file must be allocated entirely onto one disk. This does not restrict the generality of our scheme as each file partition can be treated as a stand-alone file.

3. DESIGN AND IMPLEMENTATION OF SOR

In this section, we first formulate the FAP problem and system model, which is followed by a detailed description of the SOR algorithm. Then we prove the worst case time complexity of SOR, as well as an overall comparison between SOR and Greedy, SP, and HP via an example.

3.1 Problem Formulation and System Architecture

The architecture of the FAP problem is illustrated in Figure 1(a). A parallel I/O system in its most general form consists of a linked group, for example, $D = \{d_1, \ldots, d_j, \ldots, d_n\}$, of independent homogeneous disk drives. The set of files can be represented as $F = \{f_1, \ldots, f_i, \ldots, f_m\}$. In the system, a disk d_j is modeled as a three-element tuple $d_j = (c_j, t_j, l_j)$, where c_j, t_j, l_j are the disk capacity in GByte, transfer rate (read speed) in Mbyte/second, and load (total sum of files' heat on the disk). We assume that disks are always large enough to accommodate files to be assigned on them. Similarly, a file f_i is modeled as

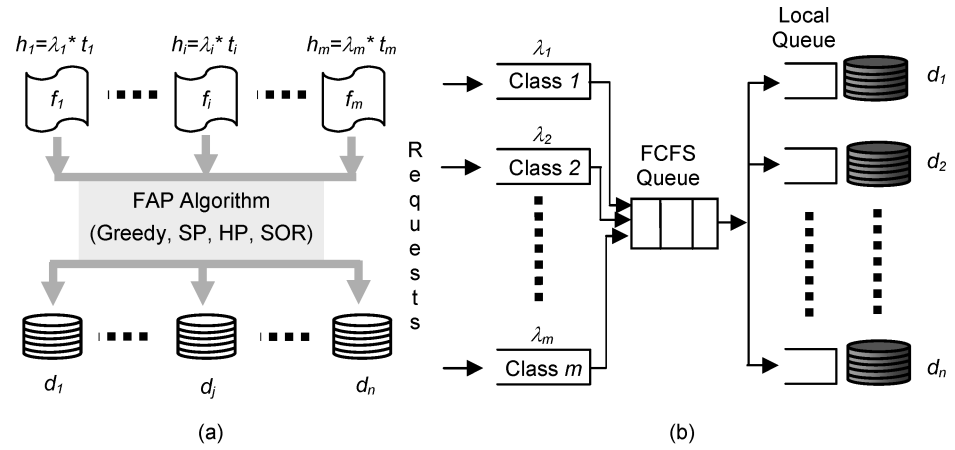

Fig. 1. System model of FAP and subsequent requests scheduling.

a set of rational parameters, for example, $f_i = (s_i, \lambda_i, es_i, h_i)$, where s_i, λ_i, es_i, h_i are the file's size in Mbyte, access rate, expected service time, and heat. In this paper, disk accesses to a file f_i are modeled as a Poisson process with a mean access rate λ_i. Also, we assume a fixed service time es_i for file f_i. This assumption is realistic for the following two reasons. First, each access to file f_i could be a sequential read of the entire file, which is a typical scenario in most file systems or WWW servers [Kwan et al. 1995]. Second, for large files, when the access unit is the entire file, the seek times and rotation latencies are negligible compared to the transfer time. Thus, es_i is determined by s_i and t_j if f_i is allocated on d_j

$$es_i = \frac{s_i}{t_j}. \tag{1}$$

Since we consider a homogeneous parallel I/O system with each disk having the same transfer rate, the service time of each file is fixed. In addition, the combination of λ_i and es_i accurately gives the load of f_i, we define the heat h_i of f_i as follows [Lee et al. 2000]:

$$h_i = \lambda_i . es_i. \tag{2}$$

Consequently, the average disk load ρ can be obtained by the following equation:

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{m} h_i. \tag{3}$$

File assignment algorithms like Greedy, SP, HP, and SOR allocate a group of files onto a set of identical disks so that the mean response time can be minimized. Figure 1(b) depicts the subsequent file access request scheduling process after the file assignment process completes. While the white color of disks in Figure 1(a) means that the load of each disk is zero before file assignment, the blue color of disks in Figure 1(b) represents that each disk has been

loaded before the system starting to serve incoming requests. Here, we employ the First-Come-First-Serve (FCFS) scheduling heuristic. Suppose there are totally u requests in the request set, which is modeled as $R = \{r_1, \ldots, r_k, \ldots, r_u\}$. 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 Figure 1(a) to find the disk on which the target file of the request resides. And then it directs the request to the disk's local queue. In fact, the request workload is a multi-class workload with each class of requests having its fixed λ_i and es_i (see Figure 1(b)).

To obtain the response time of a request r_k, two important parameters, the start time and finish time of r_k on a disk d_j must be computed. We denote the start time and finish time of r_k on disk d_j by $st_j(r_k)$ and $ft_j(r_k)$, respectively. In what follows we present derivations leading to the final expressions for these two parameters. There are three cases when r_k arrives in Q_j $(1 \le j \le n)$, the local queue of disk d_j. First, d_j is idle and Q_j is empty. Second, d_j is busy and Q_j is empty. Third, d_j is busy and Q_j is not empty. Thus, $st_j(r_k)$ is expressed as

$$st_j(r_k) = \begin{cases} SK + RT + a_k, \text{if } d_j \text{ is idle and } Q_j \text{ is empty} \\ SK + RT + a_k + r_j, \text{if } d_j \text{ is busy and } Q_j \text{ is empty} \\ SK + RT + a_k + r_j + \sum\limits_{r_p \in Q_j, a_p \le a_k} t_{fid_p}, \text{otherwise} \end{cases}, \tag{4}$$

where SK denotes average seek time, RT means average rotation latency, r_j represents the remaining service time of a request currently running on d_j, and $\sum\limits_{r_p \in Q_j, a_p \le a_k} t_{fid_p}$ is the overall service time of requests in Q_j whose arrival times are earlier than that of r_k. Consequently, $ft_j(r_k)$ can be calculated by

$$ft_j(r_k) = st_j(r_k) + es_{fid_k}, \tag{5}$$

where es_{fid_k} is the service time of the file that request r_k targets on (see Eq. 1). As a result, the response time of r_k can be obtained by

$$rt_j(r_k) = ft_j(r_k) - -a_k. \tag{6}$$

Similarly, the slowdown (the ratio between r_k's response time and its service time) of request r_k is

