SOR: A Static File Assignment Strategy Immune to Workload Characteristic Assumptions in Parallel I/O Systems

Tao Xie
Department of Computer Science
San Diego State University
San Diego, California 92182
xie@cs.sdsu.edu

Abstract

The problem of statically assigning nonpartitioned files in a parallel I/O system has been extensively investigated. A basic workload characteristic assumption of 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, in this paper 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 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 and noticeably improves the performance in terms of mean response time over the existing schemes.

1. Introduction

Many real-world applications intensively read data stored in large-scale parallel I/O systems like RAID, Redundant Arrays of Inexpensive Disks [7]. To guarantee the quality of service demanded by end-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 [12][26]. Similarly, a data-intensive Web server application that publishes significant amounts of data stored in a back-end database must answer end-users' inquiries instantly before they lose patience [6][23]. 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 [12][15][16][20][29]. 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 [20][29]. 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 [10][32]. 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 [10]. 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 [10]. Thus, heuristics 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 on-line to adapt to varying

2007 International Conference on Parallel Processing (ICPP 2007)
0-7695-2933-X/07 $25.00 © 2007 IEEE

workload patterns without a prior knowledge of the files to be assigned in the future. In this paper, we address the problem of statically assigning nonpartitioned files in a parallel disk storage system where file accesses exhibit Poisson arrival rates and fixed service times.

Several previous studies [9][13] show that the distribution of web page requests generally follows a Zipf distribution [20] 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, i.e., 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 [14], SP [20], and HP [20] were developed to reduce parallel I/O systems' mean response times. 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 very weak that it can be ignored [4][24]. In other words, at least in some real-world applications, the workload assumption on the correlation does not hold. Therefore, it is necessary to re-examine 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 [14], SP [20], and HP [20] 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 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 paper 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 based on synthetic benchmarks. Section 5 concludes the paper with summary and future directions.

2. Related work

The *file assignment problem* (FAP) exists in a wide range of distributed systems including distributed file systems [29], distributed databases [33], video servers [27], content distribution networks [5] and the Grid [11]. The first research work on FAP dates back to late 1960s [8]. Since then FAP has been comprehensively investigated because the potential gain obtained by solving a FAP is significant [10]. 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 [8][10][17][21][25][28]. Greedy, originated from *longest processing time* (LPT) algorithm proposed by Graham in [14] is one of the most well-known static file assignment heuristic algorithms. Dynamic file assignment algorithms [25] [32], 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 [27], 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 recently to solve problems such as data object replica placement [18][22], data management for large-scale storage systems [2][20][31], and automatic near-optimal storage system designs [3]. 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. proposed a static file assignment algorithm called sort partition (SP) and a semi-dynamic file assignment algorithm named hybrid partition (HP) [20]. Compared with the traditional Greedy algorithm, SP significantly improves the mean response time by

taking the two minimizations into account simultaneously [20]. On the other hand, HP is a batch-based variant of SP, which can run in on-line mode. Based on our knowledge, SP is one of the best existing static file assignment algorithms so far.

All 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 [1][9][13]. However, some recent research projects [4][24] 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 [4][24] at least in some applications. Therefore, it is necessary to re-examine 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 with 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 performance compared with 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 paper, 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 with 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 or file replication 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. The SOR algorithm

A parallel I/O system in its most general form consists of a linked group, e.g., $D = \{d_1, ..., d_j, \ldots, d_n\}$, of independent homogeneous disk drives. The set of files can be represented as $F = \{f_1, \ldots, f_i, ..., f_m\}$. 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' heats 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 a set of rational parameters, e.g., $f_i = (s_i, \lambda_i, t_i, h_i)$, where s_i, λ_i, t_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 t_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 [19]. Second, for large files, when the access unit is the entire file, the seek times and rotation latencies are negligible compared with the transfer time. Thus, t_i is determined by s_i and t_j if f_i is allocated on d_j. 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. Since the combination of λ_i and t_i accurately gives the load of f_i, we define the heat h_i of f_i as follows [20]:

$$h_i = \lambda_i \cdot t_i. \quad (1)$$

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

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{m} h_i \quad (2)$$

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. 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 $R = \{r_1, ..., r_k, \ldots, r_u\}$. 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 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 t_i.

