File Assignment in Parallel I/O Systems with Minimal Variance of Service Time

Lin-Wen Lee, Peter Scheuermann, *Senior Member*, *IEEE*, and
Radek Vingralek, *Member*, *IEEE Computer Society*

Abstract—We address the problem of assigning nonpartitioned files in a parallel I/O system where the file accesses exhibit Poisson arrival rates and fixed service times. We present two new file assignment algorithms based on open queuing networks which aim at minimizing simultaneously the load balance across all disks, as well as the variance of the service time at each disk. We first present an off-line algorithm, Sort Partition, which assigns to each disk files with similar access time. Next, we show that, assuming that a perfectly balanced file assignment can be found for a given set of files, Sort Partition will find the one with minimal mean response time. We then present an on-line algorithm, Hybrid Partition, that assigns groups of files with similar service times in successive intervals while guaranteeing that the load imbalance at any point does not exceed a certain threshold. We report on synthetic experiments which exhibit skew in file accesses and sizes and we compare the performance of our new algorithms with the vanilla greedy file allocation algorithm.

Index Terms—File allocation, parallel I/O systems, load balancing, variance of service time, heuristic algorithms.

1 Introduction

Parallel I/O systems have been the object of substantial interest in recent years due to the explosive growth and availability of RAID, Redundant Arrays of Inexpensive Disks [3]. Disk arrays partition data across multiple disks and access them in parallel in order to achieve higher transfer rates for large data accesses, like those encountered in supercomputing applications, and higher I/O rates on small data accesses, like those typical in transaction processing. Most importantly, the commercial success of RAID has been ensured by the incorporation of efficient techniques for achieving reliability based on mirroring or error correcting codes.

While the partitioning of a file determines the degree of parallelism in servicing a single request to the file, the allocation of all the files (partitions) onto the disks is an equally important parameter that affects the overall performance of a parallel I/O system. In order to fully benefit from the performance capabilities of a parallel I/O system, it has been widely recognized that the load must be uniformly distributed among all disks. Otherwise, the creation of performance bottlenecks on some of the disks may severely limit the response time of requests, as well as the overall system throughput.

Algorithms for assigning data to disks in parallel or distributed systems have been extensively studied in literature [8], [24], [6], [7], [14], [25], [26], [20], [18], [23]. Typically, these algorithms assign the data to the disks of a parallel or distributed system in such a way that a particular cost function is minimized. In the most general case, the cost function may involve communication costs, storage costs, update costs, and queuing costs. However, finding the optimal solution, even for very simple cost functions, is an NP-complete problem [8]. Consequently, viable solutions must be based on heuristics.

The heuristic methods that aim at optimizing the mean response time or the system throughput concentrate on minimizing the queuing delays on the disks. Since communication delays are usually negligible in comparison with disk access times, they can be safely omitted. Minimizing the queuing delays can be achieved by minimizing the utilization of each disk and by minimizing the variance of service times at each disk. Most of the published work concentrates on minimizing the disk utilization by balancing the system load across all disks and neglects the minimization of the variance of the service time. As the following example shows, in addition to load balancing, the performance of a parallel or distributed system can be significantly improved by reducing the variance of service times at each disk.

Example 1. Assume a parallel I/O system with two identical disks. The data to be distributed among both disks consists of $n = 1,000$ files. Among the 1,000 files, $n_A = 800$ belong to Class A and $n_B = 200$ to Class B. We assume that entire files are accessed as a unit. Each file in Class A has a mean access rate of $\lambda_A = 2$ accesses per minute and a service time $s_A = 20$ ms, while each file in class B has a mean access rate of $\lambda_B = 1$ access per minute and a service time $s_B = 120$ ms. The discrepancy in service time between the two classes might be either due to the more complex nature of class B accesses or simply to the fact that class B has larger files. For

- *L.W. Lee is with Linwen Associates, Inc., 1715 RFD, Long Grove, IL 60047. E-mail: linwen@worldnet.att.net.*
- *P. Scheuermann is with the Department of Electrical and Computer Engineering, Northwestern University, Evanston, IL 60208. E-mail: peters@ecs.nwu.edu.*
- *R. Vingralek is with InterTrust Technologies Corp., 4750 Patrick Henry Dr., Santa Clara, CA 95054-1851. E-mail: rvingral@intertrust.com.*

Manuscript received 15 Sept. 1998; accepted 4 Sept. 1999.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102101.

simplicity of presentation, we will assume the latter. Consider two assignments of files to disks 1 and 2:

Assignment 1: Place 400 class A files and 100 class B files at each of the two disks.

Assignment 2: Place all class A files at disk 1 and all class B files at disk 2.

Assignment 1 assigns the same set of files to each disk. Consequently, it leads to a perfect load balance among disks 1 and 2. The utilization of each disk is $\rho = n_A \cdot \lambda_A \cdot s_A/2 + n_B \cdot \lambda_B \cdot s_B/2 = 0.46$. On the other hand, assignment 2 leads to an imbalanced load: the utilization of disk 1 is $\rho_1 = n_A \cdot \lambda_A \cdot s_A = 0.53$ and the utilization of disk 2 is $\rho_2 = n_B \cdot \lambda_B \cdot s_B = 0.4$. Nevertheless, as we show below, assignment 2 results in a better mean system response time than assignment 1.

Under assignment 1, both disks can be modeled as M/G/1 queues. The mean response time of each disk is given as

$$E(r) = E(s) + \frac{\lambda \cdot E(s^2)}{2 \cdot (1-\rho)} = 58.6 \text{ ms},$$

where $\lambda = n_A \cdot \lambda_A/2 + n_B \cdot \lambda_B/2$ is the mean arrival rate of accesses to each disk and $E(s) = p_A \cdot s_A + p_B \cdot s_B$, $E(s^2) = p_A \cdot s_A^2 + p_B \cdot s_B^2$ are the first two statistical moments of service times at each disk, where $p_A = (n_A \cdot \lambda_A/2)/(n_A \cdot \lambda_A/2 + n_B \cdot \lambda_B/2)$ and $p_B = 1 - p_A$.

Under assignment 2, the disks are modeled as M/D/1 queues with utilizations ρ_1 and ρ_2, respectively, and service times s_A and s_B, respectively. The mean response time of disk 1 is given as

$$E(r_1) = s_A + \frac{\rho_1 \cdot s_A}{2 \cdot (1-\rho_1)} = 31.3 \text{ ms}$$

and, similarly, the mean response time of disk 2 is given as

$$E(r_2) = s_B + \frac{\rho_2 \cdot s_B}{2 \cdot (1-\rho_2)} = 160.0 \text{ ms}.$$

Consequently, the mean system response time is given as

$$E(r) = p_1 \cdot E(r_1) + p_2 \cdot E(r_2) = 45.6 \text{ ms},$$

where $p_1 = n_A \cdot \lambda_A/(n_A \cdot \lambda_A + n_B \cdot \lambda_B)$ and $p_2 = 1 - p_1$.

The above example suggests that, in the presence of a multiclass workload, the minimization of the variance of the service time on each disk of a parallel I/O system is an equally important issue as load balancing in order to optimize the system response time. Minimizing service time variance is also important for capacity planning for the network infrastructure, which connects the disks (and servers) to clients consuming the data [21].

In order to address these issues, we designed two new file assignment algorithms based on open queuing networks which aim at simultaneously minimizing the load balance across all disks, as well as the variance of the service time at each disks. Without restriction of generality, we assume that each file is allocated in its entirety to one disk. We present first an off-line algorithm, Sort-Partition, which assigns to each disk files with similar service times. We then present an on-line algorithm, Hybrid Partition, which assigns to a given disk groups of files with similar service times in successive intervals, while guaranteeing that the load imbalance at any point does not exceed a certain threshold. Our discussion is oriented toward parallel I/O systems due to the fact that we had a parallel I/O system prototype, namely FIVE [28], on which we could run our experiments. However, our algorithms are also applicable to distributed file systems.

The rest of this paper is organized as follows: In Section 2, we survey the related work. In Section 3, we describe our model, present two new heuristic file assignment algorithms, Sort Partition and Hybrid Partition, and prove optimality of Sort Partition in a constrained model which assumes that a perfectly balanced file assignment can be found. In Section 4, we describe our experimental testbed and we report on synthetic experiments which exhibit a skew in file accesses and sizes and we compare the performance of our new algorithms with the vanilla greedy file allocation algorithm. Section 5 concludes the paper.

