A Static Data Placement Strategy towards Perfect Load-Balancing for Distributed Storage Clusters

Deepthi K.Madathil, Rajani B. Thota, Paulina Paul, Tao Xie

Department of Computer Science, San Diego State University
5500 Campanile Drive, San Diego, CA 92182, USA

Deepthi-madathil@rohan.sdsu.edu
Rajani-thota@rohan.sdsu.edu
ppaul@rohan.sdsu.edu
xie@cs.sdsu.edu

Abstract— **Applications like cluster-based Video-On-Demand (VOD) systems are inherently data-intensive because clients frequently retrieve data stored on a distributed storage subsystem interconnected by a high-speed local network. To meet the Quality-of-Service (QoS) imposed by the clients, quick responses to access requests are fundamental for these applications. Among the numerous ways to reduce response times, data placement, has attracted much attention from researchers due to its effectiveness and low cost. In this paper, we propose a novel load-balancing and performance oriented static data placement strategy, called *p*erfect *b*alancing (PB), which can be applied to distributed storage subsystems in clusters to noticeably improve system responsiveness. The basic idea of PB is to balance the load across local disks and to minimize the discrepancy of service times of data on each disk simultaneously. A comprehensive experimental study shows that PB reduces mean response time up to 19.04% and 8.67% over the two well-known data placement algorithms Greedy and SP respectively.**

I. Introduction

Due to their cost-effectiveness and good scalability, cluster-based data-intensive applications such as Video-on-Demand clusters and Web servers have become increasingly popular [23][31]. A common requirement imposed by the end-users of these applications is prompt response. For example, a Video-On-Demand (VOD) cluster has to quickly respond to access requests from multiple users so as to provide them with continuous glitch-free video [13][25]. 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 [4][23]. It is easy to understand that reducing mean response time of customer requests is mandatory for these applications.

There are a wide spectrum of different ways of reducing the mean response time or improving the system throughput for cluster-based applications [1][13][16][21][22]. Data placement, or file assignment, allocating of data onto an array of disks before they are accessed, is one of such avenues that can significantly affect the overall performance of a parallel I/O system [1][22][28]. In order to fully exploit the capacities of a parallel disk storage system, data placement algorithms for parallel disk systems have been extensively investigated in the literature [1][5][7][10][22]. Generally, these algorithms place data onto a parallel disk array so that a special cost function or performance metrics can be optimized. While common cost functions include communication costs, storage costs, and queuing costs, popular performance metrics are mean response time and overall system throughput [11]. It is well-known that finding the optimal solution for a cost function or a performance metric in the context of data placement on multiple disks is an NP-complete problem [11]. Thus, heuristic algorithms became practical solutions.

Typically, heuristic data placement algorithms fall into two categories: static and dynamic. Most static data placement algorithms require complete knowledge about the workload statistics such as service times and access rates of all the files. Dynamic data placement algorithms, on the other hand, generate file-disk allocation schemes on-line to adapt to varying 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 non-partitioned files in a distributed storage subsystem where file accesses exhibit Poisson arrival rates and fixed service times. Although each node in a cluster might only have one hard disk drive, the entire disk set in the cluster can be viewed as a virtual disk array. Therefore, traditional data placement algorithms like Greedy [15] and SP [22] are still valid in cluster-based date-intensive applications.

Since load-balancing is a key step towards low mean response times, existing algorithms strived to achieve an ideal load balance across disks. 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. The load, also called the heat, of a file f_i is defined as follows [22]:

$$h_i = \lambda_i * t_i\ . \tag{1}$$

This is because the combination of a file's access rate and its service time accurately gives the load of the file. Greedy, one of the most commonly used data placement heuristics, maintains load-balancing by evenly distributing files onto multiple disks so that each disk has a similar load, which is the sum of the heat of all files on that disk. Although Greedy achieved an overall load-balancing across an array of disks, it

