Adaptive Block Rearrangement

SEDAT AKYÜREK and KENNETH SALEM
University of Maryland

An adaptive technique for reducing disk seek times is described. The technique copies frequently referenced blocks from their original locations to reserved space near the middle of the disk. Reference frequencies need not be known in advance. Instead, they are estimated by monitoring the stream of arriving requests. Trace-driven simulations show that seek times can be cut substantially by copying only a small number of blocks using this technique. The technique has been implemented by modifying a UNIX device driver. No modifications are required to the file system that uses the driver.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*; D.4.8 [**Operating Systems**]: Performance—*measurements*; *modeling and prediction*; *simulation*; H.3.2 [**Information Storage and Retrieval**]: Information Storage

General Terms: Algorithms, Design, Experimentation, Performance

Additional Key Words and Phrases: Data clustering, data replication, seek optimization

1. INTRODUCTION

Because of their relatively long access times, disks can be performance bottlenecks in a storage hierarchy. When data requests arriving at the disk are small and nonsequential, as is common in multiuser operating systems and database systems, seek time is a major component of disk access time [Bitton 1987].

This article presents an adaptive technique for reducing seek time. The idea is to copy a small number of frequently referenced disk blocks from their original locations to a reserved space near the middle of the disk. The term "adaptive" means that no advance knowledge of the frequency of reference to the data blocks is required. Instead, reference frequencies are estimated by monitoring the stream of requests for data from the disk.

Several previous studies have investigated the idea of rearranging data to reduce access time. The contributions of this article are as follows:

—We show that only a tiny fraction of a disk's data needs to be rearranged to achieve significant reductions in seek time.

This work was supported by National Science Foundation Grant no. CCR-8908898 and in part by CESDIS.

Authors' address: Department of Computer Science, University of Maryland, College Park, MD 20742; email: {akyurek; salem}@cs.umd.edu.

—We study the relationship between block rearrangement and other seek time reduction techniques such as disk head scheduling and anticipatory disk arm movement.
—We characterize workloads for which block rearrangement can be expected to work well.
—We show that rearranging small pieces of data (blocks, rather than tracks or cylinders) is a good idea.
—We show that the frequently referenced blocks can be predicted accurately and with low overhead, despite the large number of blocks on the disk.

Our performance experiments show that expected seek times are reduced 30% to 85% (depending on workload) by adaptively rearranging about 3% of the data on the disk. Another benefit of block rearrangement is reduced disk-queuing times due to reduced service times.

1.1 Related Work

That a disk's performance can be improved by clustering frequently accessed data is well known. If data references are derived from an independent random process with a known, fixed distribution, it has been shown that the *organ pipe* heuristic places the data optimally [Grossman 1973; Wong 1980]. The organ pipe heuristic calls for the most frequently accessed data to be placed in the center of the disk. The next most frequently accessed data is placed to either side of the center, and the process continues until the least-accessed data has been placed at the edge of the disk. More recently, similar results have been shown for optical storage media [Ford 1991].

In practice, data references are not drawn from a fixed distribution, nor are they independent. However, variations on the organ pipe heuristic have been shown to be effective for more-realistic reference patterns. Recently, several papers have proposed adaptive applications of this idea.

Two recent studies [Ruemmler 1991; Vongsath 1990] considered "cylinder shuffling." That technique measures disk cylinder reference frequencies over some period of time (e.g., one day) and then reorders the cylinders based on the measured frequencies. Cylinders are reordered using the organ pipe heuristic. In Vongsath [1990], this technique was found to reduce mean seek times by as much as 40% to 45%. Using a wider variety of workloads and disks, Ruemmler [1991] reported disk service time reductions of up to 10%. Seek time reductions were not reported. For one set of traces, the technique actually increased the mean seek distance and seek time, resulting in degradation of the service time. The study presented in Ruemmler [1991] also considered block shuffling. For those traces for which cylinder shuffling improved performance, block shuffling was found to provide greater improvements.

A related approach is employed in the experimental iPcress file system [Staelin 1991], which monitors access to files and moves files with high "temperatures" (frequency of access divided by file size) to the center of the disk. Unlike adaptive rearrangement and shuffling, iPcress moves only file

data. File system metadata are not rearranged. Also, the technique embodied in iPcress requires file system modifications.

The technique presented here differs from those described above in at least one of the following ways:

—*Data is not shuffled, but is instead replicated and copied to a reserved portion of the disk.* Replicating data and copying it to a reserved region has several advantages over data shuffling. It does not move infrequently accessed data at all. Thus, any placement optimizations made by the file system for such data (such as sequential placement of a file's blocks) are disturbed only temporarily by block rearrangement. Hot data that cools down are always return ed to where they were originally placed by the file system. The principal disad vantage of the replicate/copy method is disk space overhead: a small portion of the disk must be reserved for copies of hot blocks.

—*Blocks, rather than cylinders or files, are moved.* The use of small granules (blocks) is advantageous because many frequently referenced blocks can be grouped onto a small number of cylinders, creating artificial "hotspots." A potential drawback of this technique is the cost of maintaining reference counts for individual blocks. However, because we can accurately estimate reference frequencies using little space, this overhead is reduced.

—*Only a small fraction of a disk's data is rearranged at any time.* Our experiments indicate that most of the benefits of rearrangement can be achieved using a very small fraction of the data. Experiments reported in Ruemmler [1991] indicate that the same is true for shuffling. (However, the technique used for partial shuffling was not described.) Costs are also reduced by moving only a fraction of the data. Daily shuffling of all of a disk's cylinders was found to take as long as half an hour [Vongsath 1990], whereas rearrangement of a small number of blocks can be accomplished in a few minutes.

Since only a small amount of data is rearranged, block rearrangement is also very easy to turn off. Only the rearranged data must be copied back to its original location. Data that has been rearranged but not updated need not be moved at all. Partial shuffling is somewhat different. Although each shuffle may involve only a fraction of the data, data are never returned to their original locations (unless they are fortuitously shuffled there). To return a disk to its original layout after several shuffles, all of the data on the disk may have to be moved.

—*The problem of efficiently measuring reference frequencies of large numbers of blocks is addressed.* Block reference counts require little space compared to the data blocks themselves. However, reference counts should be maintained in memory so that they can be updated efficiently. For a device driver or controller implementation, this means that the counts will occupy kernel or controller memory, which is often at a premium.

—*Rearrangement is transparent to the file system that manages the disk.* Like shuffling, but unlike iPcress, rearrangement can be implemented in a

device driver (our approach) or in the disk controller. No file system modifications are required.

Other techniques for seek time reduction include head scheduling and anticipatory head movement. Head-scheduling techniques attempt to reorder queued-disk requests to improve performance. Several head-scheduling algorithms have been proposed and studied [Geist 1987; Hofri 1980; Seltzer 1990]. King [1990] suggests that disk idle times may be used to reposition the disk head in anticipation of future requests. If the head can be placed closer to the location of the next request, seek time can be reduced.

Data placement techniques, such as adaptive rearrangement, are orthogonal to both head scheduling and anticipatory disk arm movement. All of these techniques can be used together. In this article, we present the results of experiments in which adaptive rearrangement is combined with several head-scheduling algorithms and with anticipatory disk arm movement.

The remainder of the article is organized as follows. In the next section we give an overview of the block rearrangement technique. Sections 3 and 4 present in detail the components of the adaptive rearrangement system. In Section 5 the traces and the disk simulator we have used in our experiments are described. The simulator was validated against measurements of a disk driver that implements adaptive rearrangement. The results of this validation are presented in Section 6. Section 7 presents the results of our main body of simulation experiments. Finally, Section 8 presents additional simulation studies driven by traces from a VMS system.

2. ADAPTIVE REARRANGEMENT OVERVIEW

Block rearrangement is motivated by the fact that access to data stored on disks is highly skewed [Floyd 1989; Ruemmler 1991; Staelin 1991]. If the "hot" (frequently accessed) blocks are spread over the surface of the disk, distant from each other, long seek delays may result.

Hot blocks can be clustered to reduce seek times. Under our block rearrangement technique, this is achieved by copying hot blocks onto a set of reserved cylinders in the middle of the disk. The block layout outside of the reserved cylinders is left undisturbed. If the reserved cylinders accommodate blocks which get most of the references, we expect that (1) the disk head will tend to linger over those cylinders and (2) seek distances will be reduced.

Figure 1 illustrates the organization of an adaptive disk driver or controller. When rearrangement is implemented on a disk, a small number of cylinders at the center of the disk are reserved to hold copies of selected blocks of data. The component labeled *Hot-Block Table* contains a list of these blocks and their locations in the reserved cylinders. Incoming requests are compared against this list and directed to the reserved cylinders if the requested block resides there.

The components labeled *Reference Stream Analyzer* and *Block Arranger* implement adaptive rearrangement by moving blocks to and from the reserved cylinders and updating the *Hot-Block Table*. The *Reference Stream Analyzer* monitors the stream of data requests. Periodically, it produces a list

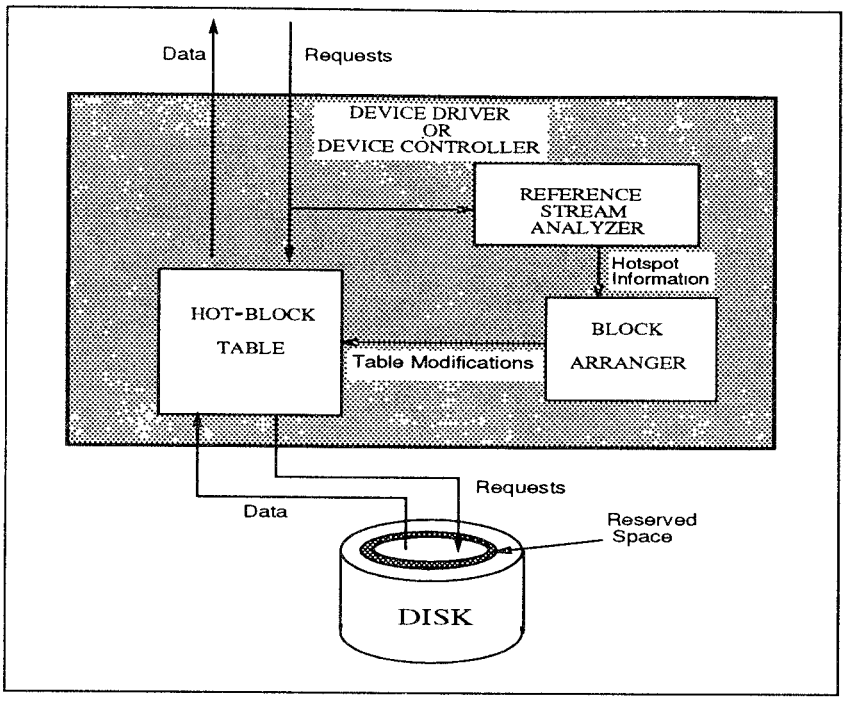

Fig. 1. An adaptive device driver or controller.

of hot (frequently referenced) blocks, ordered by frequency of reference. In the next section we discuss the production of these lists in more detail.