2 Related Work

The various algorithms proposed for the file assignment problem aim at minimizing objective functions which can be based on explicit cost functions or on implicit ones, i.e., various performance metrics which are used as proxies. The explicit cost functions used may include communication costs, storage costs, update costs, queuing costs, etc. [8], [24], [6], [7], [14], [25], [26], [20], [18], [23]. The resulting cost function is typically complex. Consequently, its extreme points can be found by using some of the standard nonlinear optimization techniques such as branch-and-bound [24], gradient descent [14], [25], [26], genetic algorithms [18], etc. However, all nonlinear optimization methods are computationally intensive and they frequently suffer from the problem of finding only local minima. This precludes their use on problems of large size which arise in large-scale parallel I/O systems.

In many applications, the most important performance measures are the mean response time and/or system throughput. Consequently, many models considered simplified cost functions which account only for queuing costs [6], [14], [27], [16], [17], [23]. As was discussed in the previous section, the queuing cost at each disk depends both on the utilization of each disk and on the variance of service times at the disks. Most of the heuristic algorithms for file assignment aim at minimizing the disks' maximal utilization either directly, i.e., by using some measure of the load, or indirectly by minimizing the total size of files assigned to each disk [6], [17], [23]. The minimization of service time variance on each disk is usually neglected.

The problem of minimizing the maximal utilization across all disks of a file system is isomorphic to the multiprocessor scheduling problem [10]. Consequently, the efficient heuristics developed for the former can be reused to solve the file assignment problem. Graham described a simple greedy algorithm for multiprocessor load balancing called LPT [11]. At each step, the algorithm greedily assigns a process to the processor having the least accumulated load. LPT can operate in either on-line or off-line modes. In on-line mode, the processes are assigned in the order of

their arrival. In off-line mode, all processes are first ordered by their load and the assignment is done in descending load order.

The LPT algorithm can be applied directly to the file assignment problem as follows: The load of each file is defined as the product of the file access rate and the access service time. This metric is frequently called the *heat* of the file [6], [23]. In what follows, we will refer to the LPT algorithm as applied to the file assignment problem as the vanilla *Greedy algorithm*. In the on-line version of the Greedy algorithm, a file is placed on the disk with the currently lowest accumulated heat and the heat of the target disk is then incremented by the heat of the new file.

The worst-case behavior of a load balancing heuristic algorithm can be expressed by its *competitive ratio*, which is defined as the ratio of the maximal load on any disk under the given heuristic algorithm and the maximal load on any disk under an optimal placement. It has been shown that the competitive ratios of the off-line and on-line versions of the Greedy algorithm are bound by $\frac{4}{3} - \frac{1}{3m} < 1.34$ and $2 - \frac{1}{m}$, respectively, where m is the number of disks [11]. The worst-case bounds can be improved, to a minor extent, by using more sophisticated algorithms [4], [2], [12]. However, it has also been shown that no on-line algorithm can achieve a competitive ratio better than $1 + \frac{1}{\sqrt{2}} \approx 1.7$ [9]. Similarly to the worst case behavior, the average-case behavior of a load balancing heuristic algorithm can be measured by its *average competitive ratio*, which is defined as the ratio of the expected value of the maximal load on any disk under the given heuristic algorithm and the expected value of the maximal load on any disk under an optimal placement. It has been shown that, for two disks, the average competitive ratio of the vanilla Greedy algorithm is bound by $1 + O(\frac{1}{n^2})$, where n is the number of files [5].

All the file assignment algorithms reported in [6], [27], [16], [17], [23] which aim at load balance optimization use either the vanilla Greedy algorithm or a variant of it. Given the known results about the good worst-case and average-case behavior of the Greedy algorithm, we use it in Section 4 as a yardstick for performance comparison with the heuristic algorithms presented developed in Section 3.

3 Heuristic Algorithms

3.1 Model Description

We consider the problem of assigning n files $f_1, f_2, \ldots, f_n$ among m disks of a parallel I/O system $d_1, d_2, \ldots, d_m$. We shall represent the solution to the file assignment problem as a partition of the set $I = \{1, \ldots, n\}$, denoted as $\{I_1, I_2, \ldots, I_m\}$, where I_i is a set of indices corresponding to the files assigned to disk d_i. For simplicity of presentation, we do not consider in this work file partitioning or file replication; thus, each file must be assigned in its entirety to one disk. This does not restrict the generality of our model since if a file is partitioned, each partition can be viewed as a stand alone file. Similarly, a device having k disks, e.g., a RAID, is modeled as k stand-alone disks. Again, this does not not restrict the generality of our model since we assume that the queuing delays on the buses or controllers of the disks are negligible when compared with the queuing delays on the disks themselves. We also assume a "flat" network topology with identical communication delays between any pair of disks. Consequently, the network delays have no impact on the file assignment.

Disk accesses to each file are modeled as a Poisson process with a mean access rate λ_i known a priori. We assume a fixed service time s_i for each file f_i. For example, each access to a file may result in a sequential scan of the entire file. Such a workload is typical in most file systems or WWW servers [19], [15]. For large files, when the unit of file access is the entire file or a large portion of it, the seek and rotation delays are negligible compared with the transfer time. In addition, we consider a homogeneous parallel I/O system with each disk having the same performance characteristics. Thus, our assumption that the service time of each file access is fixed is a valid one in the context outlined above. These two file characteristics, namely access rate and service time, can be combined in a joint metric called heat which will used by our heuristic algorithms. We define the *heat* h_i of file f_i as:

$$h_i = \lambda_i \cdot s_i.$$

The heuristic algorithms which we introduce in this section are based on an open queuing model and employ the mean response time as an objective function to be minimized. As discussed in [23], an open queuing model is more appropriate than a closed queuing model for modeling systems with large number of concurrent users.

An *on-line* file assignment algorithm must assign file f_i to disk d_j using only information about the current state of all disks and the characteristics of all previously assigned files, as well as of the incoming file f_i, i.e., λ_j, s_j, $j <= i$. The decision is made without any knowledge about the characteristics of the files f_k, $k > i$, which will be assigned in the future [1]. On the other hand, an *off-line* algorithm uses knowledge about the entire sequence $f_1, f_2, \ldots, f_n$ of files. We proceed now to discuss first our off-line algorithm Sort Partition and, then, we discuss the on-line version, Hybrid Partition.

3.2 Sort Partition Algorithm

As Example 1 suggests, significant improvements in response time can be obtained by assigning files with similar service times to the same disk, which leads to the minimization of the variance of service times at each disk. Indeed, when files of a wide variety of sizes are intermixed on each disk, it will frequently occur that small file accesses have to wait for larger file accesses that were queued ahead of them. This is inefficient, especially when the load is heavy and the queuing delays dominate the response time.

In order to address the issue of minimization of service time variance at each disk, we designed the following off-line algorithm, called *Sort Partition*. Initially, all files are ordered in a list I in descending order of their service times. The disks are selected for allocation in random order. Each disk d_k is assigned the next contiguous segment from the ordered list I, to be denoted by I_k, such that the load is distributed among the disks more or less evenly. We say that $\{I_1, I_2, \ldots, I_m\}$ is a *perfectly balanced* file assignment (PBFA) if for all k, $\sum_{i \in I_k} h_i = \rho_0$, where $\rho_0 = \frac{1}{m} \cdot \sum_{i=1}^{n} \lambda_i . \cdot s_i$.

Algorithm: Sort Partition
Input: m = number of disks
n = number of files
h_i = heat of file i
s_i = expected service time of file i
Output: assignment of files to disks $\{I_1, I_2, \ldots, I_m\}$

Step 1: Compute the average disk utilization ρ: $\rho = \frac{1}{m} \cdot \sum_{i=1}^{n} h_i$
Step 2: Sort all files into list I in descending order of their service times s_i
Step 3: Allocate to each disk d_j the next contiguous segment of I until its load, $load_j$, reaches the maximum allowed level ρ:

```
i = 1
for j = 1 to m do
    load_j = 0; I_j = ∅
    while (load_j ≤ ρ or i ≤ n) do
        I_j = I_j ∪ {i}          // assign file f_i to disk d_j
        load_j = load_j + h_i
        i = i + 1
    od
od
if (i ≤ n) then
    I_m = I_m ∪ {i, i+1, ..., n}          // assign remainder to disk d_m
fi
```

Fig. 1. Pseudocode of Sort Partition algorithm.

Sort Partition attempts to assign to each disk d_k a segment I_k so that a PBFA is obtained. In some applications where the number of files is large, PBFAs are reasonable approximations of the reality. In the actual implementation, Sort Partition will assign to disk d_k a contiguous segment of files I_k with the least number of files so that $\sum_{i \in I_k} h_i \geq \rho$, where ρ is the average disk utilization of a PBFA. In the case of a non-PBFA, the last disk, d_m, will be assigned the extra files at the "tail" of the list I. The pseudocode of the algorithm can be found in Fig. 1.

We will now show the optimality of the Sort Partition algorithm under the constraint of a PBFA. We summarize our main result in Theorem MinRT.