978-1-4244-1694-3/08/$25.00 ©2008 IEEE

overlooked the fact that the variance of the service time at each disk could significantly affect the mean response time [22]. Based on this important observation, Lee *et al.* proposed a static data placement algorithm, called Sort Partition (SP) in [22]. Initially, all files are sorted in a list I in descending order of their service times. Each disk d_k is assigned the next contiguous segment from the ordered list I until its load $load_k$ reaches the average disk utilization ρ, which is defined as:

$$\rho = \frac{1}{n} \cdot \sum_{i=1}^{m} h_i \, , \tag{2}$$

where m is the number of files and n is the number of disks. This way the load is distributed among the disks evenly. In fact, it allocates files with similar service times onto one disk while keeping load balancing across all disks. The rationale behind SP is to minimize the load variance across all disks, as well as the variance of the service time at each disk simultaneously. Performance evaluation provided by [22] demonstrated that SP noticeably improve performance in terms of mean response time. To the best of our knowledge, SP is one of the best static data placement algorithms reported in the literature so far.

However, there are two obvious drawbacks in the SP algorithm. First of all, when SP assigns files onto the last disk d_z, it allocates all the remaining files in the list I onto it. As a result, the load of disk d_z is observably higher than that of the rest disks. Thus, the principle of load balancing across all the disks is violated. Second, SP algorithm is based on two important workload characteristic assumptions: the file access rate obeys a Zipf-like distribution and the file access frequency is inversely correlated to the file size. In a Zipf-like distribution, the relative probability of a request for the i'th most popular file is proportional to $1/i^{\alpha}$, with α typically varying between 0 and 1. In other words, the most popular files are typically small in size, while the large files are relatively unpopular. These two workload assumptions were supported by several early studies on web requests [2][9][14]. It is very difficult for the SP algorithm to reach an ideal load balancing across the disks because the most popular files could overload some disks due to their relatively large values of heat. In other words, the granularity of heat of these files is large, which makes disks having these files have a load obviously larger than the average disk utilization ρ. Again, the load balancing across the disks is affected.

Motivated by the above insightful observations made by our research, we propose a new data placement strategy, called perfect balancing (PB), which overcomes the two drawbacks of SP. The PB strategy avoids the first drawback of SP by assigning the remaining files onto a subset of the disks where the average file service times are similar to that of the remaining files. In addition, PB distributes the most popular files onto different disks such that they cannot overload a particular disk.

The rest of the paper is organized as follows. In the next section we discuss the related work. In Section 3, we formulate the problem and present the PB strategy as well as the two 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.

II. Related Work

The *data placement*, or *file assignment problem* (FAP) exists in a wide range of distributed systems including distributed file systems [27], distributed databases [30], video servers [31], content distribution networks [6] and the Grid [12]. 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 [11]. Typically, solutions to FAP fall into two camps: 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][11][17]. Greedy, originated from *longest processing time* (LPT) algorithm proposed by Graham in [15] is one of the most well-known static file assignment heuristic algorithms. Dynamic file assignment algorithms [24][29], 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 [26][31], 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][20], data management for large-scale storage systems [3][22], and automatic near-optimal storage system designs [4]. 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) [22]. Compared with the traditional Greedy algorithm, SP significantly improves the mean response time by taking the two minimizations into account simultaneously [22]. 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.

Without loss of generality, we assume that (1) each data is viewed as an independent file; (2) communication delays between any pair of disks are identical and negligibly small [22]; (3) disk access (read) to each data is modelled as a Poisson process with a mean access rate λ_i; (4) a fixed service time s_i for each data; for example, each read on a data results in a sequential scan of the entire data. For large size data, this assumption is valid because when the basic unit of data access is entire data, seek time, rotation latency, and controller overhead are negligible in comparison with data transfer time; (5) the distributed disk storage system in the cluster is homogeneous in the sense that each disk has the same performance metrics.