The *Block Arranger* uses the hot-block lists to determine which blocks should be placed in the reserved cylinders. Each time a new list is produced, the arranger copies all of the blocks on the list from their original locations to the reserved cylinders (if they are not already there). Blocks that are in the reserved cylinders but that do not appear on the list are copied back to their original locations. Hot blocks are placed in the reserved cylinders using the organ pipe heuristic. As each block is placed, the arranger updates the *Hot-Block Table* appropriately so that subsequent requests for that block are directed to the proper location.

A desirable feature of this technique is that it is transparent to the file system (or other) code that uses the adaptive driver. Our implementation of the technique in a UNIX SCSI device driver (Section 6.1) required no changes to the UNIX file system.

3. REFERENCE STREAM ANALYSIS

The *Reference Stream Analyzer* is a key component of the adaptive rearrangement technique. Its task is to monitor the stream of requested data and to report which blocks are most frequently requested. The analyzer makes its

reports periodically, and each report describes request frequencies observed since the last report.

A tradeoff exists between the cost of monitoring the reference stream and the accuracy of the analyzer's reports. In general, the analyzer may make mistakes in its report. These mistakes may take the form of out-of-order blocks in the report, frequently referenced blocks that fail to appear at all, and infrequently referenced blocks that appear inadvertently. In the following, we describe the monitoring procedure used by the analyzer and discuss how these errors may arise. In the discussion, we will denote the list of hot blocks produced after the ith monitoring period by $\tilde{H}(i)$. The list of hot blocks that would be produced (after the ith monitoring period) by a *perfect* analyzer will be denoted $H(i)$.

3.1 Monitoring Technique

A simple way to monitor the reference stream is to maintain a counter for each disk block and to increment the counter each time the block is requested during the monitoring interval. At the end of the interval, the most frequently referenced blocks and their counts are reported, and the counts are cleared in preparation for the next interval.[1] The number of reported blocks is a parameter of the analyzer that is set according to the amount of disk space that is being reserved for block rearrangement.

This technique produces an exact characterization of the reference frequencies during the interval, i.e., $\tilde{H}(i) = H(i)$. However, if there is a large number of blocks, many counters must be maintained. Although the counters are much smaller than the blocks themselves, they should be maintained in memory so that they can be accessed with little overhead. Therefore the space occupied by the counters is important and should be kept small.

A more space-efficient alternative is to maintain a small group of counters for blocks that have been recently requested. When such a block is requested again, its counter is incremented. When a block that does not have a counter is requested, one of the counters may be assigned to count references to the newly referenced block. The count of references to the block that was previously assigned to the counter is lost. A variety of heuristics, such as "Least Recently Used (LRU)," can be used to determine which counter to reassign.

This simple technique takes advantage of the fact that the number of blocks that are referenced regularly is usually much smaller than the total number of blocks on the disk. In our traces, only 12% to 13% of all the blocks on the disk are referenced in each day's trace. A disadvantage is that, in general, $H(i)$ and $\tilde{H}(i)$ will not be the same. A frequently requested block may not be reported as such if a "cold spell" causes its counter to be reassigned during the monitoring interval. Conversely, a cold block may be included on the hot list if it receives a burst of references near the end of the monitoring period.

[1] Information from *several* previous intervals could also be combined using a decay function to determine the hot blocks.

In our experiments we used a variant of the LRU method to estimate block reference counts. When a fresh counter is needed, the least recently used "cold" block is replaced. A block in the list is cold if its counter's value is less than a prescribed fraction of the total number of requests so far. (The prescribed fraction is a parameter of the reference analyzer.) If there are no cold blocks in the list then the current reference is not counted.

We selected this LRU-variant method because it was both very effective on our traces and simple to implement. There is a variety of other space-efficient monitoring techniques that could be used by the reference analyzer. Several are presented and compared in Salem [1992].

3.2 Monitor Parameters

Besides the cold-block threshold described above, the monitoring technique uses two parameters: the number of counters to be maintained and the length of the monitoring intervals. The number of counters controls a tradeoff between memory use and accuracy. It must be determined by a system administrator based on the amount of memory that is available. One guideline is that the number of counters should be at least as great as the number of blocks that will be rearranged. This is because the analyzer's report to the rearranger can include no more blocks (and counts) than there are counters. For the experiments reported in this article, the number of counters was set between 5% and 10% of the number of blocks on the traced disk. We also performed simulations to explore the relationship between the accuracy of the analyzer and the performance of the rearrangement technique. The results are reported in Section 7.

The length of the monitoring interval is a more-difficult parameter to set because the relationship between the length of the interval and the performance of the system is difficult to determine. Hotspots that are monitored during interval i determine the contents of the *Hot-Block Table* during interval $i + 1$. To achieve good performance, we should select monitoring intervals such that blocks that are frequently referenced during interval i tend to be referenced frequently during interval $i + 1$ as well. Since a cost is incurred (rearrange ment of blocks) at the end of each interval, it is also desirable to make the intervals long or to make the interval boundaries coincide with periods of low load.

For our experiments, we selected 24-hour intervals because they matched the natural cycles of our workload. We did not attempt to vary the length of the interval systematically. A 24-hour interval is long enough that the overhead of rearranging hot blocks is negligible. It is short enough that hot spots are not expected to change significantly from interval to interval. Previous experiments we performed on our traces (in conjunction with other work) suggested that hot spots changed only slowly over the course of the trace. The shuffling experiments reported in Ruemmler [1991] showed that, after a disk is shuffled, performance degrades very gradually over time. Experiments with various shuffling intervals were also reported, and intervals between daily and weekly were recommended.

4. BLOCK PLACEMENT

Block rearrangement brings frequently referenced blocks closer together on a small set of cylinders in order to reduce seek distances. These distances can be further reduced if the placement of the blocks in the reserved area is done appropriately.

A simple technique is to place the blocks sequentially in the order of their original disk addresses (block numbers). We will call this technique *serial placement*. Serial placement has the advantage that it preserves placement decisions made by the file system. In other words, if a pair of rearranged blocks was originally placed close together by the file system, they will remain close together in the reserved area of the disk. For example, many UNIX file systems allocate a file's blocks close to each other [McKusick 1984]. If all of a file's blocks are hot, they will remain in close proximity after rearrangement.

Another technique is to cluster the blocks according to their rank in the hot-block list, i.e., according to their frequency of reference, using the organ pipe heuristic. Assuming that C blocks fit on a cylinder, the C hottest blocks are placed in the middle cylinder of the reserved area. The next C hottest blocks are placed on an adjacent cylinder, and so on so that the final cylinder reference distribution across the reserved cylinders is an organ pipe distribution. This technique has the advantage that it creates very hot cylinders. However, it may destroy sequential-placement decisions made by the file system.[2]

Our simulator implements both techniques. In Section 7 we report the results of experiments designed to compare them.

5. PERFORMANCE EVALUATION METHODOLOGY

Trace-driven simulation is the principal tool we have used to evaluate the performance of block rearrangement. However, we have also implemented adaptive block rearrangement in a UNIX disk driver. The implemented system has been operational on a file server for more than one year. During that time, measurements have been taken to monitor its performance.

Conceivably, we could have avoided simulation entirely and performed all of our experiments using the implemented system. However, for some experiments this would have been difficult or impossible. By using a simulator and reference traces collected at different sites, we have been able to evaluate the block rearrangement technique in different environments and under a variety of workloads. The simulator also simplified our study of the interaction of block rearrangement, disk head scheduling, and anticipatory disk arm movements.

For these reasons, we selected simulation as our primary performance evaluation tool. Measurements from the implementation have been used primarily to validate the simulator. In the remainder of this section, we

[2]The destruction is temporary. If the blocks cool, they will be restored to their original positions on the disk.

Table I. Traced Systems

Name	Type	Traced Disk	Physical Memory	Operating System	File system Block Size
sakarya	Sparcstation 2	Fuji M2266SA	32 MB	Sun-OS 4.1.1	8 KB
ballast	Sun 3	Fuji 2351	16 MB	Sun-OS 3.2	8 KB
snake	HP 9000/720	Quantum PD425S	32 MB	HP-UX 8.05	8 KB

describe the simulator and the traces used to drive our studies. In the next section we briefly describe the implementation and present a comparison of simulated and measured results for the purpose of validating the simulator. The main body of simulation results is then presented in Section 7.

5.1 Disks and Traces

For our simulations we used disk reference traces collected from three different UNIX systems. The traced disks on these systems vary in capacity, speed, and other properties. Below, we describe the systems, the disks, and the traces themselves.

5.1.1 *Traced Systems.* Table I gives some information about **ballast**, **sakarya**, and **snake**, the three UNIX systems on which the traces were collected. The first two of these systems are at the Computer Science Department of the University of Maryland in College Park, and the third one is at the University of California, Berkeley.

Ballast is a Sun 3 workstation which was running version 3.2 of Sun-OS at the time the traces were collected. Ballast served as a network file server for 45 Sun 3 workstations at the University of Maryland's Computer Science Department. The traces reflect requests generated by multiple concurrent processes running on networked workstations. The file system on the traced disk contained primarily binary files and was shared by other workstations via NFS. Traces from ballast cover approximately 16 hours of each 24-hour interval from 10:00 a.m. until 2:00 a.m. the next day.

Sakarya is a Sun Sparcstation 2 workstation which runs Sun-OS 4.1.1. Two different types of traces were gathered from sakarya's disk. First, we used the disk to store executable files and libraries that were shared by 14 Sparcstations through NFS. The file system was mounted read-only on the client workstations. The user population of the workstations consisted of about 40 faculty and graduate students in the Computer Science Department of the University of Maryland. Later we used the disk to store user files. Home directories of 20 users in a research group were stored on the disk. The primary activities of the users included running simulations, editing, and electronic mailing. Both types of traces cover 16 hours daily from 7:00 a.m. until 11:00 p.m.

Snake is a Hewlett-Packard 9000/720 workstation which acted as a file server for nine HP 9000/720 diskless workstations at the University of California, Berkeley. Snake ran HP-UNIX version 8.05. The workstations had accounts for faculty, staff, graduate students, and computer science classes.

Table II. Characteristics of the Traced Disks (The Fuji M2266SA and the Quantum disks use read-ahead buffering. The Quantum disk also uses immediate write reporting for eligible write operations.)