Theorem MinRT. *Among all PBFAs, Sort Partition finds the one with minimal mean response time.*

3.2.1 Proof of Theorem MinRT

We model each disk as a single M/G/1 queue. Consequently, the mean response time of accesses to disk d_k, $E(r_k)$ is given as

$$E(r_k) = E(s_k) + \frac{\Lambda_k \cdot E(s_k^2)}{2 \cdot (1 - \rho_k)}, \tag{1}$$

where $E(s_k)$ is the mean service time at disk d_k, $E(s_k^2)$ is the mean-square service time at disk d_k, ρ_k is the utilization of disk d_k, and Λ_k is the aggregate access rate at disk d_k, defined as

$$\Lambda_k = \sum_{i \in I_k} \lambda_i.$$

Then, the probability $p_i^{(k)}$ of access to file f_i at disk d_k is given as

$$p_i^{(k)} = \frac{\lambda_i}{\Lambda_k}. \tag{2}$$

Using $p_i^{(k)}$, the mean and mean-square service time for files at disk d_k can be computed as

$$E(s_k) = \sum_{i \in I_k} p_i^{(k)} \cdot s_k = \frac{1}{\Lambda_k} \cdot \sum_{i \in I_k} \lambda_i \cdot s_i$$

and

$$E(s_k^2) = \sum_{i \in I_k} p_i^{(k)} \cdot s_k^2 = \frac{1}{\Lambda_k} \cdot \sum_{i \in I_k} \lambda_i \cdot s_i^2.$$

Assuming a perfect load balance, we have

$$\rho_k = \sum_{i \in I_k} \lambda_i \cdot s_i = \rho_0,$$

where ρ_0 is a constant. Consequently, (1) can be simplified as

$$E(r_k) = \frac{\rho_0}{\Lambda_k} + \frac{\Lambda_k \cdot E(s_k^2)}{2 \cdot (1 - \rho_0)} = \frac{\rho_0}{\Lambda_k} + \frac{\sum_{i \in I_k} \lambda_i \cdot s_i^2}{2 \cdot (1 - \rho_0)}.$$

The overall mean response time associated with the PBFA $\{I_1, I_2, \ldots, I_k\}$ is therefore given as

$$E(r) = \sum_{k=1}^{m} \frac{\Lambda_k}{\Lambda} \cdot E(r_k) = \frac{m\rho_0}{\Lambda_k} + \frac{1}{2\Lambda(1 - \rho_0)} \cdot \sum_{k=1}^{m} \Lambda_k \sum_{i \in I_k} \lambda_i s_i^2, \tag{3}$$

where

$$\Lambda = \sum_{k=1}^{m} \Lambda_k$$

is the aggregate access rate.

From (3), it is clear that the best PBFA must minimize the following objective function

$$Y_m(\{I_1, I_2, \ldots, I_m\}) = \sum_{k=1}^{m} \Lambda_k \sum_{i \in I_k} \lambda_i s_i^2. \tag{4}$$

Minimizing the objective function (4) is equivalent to minimizing the variance of service time because (4) can be rewritten as

$$Y_m(\{I_1, I_2, \ldots, I_m\}) = m \cdot \rho_0^2 + \sum_{k=1}^{m} \Lambda_k^2 \cdot Var(s_k),$$

where the first term is a constant and $Var(s_k)$ is the variance of service time at disk d_k.

We will show now that the PBFAs found by Sort Partition algorithm indeed minimize (4). The proof will proceed by induction on the number of disks m. We first show the induction basis for $m = 2$ and then the inductive step. In order to establish the inductive basis, we prove several auxiliary lemmas.

Lemma 1. *Let $\{I_1^*, I_2^*, \ldots, I_m^*\}$ be a PBFA found by the Sort Partition algorithm and $I = \bigcup_{k=1}^{m} I_k^*$. Let the files' access rates and service times satisfy*

$$\sum_{i \in I} \lambda_i \cdot s_i = m \cdot \rho_0.$$

Then, the following holds for any $J \subset I$ with $\sum_{i \in I} \lambda_i \cdot s_i = \rho_0$:

$$\sum_{i \in J} \lambda_i \cdot s_i^2 \leq \sum_{i \in I_1^*} \lambda_i \cdot s_i^2$$

Proof. Let $K = J \cap I_1^*$. Since $\sum_{i \in J} \lambda_i \cdot s_i = \sum_{i \in I_1^*} \lambda_i \cdot s_i = \rho_0$, we have

$$\sum_{i \in J-K} \lambda_i \cdot s_i = \sum_{i \in I_1^*-K} \lambda_i \cdot s_i.$$

Since $max_{i \in J-K} s_i \leq min_{i \in I_1^*-K} s_i$, it follows that

$$\begin{aligned} \sum_{i \in J-K} \lambda_i \cdot s_i^2 &\leq \sum_{i \in J-K} \lambda_i \cdot s_i \cdot (max_{i \in J-K} s_i) \\ &\leq \sum_{i \in I_1^*-K} \lambda_i \cdot s_i \cdot (min_{i \in I_1^*-K} s_i) \leq \sum_{i \in I_1^*-K} \lambda_i \cdot s_i^2. \end{aligned}$$

Therefore,

$$\begin{aligned} \sum_{i \in J} \lambda_i \cdot s_i^2 &= \sum_{i \in J-K} \lambda_i \cdot s_i^2 + \sum_{i \in K} \lambda_i \cdot s_i^2 \\ &\leq \sum_{i \in I_1^*-K} \lambda_i \cdot s_i^2 + \sum_{i \in K} \lambda_i \cdot s_i^2 = \sum_{i \in I_1^*} \lambda_i \cdot s_i^2 \end{aligned}$$

and, thus, the lemma is proven. □

Lemma 2. *Let I be the set of indices representing files whose access rates and service times satisfy $\sum_{i \in I} \lambda_i \cdot s_i = 2\rho_0$. Let $\{I_1^*, I_2^*\}$ be a PBFA found by Sort partition. Then, for any PBFA $\{I_1, I_2\}$, the following inequalities hold*

$$\sum_{i \in I_1^* \cap I_1} \lambda_i \leq \sum_{i \in I_2^* \cap I_2} \lambda_i \tag{5}$$

$$\sum_{i \in I_1^* - I_1^* \cap I_1} \lambda_i \leq \sum_{i \in I_2^* - I_2^* \cap I_2} \lambda_i. \tag{6}$$

Proof. We first prove that

$$\sum_{i \in I_1^* \cap I_1} \lambda_i \cdot s_i = \sum_{i \in I_2^* \cap I_2} \lambda_i \cdot s_i.$$

Since both $\{I_1, I_2\}$ and $\{I_1^*, I_2^*\}$ are binary partitions of I, we have $I_1^* - I_1^* \cap I_1 \subseteq I_2$. On the other hand, we have $(I_1^* - I_1^* \cap I_1) \cap I_2^* = \emptyset$, since $I_1^* \cap I_2^* = \emptyset$. Hence,

$$I_2^* \cap I_2 \subseteq I_2 - [I_1^* - (I_1^* \cap I_1)].$$

It follows that

$$\begin{aligned} \sum_{i \in I_2^* \cap I_2} \lambda_i \cdot s_i &\leq \sum_{i \in I_2 - [I_1^* - (I_1^* \cap I_1)]} \lambda_i \cdot s_i \\ &= \sum_{i \in I_2} \lambda_i \cdot s_i - \sum_{i \in I_1^* - (I_1^* \cap I_1)} \lambda_i \cdot s_i \\ &= \sum_{i \in I_2} \lambda_i \cdot s_i - \sum_{i \in I_1^*} \lambda_i \cdot s_i + \sum_{i \in I_1^* \cap I_1} \lambda_i \cdot s_i \\ &= \sum_{i \in I_1^* \cap I_1} \lambda_i \cdot s_i, \end{aligned}$$

where the last equality results from the fact that $\sum_{i \in I_2} \lambda_i \cdot s_i = \sum_{i \in I_1^*} \lambda_i \cdot s_i = \rho_0$.