$$sd_j(r_k) = \frac{rt_j(r_k)}{es_{fid_k}}. \tag{7}$$

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

$$mrt(R) = \sum_{k=1,1 \le j \le n}^{u} rt_j(r_k) \Big/ u. \tag{8}$$

Similarly, the mean slowdown of the request set R can be obtained by

$$msd(R) = \sum_{k=1,1 \le j \le n}^{u} sd_j(r_k) \Big/ u. \tag{9}$$

```
Input: A parallel I/O system D with n disks, a collection of m files in a queue F
Output: A file allocation matrix X (n, m)
1. Use Eq. 3 to compute the average disk load ρ
2. for each disk j do
3.         load_j = 0; X(d_j, :) = 0 //  initialize the load and the allocation map for each disk
4. end for
5. f_i = 1                          // The file counter f_i is initially set to the first file in F
6. d_j = 1                          // The disk counter d_j is initially set to the first disk in D
7. Sort all files in F in ascending order of their service time es_i   (Eq. 1)
8. while f_i ≤ m do
9.         if  load_j ≤ ρ
10.                  X(d_j, :) = f_i                  // File f_i is allocated on disk d_j
11.                load_j = load_j + load(f_i)        // The load of file f_i is added to disk d_j
12.                f_i = f_i + 1                      // Process the next file
13.                if  d_j = = (n -1)                 // Process the next disk
14.                          d_j = 1
15.                else
16.                          d_j = d_j + 1
17.                end if
18.       else
19.           Search for a disk d_k among the remaining disks to accommodate file f_i
20.           If  successful
21.                  X(d_k, :) = f_i                  // File f_i is allocated on disk d_k
22.                load_k = load_k + load(f_i)
23.                f_i = f_i + 1
24.           else
25.                  X(d_n, :) = f_i                  // File f_i is allocated on the last disk d_n
26.                load_n = load_n + load(f_i)
27.                f_i = f_i + 1
28.           end if
29.        end if
30.   end while
```

Fig. 2. The SOR algorithm.

The FAP problem now can be formulated as: given a set of files F and a parallel I/O system D, find an allocation scheme X that optimizes the mean response time (see Equation (8)).

3.2 Algorithm Description

Figure 2 outlines the SOR algorithm with some detailed explanations. It is recognized that even distribution of workload among all disks and minimization of the variance of the service time at each disk are two important paths towards the goal of minimizing the queuing delay [Lee et al. 2000]. To reach that goal, SOR computes the average disk load ρ in step 1 and enforces the load on each disk not to exceed ρ (step 9). In step 7 SOR sorts the file set F in file size so that files with similar sizes can be allocated onto the same disk [Lee et al. 2000]. Next, SOR separates the most popular files onto different disks by utilizing a round-robin manner. In fact, the round-robin fashion used by SOR is actually

a *partial* round-robin in the sense that only the first $n-1$ disks are involved in the round-robin file assignment process (step 14). If a file f_i cannot be allocated onto disk d_j, SOR searches a disk d_k that is closest to d_j to accept it (Steps 19–23). If failed, which means f_i is a big file, it will be put into disk d_n, a disk dedicated for these unusual size files (Steps 25–27).

Two most desirable features of SOR that make it superior to traditional SP and HP algorithms are allocating popular files onto different disks in a round-robin manner and aggregating large files on one particular disk. The rationale behind the first characteristic of SOR is that files with higher load (heat) values will be evenly distributed onto distinct disks so that no disk can have a large number of popular files. Consequently, the overall load balancing could be further improved. By contrast, after sorting files based on their popularities, both SP and HP simply assign a consecutive collection of popular files onto each disk until its load reaches the average disk load ρ. The disadvantage of this consecutive popular file allocation on one disk is that the collection of popular files would overload a disk, which turns it into a performance bottleneck. Note that the round-robin fashion used by SOR is actually a *partial* round-robin in the sense that only the first $n-1$ disks are involved in the round-robin file assignment process (Step 14). The reason why we exclude disk d_n out of the round-robin procedure is that it will be exclusively used by files with very large sizes. Aggregating large files on one disk will prevent them from severely block responses to the requests for small files, which could happen if they are mixed together with small files on the same disk. Moreover, SOR overcomes a hidden drawback of SP where the allocation of files onto disks is not even in terms of number of files on each disk.

3.3 Time Complexity of SOR

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

THEOREM 1. *Given a parallel disk array system $D = \{d_1, d_2, \ldots, d_j, \ldots, d_n\}$ and a collection of files represented by a file queue $F = (f_1, f_2, \ldots, f_i, \ldots, f_m)$, the worst-case time complexity of SOR is $O(mlgm + mn)$, where m is the number of files in Q, n is the number of disk in the system P.*

PROOF. It takes $O(n)$ time to initialize the load and the allocation map for each disk (see Steps 2–4). The time complexity of sorting the file set F is $O(mlgm)$ as we have m files (see Step 7). To discover an appropriate disk in D for a file f_i to be allocated, the worst case for SOR is to visit each of the n disks in D. Consequently, the worst case time complexity for allocating one file is $O(n)$ (see Steps 19–28). Since we totally have m files, the worst case time complexity for allocating all files in Q is $O(mn)$. Other steps simply take $O(1)$ time. Hence, the worst-case time complexity of the SOR algorithm is: $O(n)+O(mlgm)+O(mn) = O(mlgm + mn)$. □

Theorem 1 indicates that the time complexity of the SOR algorithm is typically low. For example, in our synthetic simulation experiments, the value of

Table I. System Parameters for Overall Comparison Example

Parameter	Value
Number of files	50
File access distribution (Zipfian)	Skew degree X:Y = 70:30
File size distribution (Zipfian)	Skew degree X:Y = 70:30
Coverage of the file system	100% – each file is at least accessed once
Number of batches (for HP only)	2 – each batch has 25 files
Number of disks	8
Number of requests	975
Simulation duration	50 seconds
Average disk load (heat) ρ	213.6
Disk transfer rate	31 Mbytes/second
Aggregate access rate $\sum_{i=1}^{50} \lambda_i$	19.5 (1/second)

n is set to 5000 and the value of m is in the range [8, 24], which should take less than hundreds of microseconds to complete the SOR algorithm in modern processors. An implication of Theorem 1 is that SOR has the potential to be extended to be applied in real-world applications because of its low complexity.

3.4 A Performance Comparison Example

In this section, we compare the overall performance of SOR against Greedy, SP, and HP via an example. The purpose of this example is to intuitively show how SOR works through experimental results, which come from snapshots generated by our simulator (see Section 4.1). Table I summarizes the system parameters configured for this example with 8 disks, 50 files, and 975 requests arrival within 50 seconds. The distribution of file size was inversely correlated with the distribution of file access frequency using the same skew degree value 70:30 (see Figure 3(a)–3(b)). The three algorithms are briefly described below.

(1) *Greedy*: The most common heuristic for multiple disks load balancing. It can operate in either online mode or offline mode. Here, we only consider its offline mode because SOR is an offline 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.
(2) *SP* (*Sort Partition*): It first computes the average disk utilization using Equation (3). 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 remainder files (if any) after one round allocation will be assigned to d_n. It improves the performance of the Greedy algorithm by minimizing the variances of service times at each disk.
(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.