Type	Capacity	Cylinders	Heads	Sectors/ Track	Rot. Bytes/ Sector	Speed (RPM)	Track Buffer	Interface
Fuji M2266SA	1 GB	1658	15	85	512	3600	256 KB	SCSI-2
Fuji 2351	353 MB	785	20	46	512	3600	–	SCSI
Quantum PD425S	407 MB	1520	9	78	512	3605	56 KB	SCSI-2

The disk stored the root file system and the swap partition. The main use of the system was for compilation and editing. Traces from snake cover the full 24-hour interval daily. They were provided to us by HP Laboratories. Further information about the snake system and the traces themselves can be found in Ruemmler [1993].

5.1.2 *Disks.* The traced systems used three different types of disks. The characteristics of these disks are summarized in Table II. All of the disks have SCSI (Small Computer System Interface) interfaces. The Quantum disk has a feature called immediate write reporting. This means that the disk reports a write operation as completed as soon as the data is transferred from the host to the disk's buffer. This feature is enabled only when the write is a physical extension of the last write operation. The M2266SA and Quantum disks both support read-ahead buffering. With read-ahead buffering, when requested data is read off the recording media into the disk's buffer, the disk continues reading data into its buffer even after the requested piece of data is read. Later, if blocks that are already in the buffer are requested they are simply transferred to the host from disk's buffer. This makes read operations complete faster.

Both of these features affect the service time perceived by the host machine. Both are implemented by our simulator. In our experiments, simulation of each feature is enabled only if the feature is supported by the disk from which the input trace was generated.

5.1.3 *Traces.* In Table III we summarize the characteristics of the traces which were used in the simulations. Four days worth of trace data from each file system were used.

The traces contain a trace record for each disk I/O operation performed during the tracing period. I/O requests that are satisfied from the disk buffer cache in the main memory are not included. Each trace record includes a beginning address on the disk for the I/O request together with a requested transfer size. Each record also includes a read/write flag and a timestamp indicating the submission time of the request. Timestamps have a resolution of 1 microsecond, 5 microseconds, and 10 milliseconds, on traces from snake, sakarya, and ballast, respectively.

Several interesting features of these traces are apparent from Table III. The first is that the sakarya1 and ballast traces include significant numbers of write requests. This is despite the fact that the traced file systems are mounted "read-only" on the client workstations. These writes are generated

Table III. Trace Sets Used in the Simulations

Name	Description	Days	Number of Requests	Percentage of Reads
sakarya1	binaries and libraries	4	177269	56
sakarya2	user files	4	124556	21
ballast	binaries and libraries	4	401477	85
snake	root file system and swap	4	643872	36

by the file system itself, for the purpose of updating its data structures (i-nodes). In particular, the time at which a file was most recently read is recorded in the file's i-node.

A second feature is the lower percentage of reads in the sakarya1 traces as compared to the ballast traces. The implementation of the buffer cache changed between SunOS 3.2 (ballast) and SunOS 4.1.1 (sakarya1). We attribute the lower percentage of reads primarily to improved buffer performance in the later version of the operating system. None of the traces include read requests that hit the file system buffer cache.

Finally, we note that with one minor exception, the maximum request size in all of the traces is 8KB, the file system block size. The exception is that a total of nine requests (out of 643,872) in the snake traces were for more than 8KB of data. Our implementation of block rearrangement handles such requests by splitting them into multiple, smaller requests. The simulator, however, does not simulate request splitting. In our simulations, the lengths of these nine requests were truncated to 8KB each. Because of the small number of large requests, we do not feel that this simplification had a significant impact on our results. However, large requests could have an impact if they were more prevalent. This is because block rearrangement may split such requests into physically discontiguous subrequests. We return briefly to this problem in Section 8, where we describe block rearrangement under the VMS file system.

5.2 Simulator

The disks and disk driver are modeled using a discrete-event simulator which is implemented using the CSIM simulation library [Schwetman 1987]. The simulator reads a trace file sequentially and simulates the arrival of the traced requests using the timestamps. The driver's request queue is managed using a head-scheduling policy which is a parameter of the simulator. By default the *CLOOK* head-scheduling policy is used. This is the policy used by the traced UNIX systems.

When the request at the head of the queue is dispatched, the simulator calculates the disk service time for the request as a combination of several components:

—**Controller overhead** is a fixed time for the disk controller to initiate a request.

—**Seek time** is the time to position the head assembly over the requested block's cylinder. It is calculated by applying the seek time function for the disk to the current seek distance.

—**Rotational latency** is the time interval between the end of the seek and the time at which the requested data begins to pass beneath the disk head. Rotational position is modeled assuming that rotational speed of the disk platters is constant at the nominal velocity listed in Table II.

—**Media transfer time** is the time needed to transfer data between the platters and the disk's buffer. The transfer rate is calculated using the rotational speed of the disk and the number of bytes per disk track. Head switch times are also included in the media transfer time in case a request crosses track or cylinder boundaries.

—**I/O channel transfer time** is the time to transfer data between the host and the disk over the channel. In our case the channel is the SCSI bus. The transfer rate is taken as the maximum transfer rate of the SCSI bus used in the traced system. We did not attempt to simulate channel contention.

The seek time functions and transfer times for each of the disks are given in the Appendix.

The simulator combines these service time components differently depending on the type of request and the features of the disk drive. For normal read requests (those that do not hit the read-ahead buffer), media transfer and channel transfer overlap. Service time is calculated as the sum of seek time, rotational latency, controller overhead, and the maximum of the two transfer times. For write requests, seek time and data transfer overlap. Service time for normal writes (those for which immediate reporting is not in effect) is calculated as the maximum of the seek time and the channel transfer time, plus the sum of the remaining three components. The simulator records the distributions of disk queue waiting times, service times, seek distances, and seek times separately for read and write requests.

Immediate reporting was simulated for the snake traces. Immediate reporting is enabled only for asynchronous write requests, only if the previous request was also a write, and only if the current write is a physical extension of the previous one. For such writes, the service time is taken to be the sum of the controller overhead and I/O channel transfer time only, since the controller does not wait for the data to be written to the disk before responding to the host. The controller is assumed to begin the actual disk write operation immediately after responding to the host. In the event that the next request arrives at the controller before the write operation is complete, it is delayed there until the write is finished. This delay is included in the service time of the delayed request.

A read-ahead buffer is simulated for the sakarya1, sakarya2, and snake traces. After a read request, the controller reads additional data into its buffer, continuing from the point at which the read request left off. Read ahead continues until the read buffer is full or until a new read request that cannot be satisfied from the buffer arrives. To avoid potential consistency problems, the contents of the read buffer are discarded when a write request

arrives. The Fuji M2266SA is modeled using six independent track-sized buffers. A simple round-robin replacement policy is used to select a buffer when new read-ahead data must be stored. The Quantum disk is modeled using a single buffer.

If a read request for buffered data is received, the request can be satisfied without additional access to the platters. In that case, the read service time is taken as the sum of the controller overhead and the channel transfer time only. In case only part of the requested data is buffered, the request is treated as an unbuffered read.

The traces from sakarya were collected from a disk on which block rearrangement was already implemented. This disk has a reserved area already set aside on it for placing the frequently accessed blocks. However, the disks on ballast and snake did not include reserved regions. To simulate block rearrangement, a reserved region is inserted at the center of the simulated disk. Thus, the simulated disk is slightly larger than the actual disk. These extra cylinders make the simulated seek times slightly pessimistic because seeks that cross the additional cylinders require additional time. However, because of the small number of extra cylinders, the nonlinear relationship between seek time and seek distance, and the infrequency of such seeks, their impact is not substantial.

The simulated performance metrics do not include the cost of periodic rearrangement of the data. We feel that this is justified by the fact that rearrangement is inexpensive and occurs very infrequently. In all of our simulations we assume that blocks are rearranged only once per day. In our implementation (described below), rearrangement was scheduled to occur late at night, during a period of low disk activity. Rearrangement typically took only a few minutes. Furthermore, the adaptive driver is implemented so that data movement caused by rearrangement is interleaved with I/O requests from the file system. There is no need to take the disk off-line while it is being rearranged.

6. MEASUREMENTS AND SIMULATOR VALIDATION

The adaptive block rearrangement system was implemented under SunOS 4.1.1 and has been operational on sakarya for more than a year. During this time we have been monitoring the performance of the system and tracing the I/O requests on the disks. In this section we provide a brief description of the implemented system. Additional implementation details can be found in Akyurek [1993].

6.1 Implementation

The rearrangement system was implemented through modifications to the SCSI disk device driver of SunOS 4.1.1. In addition, several user-level programs were written to control the modified driver. The system design is similar to that shown in Figure 1. The only difference is that the Reference Stream Analyzer and a part of the Block Arranger are implemented as user-level programs. These programs interact with the modified driver through system calls.

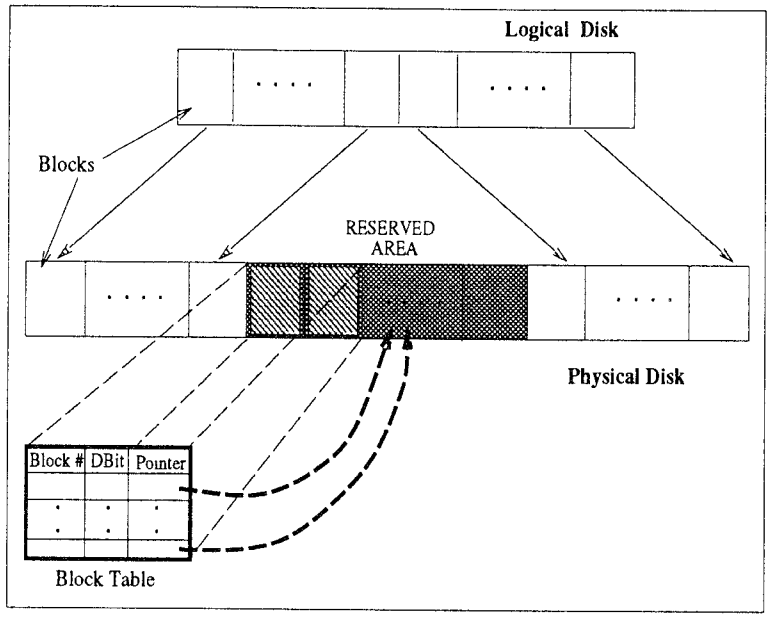

Fig. 2. Block mapping on a rearranged disk.

The modified driver implements the following functions of the rearrangement system:

—**Reserving Space**. The driver conceals a portion of the disk from the rest of the operating system. This area is used as the reserved space for rearrangement. In this way the rearranged disk is made to look smaller than it actually is. The driver implements the mapping between the smaller *logical disk* and the physical disk (Figure 2).