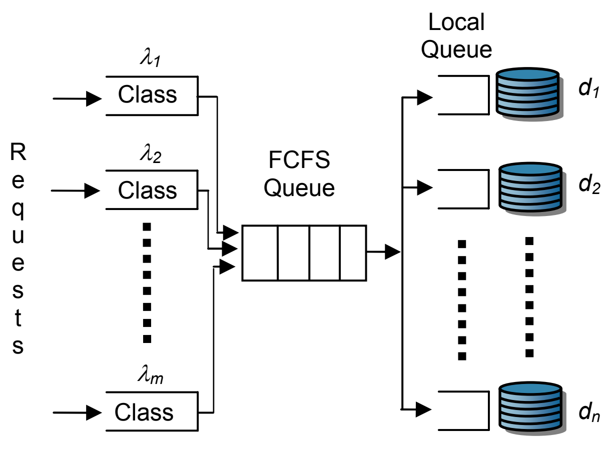

Fig. 1. System architecture

III. Design and implementation of PB

In this section, we first formulate the FAP problem and present the system model, which is followed by a detailed description of the PB algorithm. Then we prove the time complexity of PB.

A. Problem Formulation and System Architecture

A distributed storage system consists of a linked group, e.g., $D = \{d_1, ..., d_j, ..., d_n\}$, 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.

The architecture of the data placement problem is illustrated in Fig. 1. Note that in this paper we assume that each node in a cluster only has one hard disk drive attached. Also, each node maintains a Local Queue, where requests target on files on that node are stored in their arrival order. Data placement algorithms like Greedy, SP, and PB allocate a group of files onto a set of identical disks so that the mean response time can be minimized. Fig. 1 depicts the subsequent file access requests scheduling process after the file assignment process completes. 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 $R = \{r_1, ..., r_k, ..., r_u\}$. 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 by data placement algorithms 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 (see Fig. 1).

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

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

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

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

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

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

The FAP problem now can be formulated as: given a set of files F and a parallel disk set D, find an allocation scheme X that optimizes the mean response time expressed by Eq. 6.

B. Algorithm Description

Fig. 2 outlines the PB 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 [22]. PB takes these two critical aspects into account as well. Specifically, PB computes the average disk load ρ in step 3 and enforces the load on each disk not to exceed ρ (Step 7). Step 4 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 as well [22]. Besides, PB separates the most popular n-1 files onto different disks rather than a consecutive allocation of a sorted file set, which was adopted by SP (Step 5). The reason why we exclude disk d_n out of Step 5 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 dispersing the n-1 most popular

```
Input:    $n$ = number of disks, $m$ = number of files, $h_i$ = heat of file $f_i$, $s_i$ = expected service time of file $f_i$
Output:   A file allocation matrix $X(n, m)$
Step 1-   Compute the heat $h_i$ for each file $f_i$
Step 2-   Initialize the load and the allocation map for each disk
          for each disk $d_j$ do
                  $load_j = 0$; $X(d_j, :) = 0$
          end for
Step 3-   Use Eq. 2 to compute the average disk load $\rho$
Step 4-   Sort the service time of all files in ascending order
Step 5-   Assign the most popular $n$-1 files $\{f_1, f_2, .., f_{n-1}\}$ to disks $\{d_1, d_2, …, d_{n-1}\}$, respectively
          $f_i = 1$
          for ( $d_j = 1$; $d_j \le n-1$; $d_j$++)  do
                  $load_j = load_j + h_i$
                  $X(d_j, :) = f_i$
                  $f_i = f_i + 1$
         end for
Step 6-   Assign the most unpopular files $\{f_k, f_{k+1}, .., f_m\}$ onto the disk $d_n$ until its load reaches $\rho$
Step 7-   Assign the rest files to disks until the disk load reaches the predetermined threshold value $\rho$
          $f_i = n$;
          for ( $d_j = 1$; $d_j \le n-1$; $d_j$++)  do
              $load_j = 0$;
              while ($load_j \le \rho$ and $f_i \le k$) do
                  $X(d_j, :) = f_i$                    // File $f_i$ is allocated on disk $d_j$
                  $load_j = load_j + h_i$
                  $f_i = f_i + 1$
               end while
           end for
Step 8-   Assign the remaining files to disks in the range $\{d_1, …, d_{n-1}\}$
           if ($f_i \le k$)
              for each remaining file do
                  Find the lowest loaded disk $d_j$ where $1 \le j \le n-1$
                  $X(d_j, :) = f_i$ // File $f_i$ is allocated on disk $d_j$
                  $load_j = load_j + h_i$
              end for
          end if
```