Table II. File Sizes (Mbytes) and Request Numbers

1∼10	1600,125	1623,77	1646,57	1671,47	1696,40	1723,35	1750,31	1779,29	1809,26	1839,24
11∼20	1872,23	1905,21	1941,20	1977,19	2016,18	2056,17	2099,17	2143,16	2190,15	2239,15
21∼30	2292,14	2347,14	2406,13	2468,13	2535,13	2606,12	2681,12	2763,12	2851,11	2946,11
31∼40	3049,11	3161,10	3283,10	3418,10	3567,10	3733,10	3918,9	4128,9	4367,9	4643,9
41∼50	4965,9	5347,9	5810,8	6382,8	7113,8	8087,8	9462,8	11586,8	15412,8	25101,7

Table III. Allocation Scheme for SP

Disk 1	1	2	—	—	—	—	—	—	—	—
Disk 2	3	4	5	6	—	—	—	—	—	—
Disk 3	7	8	9	10	11	12	13	—	—	—
Disk 4	14	15	16	17	18	19	20	21	22	—
Disk 5	23	24	25	26	27	28	29	30	31	32
Disk 6	33	34	35	36	37	38	39	40	—	—
Disk 7	41	42	43	44	45	46	—	—	—	—
Disk 8	47	48	49	50	—	—	—	—	—	—

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 (heat). 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.

Table II specifies each file's size and the number of requests that target on it within 50 seconds. For example, the size of file f_2 is 1623 Mbytes and there are 77 requests targeting on it. After running the four algorithms, the resulting file allocation schemes were given by Table III–Table VI. Table VII demonstrates the load (heat) distribution on each disk and mean response time for each algorithm after serving 975 requests.

One can easily draw the following conclusions based on the observations from the four file allocation schemes (Table III–Table VI). First, Greedy delivers the worst performance in terms of mean response time among the four algorithms because it mixes large files with small files in each disk. Although large files have fewer requests, these requests could severely delay all subsequent requests that access small files on the same disk due to their long service times. As a result, the mean response time is dramatically enlarged. This conclusion is verified by the experimental results shown in Table VII. Second, the mean response times of HP are somewhere between Greedy and SP because it also mixes large files with small files in some disks. This is because HP can only separate large files from small files within each batch and it lacks of the knowledge of files in the future batches. Third, SP performs best among the three existing algorithms for it appropriately keeps large files apart from small files. Last, SOR outperforms SP due to the following several facts. First of all, SOR separates large files from small files just as SP does. Next, SOR scatters the first files, which have relatively large values in terms of heat, onto distinct disks to further improve the overall load balancing. Finally, SOR solves the uneven problem in terms of number of files in each disk experienced by SP. Take disk d_1 for instance, d_1 in SP can only accept f_1 and f_2 because the next file f_3 has a relatively big value in heat. Since $\lambda_3 = 57/50 = 1.14$ (1/second) and

Table IV. Allocation Scheme for HP

Disk 1	3	7	10	12	11	17	—	—	—	—
Disk 2	9	15	21	39	41	4	8	—	—	—
Disk 3	28	29	30	16	18	13	14	31	32	38
Disk 4	34	35	40	26	37	47	48	—	—	—
Disk 5	42	45	19	20	22	24	23	25	27	—
Disk 6	50	33	36	49	—	—	—	—	—	—
Disk 7	2	44	43	46	—	—	—	—	—	—
Disk 8	5	6	1	—	—	—	—	—	—	—

Table V. Allocation Scheme for Greedy

Disk 1	23	13	41	50	—	—	—	—	—	—	—	—
Disk 2	5	16	38	12	44	43	—	—	—	—	—	—
Disk 3	48	39	8	36	27	—	—	—	—	—	—	—
Disk 4	1	6	18	31	—	—	—	—	—	—	—	—
Disk 5	21	22	30	25	9	49	—	—	—	—	—	—
Disk 6	46	40	47	7	10	28	—	—	—	—	—	—
Disk 7	33	20	15	45	14	29	3	—	—	—	—	—
Disk 8	26	35	42	19	4	32	2	11	17	37	24	34

Table VI. Allocation Scheme for SOR

1	8	15	22	—	—	—	—
2	9	16	23	29	30	—	—
3	10	17	24	31	32	33	—
4	11	18	25	34	35	36	—
5	12	19	26	37	38	39	40
6	13	20	27	41	42	43	—
7	14	21	28	44	45	46	—
47	48	49	50	—	—	—	—

Table VII. Disk Load (heat) Distribution and Mean Response Times

Disk	Greedy	SP	HP	SOR
1	187.4	209.7	200.6	206.9
2	179.0	193.8	212.6	194.8
3	163.3	205.7	222.5	195.9
4	211.7	200.1	223.6	190.9
5	193.9	207.6	214.7	209.9
6	202.4	189.4	238.2	196.2
7	207.9	201.3	185.3	212.7
8	363.4	301.5	211.7	320.3
Mean response time (second)	5859.9	5124.6	5239.3	5092.4

$t_3 = 1646/31 = 53.1$ seconds, the load of file f_3 is $load(f_3) = 60.5$. Meanwhile, the discrepancy between the current load $load_1$ and the average disk load ρ is $213.6-(\lambda_1 \cdot t_1+\lambda_2 \cdot t_2) = 213.6-(129+80.6) = 4$, which is smaller than $load(f_3)$. Consequently, f_3 is rejected by d_1. In contrast to SP, SOR can accommodate 4 files (f_1, f_8, f_{15} and f_{22}, see Table VI). In addition, comparing Table III with Table VI, one can see that the range of number of files on each disk in SP is [2, 10], whereas it is only [4, 8] in SOR. SOR reduces the variance of number of files across disks by 50% compared to SP. In other words, compared to SP, SOR can more evenly distribute files onto disks in terms of number of files on each disk.

Table VIII. Main Characteristics of the Cheetah Disk

Description	Value	Description	Value
Disk model	Seagate Cheetah ST39205LC	Standard interface	SCSI
Storage capacity	9.17 GBytes	Rotational speed	10,000 rpm
Ave. seek time (*SK*)	5.4 msecs	Number of platters	1
Ave. rotation latency (*RT*)	3 msecs	Transfer rate	31 Mbytes/second

The conclusion is that an even file allocation scheme in terms of number of files on each disk can further decrease the mean response time. A straightforward explanation is that under the same level of disk load (heat), a large number of requests with relatively shorter service times can result in a longer mean response time compared to a small number of requests with relatively longer service times. The reason is twofold. First, a small number of requests tend to generate a sparser request arrival list, and thus, decrease the possibilities that arrived requests compete with each for the same disk. Second, a small number of requests avoid a delay accumulation effect, which might be experienced by a large number of requests due to the relatively larger number of requests.