—**Hot-Block Table**. The driver maintains the hot-block table, which records the rearranged blocks and their positions in the reserved area on the disk. A copy of the table is also kept on the disk so that the memory copy can be recreated after a failure or shutdown.

—**Block Mapping**. When a request arrives, the driver performs a look-up in the hot-block table. If the requested block has been rearranged, the driver maps the request to the copy in the reserved area.

—**Block Movement**. The driver provides kernel entry points through the UNIX *ioctl* system call for user-level processes to control the movement of blocks into and out of the reserved area. Using one call, a given block can be copied to a given position in the reserved area. An entry is also made in the block table for this block. Using another call, all of the blocks in the reserved area can be copied back to their original positions. Block movement within the driver is performed one block at a time. Requests to a block that is in flight are temporarily delayed until the block is moved.

—**Request Monitoring**. The driver records block requests in a small table. The contents of the table can be read by a user-level process using a system call.

We have also incorporated some routines in the driver for performance monitoring. Although they are not a part of the block rearrangement system, these routines helped us to access the performance of the system. We are able to measure statistics such as disk queue waiting times and disk service times. Seek distances are inferred from the addresses of the currently requested block and the previous one. These statistics are recorded in a table in the driver and can be read through a system call by user-level processes. Seek times cannot be measured directly. Instead, we infer the seek time distribution from the measured seek distance distribution and the seek time function for each disk (see the Appendix).

File allocation in UNIX systems is done in terms of file system blocks. Each block consists of several disk sectors. A *block* for the block rearrangement system is the same thing as a UNIX file system block. On all of the traced systems, the file system block size was 8 kilobytes (16 disk sectors).

6.2 Comparison of Measured and Simulated Performance

To validate the simulator, we compared simulated performance against the measured performance of this system. While the sakarya1 and sakarya2 traces were being collected, we monitored the performance of the adaptive driver. As described in Table III, these trace sets cover periods of four days each. During each of these periods, block rearrangement was applied on alternate days only. This was done to allow us to test the accuracy of the simulator with and without rearrangement. In the remainder of this section we present the results of these tests.

Table IV compares simulated and measured performance for the sakarya1 and sakarya2 traces. On days two and four, 3500 blocks were rearranged. On days one and three, none were. The table lists both measured and simulated values. Figure 3 shows the measured and simulated service time distributions for days one and two of each type of trace.

The data show that the simulator is reasonably accurate. Differences between simulated and measured seek distances (and times) are probably caused by the interaction of the disk-head-scheduling policy with small errors in the simulated service times. This can result in minor variations in request ordering between the simulation and the real system.

Simulated waiting times are somewhat less accurate than the service times, particularly for the sakarya2 traces. Waiting time is more difficult than service time to simulate accurately. This is because small errors in simulated service time may accumulate and contribute to large waiting-time errors when the request queue is long. This effect is most noticeable in traces with large numbers of write requests, since writes often occur in bursts. The sakarya2 trace has a much higher percentage of write requests than any of the other traces we have studied.

Table IV. Comparison of Measured Values and Simulation Results for the Sakarya Traces (Seek distances are in cylinders and times are in milliseconds.)

Trace Type	Day	Rearrangement	Result Type	Mean Seek Dist	Mean Seek Time	Mean Service Time	Mean Waiting Time
sakarya1	1	Off	Measured	315	8.01	21.15	69.98
			Simulated	333	8.04	20.71	67.47
	2	On	Measured	27	1.16	14.08	35.65
			Simulated	21	0.99	14.11	38.34
	3	Off	Measured	304	7.90	21.89	61.47
			Simulated	304	7.76	20.34	56.47
	4	On	Measured	22	1.10	13.83	44.52
			Simulated	21	1.03	14.08	49.51
sakarya2	1	Off	Measured	147	4.79	17.09	4.33
			Simulated	145	4.71	16.61	5.50
	2	On	Measured	41	2.26	14.63	8.75
			Simulated	40	2.21	14.67	11.27
	3	Off	Measured	125	4.45	17.24	6.41
			Simulated	121	4.34	16.77	7.82
	4	On	Measured	66	2.97	15.27	3.96
			Simulated	65	2.90	14.90	5.54

7. SIMULATION RESULTS

In this section we will present the results of trace-driven simulations we have run to evaluate the performance of our block rearrangement technique under different workloads and using different types of disks. We also present an analysis of the sensitivity of these results to changes in the parameters and policies used by the block rearrangement system. Finally, we consider the interaction of block rearrangement with other seek time reduction techniques, namely disk head scheduling and anticipatory disk arm movement.

7.1 Different Workloads

Simulation results using four different types of traces are summarized in Table V. Results from three of the four trace days are shown for each type of trace, since the first day's worth of trace data was used only for determining the list of hot blocks for the second day. For simulations with rearrangement, the reserved area held 3500 blocks for traces from sakarya, 1400 blocks for ballast traces, and 1600 blocks for snake traces. This represents a space overhead of approximately 3% for each of the simulated disks. Blocks were placed in the reserved area using the organ pipe technique. The Reference Stream Analyzer used the LRU-variant algorithm described previously to determine hot blocks. The number of counters used by the analyzer was set to 3500 for ballast and snake traces and to 6000 for sakarya traces. The guideline we used was to set the number of counters between 5% and 10% of the total number of blocks on a particular disk.

For the various trace types, seek distance reductions resulting from block rearrangement ranged from about 55% to over 90%. More important were the reductions in mean seek times. Because of the shape of the seek time functions, large reductions in distance do not necessarily imply correspond-

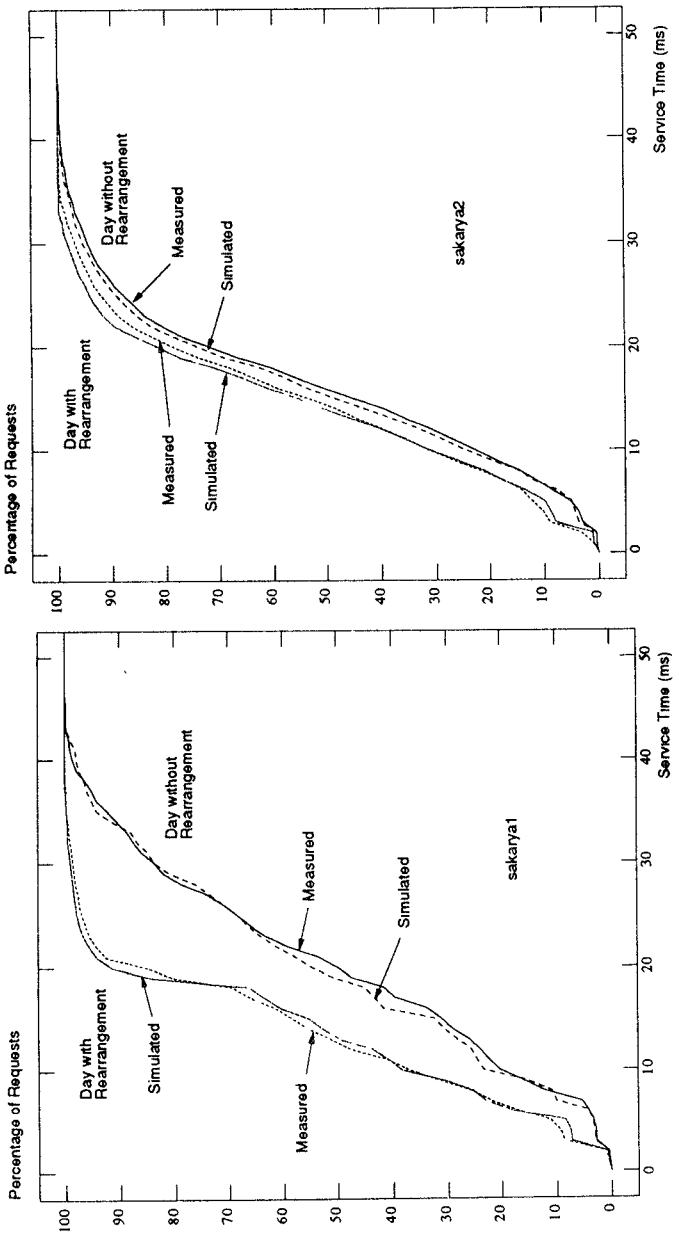

Fig. 3. Service time distributions for two days from the sakarya1 and sakarya2 traces. Both measured and simulated service time distributions are plotted.

Table V. Results of On/Off Simulations for All Trace Types (The minimum, average, and maximum daily mean values over the days of each trace type are reported. Times are in milliseconds; seek distances are in cylinders.)

Trace Type		Rearrangement Not Applied			Rearrangement Applied		
		Min	Avg	Max	Min	Avg	Max
sakarya1	Daily Mean Seek Distance	264	**287**	303	21	**25**	32
	Zero-length Seeks (%)	26	**31**	35	69	**74**	78
	Daily Mean Seek Time	6.78	**7.2**	7.76	0.99	**1.18**	1.53
	Daily Mean Service Time	19.41	**19.**	19.91	14.11	**14.63**	14.98
	Daily Mean Waiting Time	56.47	**65.**	76.89	38.34	**44.38**	49.51
sakarya2	Daily Mean Seek Distance	121	**131**	141	40	**56**	65
	Zero-length Seeks (%)	28	**32**	35	40	**44**	51
	Daily Mean Seek Time	4.26	**4.3**	4.52	2.21	**2.72**	3.05
	Daily Mean Service Time	16.51	**16.**	16.88	14.67	**15.03**	15.51
	Daily Mean Waiting Time	7.82	**11.**	15.87	5.54	**8.46**	11.27
ballast	Daily Mean Seek Distance	114	**116**	118	30	**31**	33
	Zero-length Seeks (%)	32	**33**	34	49	**51**	52
	Daily Mean Seek Time	9.49	**9.60**	9.71	4.30	**4.47**	4.73
	Daily Mean Service Time	25.28	**25.40**	25.47	20.98	**21.15**	21.44
	Daily Mean Waiting Time	29.09	**29.55**	30.17	24.30	**24.42**	24.63
snake	Daily Mean Seek Distance	157	**165**	170	28	**30**	34
	Zero-length Seeks (%)	35	**36**	37	38	**44**	51
	Daily Mean Seek Time	6.23	**6.41**	6.51	2.29	**3.19**	3.51
	Daily Mean Service Time	20.54	**20.70**	20.79	17.26	**17.44**	17.59
	Daily Mean Waiting Time	32.15	**34.25**	35.44	22.30	**24.38**	25.66

ingly large reductions in time. In particular, there is a constant time overhead when a seek operation is initiated. Despite this, mean seek time reductions were substantial, ranging from a low of 35% for a sakarya2 trace to highs of over 80% for sakarya1 traces.