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

$$st_j(r_k) = \begin{cases} a_k, \text{if } d_j \text{ is idle and } Q_j \text{ is empty} \\ a_k + r_j, \text{if } d_j \text{ is busy and } Q_j \text{ is empty} \\ a_k + r_j + \sum_{r_p \in Q_j, a_p \le a_k} t_{fid_p}, \text{otherwise} \end{cases} \quad (3)$$

where r_j represents the remaining service time of a request currently running on d_j, and $\sum_{r_p \in Q_j, a_p \le a_k} t_{fid_p}$ 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

$$ft_j(r_k) = st_j(r_k) + t_{fid_k}, \quad (4)$$

, where t_{fid_k} is the service time of the file that request r_k targets on. As a result, the response time of r_k can be obtained by

$$rt_j(r_k) = ft_j(r_k) - st_j(r_k). \quad (5)$$

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

$$\text{mrt(R)} = \sum_{k=1,1 \le j \le n}^{u} rt_j(r_k) \Big/ u \cdot \quad (6)$$

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 expressed by Eq. 6.

Fig. 1 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 [20]. SOR takes these two critical aspects into account as well. Specifically, SOR computes the average disk load ρ in step 1 and enforces the load on each disk not to exceed ρ (Step 9). Step 7 sorts the file set F in file size so that files with similar sizes can be allocated onto the same disk, a clever strategy that was employed by SP and HP as well [20]. Besides, SOR separates the most popular files onto different disks by utilizing a round-robin

Input: A parallel I/O system *D* with *n* identical disks, a collection of *m* files in a queue *F*
Output: A file allocation matrix *X* (*n*, *m*)
1. Use **Eq. 2** to compute the average disk load
2. **for** each disk *j* **do**
3. $load_j = 0$; X(d_j, :) = 0
4. **end for**
5. $f_i = 1$
6. $d_j = 1$
7. Sort all files in *F* in ascending order of their service time s_i
8. **while** f_i • *m* **do**
9. **if** $load_j$ •
10. X(d_j, :) = f_i
11. $load_j = load_j + load(f_i)$
12. $f_i = f_i + 1$
13. **if** d_j = = (*n* -1)
14. $d_j = 1$
15. **else**
16. $d_j = d_j + 1$
17. **end if**
18. **else**
19. Search for a disk d_k from the rest disks to accommodate file f_i
20. **If** successful
21. X(d_k, :) = f_i
22. $load_k = load_k + load(f_i)$
23. $f_i = f_i + 1$
24. **else**
25. X(d_n, :) = f_i
26. $load_n = load_n + load(f_i)$

Fig. 1. The SOR algorithm.

manner rather than a consecutive allocation of a sorted file set, which was adopted by SP. 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 big sizes. Confining very big files in 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. The advantage of a round-robin file assignment strategy is that files with higher load (heat) values will be distributed onto distinct disks so that the overall load balancing could be further improved. 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. If a file f_i cannot be allocated onto disk d_j, SOR searches a disk d_k that is closet 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).

4. Performance evaluation

Now we are in a position to evaluate the effectiveness of the proposed SOR algorithm. To demonstrate the strength of SOR, we compare it with Greedy, SP, and HP. The three algorithms are briefly described below.

(1) *Greedy*: It can operate in either on-line mode or off-line mode. Here, we only consider its off-line mode because SOR is an off-line 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 Eq. 2. 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. For each batch, HP assigns files to disks in distinct allocation intervals. The algorithm selects, for each allocation intervals 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.

4.1. Simulation setup

We have developed an execution-driven simulator that models an array of conventional Cheetah ST39205LC disks. 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. Note that the mean response times are normalized in the scale [0, 1] for all graphs in this section.
- *Mean response time improvement*: decrease (in seconds) of mean response time gained by SOR compared with the three existing algorithms.
- *Mean slowdown*: the ratio between average request turnaround time and average request service time.

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.

Table 1 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 4.2 to Section 4.3 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.

Parameter	Value (Fixed) – (Varied)
Number of files	(5000)
File load (heat)	Each file imposes a load (heat) that is defined as $h_i = {}_i * t_r$.
Coverage of the file system	(100%) – each file is at least accessed once
Number of batches (HP)	(4) – each batch has 1250 files
Number of disks	(16) – (8, 12, 16, 20, 24)
Aggregate access rate (1/second)	(200) – (25, 50, 100, 200, 300, 400, 500, 600, 700, 800, 900,1000)

Table 1. System parameters.