4. PERFORMANCE EVALUATION

Here we evaluate the effectiveness of the proposed SOR file assignment scheme using extensive simulations. The advantage of using simulation is that we can easily vary parameters, which is a key component of this paper. We first introduce experimental settings in Section 4.1 and Section 4.2. Next, in Section 4.3 we evaluate the four algorithms in Zipf-like file size distribution followed by a study of the algorithms in uniform file size distribution in Section 4.4. Finally, we assess the performance of the four algorithms using three real-world traces in Section 4.5. Some preliminary results in Section 4.3 were presented in Xie [2007].

4.1 Execution-Driven Simulator and Parameter Space

We have developed an execution-driven simulator that models an array of conventional Cheetah disks. The main characteristics of the conventional disk are shown in Table VIII. 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 I/O system (see Equation (8)). Note that the mean response times are normalized in the scale [0, 1] for all graphs in Sections 4.3–4.4.

—*Mean Response Time Improvement*. decrease (in seconds) of mean response time gained by SOR compared to the three existing algorithms.

—*Mean Slowdown*. the ratio between average request response time and average request service time (see Equation (9)).

—*Mean Disk Utilization*. average ratio between a disk's total service time and its total operation time. The operation time is defined as the time period between the arrival time of the first file access request and the finish time of the last file access request.

Two categories of parameters directly influence the file assignment algorithms that we investigate: workload characteristics and disk drive characteristics. Among the large number of parameters that specify a workload, we identified five key characteristics: number of files, request rate, file popularity weight, file size distribution, and the coverage of the file system.

—*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 5000 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. Each file was allocated to a single disk. No files can be partitioned or replicated.

—*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 $\sum_{i=1}^{5000} \lambda_i$. The value of the aggregate arrival rate represents the intensity of the total access requests submitted to the disk array where 5000 files have been assigned across.

—*File Popularity Weight*. File popularity weight relates to the frequency with which file requests arrive at the parallel I/O 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 $\theta = log\frac{X}{100} / log\frac{Y}{100}$, where X percent of all accesses were directed to Y percent of files [Lee et al. 2000]. The value of X:Y is called *skew degree* in this article and $\alpha = 1 - \theta$ (see Section 1 for α). Figure 3(a) shows a Zipf-like distribution of file access rate on the 5000 files with $X:Y = 70{:}30$ assuming that file f_1 is the most popular file and f_{5000} is the most unpopular one. In our simulations, we tested four values of θ with *skew degree* (X:Y) changing from 50:50 to 80:20.

—*File Size Distribution*. We considered two cases for file size distribution: Zipf-like distribution and uniform distribution. In the first case, the distribution of access rates across the files and the distribution of file sizes were inversely correlated with the same skew parameter θ, as shown in Figure 3(b). The file size distribution is reasonable because the phenomena that popular files are generally small ones can be frequently observed. On the other hand, however, some Web proxy traces demonstrated that the correlation between file access frequency and file size could be very weak and can be ignored [Breslau et al. 1999]. Therefore, we also measured the system performance when file size is independent of file access frequency and it obeys a uniform distribution (Figure 3(c)). Section 4.3 measures the performance results assuming a Zipf-like file size distribution, whereas Section 4.4 examines SOR assuming a uniform file size distribution.

—*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.

Table IX. Simulation Parameters

Parameter	Value (Fixed) – (Varied)
Number of files	(5000)
File request rate	Each file has a fixed Poisson request arrival rate λ_i (1/second).
File access distribution (Zipf-like)	Skew degree (X:Y = 70:30) – (50:50, 60:40, 70:30, 80:20)
File size distribution	Zipfian: X:Y = (70:30) – (50:50, 60:40, 70:30, 80:20)
	Uniform: ([1,1000]) – ([1,1000], [1,1100], [1,1200], [1,1300], [1,1400], [1,1500], [1,1600], [1,1700], [1,1800], [1,1900], [1,2000]) MB
Number of batches (HP)	(4) – each batch has 1250 files
Number of disks	(16) – (8, 12, 16, 20, 24)
Aggregate access rate	(200) – (25, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000) (1/second)
Simulation duration	(1000) seconds

We set the coverage of the system to 100% in our simulations, which means all files in the parallel I/O system are accessed at least once.

For disk drive characteristics, we only focused on transfer rate (see Table IX) because transfer rate combined with the file size decides a file access request's service time, which in turn affects the mean response time. Table IX summarizes the configuration parameters of a simulated parallel I/O system used in our experiments and characteristics of the synthetic workload. All synthetic workload used from Section 4.3 to Section 4.4 were created by our trace generator. Although number of disks, aggregate access rate, and size of files are synthetically generated, we examined impacts of these important parameters on system performance by controlling the parameters.

4.2 Synthetic Workload Generator

To study the impact of the parameters we discussed in Section 4.2, we built a workload generator that can produce file traces as well as request traces with user-specified request rate, popularity weight, file sizes, number of files, coverage, and 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 $\theta = log\frac{X}{100}/log\frac{Y}{100}$. 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 it access rate (Figure 3(a)). The same skew degree value was used to produce file sizes when the file size obeys a Zipf-like distribution (Figure 3(b)). In case the file sizes are irrelevant to file access rates, the file sizes were generated based on a uniform distribution (Figure 3(c)). Therefore, each file has the following four attributes: popularity weight, access rate, file size in Zipf-like distribution, and

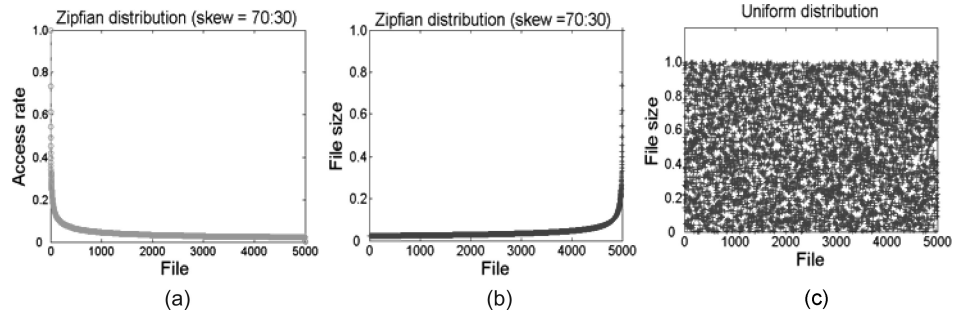

Fig. 3. (a) File access rate distribution; (b) Zipfian file size distribution; (c) Uniform file size distribution.

file size in uniform 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 (1000 seconds) according to each file's access rate. Since we set the *coverage of the file system* to 100%, each file has at least one request. The inter-arrival 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 mix all 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.