One reason that block rearrangement results in large seek time reductions is that it decreases the number of seek operations. By placing frequently accessed blocks on the same cylinder, the need to reposition the disk head was often eliminated completely. For example, on sakarya1 traces the percentage of requests that did not require a seek was increased from 31% to 74%. On sakarya2 traces it was increased from 32% to 44%. This is an advantage of rearranging blocks rather than cylinders. Cylinder rearrangement cannot reduce the number of seek operations.

The four types of traces we have selected are representative of different types of workloads. The sakarya1 and ballast traces are from file servers that hold primarily read-only data. Most of the write operations in these traces are for updating file system data structures (i-nodes). They are concentrated on a small group of blocks, and they tend to arrive in bursts. The sakarya2 traces are from a disk which stores user files. Request patterns in this trace tend to be more sequential and localized. As a result, seek distances (without rearrangement) are shorter in these traces. The snake traces include requests to both a read-only file system and a swap area—they exhibit a blend of the characteristics of the other trace types.

Block rearrangement produced significant seek time reductions under all of the workloads. However, it performed best on traces in the first group

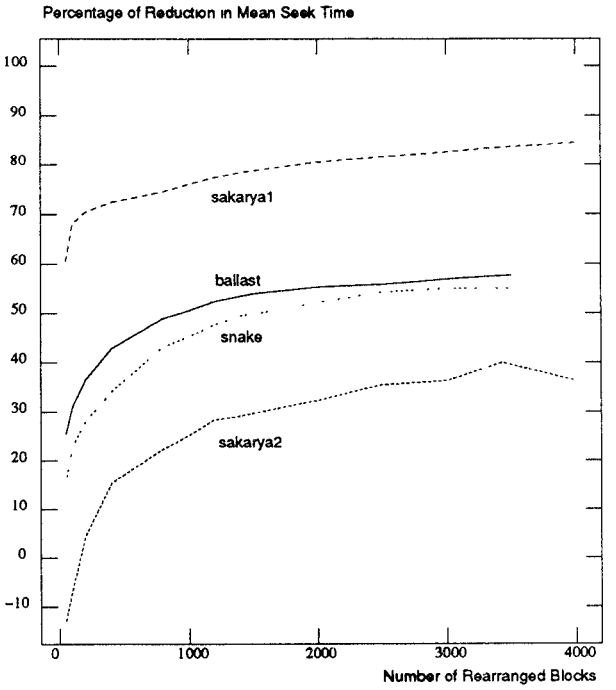

Fig. 4. The change in daily mean seek time reduction as the number of rearranged blocks is varied.

(sakarya1 and ballast). One reason for the poorer performance on the remaining traces is that they include write requests originating from file creation and expansion operations. Rearrangement does not help in such cases, since a previously unused block is unlikely to have been arranged.

A secondary effect of shorter seek times is shorter disk queues. Since the disk's mean service time is reduced by block rearrangement, waiting times in the disk's queue should decrease as well. We found that, with the exception of one day from the sakarya2 traces, block rearrangement always reduced the daily mean waiting times. Although the utilization of the disks was low (less than 5%), the mean waiting times were high. The long waiting times were caused primarily by bursts of write request arrivals. The bursts occur because of the UNIX file system's policy of delaying some writes and sending them to disk at 30-second intervals. The waiting-time reductions achieved by block rearrangement indicate that it may be particularly beneficial for heavily utilized disks or for those with very bursty workloads.

7.2 Disk Space Overhead

The seek time reductions achieved by block rearrangement depend on the amount of reserved space available to hold rearranged blocks. In Figure 4 we have plotted the percentage reduction in average seek time as the number of

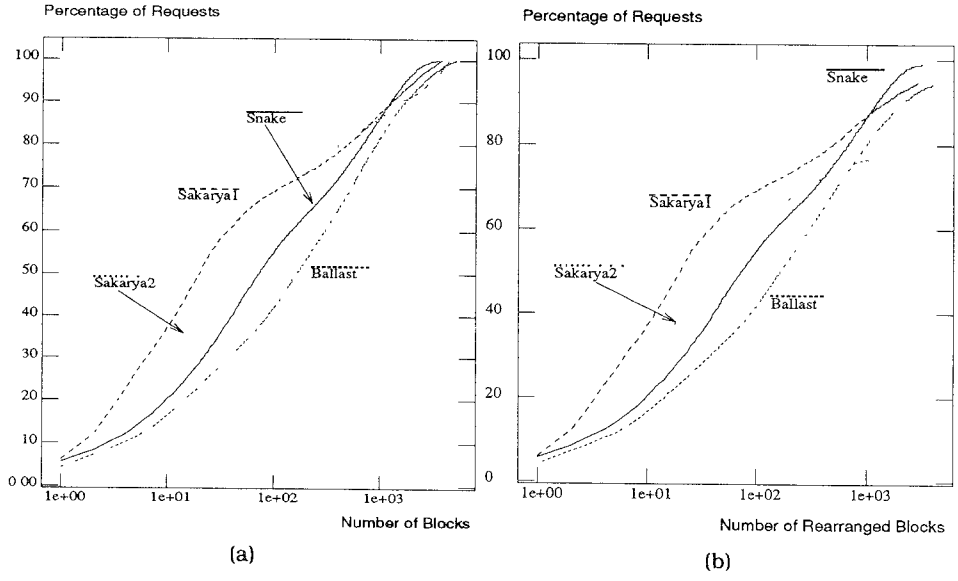

Fig. 5. Request distributions for all the blocks and for the rearranged blocks. To show the distributions more clearly, a log scale has been used on the x-axis.

rearranged blocks is varied. The plotted values are averaged over all traces of each type.

Most of the seek time reductions are due to the first thousand rearranged blocks. These blocks amount to between 1% and 2% of all the blocks on the traced disks. This result can be better understood with the help of Figures 5(a) and 5(b). In Figure 5(a) we have plotted the block request frequency distributions for one day's worth of each type of trace. Figure 5(b) shows the request distribution for the rearranged blocks only, for the same trace days. In each figure, the blocks are sorted in descending order of reference frequency.

Figure 5(a) shows that the block reference distributions are very skewed for all types of traces. The first thousand blocks in each type of trace absorb around 90% of all of the requests. Figure 5(b) shows that the block rearrangement system is successful in capturing the frequently accessed blocks in the reserved disk cylinders. The first thousand blocks placed in the reserved cylinders capture between 70% to 85% of all the requests.

Reference skew can have a significant impact on the performance of the rearrangement technique. To study its effects further, we selected several days' worth of traces that exhibit different degrees of skew. All were selected from the sakarya1 trace set. We then ran experiments to determine how variations in the skew would affect the performance of the block rearrangement system.

Figure 6(a) shows the bock request distributions for the traces we selected. As before, blocks are sorted in descending order of reference frequency.

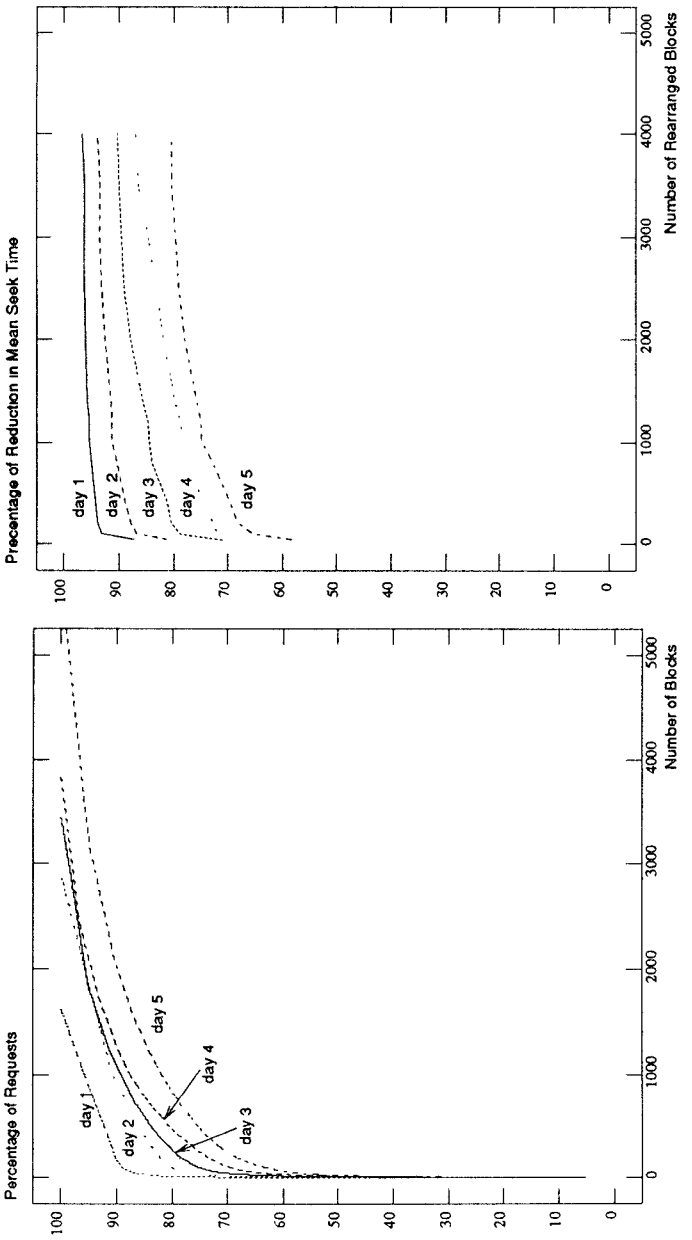

Fig. 6. Mean seek time reductions versus number of rearranged blocks on days with different degrees of reference skew.

Figure 6(b) shows the percentage reduction in mean seek time that was achieved by block rearrangement as the number of rearranged blocks was varied from 50 to 4000.

Bock rearrangement becomes more effective as the reference distribution becomes more skewed. Skew affects performance in two ways. First, increased skew increases the fraction of requests that is absorbed by the reserved region. This allows the disk arm to spend more time there. Second, when the block request skew becomes greater, the distribution of requests over the cylinders in the reserved area also becomes more skewed.

7.3 Data Placement

We performed experiments to compare organ pipe placement to serial placement. The results are summarized in Table VI. The placement policy had only a weak effect on performance. For the trace sets for which rearrangement worked best (e.g., sakarya1), organ pipe placement produced slightly better seek times than serial placement. For the remaining traces, the two policies were nearly indistinguishable. We attribute organ pipe's superior performance on the sakarya1 traces to the fact that they exhibited greater reference skew than any of the others. This allows the organ pipe policy to produce extremely hot cylinders in the reserved region. By grouping hot blocks together, the organ pipe policy may also enhance the effectiveness of read-ahead buffering at the disk, since a request for a single hot block will cause others to be read into the buffer. For the sakarya1 traces, we did not observe any substantial degradation of performance caused by disturbances to rotational optimizations performed by the file system. This is probably because many of the hot blocks contain file system metadata (i-nodes), whose placement is not rotationally optimized by the file system.