4.2. Impact of aggregate access rate

The goal of this experiment is to compare the proposed SOR algorithm against the three well-known file assignment schemes, and to understand the sensitivity of the four heuristics to the aggregate access rate in a parallel I/O system, where an array of identical disk drives serve incoming requests simultaneously. The aggregate access rate varies from 25 (1/second) to 1000 (1/second) and the file sizes were distributed according to Zipf's law with skew degree 70:30.

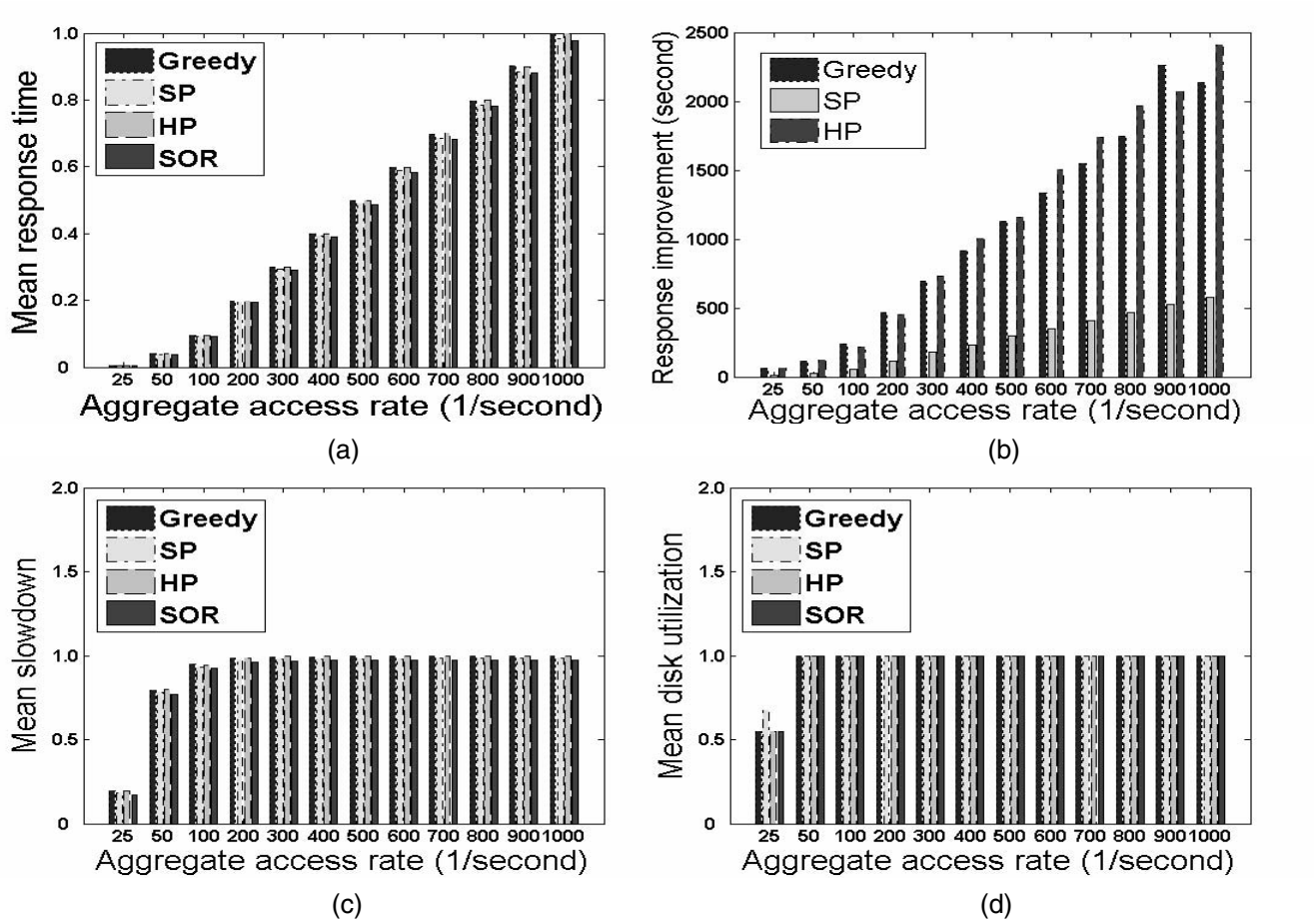

Fig. 4. Impact of aggregate access rate in Zipfian file size distribution.