4.3 Experimental Results from Zipf-like File Size Distribution

The goal of this group of experiments (Figure 4–Figure 6) is to compare the proposed SOR algorithm against the three well-known file assignment schemes when file size follows a Zipf-like distribution. The aggregate access rate varies from 25 to 1000 per second and the file sizes were distributed according to Zipf's law with skew degree 70:30.

—*Impact of Aggregate Access Rate*. Figure 4 shows the simulation results for the four algorithms on a parallel I/O disk array with 16 disk drives. We observe from Figure 4(a) that SOR consistently outperforms the three exiting approaches in terms of mean response time. This is because SOR considers both minimizing variance of service time for each disk and fine-tuning load balancing degree. Consequently, the sorted files were continuously assigned to disks such that a more evenly distributed workload allocation scheme was generated. SP takes the second place in mean response time metric, which is consistent with our expectation because it is one of the best existing static file assignment heuristics. To clearly demonstrate the performance improvement, Figure 4(b) provides mean response time decrease gained by SOR compared to Greedy, SP, HP, respectively. In particular, SOR can reduce mean response time on average by 1118.3, 1052.8, and 269.6 seconds, compared to HP, Greedy, and SP, respectively. An interesting observation is that the mean response time improvement becomes more significant when the overall

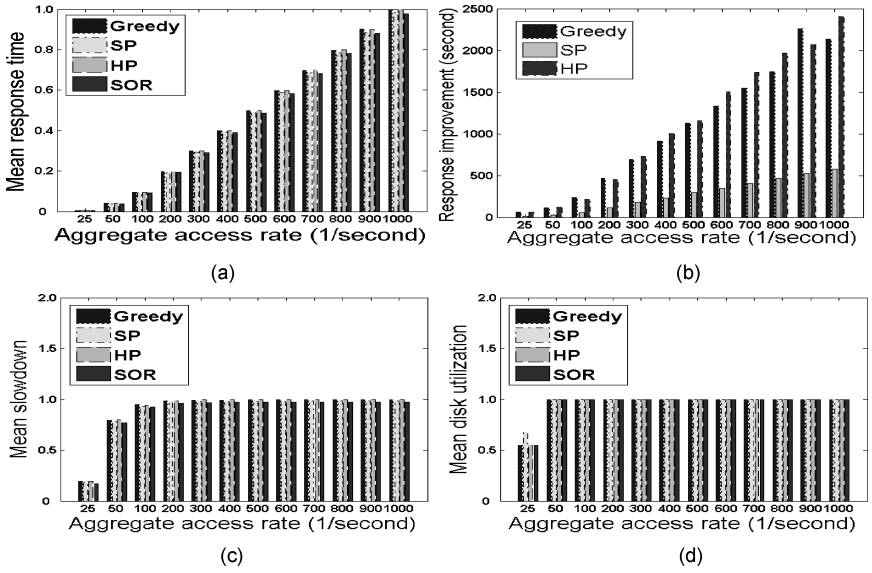

Fig. 4. Impact of aggregate access rate in Zipfian file size distribution.

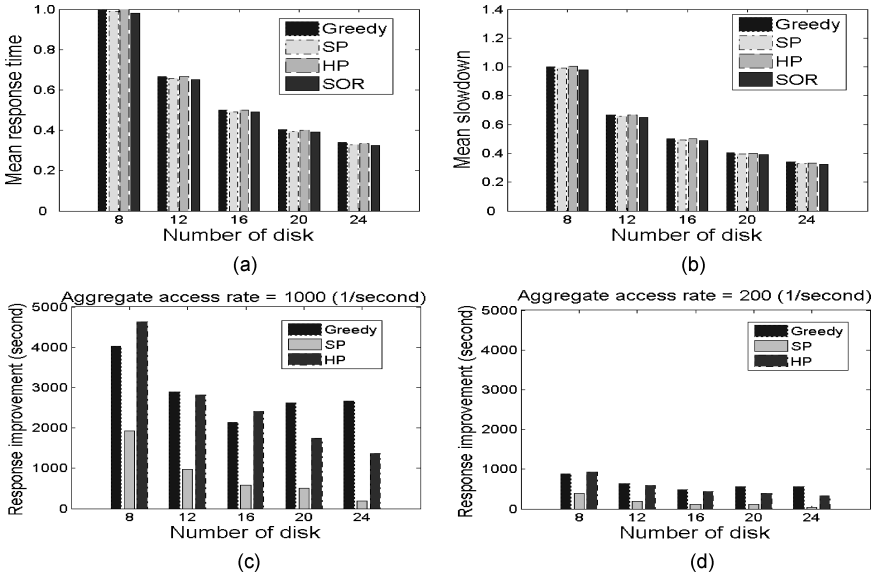

Fig. 5. Scalability in Zipfian file size distribution.

workload represented by the aggregate access rate increases. The implication is that SOR exhibits its strength in situations where system workload is heavy. In terms of mean slowdown, SOR also performs best among the four heuristics (Figure 4(c)), which is consistent with the results shown in Figure 4(a). Since the total workload is relatively heavy, the mean disk utilization in Figure 4(d) quickly rises to 1 when aggregate access rate is larger than 25 (1/second).

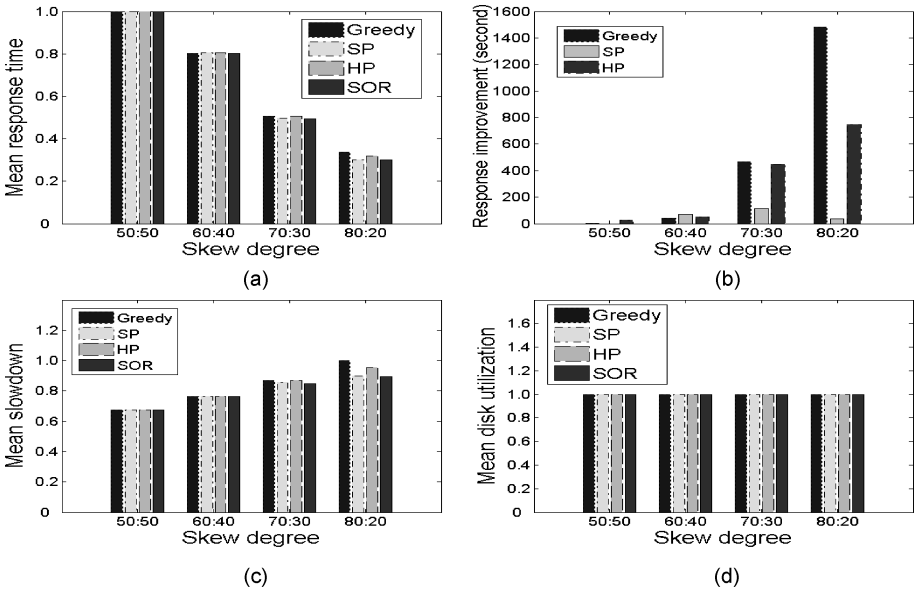

Fig. 6. Sensitivity to skew parameter θ in Zipfian file size distribution.