7.4 Accuracy of the Reference Analyzer

There are two sources of error during reference analysis. One source is the limited number of counters, causing $\bar{H}(i)$ to differ from $H(i)$. The second is temporal variations in reference patterns, meaning that $H(i)$ differs from $H(i + 1)$.

We ran several experiments to determine how these sources of error affected the performance of the block rearranger. To determine the impact of the number of counters, we ran experiments in which the number was varied. To determine the impact of temporal variations, we ran experiments in which the analyzer was allowed to scan the reference trace in advance. In other words, $\bar{H}(i)$, rather than $\bar{H}(i - 1)$, was used to rearrange the blocks before interval i. Although such an analyzer is not practical, it does allow us to gauge the impact of temporal changes in the reference counts.

Figure 7 summarizes the results of these experiments. The graphs show the mean percentage seek time reductions achieved for each type of trace as the number of rearranged blocks is varied. Results from one day's worth of data from each type of trace are shown. Each curve is labeled with the number of counters used for analysis, plus an indication of whether the

Table VI. Results of Experiments with Different Block Placement Policies (The minimum, average, and maximum daily mean values over the days of each trace type are reported. Times are in milliseconds, and distances are in cylinders.)

Trace type		Daily mean seek distance			Daily mean seek time			Daily mean service time		
		min	avg	max	min	avg	max	min	avg	max
sakarya1	No Rearrangement	264	287	303	6.78	7.28	7.76	19.41	19.89	19.91
	Rearrangement with Organpipe Placement	21	**25**	32	0.99	**1.18**	1.53	14.11	**14.63**	14.98
	Rearrangement with Serial Placement	24	**34**	41	2.27	**2.56**	2.84	15.27	**15.48**	15.69
sakarya2	No Rearrangement	121	131	141	4.26	4.37	4.52	16.51	16.72	16.88
	Rearrangement with Organpipe Placement	40	**56**	65	2.21	**2.72**	3.05	14.67	**15.03**	15.51
	Rearrangement with Serial Placement	40	**56**	66	2.28	**2.77**	3.11	14.62	**15.14**	15.58
ballast	No Rearrangement	114	116	118	9.49	9.60	9.71	25.40	25.28	25.47
	Rearrangement with Organpipe Placement	30	**31**	33	4.30	**4.47**	4.73	20.98	**21.15**	21.44
	Rearrangement with Serial Placement	30	**32**	35	5.08	**5.23**	5.44	21.63	**21.74**	21.91
snake	No Rearrangement	157	165	170	6.23	6.41	6.51	20.54	20.70	20.79
	Rearrangement with Organpipe Placement	28	**30**	34	2.29	**3.19**	3.51	17.26	**17.44**	17.59
	Rearrangement with Serial Placement	31	**33**	36	3.47	**3.56**	3.61	17.22	**17.46**	17.62

reference trace was scanned in advance or not. In each graph, the highest curve illustrates the performance achieved by a perfect analyzer, i.e., one for which exact reference counts for each block are known in advance.

Several results are apparent from these experiments. First, reference counters in excess of the number of blocks being rearranged do not contribute much to the system's performance. Certainly, no more than twice as many counters as blocks should be used. Second, for traces other than sakarya2, the performance achieved with the practical reference analyzer was quite close to that achieved by the perfect analyzer. The sakarya2 traces showed more-temporal variation than the others. In fact, Figure 7 suggests that day-to-day variation of block reference counts is a principal culprit in the relatively poor performance on the sakarya2 traces. One reason for this variation is the substantial number of references to newly created files in those traces. Clearly, the success of block rearrangement depends on the file system that changes slowly over time.

7.5 Disk Head Scheduling

Disk head scheduling is a widely used technique for reducing seek times. Several different head-scheduling algorithms have been proposed and implemented. However, head scheduling is most effective when the disk queues are long, and long disk queues are something we want to avoid in a disk system.

In the traces we have used in this article, the daily mean queue lengths observed by an arriving request (excluding any request being serviced) did not exceed 2.3. The highest mean queue lengths were on the sakarya1 traces,

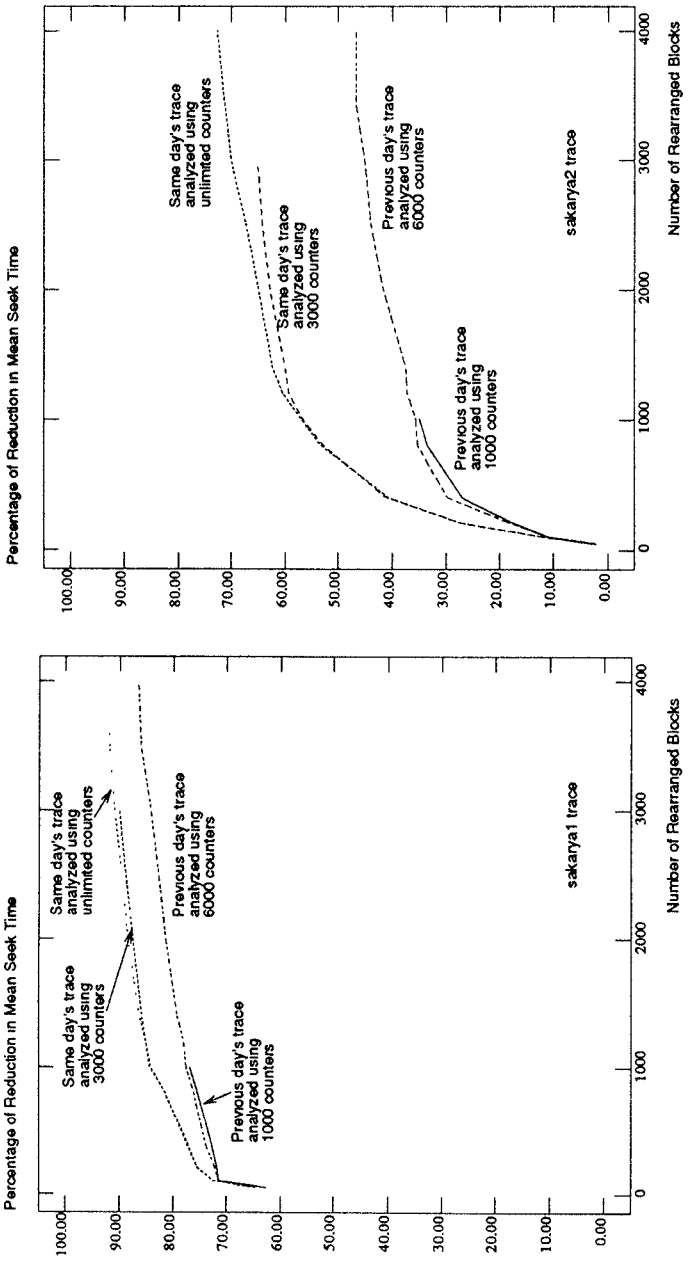

Fig. 7. Reductions in mean seek times when the reference analyzer used different number of counters for estimating hot blocks. The reductions are also shown for the case when the block arranger was provided with exact block frequencies in advance. Results of experiments when the analyzer was allowed to prescan the same day's trace are also plotted.

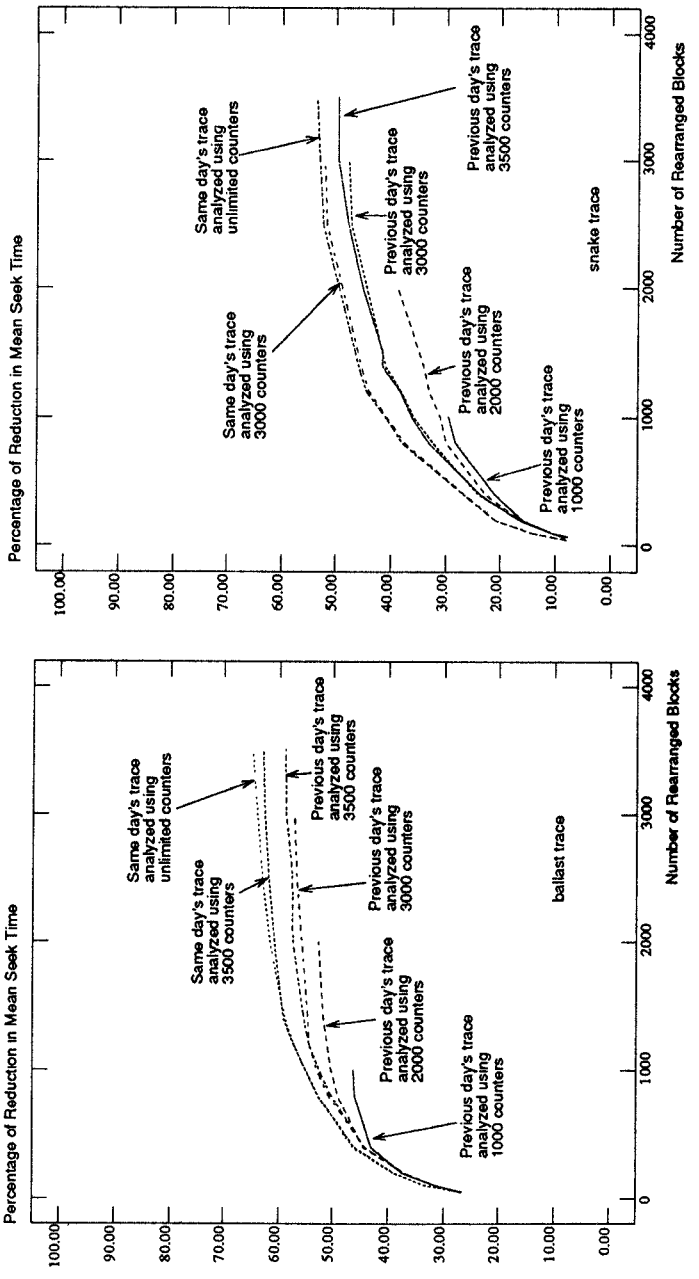

Fig. 7. continued.

where they varied between 1.5 and 2.3. On the snake and the ballast traces the mean queue lengths were between 0.5 and 0.7, and on the sakarya2 traces they were less than 0.5. This means that on average the arriving requests did see a disk queue. Occasionally, the queue can get long because of bursty arrivals (especially writes).