Fig. 2. The PB algorithm

files is that files with high load (heat) values will be distributed onto distinct disks so that the overall load balancing could be further improved. Moreover, the most unpopular files with large sizes will be allocated on the last disk d_n (Step 6). PB overcomes a hidden drawback of SP where the allocation of the remaining files onto the last disk makes it overloaded. In stead of allocating all the remaining files on the last disk, for each remaining file PB assigns it onto the disk within the range $[d_1, d_{n-1}]$, which currently has lowest load (Step 8). This way the load of the remaining files can be evenly shared by the n-1 disks.

C. Time Complexity Analysis

Before qualitatively comparing our scheme with two existing algorithms, we demonstrate the time complexity of the PB algorithm.

Theorem 1. Given a parallel disk array system $D = \{d_1, d_2, ..., d_j, ..., d_n\}$ and a collection of files represented by a file queue $F = (f_1, f_2, ..., f_i, …, f_m)$, the time complexity of PB is $O(mlgm+2m+n)$, where m is the number of files in Q, n is the number of disk in the system P.

Proof. It takes $O(m)$ time to compute the heat h_i for each file f_i (Step 1). Also, it takes $O(n)$ time to initialize the load and the allocation map for each disk (Step 2). The time complexity of sorting the file set F is $O(mlgm)$ as we have m files (Step 4). Step 5 takes time $O(n-1)$ to be completed. Similarly, Step 6 consumes time $O(m-k+1)$. Step 7 takes $O(k-n)$ to assign the rest files on disks. To discover an appropriate disk in $\{d_1, …, d_{n-1}\}$ for a remaining file to be allocated, the worst case for PB is to visit each of the $n-1$ disks in $\{d_1, …, d_{n-1}\}$ (Step8). Consequently, the worst case time complexity for allocating one remaining file is $O(n-1)$ (see Step 8). Assume we have w remaining files ($1 \le w \le\le m$). Thus, Step 8 takes $O(w(n-1))$. Since w is far smaller than m and n is usually a very small number compared with m, $O(w(n-1))$ is much smaller than $O(m)$. Therefore, time complexity of Step 8 can be ignored.

Other steps simply take $O(1)$ time. Hence, the worst-case time complexity of the PB algorithm is: $O(m)+O(n) + O(mlgm)+ O(n-1)+O(m-k+1)+O(k-n) = O(mlgm+2m+n)$. □

Theorem 1 indicates that the time complexity of the PB algorithm is typically low. For example, in our experiments, the value of m is set to 5000 and the value of n is in the range [8, 32], which should take less than hundreds of microseconds to complete the PB algorithm in modern processors. An implication of Theorem 1 is that PB has potential to be extended to be applied in real-world applications because of its low complexity.

IV. Simulation Study

Now we are in a position to evaluate the effectiveness of the proposed PB data placement scheme using extensive simulations. The advantage of using simulation is that we can easily vary parameters, which is a key component of this paper. The two baseline algorithms that were used to compare with our PB strategy are briefly described below.

(1) *Greedy*: The most common heuristic for multiple disks load balancing. It can operate in either on-line mode or off-line mode. Here, we only consider its off-line mode because PB 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.