Fig. 2 shows the simulation results for the four algorithms on a parallel I/O disk array with 16 disk drives. We observe from Fig. 2a 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, Fig. 2b provides mean response time decrease gained by SOR compared with Greedy, SP, HP, respectively. In particular, SOR can reduce mean response time on average by 1118.3, 1052.8, and 269.6 seconds, compared with HP, Greedy, and SP, respectively. An interesting observation is that the mean response time improvement becomes more significant when the overall 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 (Fig. 2c), which is consistent with the results shown in Fig. 2a. Since the total workload is relatively heavy, the mean disk utilization in Fig. 2d quickly arises to 1 when aggregate access rate is larger than 25 (1/second).

4.3. 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. Fig. 3 plots the performance of the four algorithms as functions of the number of disks. The results show that SOR exhibits a good scalability.

Fig. 3 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

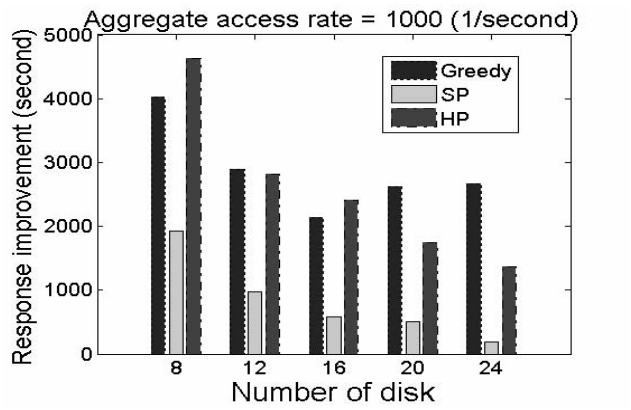

Fig. 3. Scalability test.

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. From Fig. 3 we can see that the mean response time improvement becomes more pronounced when the aggregate access rate is large.

5. Conclusions

In this paper, we address the issue of statically allocating non-partitioned 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 synthetic benchmarks. 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, SOR consistently improves the performance of parallel I/O systems in terms of mean response time over three well-known file assignment algorithms. Compared with SP, one of the best existing static non-partitioned file assignment algorithms, SOR achieves improvement in mean response time on averages of 269.6 seconds. The improvement of SOR in mean response time over Greedy and HP are 1052.8 seconds and 1118.3 seconds on average, respectively.

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 re-allocated 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.

Acknowledgements

This work was supported by the National Science Foundation under grant number CCF-0702781.

References

[1] V. Almeda, M. Cesario, R. Fonseca, W. Meira Jr. and C. Murta, "Analyzing the behaviour of a proxy server," *3rd Int'l WWW Caching Workshop*, 1998.

[2] G.A. Alvarez et al., "Minerva: An automated resource provisioning tool for large-scale storage systems," *ACM Trans. Computer Systems*, Vol. 19, No. 4, pp. 483 - 518, Nov. 2001.

[3] E. Anderson, S. Spence, R. Swaminathan, M. Kallahalla and Q. Wang, "Quickly finding near-optimal storage designs," *ACM Trans. Computer Systems*, Vol. 23, No. 4, pp. 337 - 374, Nov. 2005.

[4] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web Caching and Zip-like Distributions: Evidence and Implications," *IEEE INFOCOM*, pp. 126 - 134, 1999.

[5] S. Buchholz and T. Buchholz, "Replica placement in adaptive content distribution networks," *ACM Symp. Applied Computing*, pp. 1705 - 1710, 2004.

[6] E.V. Carrera, E. Pinheiro and R. Bianchini, "Conserving disk energy in network servers," *Proc. 17th Annual Int'l Conf. Supercomputing*, pp. 86 - 97, 2003.

[7] 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.

[8] W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Computers*, Vol. 18, No. 10, pp. 885 - 889, 1969.

[9] C. Cunha, A. Bestavros and M. Crovella, "Characteristics of WWW Client-based Traces," *Technical Report*, 1995-010, Boston University, 1995.