We ran experiments to investigate the interaction between head scheduling and block rearrangement. We consider three head-scheduling algorithms, as described below:

—**First Come First Served (FCFS)** is the simplest scheduling algorithm. Requests are served in the order in which they arrive. When there is no queuing, every scheduling algorithm degenerates to FCFS. We included FCFS as a base case against which we can compare other policies.

—**LOOK** sweeps the disk from one end to the other serving the requests on the way. When there are no further requests in the current sweep direction, the direction is reversed.

—**CLOOK** is similar to LOOK, but the disk is swept in only one direction. When there are no further requests in the sweep direction, a new sweep is begun, starting with the request furthest from the current position of the disk arm. The CLOOK algorithm was used in all of the simulations we have presented so far.

We ran simulations using all three head-scheduling policies both with and without rearrangement. Table VII shows the results of these experiments. We found that block rearrangement did about as well with FCFS as with CLOOK or LOOK, i.e., the scheduling discipline did not interact strongly with block rearrangement. This can be seen by comparing seek times in Table VII when rearrangement is on. When block rearrangement is used, it reduces head movement by capturing the active blocks on a small set of cylinders. Since the disk arm tends to linger over the reserved region, it is likely that queued requests will be for blocks residing on nearby cylinders. Thus, the additional benefit of head scheduling is small when block rearrangement is being used. However, head scheduling may have a greater impact in systems more heavily loaded in which disk queues are likely to be longer than those we observed.

7.6 Utilization of Idle Times

Anticipatory disk head positioning is another technique that can be used to shorten seek times. It is argued in King [1990] that disks are idle most of the time and that idle time can be used to reposition the disk head in anticipation of future requests. If the head can be moved closer to the next requested block, seek times can be reduced.

Like King, we observed that our traced disks were idle most of the time. We ran several experiments to study the effects of anticipatory head movement. As with disk head scheduling, we had two goals for these experiments. First, we wished to determine how the anticipatory movement compared with block rearrangement at reducing seek times. Second, since both techniques can be used concurrently, we wished to study their interaction.

Table VII. Results of Experiments with Different Head-Scheduling Algorithms (The minimum, average, and maximum daily mean values over the days of each trace type are reported. Times are in milliseconds, and distances are in cylinders.)

Trace type	Policy	Daily mean seek distance						Daily mean seek time					
		Rearrangement											
		off			on			off			on		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
sakarya1	FCFS	406	**423**	434	22	**27**	37	9.62	**9.79**	9.93	1.03	**1.26**	1.67
	CLOOK	264	**287**	303	21	**25**	32	6.78	**7.28**	7.76	0.99	**1.18**	1.53
	LOOK	240	**265**	290	20	**24**	32	6.48	**7.00**	7.55	0.99	**1.18**	1.52
sakarya2	FCFS	129	**140**	148	42	**59**	69	4.51	**4.62**	4.76	2.27	**2.81**	3.23
	CLOOK	121	**131**	141	40	**56**	65	4.26	**4.37**	4.52	2.21	**2.72**	3.05
	LOOK	119	**126**	135	40	**53**	64	4.16	**4.29**	4.42	2.20	**2.70**	3.03
ballast	FCFS	131	**133**	135	29	**33**	35	10.34	**10.50**	10.61	4.42	**4.61**	4.86
	CLOOK	114	**116**	118	30	**31**	33	9.49	**9.60**	9.71	4.30	**4.47**	4.73
	LOOK	111	**113**	116	29	**31**	34	9.38	**9.50**	9.60	4.28	**4.46**	4.71
snake	FCFS	186	**196**	201	30	**32**	36	6.96	**7.19**	7.33	3.09	**3.50**	3.83
	CLOOK	157	**165**	170	28	**30**	34	6.23	**6.41**	6.51	2.29	**3.19**	3.51
	LOOK	149	**158**	161	27	**29**	33	6.13	**6.31**	6.41	2.78	**3.20**	3.50

Although anticipatory head positioning is a simple idea, it may be difficult to implement in practice. One problem is how to decide where to reposition the disk head during idle period. Another is how to decide whether an idle period will be long enough for an anticipatory seek. Seek operations are not interruptible on most disks. If an anticipatory seek does not end within the idle period, it will delay the next request.

We first studied an idealized version of the technique that avoids these problems. Specifically, we assumed that the length of each idle period was known in advance, as was the location of the next block to be requested. The idealized head-positioning technique causes the disk head to be moved toward the location of the next request. The anticipatory move is only as long as the idle period will allow. If the idle period is not long enough, the head may stop short of the location of the next request. Clearly, this technique is not implementable. However, its performance does provide us with a bound on how well a more-practical technique might perform.

Table VIII shows the results of our experiments. For each trace type, the first row shows seek times and distances when neither block rearrangement nor anticipatory head positioning were used. The next three rows show the performance when one or both of the techniques are used. A fifth row, which we will explain below, is also included.

The table shows that block rearrangement performs better when combined with idealized anticipatory head positioning. Given this, we were interested in determining whether we would still get an improvement using a more-practical anticipatory movement technique. In particular, we experimented with a "move-to-center" heuristic. This means that when the disk becomes idle, the disk arm is moved toward the center of the disk as far as the duration of the idle period will permit. This heuristic does not presume advance knowledge of the next request. However, it does use knowledge of

Table VIII. Results of Experiments with Idle Time Utilization (The minimum, average, and maximum daily mean values over the days of each trace type are reported. Times are in milliseconds, and distances are in cylinders.)

Trace Type	Technique Used	Daily mean seek distance			Daily mean seek time		
		min	avg	max	min	avg	max
sakarya1	None	264	**287**	303	6.78	**7.28**	7.76
	Perfect Idle Time Utilization	158	**166**	182	4.22	**4.35**	4.58
	Rearrangement Only	21	**25**	32	0.99	**1.18**	1.53
	Perfect Idle Time Utilization & Rearrangement	8	**11**	17	0.33	**0.48**	0.71
	Seek to Center & Rearrangement	20	**24**	33	1.04	**1.23**	1.58
sakarya2	None	121	**131**	141	4.26	**4.37**	4.52
	Perfect Idle Time Utilization	49	**51**	53	1.77	**1.85**	1.91
	Rearrangement Only	40	**56**	65	2.21	**2.72**	3.05
	Perfect Idle Time Utilization & Rearrangement	20	**32**	42	1.06	**1.43**	1.83
	Seek to Center & Rearrangement	50	**62**	71	2.51	**2.93**	3.28
ballast	None	114	**116**	118	9.49	**9.60**	9.71
	Perfect Idle Time Utilization	39	**41**	42	3.29	**3.37**	3.42
	Rearrangement Only	30	**31**	33	4.30	**4.47**	4.73
	Perfect Idle Time Utilization & Rearrangement	8	**10**	12	1.07	**1.19**	1.35
	Seek to Center & Rearrangement	34	**40**	50	5.22	**5.52**	6.04
snake	None	157	**165**	170	6.23	**6.41**	6.51
	Perfect Idle Time Utilization	95	**103**	109	3.63	**4.02**	4.28
	Rearrangement Only	28	**30**	34	2.29	**3.19**	3.51
	Perfect Idle Time Utilization & Rearrangement	17	**19**	21	1.56	**1.77**	1.97
	Seek to Center & Rearrangement	25	**28**	32	2.72	**3.19**	3.50

the idle period to limit the length of the move. We hoped that this heuristic would interact well with block rearrangement, since rearrangement clusters frequently requested blocks near the disk's center.

The results of experiments with the move-to-center heuristic are shown in the fifth row for each trace type in the tables. Contrary to our intuition, the anticipatory seeks actually increased the mean seek time for all of the traces. This was due to sequential dependencies in the request streams. When the disk arm moves away from the reserved region, it often services requests for more than one infrequently referenced block before moving back to the disk's center. Although the combination of block rearrangement and anticipatory movement appears to have potential, it is not clear how to realize it.

8. BLOCK REARRANGEMENT UNDER VMS

In addition to the UNIX reference traces, we had access to a set of traces collected from disks supporting a commercial database application running on DEC's VMS operating system. Unlike the UNIX file systems we studied, the VMS file system is extent based. We were interested in determining whether block rearrangement would also work well under that system.

Unfortunately, each trace in the VMS set covers only a one-hour period. Rather than trying to divide such a short time period into intervals for the purpose of estimating block reference frequencies, we performed experiments similar to those in Section 7.4. Specifically, we allowed the analyzer to scan each trace in advance to determine reference counts. The analyzer's output was then used to rearrange the disk blocks before the traced requests were simulated.

Clearly, this method will result in an optimistic assessment of the behavior of block arrangement in the VMS environment. The experimental results must be viewed with a degree of caution. In Section 7.4 we found that for our UNIX traces, the performance obtained by predicting hot spots from the previous day's trace was close to the performance obtained with an advance scan of the upcoming request stream. Although this result may not apply directly to the VMS traces, we hoped that these experiments would provide us with a rough assessment of block rearrangement's potential under VMS.

8.1 Disks and Traces

We used traces from two disks connected to the traced VAX system. Each disk is a DEC model RA81. Both disks were connected to the host through a HSC (Hierarchical Storage Controller). The connection between the controller and the disks was via an SDI (Standard Disk Interconnect) bus. The seek time function for the RA81 disk is given in the Appendix.

Table IX gives some information about the RA81 disk and the traces. Each trace record includes a beginning sector number on the disk and a request length expressed in sectors. Trace records also contain a read/write flag and a timestamp. In our traces, the vast majority of the requests are read operations. Request sizes vary from 1 sector to 120 sectors, but many requests are small. Around 40% of the requests are for 1 sector only. The workload produced disk utilization of 35% and 48% (without block rearrangement) for the two traces.

8.2 Simulator

We made two minor modifications to our simulator to accommodate the VMS traces. The first was made because of the extent-based disk space allocation technique used by the VMS file system. Extents are allocated in multiples of C disk sectors. We will refer to each group of C sectors as a *chunk*. The rearrangement system rearranges blocks of data. For our simulations, we set the rearrangement block size to match the chunk size. For our traces, the chunk size is three sectors.