A. 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 I.

TABLE I
MAIN CHARACTERISTICS OF THE CHEETAH DISK

Description	Value
Disk model	Seagate Cheetah ST39205LC
Storage capacity	9.17 GBytes
Average seek time	5.4 msecs
Average rotation time	3 msecs
Standard interface	SCSI
Rotational speed	10000 rpm
Number of platters	1
Transfer rate	31 Mbytes/second

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 distributed storage cluster.
- *Mean response time improvement*: decrease in percentage of mean response time gained by PB compared with the two existing algorithms.

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.

1. *Number of files*: Since the total number of files to be assigned onto a parallel disk array directly determines the disk array's load, we set it to 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.

2. *Request rate*: Each file access represents a sequential read of the entire file. Hence, the service time of a file access request is proportional to the file's size. We assume that each file has a fixed request arrival rate λ_i and the arrival interval times are exponentially distributed. The aggregate arrival rate of the entire system is defined as $\sum_{i=1}^{5000} \lambda_i$. 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.

3. *File popularity weight*: File popularity weight relates to the frequency with which file requests arrive at the distributed storage cluster. 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 [22]. The value of $X{:}Y$ is called skew degree in this paper and $\alpha = 1 - \theta$ (see Section 1 for α). Fig. 3 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 70:30.

4. *File size distribution*: 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 Fig. 3. The file size distribution is reasonable because the phenomena that popular files are generally small ones can be frequently observed.

5. *Coverage of the system:* The file system coverage is defined as the percentage of the entire file repository that is actually accessed by the request workload. We set the coverage of the system to 100% in our simulations, which means all files in the system are accessed at least once.

TABLE II
CHARACTERISTICS OF SYSTEM PARAMETERS

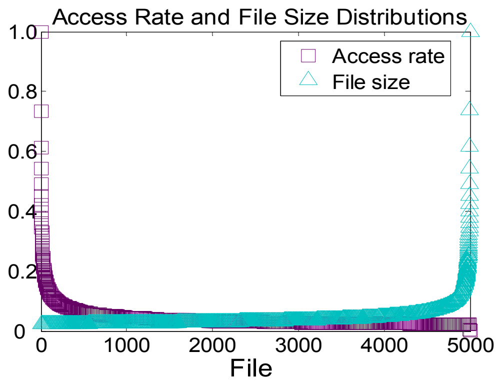

Fig. 3. Access rate and file size distributions

Parameter	Value (Fixed) – (Varied)
Number of files	(5000)
Skew degree	(70:30) – (50:50, 60:40, 70:30)
Coverage	(100%)
Number of disks	(16) – (8, 12, 16, 20, 24,28,32)
Aggregate access rate	(10) – (10, 15, 20, 25, 30, 35, 40, 45, 50) (1/second)
Simulation duration	(1000) seconds

Table II summarizes the configuration parameters of a simulated distributed storage cluster used in our experiments and characteristics of the synthetic workload. All synthetic workload used from Section IV.B to Section IV.D 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.

B. Impact of Aggregate Access Rate

The goal of this experiment is to compare the proposed PB algorithm against the two well-known file assignment schemes, and to understand the sensitivity of the three heuristics to the aggregate access rate in a distributed storage cluster, where an array of identical disk drives serve incoming requests simultaneously. The aggregate access rate varies from 10 (1/second) to 50 (1/second) and the file sizes were distributed according to Zipf's law with skew degree 70:30.