Similarly, we may establish that

$$\sum_{i\in I_1^*\cap I_1}\lambda_i\cdot s_i \le \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i.$$

Therefore, we obtain

$$\sum_{i\in I_1^*\cap I_1}\lambda_i\cdot s_i = \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i. \tag{7}$$

Besides,

$$\sum_{i\in I_1^*-I_1^*\cap I_1}\lambda_i\cdot s_i = \sum_{i\in I_2^*-I_2^*\cap I_2}\lambda_i\cdot s_i \tag{8}$$

follows from the fact that

$$\sum_{i\in I_1^*\cap I_1}\lambda_i\cdot s_i + \sum_{i\in I_1^*-I_1^*\cap I_1}\lambda_i\cdot s_i = \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i + \sum_{i\in I_2^*-I_2^*\cap I_2}\lambda_i\cdot s_i = \rho_0.$$

Using (7) and $min_{i\in I_1^*}s_i \ge max_{i\in I_2^*}s_i$, we obtain

$$\begin{aligned}\sum_{i\in I_1^*\cap I_1}\lambda_i &\le \sum_{i\in I_1^*\cap I_1}\lambda_i\cdot s_i\cdot\frac{1}{min_{i\in I_1^*}s_i}\\ &= \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i\cdot\frac{1}{min_{i\in I_1^*}s_i}\\ &\le \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i\cdot\frac{1}{max_{i\in I_2^*}s_i}\\ &\le \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot s_i\cdot\frac{1}{s_i} = \sum_{i\in I_2^*\cap I_2}\lambda_i\end{aligned}$$

which completes the proof of Part (5). Part (6) can be proven in a similar manner using (8) and $min_{i\in I_1^*}s_i \ge max_{i\in I_2^*}s_i$. □

Lemma 3. *Let I be the set of indices representing files whose access rates and service times satisfy $\sum_{i\in I}\lambda_i\cdot s_i = 2\rho_0$. Let $\{I_1^*, I_2^*\}$ be a PBFA found by Sort partition. Then, for any PBFA $\{I_1, I_2\}$, the following holds*

$$Y_2(\{I_1, I_2\}) \ge Y_2(\{I_1^*, I_2^*\}),$$

where $Y_2(\cdot)$ is the objective function defined in (4).

Proof. Let

$$\begin{aligned}Y_2 &= Y_2(\{I_1, I_2\}),\\ Y_2^* &= Y_2(\{I_1^*, I_2^*\}),\\ A_1 &= \sum_{i\in I_1}\lambda_i\cdot s_i^2,\\ A_2 &= \sum_{i\in I_2}\lambda_i\cdot s_i^2,\\ A_1^* &= \sum_{i\in I_1^*}\lambda_i\cdot s_i^2,\end{aligned}$$

and

$$A_2^* = \sum_{i\in I_2^*}\lambda_i\cdot s_i^2.$$

We can rewrite Y_2 and Y_2^*, respectively, as

$$\begin{aligned}Y_2 &= \sum_{i\in I_1}\lambda_i\cdot A_1 + \sum_{i\in I_2}\lambda_i\cdot A_2\\ &= \sum_{i\in I_1\cap I_1^*}\lambda_i\cdot A_1 + \sum_{i\in I_1^*-I_1\cap I_1^*}\lambda_i\cdot A_2 + \sum_{i\in I_2^*-I_2\cap I_2^*}\lambda_i\cdot A_1 + \sum_{i\in I_2\cap I_2^*}\lambda_i\cdot A_2\end{aligned}$$

and

$$Y_2^* = \sum_{i\in I_1\cap I_1^*}\lambda_i\cdot A_1^* + \sum_{i\in I_1^*-I_1\cap I_1^*}\lambda_i\cdot A_1^* + \sum_{i\in I_2^*-I_2\cap I_2^*}\lambda_i\cdot A_2^* + \sum_{i\in I_2\cap I_2^*}\lambda_i\cdot A_2^*.$$

It follows that

$$Y_2 - Y_2^* = \sum_{i\in I_1\cap I_1^*}\lambda_i\cdot(A_1 - A_1^*) + \sum_{i\in I_1^*-I_1\cap I_1^*}\lambda_i\cdot(A_2 - A_1^*) - \sum_{i\in I_2^*\cap I_2}\lambda_i\cdot(A_2^* - A_2) - \sum_{i\in I_2^*-I_2^*\cap I_2}\lambda_i\cdot(A_2^* - A_1).$$

Now, since $A_1 + A_2 = A_1^* + A_2^* = \sum_{i\in I}\lambda_i\cdot s_i^2$ and, thus, $A_1 - A_1^* = A_2 - A_2^*$ and $A_2 - A_1^* = A_2^* - A_1$, we have

$$Y_2 - Y_2^* = \left(\sum_{i\in I_1\cap I_1^*}\lambda_i - \sum_{i\in I_2^*\cap I_2}\lambda_i\right)\cdot(A_1 - A_1^*) + \left(\sum_{i\in I_1^*-I_1\cap I_1^*}\lambda_i - \sum_{i\in I_2^*-I_2^*\cap I_2}\lambda_i\right)\cdot(A_2 - A_1^*).$$

From Lemma 1, it follows that $A_1 - A_1^* \le 0$ and $A_2 - A_1^* \le 0$. Furthermore, from Lemma 2, we obtain

$$\sum_{i\in I_1^*\cap I_1}\lambda_i - \sum_{i\in I_2^*\cap I_2}\lambda_i \le 0$$

and

$$\sum_{i\in I_1^*-I_1^*\cap I_1}\lambda_i - \sum_{i\in I_2^*-I_2^*\cap I_2}\lambda_i \le 0.$$

We therefore conclude that $Y_2 - Y_2^* \ge 0$ and the proof of Lemma 3 is complete. □

Lemma 3 establishes the induction basis for the proof of Theorem 1. In order to prove the inductive step, we develop an algorithm which correlates the file assignment found by Sort Partition for m disks with the assignment found by Sort Partition for $m+1$ disks.

Let I be a set of file indices. We denote the partition of I, $\{I_1^*, I_2^*, \ldots, I_m^*\}$ found by the Sort Partition algorithm as $SP_m(I)$. Let I_{m+1} be a set of file indices satisfying $\sum_{i\in I_{m+1}}\lambda_i\cdot s_i = \rho_0$. We define a polynomial-time algorithm *Extension* which extends $SP_m(I)$ to $SP_{m+1}(I\cup I_{m+1})$. The pseudocode of the algorithm can be found in Fig. 2.

Basically, *Extension* works by interleaving the files assigned to I_{m+1} to successive elements of the partition created by $SP_m(I)$. After k iterations of this procedure, the partition elements starting with I_{m+1-k}^{**} to the end of the sequence are already sorted. The next iteration of the procedure moves the files out of order in element I_{m+1-k}^* to the element in position *m-k* in the corresponding partition created by $SP_m(I)$.

Algorithm: Extension
Input: $SP_m(I)$ and I_{m+1}
Output: $SP_{m+1}(I \cup I_{m+1})$

set $\{I_m^+, I_{m+1}^{**}\} = SP_2(I_m^* \cup I_{m+1})$
for $k = 1$ **to** $m-1$ **do**
  set $\{I_{m-k}^+, I_{m+1-k}^{**}\} = SP_2(I_{m-k}^* \cup I_{m+1-k}^+)$
od
set $I_1^{**} = I_1^+$
return $\{I_1^{**}, I_2^{**}, \ldots, I_m^{**}, I_{m+1}^{**}\}$

Fig. 2. Pseudocode of extension algorithm.

In order to show that the output of the Extension algorithm is indeed $SP_{m+1}(I \cup I_{m+1})$, we first establish the following lemma:

Lemma 4. *Let $\{I_1^*, I_2^*\} = SP_2(I_1 \cup I_2)$, where I_1 and I_2 represent files such that $\sum_{i \in I_k} \lambda_i \cdot s_i = \rho_0$, $k = 1, 2$. Then, for $k = 1, 2$, the following holds:*

$$max_{i \in I_2^*} s_i \leq max_{i \in I_k} s_i$$

Proof. Let $s_j = max_{i \in I_2^*} s_i$. Assume by contradiction that $max_{i \in I_k} s_i < s_j$. Consequently, from the definition of the Sort Partition algorithm, it follows that $I_k \cap I_1^* = \emptyset$ and, thus, $I_k \subset I_2^*$.