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

[11] I. Foster, "The Grid: Blueprint for a New Computing Infrastructure," *Morgan Kaufmann*, 2 Ed., 2004.

[12] S. Ghandeharizadeh, S.H. Kim, and C. Shababi, "On disk scheduling and data placement for video servers," *ACM Sigmetrics Performance Evaluation*, Vol. 23, Issue 1, pp. 37 - 46, 1995.

[13] S. Glassman, "A caching relay for the World Wide Web," *First conf. World-Wide Web*, pp. 165 - 173, 1994.

[14] R.L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal Applied Math.*, Vol. 7, No. 2, pp. 416 – 429, 1969.

[15] W.W. Hsu, A.J. Smith, and H.C. Young, "The automatic improvement of locality in storage systems," *ACM Transactions on Computer Systems*, Vol. 23, Issue 4, pp. 424 - 473, 2005.

[16] 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 SOSP*, pp. 263-276, 2005.

[17] J. Kangasharju, J. Roberts, and K. Ross, "Object replication strategies in content distribution networks," *Computer Communications*, Vol. 25, No. 4, pp. 367 - 383, 2002.

[18] M. Karlsson and C. Karamanolis, "Choosing replica placement heuristics for wide-area systems," *Proc. 24th Int'l Conf. Distributed Computing Systems*. pp. 350 - 359, 2004.

[19] 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.

[20] 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, Feb. 2000.

[21] T. Loukopoulos and I. Ahmad, "Static and adaptive data replication algorithms for fast information access in large distributed systems," *Proc. ICDCS*, pp. 385 - 392, April 2000.

[22] T. Loukopoulos, P. Lampsas and I. Ahmad, "Continuous replica placement schemes in distributed systems," *Proc. 19th Int'l Conf. Supercomputing*, pp. 284 - 292, 2005.

[23] P. Merialdo, P. Atzeni, and G. Mecca, "Design and development of data-intensive web sites: The Araneus approach," *ACM Transactions on Internet Technology*, Vol. 3, Issue 1, pp. 49 - 92, Feb. 2003.

[24] N. Nishikawa, T. Hosokawa, Y. Mori, K. Yoshida and H. Tsuji, "Memory-based architecture for distributed WWW caching proxy," *Proc. 7th Int'l Conf. World Wide Web*, pp. 205 - 214, 1998.

[25] L. Qiu, *et al.*, "On the placement of web server replicas," *Proc. IEEE INFOCOM*, pp. 1587 - 1596, April 2001.

[26] N.J. Sarhan and C.R. Das, "Adaptive Block Rearrangement Algorithms for Video-On-Demand Servers," *Proc. 30th International Conference on Parallel Processing*, pp. 452 -462, 2001.

[27] P. Scheuermann, G. Weikum, and P. Zabback, "Data Partitioning and Load Balancing in Parallel Disk Systems," *VLDB.*, Vol.7, No.1, pp. 48 - 66, 1998.

[28] X. Tang and J. Xu, "On replica placement for QoS-aware content distribution," *23rd Annual Joint Conf. IEEE Computer and Communications Societies (INFOCOM)*, Vol. 2, pp. 806 - 815, Mar. 2004.

[29] R. Tewari, "Distributed file allocation with consistency constraints," *Proc. ICDCS*, pp. 408 - 415, June 1992.

[30] P. Triantafillou, S. Christodoulakis, and C. Georgiadis, "Optimal data placement on disks: a comprehensive solution for different technologies," *IEEE Trans. Knowledge Data Eng..*, Vol. 12, Issue. 2, pp. 324 - 330, 2000.

[31] S.A. Weil, S.A. Brandt, E.L. Miller and C. Maltzahn, "CRUSH: controlled, scalable, decentralized placement of replicated data," *Proc. ACM/IEEE conf. Supercomputing*, No. 122, 2006.

[32] J. Wolf, K. Pattipati, "A File Assignment Problem Model for Extended Local Area Network Environments," *Proc. 10th Int'l Conf. Distributed Computing Systems*, pp. 554 - 561, 1990.

[33] O. Wolfson, S. Jajodia, and Y. Huang, "An adaptive data replication algorithm," *ACM Trans. Database Systems*, Vol. 22, No. 4, pp. 255 - 314, 1997.