Fig. 4 shows the simulation results for the three algorithms on a simulated distributed storage cluster with 16 disk drives (nodes). We observe from Fig. 4a that PB consistently outperforms the two exiting approaches in terms of mean response time. This is because PB 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

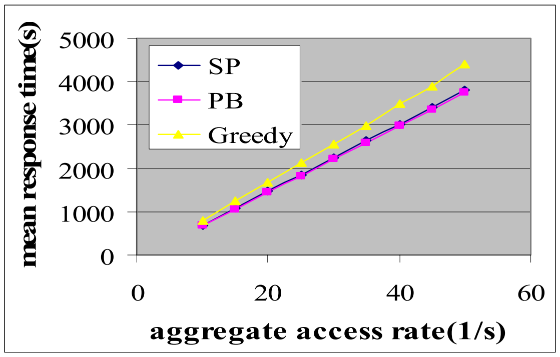

Fig. 4. Impact of aggregate access rate

TABLE III
DETAILED COMPARISONS IN AGGREAGE ACCESS RATE

Access rate	Greedy	SP	PB	Improved over (SP)	Improved over (Greedy)
10	797.76	690.78	678.41	1.79%	14.96%
15	1241.48	1077.94	1060.59	1.61%	14.57%
20	1669.48	1469.33	1446.61	1.55%	13.35%
25	2116.73	1855.27	1826.31	1.56%	13.72%
30	2555.54	2247.66	2211.35	1.62%	13.47%
35	2977.21	2634.41	2592.83	1.58%	12.91%
40	3488.79	3023.85	2975.10	1.61%	14.72%
45	3898.65	3407.68	3356.71	1.49%	13.90%
50	4397.67	3802.93	3741.20	1.62%	14.93%

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, Table III provides mean response time decrease gained by PB compared with Greedy and SP, respectively. In particular, PB can reduce mean response time on average by 14.1%nd 1.1%, compared with Greedy and SP, respectively.

C. Scalability

This experiment is intended to investigate the scalability of the three algorithms. We scale the number of disks in the system from 8 to 32. The aggregate access rate is configured to 10 (1/second). The skew degree is still set to 70:30. Fig. 5 plots the performance of the three algorithms as functions of the number of disks. The results show that PB exhibits a good scalability.

Fig. 5 shows that all of the three algorithms deliver better performance in mean response time 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 PB outperforms the rest two approaches in all tested cases. Comparing the results from Table IV, we can see that the mean response time improvement becomes more pronounced when the number of disks becomes larger. In addition, the implication of results from Table IV is that PB scales well when the number of disk in a cluster increases. Another observation is that the number of disks significantly

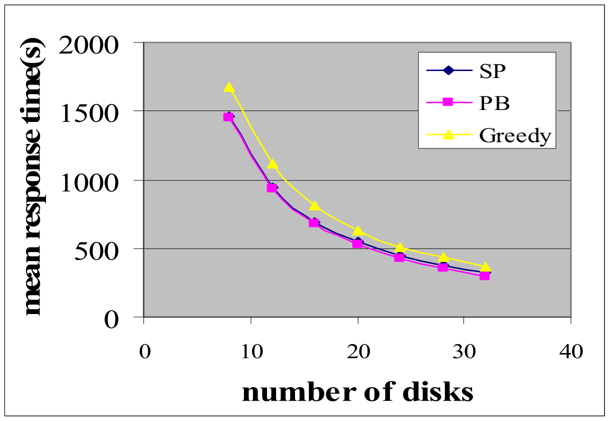

Fig. 5. Impact of the number of disks

TABLE IV

DETAILED COMPARISONS IN NUMBER OF DISKS

Disk number	Greedy	SP	PB	Improve (SP)	Improve (Greedy)
8	1678.14	1457.27	1447.89	0.64%	13.72%
12	1113.20	947.37	934.45	1.36%	16.06%
16	809.67	689.91	678.02	1.72%	16.26%
20	626.18	544.28	526.78	3.21%	15.87%
24	504.81	445.13	424.36	4.66%	15.94%
28	432.49	370.62	350.46	5.44%	18.96%
32	364.99	323.56	295.48	8.67%	19.04%

affects system performance and our PB scheme can better exploit the increased number of disks when compared to Greedy and SP.

Table IV gives detailed experimental result comparisons among the three algorithms. Clearly, the improvement gained by PB comparing SP noticeably escalates when the system was scaled up. In short, compared with Greedy and SP, PB improves mean response time up to 19.04% and 8.67%, respectively.