Therefore, $\{j\} \cup I_k \subseteq I_2^*$. Since $\sum_{i \in I_k} \lambda_i \cdot s_i = \rho_0$, we have $\sum_{i \in I_2^*} \lambda_i \cdot s_i > \rho_0$, which contradicts the definition of I_2^*. □

Lemma 5. *The output of algorithm Extension satisfies $\{I_1^{**}, I_2^{**}, \ldots, I_{m+1}^{**}\} = SP_{m+1}(I \cup I_{m+1})$.*

Proof. We need to prove that

$$min_{i \in I_k^{**}} s_i \geq max_{i \in I_{k+1}^{**}} s_i, \qquad 1 \leq k \leq m. \tag{9}$$

From the definition of algorithm Extension, we have

$$max_{i \in I_{k+1}^{**}} s_i \leq min_{i \in I_k^+} s_i. \tag{10}$$

From Lemma 4, it follows that

$$max_{i \in I_{k+1}^{**}} s_i \leq max_{i \in I_k^*} s_i \leq min_{i \in I_{k-1}^*} s_i, \tag{11}$$

where the last inequality holds because $\{I_{k-1}^*, I_k^*\} \subset SP_m(I)$. Combining (10) and (11) yields

$$max_{i \in I_{k+1}^{**}} s_i \leq min_{i \in I_k^+ \cup I_{k-1}^*} s_i. \tag{12}$$

Since $I_k^{**} \in SP_2(I_k^+ \cup I_{k-1}^*)$, we have

$$min_{i \in I_K^{**}} s_i \leq min_{i \in I_k^+ \cup I_{k-1}^*} s_i. \tag{13}$$

We conclude from (12) and (13) that

$$max_{i \in I_{k+1}^{**}} s_i \leq min_{i \in I_k^{**}} s_i.$$

Hence, the proof of Lemma 5 is complete. □

We proceed with the proof of the inductive step. We assume that, for any PBFA to m disks, $\{J_1, J_2, \ldots, J_m\}$, the following holds

$$Y_m(\{J_1, J_2, \ldots, J_m\}) \geq Y_m(\{J_1^*, J_2^*, \ldots, J_m^*\}),$$

where $\{J_1^*, J_2^*, \ldots, J_m^*\}$ is a PBFA found by Sort Partition and $Y_m(\cdot)$ is the objective function defined in (4). We need to prove the claim for $m+1$ disks.

Consider a set of file indices I such that $\sum_{i \in I} \lambda_i \cdot s_i = (m+1) \cdot \rho_0$. For an arbitrary perfectly balanced partition of I, $\{I_1, I_2, \ldots, I_m, I_{m+1}\}$, we have

$$\begin{aligned} &Y_{m+1}(\{I_1, I_2, \ldots, I_m, I_{m+1}\}) \\ &= \sum_{k=1}^{m} \Lambda_k \sum_{i \in I_k} \lambda_i \cdot s_i^2 + \Lambda_{m+1} \sum_{i \in I_{m+1}} \lambda_i \cdot s_i^2, \end{aligned} \tag{14}$$

where $\Lambda_k = \sum_{i \in I_k} \lambda_i$.

Let $SP_{m+1}(I) = \{I_1^{**}, I_2^{**}, \ldots, I_{m+1}^{**}\}$ and

$$Y_{m+1}^{**} = Y_{m+1}(\{I_1^{**}, I_2^{**}, \ldots, I_{m+1}^{**}\}).$$

We will show that

$$Y_{m+1}(\{I_1, I_2, \ldots, I_{m+1}\}) \geq Y_{m+1}^{**}.$$

Let $L = \bigcup_{k=1}^{m}$ and $\{I_1^*, I_2^*, \ldots, I_m^*\} = SP_m(L)$. By the inductive assumption, we have

$$Y_m(\{I_1^*, I_2^*, \ldots, I_m^*\}) \geq Y_m(\{I_1, I_2, \ldots, I_m\}). \tag{15}$$

It follows from (14) and (15) that

$$\begin{aligned} &Y_{m+1}(\{I_1, I_2, \ldots, I_{m+1}\}) \\ &\geq \sum_{k=1}^{m} \Lambda_k^* \sum_{i \in I_k^*} \lambda_i \cdot s_i^2 + \Lambda_{m+1} \sum_{i \in I_{m+1}^*} \lambda_i \cdot s_i^2 \\ &= Y_{m+1}(\{I_1^*, I_2^*, \ldots, I_m^*, I_{m+1}\}), \end{aligned}$$

where $\Lambda_k^* = \sum_{i \in I_k^*} \lambda_i$.

We use Lemma 5 to compare $Y_{m+1}(\{I_1^*, I_2^*, \ldots, I_m^*, I_{m+1}\})$ with Y_{m+1}^{**}. It can be easily seen that algorithm Extension generates $\{I_1^{**}, I_2^{**}, \ldots, I_m^{**}, I_{m+1}^{**}\}$ on input

$$\{I_1^*, I_2^*, \ldots, I_m^*, I_{m+1}\}.$$

Each iteration of algorithm Extension executes Sort Partition with the number of disks $m = 2$. Consequently, by Lemma 3, the objective function does not increase after each iteration. Therefore, the theorem holds for $m+1$ disks. This concludes the proof of Theorem 1. □

3.3 Hybrid Partition Algorithm

In the previous section, we have shown that, assuming a PBFA, Sort Partition achieves superior performance compared to Greedy in terms of response time. This is achieved by assigning files with similar service times to the same disk. However, Greedy can operate in on-line mode requiring no a priori knowledge of the files to be assigned in the future. On the other hand, Sort Partition is an off-line algorithm which requires complete knowledge about the service times and access rates of all the files. This may be

Algorithm: Hybrid Partition
Input: m = number of disks
n = number of files
h_i = heat of file i
b = number of sorted batches of files
B_i = i-th batch of files sorted in descending order of service times
B = list $(B_1, B_2, \ldots, B_b)$
Output: assignment of files to disk $\{I_1, I_2, \ldots, I_m\}$

Step 1: Initialize the load on each disk d_i: $load_i = 0$ and set allocation list $I_i = \emptyset$
Step 2: **while** (list of batches B has not been exhausted) **do**
Step 2.1: Select next batch B_{next} not yet assigned
while (B_{next} has not been completely assigned) **do**
Step 3: Determine the next disk, d_k, to be used during the next allocation interval, l:
d_k = disk such that $load_k = min_{s=1,\ldots,m}(load_s)$
Step 2.1: Determine the threshold θ_k to which the load can be increased on disk d_k during this interval:
$\theta_k = 1 - \frac{1-load_k}{overflow}$ // $overflow$ is a constant explained below
Step 2.2: During allocation interval l assign to disk d_k a contiguous segment of files from B_{next} until its load $load_k$ reaches threshold θ_k
Update the allocation list I_k
$I_k = I_k \cup \{\text{indices of files allocated in interval } l\}$
od
od

Fig. 3. Pseudocode of Hybrid Partition algorithm.

clearly inappropriate in many situations where the files are being generated dynamically, i.e., on the fly.

To address these issues, we have designed a new on-line algorithm, called Hybrid Partition, which attempts to simultaneously minimize the load variance across all disks, as well as the service time variance at each disk. Hybrid Partition requires that all files be assigned in descending order of service times, but, in contrast to Sort Partition, it does not require any knowledge about the statistics of files to be allocated in the future. In many practical situations, files arrive in batches, which can be sorted prior to their assignment, but with no correlation between the file service times in different batches. Hybrid Partition is intended as an on-line alternative to Greedy for batches of reasonable size.

The Hybrid Partition 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. Like Greedy, the algorithm selects the disk with the smallest accumulated load (heat), denoted as $load_k$. During one allocation interval, a number of files are allocated to the target disk d_k until its load reaches a given threshold θ_k. The files to be allocated to disk d_k are a contiguous segment of files from the current batch B_{next}. This is similar in spirit to Sort Partition, except that the number of batches can be larger than one. A high level version of our algorithm is given in Fig. 3.

Hybrid Partition attempts to reconcile between two conflicting goals: minimizing the load variance across the disks and minimizing service time variance at each disk in the following way: When the overall disk utilization is low, the load imbalance does not significantly impact the response time and, thus, the algorithm gives priority to minimizing service time variance. This is achieved by assigning to one disk a "relatively" large segment of files from B_{next} with similar service times. On the other hand, once the system utilization is high, any load imbalance might significantly affect the system response time. In order to give priority to load balancing, smaller segments of files

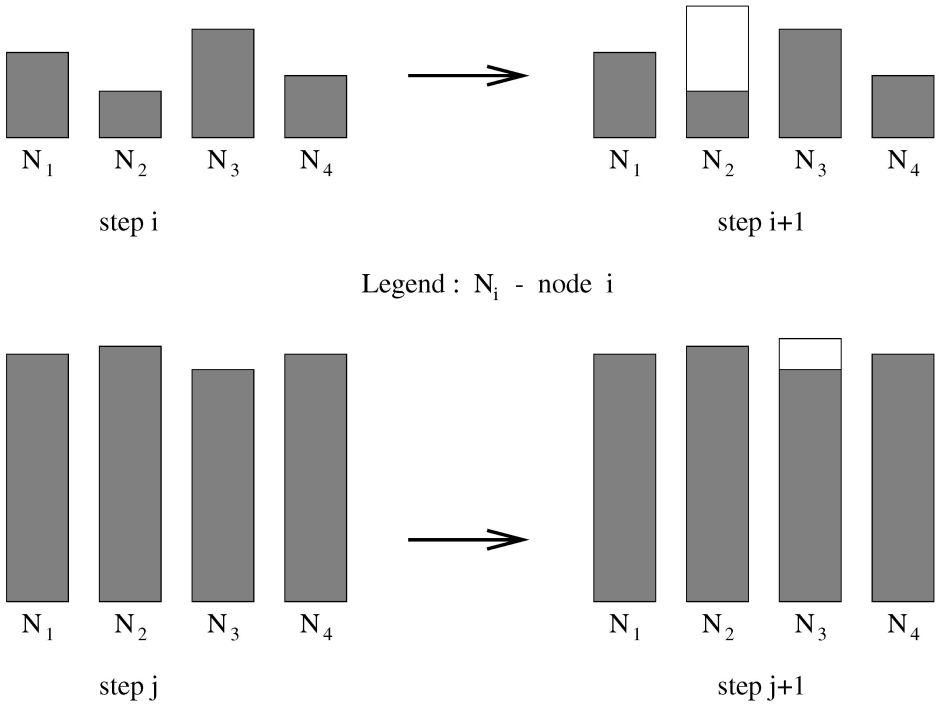

Fig. 4. Illustration of Hybrid Partition algorithm.