—*Scalability*. This experiment is intended to investigate the scalability of the four algorithms. We scale the number of disks in the system from 8 to 24. The aggregate access rate is configured to 200 (1/second) and 1000 (1/second). The skew degree is still set to 70:30. Figure 5 plots the performance of the four algorithms as functions of the number of disks. The results show that SOR exhibits a good scalability.

Figure 5 shows that all of the four algorithms deliver better performance in both mean response time and 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. One important observation is that SOR outperforms the rest three approaches in all tested cases. The implication of this observation is that SOR is suitable for a parallel I/O system where the number of disks is not sufficient for a heavy workload. Comparing the results from Figure 5(c) and Figure 5(d), we can see that the mean response time improvement becomes more pronounced when the aggregate access rate is large. In addition, the implication of results from Figure 5(c) and Figure 5(d) is that the aggregate access rate affects the mean response time more significantly compared to the influence imposed by the number of disks.

—*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 set to 50:50, SOR degraded to SP and only marginally outperforms Greedy and HP in terms of mean response time (Figure 6(b)). This is because the skew parameter θ is equal to 1, which means the access requests were evenly distributed across all files without any request skew. On the other hand, when the skew degree was enlarged to 70:30, SOR can reduce mean response time by 466.9, 115.2, 448.7 seconds, compared to Greedy, SP,

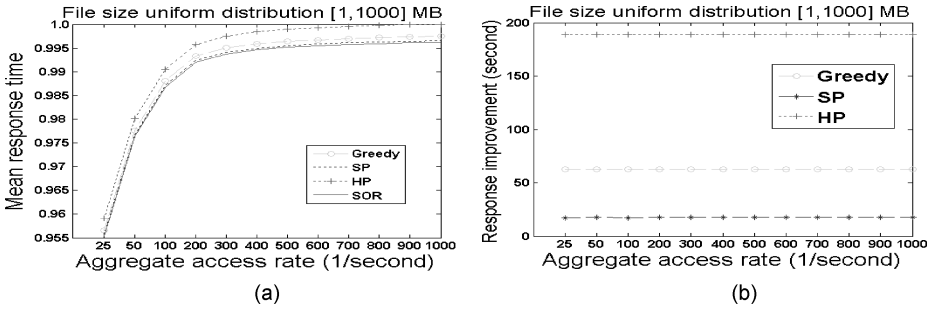

Fig. 7. Impact of aggregate access rate in uniform file size distribution.

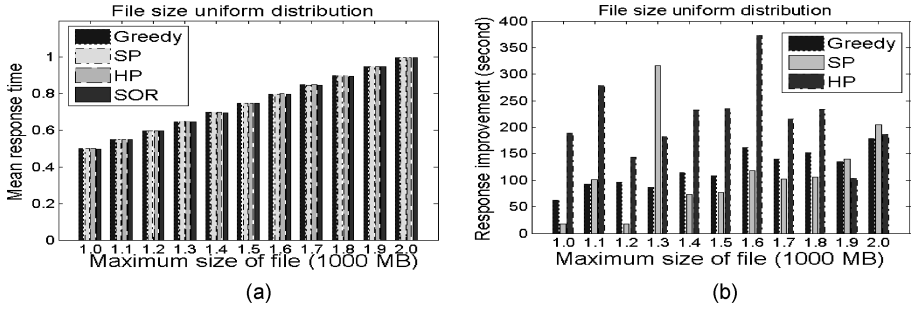

Fig. 8. Impact of file size range in uniform file size distribution.

and HP, respectively. We observe from Figure 6 that SOR achieves the best mean response time improvement when the skew degree is 70:30.

4.4 Experimental Results from Uniform File Size Distribution

The purpose of this group of experiments (Figure 7, Figure 8) is to evaluate the proposed SOR algorithm when file size follows a uniform distribution.

—*Impact of aggregate access rate*. In this section we examine the performance impact of aggregate access rate when the size of files exhibits a uniform distribution with minimum file size 1 MB and maximum file size 1000 MB. In case where the sizes of files obey a uniform distribution, SOR consistently outperforms all the other algorithms in mean response time. The decrease of mean response time gained by SOR is not pronounced as on average SOR only reduces mean response time by 188.8, 62.3, and 17.5 seconds, compared to HP, Greedy, and SP, respectively (Figure 7). However, this experiment manifests that SOR is a more general file assignment scheme, which works well in both file size distributions. On the contrary, compared to the results from Figure 4(a), Greedy, SP, and HP significantly degrade their mean response times on average by 2.44, 2.48, and 2.45 times, respectively. The results of this experiment demonstrate that existing file assignment algorithms perform much worse when the inverse correlation between file access frequency and file size does not hold.

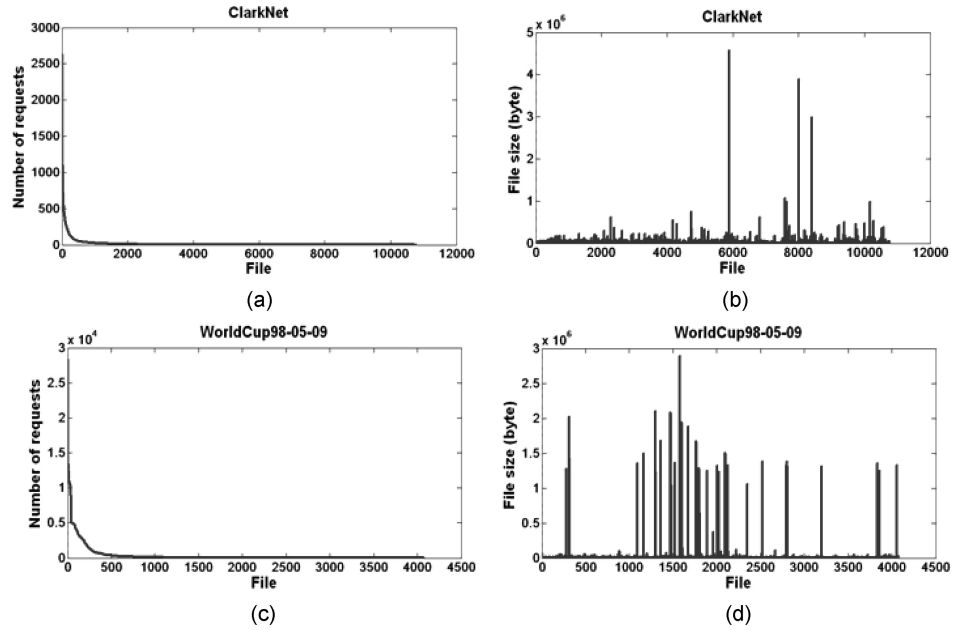

Fig. 9. Workload characterization of the two traces: number of requests and file size distribution.