Each request in the traces spans one or more physically contiguous chunks. Since chunks are the unit of rearrangement, it may sometimes happen that a request includes some chunks that have been rearranged and others that have not. In such cases, the simulator divides the request into several smaller subrequests such that each subrequest spans a physically contiguous set of chunks. It then services the subrequests in sequence. The seek and service

Table IX. Specifications of the RA81 Disk and Information about the VMS Traces

Disk	DEC RA81
Capacity	456 MB
Cylinders	1248
Heads	14
Track size	51 sectors
Sector size	512 bytes
Speed	3600 RPM
Interface	SDI

Trace	Average Request Size (sectors)	Number of Requests	Percentage of Reads
disk 1	5.9	41032	97
disk 2	5.4	62066	90

Table X. Results of Experiments with VMS Traces

	disk 1				disk 2			
	mean seek dist	mean seek time	mean service time	mean waiting time	mean seek dist	mean seek time	mean service time	mean waiting time
Original	344	19.39	31.28	23.01	223	16.38	28.13	31.67
Rearrangement with Organ-pipe Placement	10	5.13	22.99	19.70	18	6.88	23.52	27.97
Rearrangement with Serial Placement	15	5.95	17.52	3.88	21	6.47	18.02	6.80

times for the full request are taken to be the sums of the times for the subrequests.

8.3 Simulation Results

Table X summarizes the results of the experiments with VMS traces. In the table, mean values for several performance metrics are shown for experiments with and without block rearrangement. Five thousand blocks were rearranged, which amounts to less than 2% of the total disk space. We experimented with both serial and organ pipe placement. The reference analyzer used 7000 counters to analyze the reference stream.

Rearrangement results in substantial reductions in seek time and distance. In the best case, seek distances were reduced by 97% and seek times by 73%. Although these results are optimistic, they certainly suggest that block rearrangement is potentially useful under extent-based file systems such as VMS's.

Although the two block placement heuristics produced comparable seek time reductions, we found that service time reductions were greater when serial placement was used. This is because the serial policy is more likely to preserve file system's extent-based layout decisions within the reserved area. In contrast, the organ pipe heuristic may copy two contiguous hot chunks into

noncontiguous locations in the reserved region. As a result, a request for several contiguous chunks is more likely to have to be split into subrequests when organ pipe placement is used by the driver. Such requests introduce extra rotational delays (and seeks) which drive up the mean service time. These experiments suggest that serial placement is more appropriate for use under extent-based file systems.

9. SUMMARY AND CONCLUSIONS

We have presented an adaptive technique for rearranging disk blocks to reduce access times. Data requests are monitored to determine which blocks are frequently accessed. These blocks are then grouped together on a set of reserved cylinders in the center of the disk. Access frequencies are periodically reevaluated to reflect changes in data reference patterns over time.

The adaptive technique was evaluated using trace-driven simulations employing traces from several different systems. We found that the technique can cut seek times substantially by rearranging only a small fraction of the data on a disk. Adaptive rearrangement worked best under read-only file systems with a large number of users producing a stable workload. Such an environment is common on network file servers. Improvements were not as great under workloads that included substantial numbers of file updates, deletions, and creations. Our simulations also showed that the success of the block rearrangement technique is closely tied to the skew in the block access distributions. The more skewed the distributions, the greater the seek time reductions that can be achieved by rearranging blocks.

We experimented with two policies for arranging blocks in the reserved cylinders. We found that these policies had only a weak effect on performance, with the organ pipe heuristic performing slightly better on those traces for which the rearrangement technique achieved its best performance. However, additional experiments with a set of VMS traces suggested that the serial policy might be better suited for use under extent-based file systems.

Other techniques for seek time reduction can be combined with block rearrangement. We studied the interaction of block rearrangement with head scheduling and anticipatory disk head positioning. Our experiments found little interaction between head scheduling and block rearrangement. We also found that the combina tion of block rearrangement with an idealized form of anticipatory disk head mo vement performed better than either technique used alone. However, it may be difficult to develop a practical anticipatory movement technique that works as well as the idealized one.

A possible extension to this work is to explore more-sophisticated policies for determining which copy of a rearranged block to seek to when a choice is available. Since our rearrangement technique copies blocks from their original locations to the reserved cylinders, two copies of the data are available on the disk (unless the block is updated). When the block is requested, the driver (or controller) can cause the disk to seek to either copy. The techniques described in this article do not attempt to utilize the original copy of a block once it has been copied into the reserve region.

Table XI. Transfer Rates and Seek Time Functions Used in the Simulations

Disk Type	Fujitsu M2266AS
Disk Media Transfer Rate	2.5 MB/s
Channel Transfer Rate	4.2 MB/s

$$seektime(d) = \begin{cases} 0 & \text{if } d = 0 \\ 1.3040 + 0.584\sqrt{d} & \text{if } 0 < d \leq 10 \\ 1.205 + 0.65\sqrt{d} & \\ -0.734\sqrt[3]{d} + 0.659 \log d & \text{if } d \leq 225 \\ 7.44 + 0.0114d & \text{if } d > 225 \end{cases}$$

Disk Type	Fujitsu 2351 Eagle
Disk Media Transfer Rate	1.3 MB/s
Channel Transfer Rate	2 MB/s

$$seektime(d) = \begin{cases} 0 & \text{if } d = 0 \\ 4.6 + 0.87\sqrt{d} & \text{if } d \leq 239 \\ 18 + 0.028(d - 239) & \text{if } d > 239 \end{cases}$$

Disk Type	Quantum PD425S
Disk Media Transfer Rate	2.2 MB/s
Channel Transfer Rate	5 MB/s

$$seektime(d) = \begin{cases} 0 & \text{if } d = 0 \\ 3.521 + 0.486\sqrt{d} & \text{if } d \leq 488 \\ 8.884 + 0.011d & \text{if } d > 488 \end{cases}$$

Disk Type	DEC RA81
Disk Media Transfer Rate	1.5 MB/s
Channel Transfer Rate	3 MB/s

$$seektime(d) = \begin{cases} 0 & \text{if } d = 0 \\ 4.766 + 1.234\sqrt{d} & \text{if } d > 0 \end{cases}$$

APPENDIX. DISK DELAY PARAMETERS

The transfer rates and the seek time functions used in the simulations are listed in Table XI. The disk media transfer rate is calculated using the rotational speed of a particular disk and the number of sectors per track. For the Fujitsu M2266AS, the channel transfer rate was determined by measuring the aggregate transfer rate (media transfer plus channel transfer) and subtracting the calculated media rate. For the remaining disks, the maximum nominal transfer rate for the channel was used.

Seek time (in milliseconds) is a function of seek distance. The seek time function for the Fujitsu M2266AS was derived by the authors using measurements taken on the disk. The function for the Fujitsu 2351 Eagle is borrowed from Seltzer [1990]. The seek time function for the Quantum disk was provided by HP Labs together with the traces. Finally, the seek time function for the RA81 disk was derived from the manufacturer's specifications.

ACKNOWLEDGMENTS

Traces from snake were provided to us by Hewlett Packard Laboratories. We are grateful to John Wilkes for his efforts in making the traces available to us. We are also grateful to Scott Carson and the Digital Equipment Corporation for providing access to the VMS traces.

REFERENCES

AKYÜREK, S. AND SALEM, K. M. 1993. Adaptive block rearrangement under UNIX. In *Proceedings of the USENIX Summer 1993 Technical Conference* (Cincinnati, Ohio, June). USENIX Assoc., Berkeley, Calif., 307–321.

BITTON, D. 1987. Technology trends in mass-storage systems. In *Proceedings of the SIGMOD 1987 Annual Conference* (San Francisco, Calif.). ACM, New York, 7–8.

FLOYD, R. A. AND ELLIS, C. S. 1989. Directory reference patterns in hierarchical file systems. *IEEE Trans. Knowl. Data Eng. 1*, 2 (June), 238–247.

FORD, D. A. AND CHRISTODOULAKIS, S. 1991. Optimizing random retrievals from CLV format optical disks. In *Proceedings of the 17th International Conference on Very Large Data Bases* (Barcelona, Spain, Sept.). VLDB Endowment, Saratoga, Calif., 413–422.

GEIST, R. AND DANIEL, S. 1987. Continuum of disk scheduling algorithms. *ACM Trans. Comput. Syst. 5*, 1 (Feb.), 77–92.

GROSSMAN, D. D. AND SILVERMAN, H. F. 1973. Placement of records on a secondary storage device to minimize access time. *J. ACM 20*, 3 (July), 429–438.

HOFRI, M. 1980. Disk scheduling: FCFS vs. SSTF revisited. *Commun. ACM 23*, 11, 645–653.

KING, R. P. 1990. Disk arm movement in anticipation of future requests. *ACM Trans. Comput. Syst. 8*, 3 (Aug.), 214–229.

MCKUSICK, M. K., JOY, W. N., LEFFLER, S. J., AND FABRY, R. S. 1984. A fast file system for UNIX. *ACM Trans. Comput. Syst. 2*, 3 (Aug.), 181–197.

OUSTERHOUT, J. K., DA COSTA, H., HARRISON, D., KUNZE, J. A., KUPPER, M., AND THOMPSON, J. G. 1985. A trace driven analysis of the UNIX 4.2 BSD file system. In *Proceedings of the 10th ACM Symposium on Operating System Principles*. ACM, New York, 15–24.

RUEMMLER, C. AND WILKES, J. 1993. UNIX disk access patterns. In *Proceedings of the Winter 1993 USENIX Conference* (San Diego, Calif., Jan.). USENIX Assoc., Berkeley, Calif., 405–420.

RUEMMLER, C. AND WILKES, J. 1991. Disk Shuffling. HPL-91-156, Hewlett-Packard Laboratories, Palo Alto, Calif. Oct.

SALEM, K., BARBARÁ, D., AND LIPTON, R. J. 1992. Probabilistic diagnosis of hot spots. In *Proceedings of 8th International Conference on Data Engineering* (Tempe, Ariz., Feb.). 30–39.

SELTZER, M., CHEN, P., AND OUSTERHOUT, J. 1990. Disk scheduling revisited. In *Proceedings of the Winter 1990 USENIX Conference* (Washington, D.C.). USENIX Assoc., Berkeley, Calif.

SCHWETMAN, H. 1987. CSIM reference manual. Tech. Rep. ACA-ST-257-87. Microelectronics and Computer Technology Corporation, Austin, Tex.

STAELIN, C. AND GARCIA-MOLINA, H. 1991. Smart filesystems. In *Proceedings of the Winter 1991 USENIX Conference* (Dallas, Tex.). USENIX Assoc., Berkeley, Calif., 45–51.

STAELIN, C. AND GARCIA-MOLINA, H. 1990. Clustering active disk data to improve disk performance. Tech. Rep. CS-TR-283-90, Dept. of Computer Science, Princeton Univ., Princeton, N.J. Sept.

VONGSATHORN, P. AND CARSON, S. D. 1990. A system for adaptive disk rearrangement. *Softw. Prac. Exp. 20*, 3 (Mar.), 225–242.

WONG, C. K. 1980. Minimizing expected head movement in one-dimensional and two-dimensional mass storage systems. *ACM Comput. Surv. 12*, 2 (June), 167–178.

Received November 1993; revised July 1994; accepted November 1994