from B_{next}, i.e., fewer files, are assigned to the target disk during one allocation interval. This basic principle is illustrated in Fig. 4, where two allocation intervals are depicted. The load of each disk is proportional to to its height and the white portion of a disk represents the additional load assigned to it during the current allocation interval. Observe also that, like Greedy, Hybrid Partition may select the same disk during different allocation intervals.

In order to implement this trade-off between load imbalance and service time minimization, our algorithm "dynamically" adjusts the threshold θ_k which the load can reach during the current allocation interval on disk d_k. As $load_k$ increases, the difference $\theta_k - load_k$ decreases. Ideally, the value θ_k should be selected in such a way that the ratio between the mean response times on disk d_k after and before the file assignments in this interval does not exceed a fixed constant, $overflow$. The constant gives the maximal increase in response time one is willing to tolerate in order to minimize the service time variance on each server. The value of θ_k can be computed by approximating the behavior of each disk by a M/M/1 queue. In this case, we obtain that $E(r_1)$ and $E(r_2)$, the mean response times at disk k before and after the assignments in the current allocation interval, respectively, are given as:

$$E(r_1) = E(s)/(1 - load_k)$$

and

$$E(r_2) = E(s)/(1 - \theta_k).$$

By setting $E(r_2)/E(r_1) = overflow$, we obtain

$$\theta_k = 1 - \frac{1 - load_k}{overflow}.$$

In the experiments reported in Section 4, we set the value of $overflow$ to 1.05.

4 Experimental Evaluation

In this section, we present an experimental performance evaluation of Sort Partition and Hybrid Partition algorithms. We compare their performance with the vanilla Greedy algorithm.

4.1 Experimental Setup and Workload Characteristics

The experimental testbed is based on a parallel I/O system prototype FIVE [28], [23]. FIVE was designed to manage files striped across several disks which are connected to a single host. In all experiments, the contention on the host's controller and bus was minimal. Consequently, each disk can be viewed as connected to a single disk of a parallel I/O system. FIVE can either manage real data on real disks or it can simulate each disk when the former is not available. The disk simulator keeps track of exact arm positions, as well as rotational positions of the disk head. It also considers head switch delays, realistic seek time as a nonlinear function of the seek distance, and also other details of real disks [23]. In all experiments, we used simulated disks with the configuration parameters described in Fig. 5.

All tests are based on synthetic workloads. In all experiments, we distributed 5,000 files across the 16 available disks. Each file was allocated to a single disk; the files were not partitioned or replicated. Each file access represented a sequential read of the entire file. In order to

#disks	16	capacity of one disk	539 MBytes
#cylinders per disk	1435	capacity of the disk system	8.6 GBytes
#tracks per cylinder	11	revolutions per minute	4400 rpm
block size	4 KBytes	avg. disk seek time	12 ms
track size	8 blocks	disk transfer rate	2.44 MBytes/s

Fig. 5. Simulated hardware configuration.

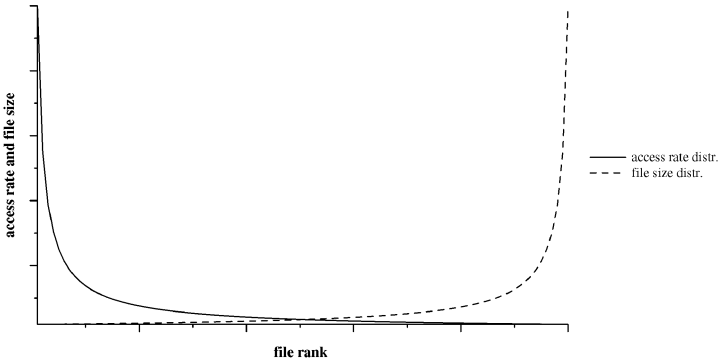

Fig. 6. Workload characterization: access rate and file size distribution.

study the efficiency of our algorithms in a realistic setting, the distributions of file sizes and file accesses across the files exhibit a skew. The sizes of the files were distributed according to a Zipfian 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 [13]. We conducted experiments with the skew parameter θ corresponding to either 60-40 or 70-30 distributions. The interarrival times of accesses to file f_i were exponentially distributed with a fixed mean $1/\lambda_i$. On the other hand, in order to reflect the scenario where different files have different access rates, we modeled the distribution of access rates across the files also with a Zipfian distribution having the same skew parameter θ as the distribution of file sizes.

As observed in real system traces [19], [15], the most popular files are typically small in size, while the large files are relatively unpopular. Therefore, the distributions of access rates across the files and file sizes were inversely correlated, as shown in Fig. 6. In each experiment, either all files were assigned at the same time or the files were assigned in several batches. The files in each batch were randomly selected. Consequently, even after sorting each batch (in descending order of service times), the batches did not form an ordered sequence. We conducted experiments with four and 64 batches each having 78 and 1,250 files, respectively. In each series of experiments, we increased the aggregate access rate $\Lambda = \sum_{i=1}^{5000} \lambda_i$ until the point when the model started thrashing. Each experiment simulated approximately a 15 minute interval.

4.2 Experimental Results

We compared the performance of Sort Partition and Hybrid Partition against the performance of Greedy. We concentrated on the average response time as the primary performance metric. Because the response time grows by more than one order of magnitude as the aggregate access rate Λ approaches the thrashing point, the average response times are represented on a logarithmic scale for all graphs reported in this section.

We found that Sort Partition consistently provided the best response time among all algorithms. This result would be trivial under a PBFA. Fig. 7 shows a sample of coefficient of variation[1] of disk load under various assignments and $\Lambda = 200s^{-1}$. These results confirm our intuitive expectation that Greedy leads to the best load balance because load balancing is its only goal. On the other hand, Sort Partition leads to the worst load balance because it does not explicitly attempt to balance the load; rather, it assumes that a PBFA can be found. Finally, Hybrid Partition's load balance is a compromise between that of Greedy and Sort Partition as its priorities alternate between load balancing and minimizing the variance of service time.

Although Sort Partition leads to assignments with worst load balance, it still provides the best response time among all three assignments for both values of skew θ as shown in Figs. 8 and 9. For example, for θ corresponding to a 70/30 distribution and $\Lambda = 200\text{s}^{-1}$, Sort Partition provides a 50 percent improvement over the response time of Greedy and 24 percent improvement over the response time of Hybrid Partition. Thus, the experiments justify our claims about the importance of minimizing the variance of the service time at each disk of a parallel I/O system. Fig. 8 also shows that the higher the aggregate access rate Λ is, the more significant is the improvement achieved by Sort Partition. Thus, paying attention to minimization of service time variance is especially important when the system is under a heavy

1. Standard deviation normalized by mean.

coefficient of variation		
Greedy	Hybrid Partition	Sort Partition
0.037	0.126	0.147

Fig. 7. Disk load variance.

load. We also found that the response times of Hybrid Partition were between those of Greedy and Sort Partition. This is again in accordance with our expectations because Hybrid Partition trades off its performance for the ability to do on-line processing.

As Figs. 8 and 9 show, the qualitative ranking of the three algorithms does not change for different access rates and skew parameters, θ. The differences in response times decrease slowly with smaller values of the skew parameter θ. However, for θ corresponding to 60/40 distribution and $\Lambda = 200\text{s}^{-1}$, Sort Partition still provides a 44 percent improvement over Greedy and a 19 percent improvement over Hybrid Partition.