—*Impact of file size range*. To measure the impacts of file size range in uniform file size distribution, we evaluate the four algorithms while changing the range of file size from [1, 1000] Mbytes to [1, 2000] Mbytes. Performance patterns plotted in Figure 8 are similar to those reported in previous sections, thereby verifying that SOR can gain performance improvements for a wide range of file size. SOR always offers the smallest mean response time in all cases.

4.5 Trace-Driven Simulations

To validate the results from the synthetic workload, we evaluate in this experiment the SOR algorithm using three real-world Web I/O traces (ClarkNet-HTTP log [Carrera et al. 2003], WorldCup98-05-09, and WorldCup98-06-11 [Arlitt and Jin 1999]), which have been widely used in the literature. The ClarkNet-HTTP log was collected by ClarkNet, an Internet service provider, for a week from 09/04/95 to 01/10/95 with total 3,328,587 requests. Since the simulation times in our experiments are much shorter compared with the time span of the ClarkNet-HTTP trace, we only choose one day (09/04/95) data, which has 186,397 re-quests. Similarly, we select two days' data, 05/09/98 and 06/11/98, from the WorldCup98 trace, which presents one of the largest Web workload analyzed so far [Arlitt and Jin 1999]. While the ClarkNet-HTTP log presents a light workload, the two WorldCup98 traces introduce a heavy and an extremely heavy workload condition, respectively. Table X summarizes the statistics of the three real traces.

The distributions of access rates across the files and file sizes of the ClarkNet trace and the WorldCup98-05-09 trace were plotted in Figure 9. Files on the

Table X. Statistics of the Three Real Traces

Trace Name	Files	Requests	Ave. Arrival Interval	Mean File Size
Clarknet-HTTP log	10798	186397	463.5 ms	13097 bytes
World Cup 98–05-09	4079	1480081	58.4 ms	20021 bytes
World Cup 98–06-11	4261	59201342	1.46 ms	20086 bytes

x-axis of all figures in Figure 9 were sorted by their popularity in terms of number of requests received with file 1 being the most popular file. Since the WorldCup98-06-11 trace has a similar file popularity and file size distribution to WorldCup98-05-09, we show only WorldCup98-05-09 in Figure 9. From Figure 9(a) and Figure 9(c), one can see that both ClarkNet and WorldCup98-05-09 exhibit a Zipf-like file requests distribution. Nevertheless, the ClarkNet trace displays a clear correlation between file access frequency and file size. More precisely, all popular files in ClarkNet are small files (see Figure 9(b)). On the other hand, the WorldCup98-05-09 trace demonstrates that the access frequency of files has no correlation with file size (see Figure 9(d)). In other words, large files could be either popular files or unpopular files in WorldCup98-05-09. Thus, by using the three traces, all four algorithms can be examined in two distinct scenarios: (1) there is an inverse correlation between file access frequency and file size; and (2) file access frequency and file size has no correlation. Trace-driven simulation results (see Figure 10) demonstrate that SOR outperforms the three baseline algorithms in almost all cases. Especially, when workload becomes heavier, the improvement of SOR obviously increases. More importantly, results from Figure 10 are consistent with synthetic simulations discussed in Section 4.3 and Section 4.4, which validates the effectiveness of our synthetic experiments.

5. CONCLUSIONS

In this article, we address the issue of statically allocating nonpartitioned files onto a parallel I/O system where the file access requests exhibit Poisson arrival rates and fixed service times. We found that the performance of existing file assignment algorithms in terms of mean response time dramatically degraded when the inverse correlation between file access frequency and file size does not hold. Therefore, a *s*tatic *ro*und-*r*obin (SOR) file assignment strategy is developed to generate optimized file allocations that minimize mean response time no matter the correlation exists or not. To quantitatively evaluate the effectiveness and practicality of the proposed SOR scheme, we conducted extensive experiments using both synthetic benchmarks and real-world traces. Experimental results show that when the distribution of access rates across the files and the distribution of file sizes were inversely correlated with the same skew parameter θ (Figure 3(a)–3(b)), SOR consistently improves the performance of parallel I/O systems in terms of mean response time over three well-known file assignment algorithms. Compared to SP, one of the best existing static nonpartitioned file assignment algorithms, SOR obviously achieves improvement in mean response time. More importantly, when the

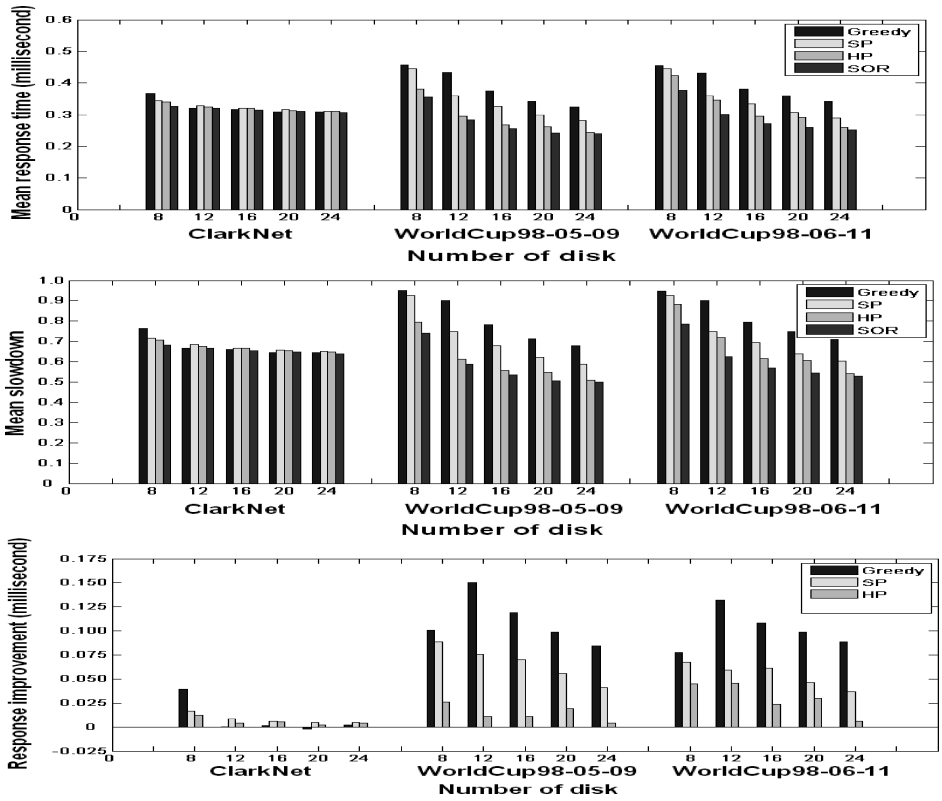

Fig. 10. Trace-driven simulation results.

correlation between file access frequency and file size is negligible, SOR still consistently performs better when file size exhibits a uniform distribution.