D. *Impact of Skew Degree*

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, PB degraded to SP and Greedy in terms of mean response time (Fig. 6). 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, PB can reduce mean response time by 1.67% and 16.26%, compared with SP and Greedy, respectively. We observe from Fig. 6 that PB achieves the best mean response time improvement when the skew degree is 70:30. In Fig. 6, the “1”, “2”, and “3” on X axis represents skew degree 50:50, 60:40, and 70:30, respectively.

V. CONCLUSIONS

In this paper, we address the issue of statically allocating non-partitioned files onto a distributed storage cluster where the file access requests exhibit Poisson arrival rates and fixed service times. We found that the performance of existing data placement algorithms can be observably improved by improving load-balancing across the disks. Therefore, a *p*erfect *b*alancing (PB) data placement strategy is developed to generate optimized file allocations that minimize mean response time. To quantitatively evaluate the effectiveness and practicality of the proposed PB 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 (Fig. 3), PB consistently improves the performance of distributed storage clusters in terms of mean response time over two well-known file assignment algorithms. In comparison PB, achieves improvement in mean response time by up to 8.67% over one of the best existing static non-partitioned file assignment algorithms SP. The improvement of PB in mean response time over Greedy on average is 14.06%.

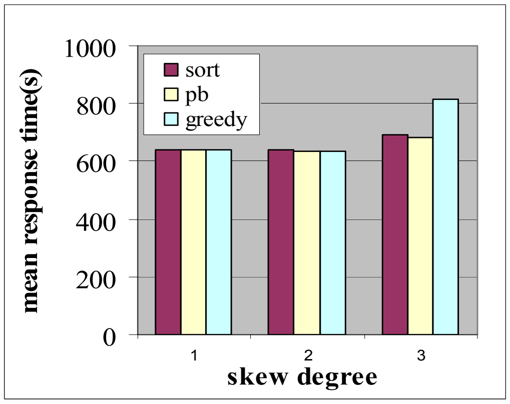

Fig. 6. Impact of skew degree

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 PB 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.

ACKNOWLEDGMENT

This work was supported by the National Science Foundation under grant number CCF-0742187.

REFERENCES

[1] Akyürek and K. Salem, “Adaptive block rearrangement,” *ACM Trans. Computer Systems*, Vol. 13, Issue 2, pp. 89-121, 1995.

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

[3] 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.

[4] 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.

[5] A. Brinkmann, K. Salzwedel, and C. Scheideler, "Efficient, distributed data placement strategies for storage area networks," *Proc. 12th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 119-128, 2000.

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

[7] Y. Cho, M. Winslett, Y. Chen, and S.W. Kuo, "Parallel I/O performance of fine grained data distributions," *Proc. 7th Int'l Symp. High Performance Distributed Computing*, pp. 163-170, 1998.

[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] C.H.Q. Ding and Y. He, "Data organization and I/O in a parallel ocean circulation model," *Proc. 13th Annual Int'l Conf. Supercomputing*, 1999.

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

[12] I. Foster, "The Grid: Blueprint for a New Computing Infrastructure," *Morgan Kaufmann*, 2 Ed., 2003.//change

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

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

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

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

[21] J.N. Matthews, D. Roselli, A.M. Costello, R.Y. Wang, and T.E. Anderson, "Improving the performance of log-structured file systems with adaptive methods," *Proc. 16th ACM Symposium on Operating Systems Principles*, pp. 238-251, 1997.

[22] 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, Issue 2, pp. 127-140, 2000.

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

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

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

[26] 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.

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

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

[29] 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.

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

[31] X. Zhou and C. Xu, "Optimal video replication and placement on a cluster of video-on-demand servers," *Proc. Int'l Conf. Parallel Processing (ICPP)*, pp. 547 - 555, 2002.