We have also compared the performance of the two on-line algorithms, Hybrid Partition and Greedy, on workloads with multiple batches. The results for a skew θ corresponding to a 70/30 distribution can be found in Figs. 10 and 11. Hybrid Partition always provides a better response time than Greedy. However, the improvement diminishes as the number of batches grows and, consequently, their size decreases. For example, as shown in Fig. 10, given an arrival rate $\Lambda = 200\text{s}^{-1}$, and with four runs, each consisting of 1,250 files, Hybrid Partition provides a 21 percent improvement of response time over Greedy. Once the number of batches grows to 64 with each batch having 78 files, the improvement in response time of Hybrid Partition over Greedy drops to 6 percent for the same value of Λ, as shown in Fig. 11. The results for a skew corresponding to a 60/40 distribution were qualitatively similar.

Based on these experimental results, we arrived at the following conclusions:

- *Sort Partition* should be used whenever an off-line algorithm is feasible, i.e., when the characteristics of all files (service times and heat) are known in advance.
- *Hybrid Partition* should be used whenever the files arrive dynamically in reasonably large batches, which can be sorted in descending order of service times prior to their assignment. However, as the batch size decreases, the response time improvements of Hybrid Partition over Greedy become negligible.
- *Greedy* can be used whenever the files to be assigned arrive one-at-a-time. In such an environment, Greedy provides practically the same response time as Hybrid Partition.

5 Conclusion

We have presented two novel file assignment algorithms, Sort Partition and Hybrid Partition. Both algorithms aim at optimizing the mean system response time by simultaneously minimizing the variance of the load across all servers and the variance of the service time at each server. Specifically, Sort Partition minimizes the variance of the service time at each server by assigning to each server files with similar service times. We have shown that, among all PBFAs, Sort Partition finds the assignment guaranteeing minimal system response time. Because Sort Partition is inherently an off-line algorithm, we designed an on-line algorithm, Hybrid Partition, which approximates the behavior of Sort Partition in a given allocation interval while guaranteeing that during each interval the load imbalance does not exceed a certain threshold.

Our experimental results show that Sort Partition provides consistently better response times than the vanilla Greedy algorithm, even when the file assignment which it produces is not load balanced. Although Hybrid Partition does not achieve the performance level of Sort Partition, it still provides superior performance in comparison with the

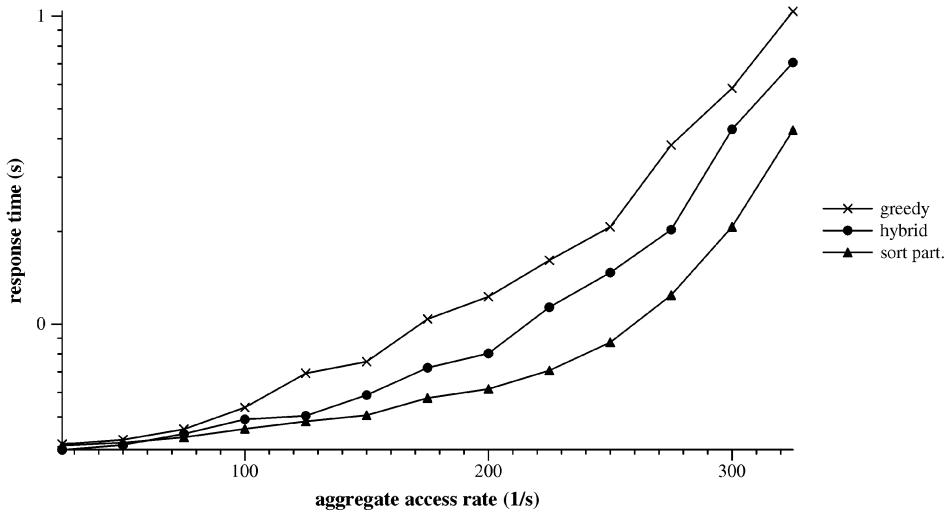

Fig. 8. Average response time for 70/30 skew, one batch.

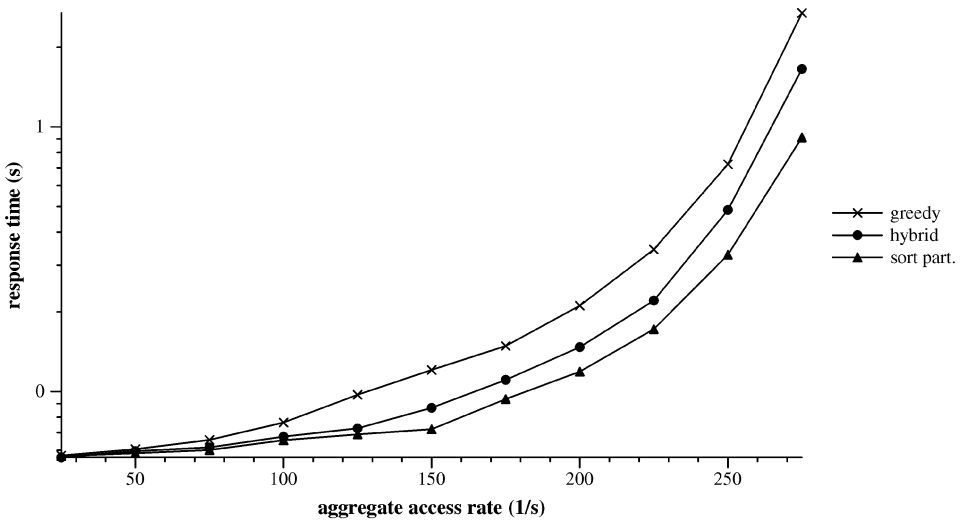

Fig. 9. Average response time for 60/40 skew, one batch.

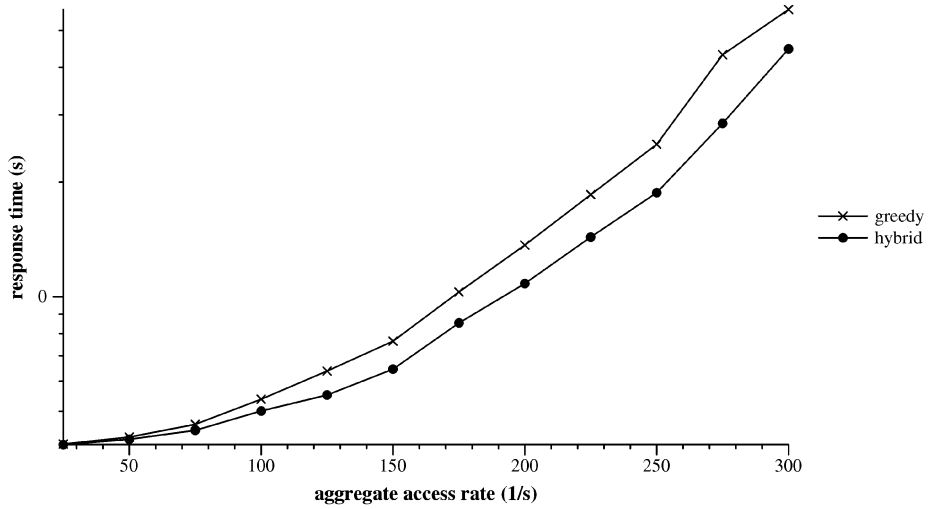

Fig. 10. Average response time for 1,250 file batch size, 70/30 skew.

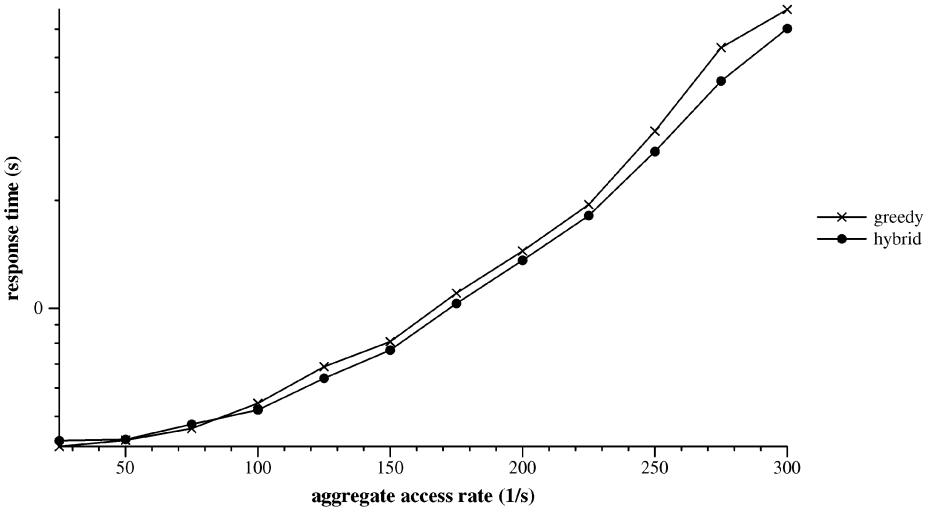

Fig. 11. Average response time for 78 file batch size, 70/30 skew.

vanilla Greedy algorithm in an environment where the files arrives arrive in sorted batches of reasonable large sizes. We are planning to extend our analysis of Hybrid Partition by including an analytical model to determine the optimal choice of the constant $overflow$, as well as to consider explicitly the impact of the batch size.