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 file assignment algorithm is mandatory so that dynamically arrived files can be reallocated by migrating files from one disk to another. Second, we intend to enable the SOR scheme to cooperate with the RAID architecture, where files are usually partitioned and then distributed across disks in order to further reduce the service time of a single request.

REFERENCES

ALMEDA, V., CESARIO, M., FONSECA, R., MEIRA, W. JR., AND MURTA, C. 1998. Analyzing the behaviour of a proxy server. In *Proceedings of the 3rd International Caching Workshop*.

ALVAREZ, G. A., BOROWSKY, E., GO, S., ROMER, T. H., BECKER-SZENDY, R., GOLDING, R., MERCHANT, A., SPASOJEVIC, M., VEITCH, A., AND WILKES, J. 2001. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Trans. Comput. Syst. 19*, 4, 483–518.

ANDERSON, E., SPENCE, S., SWAMINATHAN, R., KALLAHALLA, M., AND WANG, Q. 2005. Quickly finding near-optimal storage designs. *ACM Trans. Comput. Syst. 23*, 4, 337–374.

ARLITT, M. AND JIN, T. 1999. Workload characterization of the 1998 World Cup Web site. Tech. rep., HPL-1999-35R1, HP Labs.

Breslau, L. Cao, P., Fan, L., Phillips, G., and Shenker, S. 1999. Web caching and Zip-like distributions: Evidence and implications. In *Proceedings of the 18th Conference on Computer Communications*. 126–134.

Buchholz, S. and Buchholz, T. 2004. Replica placement in adaptive content distribution networks. In *Proceedings of the ACM Symposium on Applied Computing*. 1705–1710.

Carrera, E. V., Pinheiro, E., and Bianchini, R. 2003. Conserving disk energy in network servers. In *Proceedings of the 17th Annual International Conference on Supercomputing*. 86–97.

Chen, P. M., Lee, E. K., Gibson, G. A., Katz, R. H., and Patterson, D. A. 1994. RAID: High-performance, reliable secondary storage. *ACM Comput. Surv. 26*, 2, 145–185.

Chu, W. Optimal file allocation in a multiple computer system. *IEEE Trans. Comput. 18*, 10, 885–889.

Cunha, C., Bestavros, A., and Crovella, M. 1995. Characteristics of WWW client-based traces. Tech. rep., 1995-010, Boston University.

Dowdy, W. and Foster, D. 1982. Comparative models of the file assignment problem. *ACM Comput. Surv. 14*, 2, 287–313.

Foster, I. 2004. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, Los Altos, CA.

Glassman, S. 1994. A caching relay for the World Wide Web. In *Proceedings of the 1st International Conference on the World-Wide Web*. 165–173.

Graham, R. L. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math. 7*, 2, 416–429.

Hsu, W. W., Smith, A. J., and Young, H. C. 2005. The automatic improvement of locality in storage systems. *ACM Trans. Comput. Syst. 23*, 4, 424–473.

Huang, H., Hung, W., and Shin, K. G. 2005. FS2: dynamic data replication in free disk space for improving disk performance and energy consumption. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles*. 263–276.

Kangasharju, J., Roberts, J., and Ross, K. 2002. Object replication strategies in content distribution networks. *Comput. Comm. 25*, 4, 367–383.

Karlsson, M. and Karamanolis, C. 2004. Choosing replica placement heuristics for wide-area systems. In *Proceedings of the 24th International Conference on Distributed Computing Systems*. 350–359.

Kwan, T., McGrath, R., and Reed, D. 1995. Ncsas world wide web server design and performance. *Comput. 28*, 11, 67–74.

Lee, L. W., Scheuermann, P., and Vingralek, R. 2000. File assignment in parallel I/O systems with minimal variance of service time. *IEEE Trans. Comput. 49*, 2, 127–140.

Loukopoulos, T. and Ahmad, I. 2000. Static and adaptive data replication algorithms for fast information access in large distributed systems. In *Proceedings of the 20th International Conference on Distributed Computing Systems*. 385–392.

Loukopoulos, T., Lampsas, P., and Ahmad, I. 2005. Continuous replica placement schemes in distributed systems. In *Proceedings of the 19th Annual International Conference on Supercomputing*. 284–292.

Merialdo, P., Atzeni, P., and Mecca, G. 2003. Design and development of data-intensive web sites: The Araneus approach. *ACM Trans. Inter. Tech. 3*, 1, 49–92.

Nishikawa, N., Hosokawa, T., Mori, Y., Yoshida, K., and Tsuji, H. 1998. Memory-based architecture for distributed WWW caching proxy. In *Proceedings of the 7th International Conference on World Wide Web*. 205–214.

Pattipati, K. R. and Wolf, J. L. 1990. A file assignment problem model for extended local area network environments. In *Proceedings of the 10th International Conference on Distributed Computing Systems*. 554–561.

Qiu, L., Padmanabhan, V. N., and Voelker, G. M. 2001. On the placement of web server replicas. In *Proceedings of the 21th Annual Joint Conference on Computer and Communications*. 1587–1596.

Scheuermann, P., Weikum, G., and Zabback, P. 1998. Data partitioning and load balancing in parallel disk systems. *VLDB 7*, 1, 48–66.

Tang, X. and Xu, J. 2004. On replica placement for QoS-aware content distribution. In *Proceedings of the 23rd Annual Joint Conference on Computer and Communications*. 806–815.

TEWARI, R. 1992. Distributed file allocation with consistency constraints. In *Proceedings of the 12th International Conference on Distributed Computing Systems*. 408–415.

TRIANTAFILLOU, P., CHRISTODOULAKIS, S., AND GEORGIADIS, C. 2000. Optimal data placement on disks: A comprehensive solution for different technologies. *IEEE Trans. Knowl. Data Engin. 12*, 2, 324–330.

WEIL, S. A., BRANDT, S. A., MILLER, E. L., AND MALTZAHN, C. 2006. CRUSH: Controlled, scalable, decentralized placement of replicated data. In *Proceedings of the ACM/IEEE Conference on Supercomputing*. 122.

WOLFSON, O., JAJODIA, S., AND HUANG, Y. 1997. An adaptive data replication algorithm. *ACM Trans. Datab. Syst. 22*, 4, 255–314.

XIE, T. 2007. SOR: A static file assignment strategy immune to workload characteristic assumptions in parallel I/O systems. In *Proceedings of the 36th International Conference on Parallel Processing (ICPP)*.

XIE, T. 2008. SEA: A striping-based energy-aware strategy for data placement in RAID-structured storage systems. *IEEE Trans. Comput. 57*, 6, 748–761.

Received June 2008; revised April 2009; accepted April 2009