In practice, judicious file allocation is only one of the performance tuning issues that need to be incorporated in a file manager for parallel I/O systems. As we mentioned earlier, in RAIDs [3], files are usually partitioned into extents that are distributed across disks in order to further reduce the service time of a single request or to improve the throughput of multiple requests. Striping is the most commonly used variant of file partitioning whereby a file is divided first into fixed-sized runs of logically consecutive data units that are assigned to disks in a round-robin manner. A file extent corresponds now to all runs of a file that need to be allocated contiguously on a single disk. Although file striping and file allocation are orthogonal issues, they are not completely independent. Striping imposes an additional constraint on the file allocation problem. Namely, in order to support intrarequest parallelism [23], it is necessary to allocate the extents of a file to different disks. We plan to extend our algorithms for file allocation to cover the case when the units of allocation are file segments.

In addition, in a fully dynamic environment, not only are files to be created or deleted on the fly, but files may grow or shrink and the file access characteristics may change over time. In order to deal with all these dynamics of change, it is necessary to incorporate into a file manager another tuning component that can redistribute the load by migrating data from one disk to another. File migration is an on-line reorganization process which is performed incrementally, usually by migrating a file (or file segment) at the time [22]. Thus, file migration is a tuning step complementary to file allocation. By performing incremental migration steps, we can avoid the alternative of an expensive reallocation of all files. In [29], we have presented a model for file migration which determines whether a given file migration is beneficial at a given point in time; this is done by measuring whether its benefit, given as an objective function based on the variance of the load across the disks, exceeds the migration's cost.

ACKNOWLEDGMENTS

This work has been supported in part by the U.S. National Science Foundation under grant IRI-9303583 and by NASA-Ames under grant NAG2-846. We would like to thank Gerhard Weikum for his insightful comments on an earlier version of this paper and Sandra Irani for discussions on the feasibility of upper bounds for our on-line algorithm.

REFERENCES

[1] Y. Azar, A. Broder, and A. Karlin, "On-Line Load Balancing," *Theoretical Computer Science,* vol. 130, 1994.

[2] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra, "New Algorithms for an Ancient Scheduling Problem," *Proc. 24th ACM Symp. Theory of Computing,* pp. 51-58, 1992.

[3] 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.

[4] E. Coffman, M. Garey, and D. Johnson, "An Application of Bin-Packing to Multiprocessor Scheduling," *SIAM J. Computing,* vol. 7, no. 1, pp. 1-17, 1978.

[5] E. Coffman, G. Grederickson, and G. Lueker, "A Note on Expected Makespans for Largest-First Sequences of Independent Tasks on Two Processors," *Math. Operations Research,* vol. 9, 1984.

[6] G. Copeland, W. Alexander, E. Bougher, and T. Keller, "Data Placement in Bubba," *Proc. ACM SIGMOD Int'l Conf. Management of Data,* pp. 99-108, 1988.

[7] R. Dewan and B. Gavish, "Models for the Combined Logical and Physical Design of Databases," *IEEE Trans. Computers,* vol. 38, no. 7, pp. 955-967, July 1989.

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

[9] U. Faigle, W. Kern, and G. Turan, "On the Performance of On-Line Algorithms for Particular Problems" *Acta Cybernetica,* vol. 9, pp. 107-119, 1989.

[10] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman, 1979.

[11] R.L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM J. Applied Math.,* vol. 17, no. 2, pp. 416-429, 1969.

[12] D. Karger, S. Phillips, and E. Torng, "A Better Algorithm for an Ancient Scheduling Problem," *Proc. Fifth ACM Symp. Discrete Algorithms,* 1994.

[13] D. Knuth, *The Art of Computer Programming.* Addison-Wesley, 1973.

[14] J. Kurose and R. Simha, "A Microeconomic Approach to Optimal Resource Allocation in Distributed Computer Systems," *IEEE Trans. Computers,* vol. 38, no. 5, May 1989.

[15] 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.

[16] C. Lee and K. Hua, "A Self-Adjusting Data Distribution Mechanism for Multidimensional Load Balancing in Multiprocessor-Based Database Systems," *Information Systems,* vol. 19, no. 7, pp. 549-567, 1994.

[17] H. Lee and T. Park, "Allocating Data and Workload among Multiple Servers in a Local Area Network," *Information Systems,* vol. 20, no. 3, 1995.

[18] S. March and S. Rho, "Allocationg Data and Operations to Nodes in Distributed Database Design," *IEEE Trans. Knowledge and Data Eng.,* vol. 7, no. 2, pp. 305-317, Mar./Apr. 1995.

[19] J. Ousterhout, H. Da Costa, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson, "A Trace-Driven Analysis of the UNIX 4.2BSD File System" Technical Report CSD-85-230, Univ. of California at Berkeley, 1985.

[20] D. Rotem, G. Schloss, and A. Segev, "Data Allocation of Multidisk Databases," *IEEE Trans. Knowledge and Data Eng.,* vol. 5, no. 5, pp. 882-887, Sept./Oct. 1993.

[21] J. Salehi, Z.L. Zhang, J. Kurose, and D. Towsley, "Supporting Stored Video: Reducing Rate Variability and End-to-End Resource Requirements through Optimal Smoothing," *ACM SIGMETRICS,* 1996.

[22] P. Scheuermann, G. Weikum, and P. Zabback, "Disk Cooling in Parallel Disk Systems," *IEEE Data Eng. Bulletin,* vol. 17, no. 3, pp. 29-40, 1994.

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

[24] B. Wah, "File Placement on Distributed Computer Systems," *Computer,* vol. 17, no. 1, pp. 23-32, Jan. 1984.

[25] J. Wolf, "The Placement Optimization Program: A Practical Solution to the Disk File Assignment Problem," *Proc. Int'l Conf. Measurement and Modeling of Computer Systems,* pp. 1-10, 1989.

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

[27] J. Wolf, P. Yu, J. Turek, and D. Dias, "A Parallel Hash Join Algorithm for Managing Data Skew" *IEEE Trans. Parallel and Distributed Systems,* vol. 4, no. 12, Dec. 1993.

[28] P. Zabback, "I/O Parallelism in Database Systems—Design, Implementation, and Evaluation of a Storage System for Parallel Disks," PhD thesis, Dept. of Computer Science ETH Zurich, 1994 (in German).
[29] P. Zabback, I. Onyuksel, P. Scheuermann, and G. Weikum, "Database Reorganization in Parallel Disk Arrays with I/O Service Stealing," *IEEE Trans. Knowledge and Data Eng.*, vol. 10, no. 5, pp. 855-858, Sept./Oct. 1998.

Lin-Wen Lee received her BA in foreign languages and literature from National Taiwan University, Taiwan, in 1977. From 1978 to 1980, she was a nondegree student in computer science at Princeton University, Princeton, New Jersey. She then received her MS and PhD degrees in computer science from Northwestern University in 1985 and 1994, respectively. During 1989, she was a summer intern at AT&T Bell Laboratories. Since 1993, she has been president of Linwen Associates, Inc., Long Grove, Illinois, specializing in information and telecommunication consulting. Her research interests are in networked computing, database and file systems.

Peter Scheuermann received his PhD in computer science from the State University of New York at Stony Brook. He has been with Northwestern University since 1976, where he is currently a professor of electrical and computer engineering. He has held visiting professor positions with the Free University of Amsterdam, the University of Hamburg, and the Swiss Federal Institute of Technology, Zurich. During 1997-1998, he served as program director for operating systems and compilers in the Computer and Communications Division of the National Science Foundation.

Dr. Scheuermann has served on the editorial board of the *Communications of the ACM* and is currently an associate editor of *The VLDB Journal*. He was general chair of the ACM-SIGMOD Conference in 1988, general chair of FODO '93, and, more recently, program chair of the Seventh International Workshop on Research Issues in Data Engineering (1997). His current research interests include Web-related technologies, I/O systems, data warehousing and data mining, parallel and distributed database systems, and spatial databases.

Radek Vingralek received the MS degree from Charles University in Prague, Czech Republic, and the PhD degree from the University of Kentucky, Lexington. He is a member of the research staff at STAR Lab, InterTrust Technologies Corporation. Formerly, he held positions at Bell Laboratories, Lucent Technologies, and Oracle Corporation. His research interests include web proxy server design, wide-area data replication, transaction processing, and query processing in parallel database systems.