CS-TR-3054.1
UMIACS-TR-93-28.1
February, 1994

Adaptive Block Rearrangement Under UNIX*

Sedat Akyürek
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Kenneth Salem
Institute for Advanced Computer Studies
and
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

An adaptive UNIX disk device driver is described. To reduce seek times, the driver copies frequently-referenced blocks from their original locations to reserved space near the center of the disk. Block reference frequencies need not be known in advance. Instead, they are estimated by monitoring the stream of arriving requests. Measurements show that the adaptive driver reduces seek times and response times substantially.

1 Introduction

Disk performance can be a limiting factor in the performance of a computer system. The problem of slow secondary storage devices has come to be known as the I/O bottleneck. One of the major components of disk access times is seek time, the time required to move the drive's read/write heads to the required data. By reducing seek time, the performance of a disk can be improved.

In a recent paper [Akyurek 93] we introduced an adaptive block rearrangement technique which reduces disk seek times. Under this technique, a small number of frequently referenced data blocks are copied 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. The set of blocks in the reserved space is changed periodically to adapt to changing user access patterns.

Trace-driven simulations have shown that this technique can be very effective in reducing seek times. This paper describes an implementation of the technique using a modified UNIX device driver. It also presents the results of detailed measurements taken during the operation of the system. Our results demonstrate that adaptive block rearrangement can easily and effectively be employed under existing file systems.

This work was supported by National Science Foundation Grant No: CCR-8908898 and in part by CESDIS.
Submitted to *Software Practice & Experience.*
A preliminary version of this report appeared in the Proceedings of USENIX Summer Technical Conference, June, 1993, pages 307-321.
*UNIX is a trademark of AT&T

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 [Wong 80, Grossman 73]. 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 91].

In practice, data references are not drawn from a fixed distribution, nor are they independent. Although references are highly skewed [Floyd 89, Staelin 91, Vongsath 90, Ouster 85], request distributions change over time, and they are generally not known in advance. Nevertheless, variations of the organ pipe heuristic seem to work well in practice. Recently, several papers have proposed adaptive applications of data clustering based on this idea.

Vongsathorn and Carson [Vongsath 90] showed that dynamically clustering frequently accessed data worked better than static placement. In that study, disk cylinders are dynamically rearranged using the organ pipe heuristic, according to observed data access frequencies. Recent work in the DataMesh project [Ruemmler 91] considered rearrangement of cylinders and blocks, with mixed results. Their conclusion that block shuffling generally outperforms cylinder shuffling corroborates one of our own. A similar approach is employed in the experimental iPcress file system[Staelin 91], which monitors access to files and moves files with high "temperatures" (frequency of access divided by file size) to the center of the disk.

Our technique differs from each of the techniques mentioned above in at least one of the following respects.

Granularity Our technique moves blocks rather than cylinders or files. Blocks within a file or within a cylinder can vary in temperature. Also, block rearrangement can increase the number of zero-length seeks, while cylinder reorganization cannot. Smaller granularity also facilitates incremental rearrangement.

Data Volume Only a small fraction of blocks are rearranged at any time as opposed to reorganizing all the data on the disk. This makes rearrangement faster.

Layout Preservation Block relocation is temporary, and cooled blocks are returned to their original locations. The layout of cool blocks is not disturbed at all.

Transparency Our technique can be implemented in a device driver (or controller). No changes to the file system are required.

Other systems cluster data based on criteria other than reference frequency. The Berkeley Fast File System (FFS) [McKusick 84] used in many UNIX systems uses placement heuristics that try to cluster the blocks of a file. However, hot blocks from different files may be spread widely over the disk's surface. This can result in long random seek operations when requests for the blocks of different files are interleaved, as is the case in multi-user systems.

LFS [Rosenblum 91] and Loge [English 92] rearrange data based on the order of writes in the request stream. The primary goal of these systems is to improve write performance, not read performance. In contrast to both Loge and LFS, our adaptive block rearrangement technique makes both read and write operations faster. These systems try to improve write performance by reducing both seek and rotational delays. LFS eliminates seek and rotational delays by combining many write operations into a single large write operation. The Loge self-organizing disk controller transparently reorganizes blocks each time they are written to reduce seek and rotational delay. Simulation studies of the controller show that it can reduce write service times, but the savings come at the expense of increased read service times. Unlike Loge, the block rearrangement system described here preserves the data placement done by the file system. Finally,

although the adaptive block rearrangement system described here is implemented in the device driver on the host, it can be implemented in an intelligent I/O or disk controller, like Loge.

In the next section we will give an overview of the block rearrangement technique. In Section 3 we will give a brief introduction to UNIX file system and UNIX device drivers. Section 4 describes the implementation of the adaptive block rearrangement in a UNIX system. In Section 5 we present the results of experiments conducted while the system was operational on a network file server.

2 Overview

Although the volume of data stored on disks is increasing, experience shows that only a small fraction of this data on the disks is actively and frequently used[Floyd 89, Staelin 90, Staelin 91]. If the blocks storing hot (frequently accessed) data are spread over the surface of the disk, distant from each other, long seek delays may result.

Adaptive block rearrangement takes advantage of highly skewed data access patterns to reduce seek delays. It clusters hot blocks on a reserved set of contiguous disk cylinders to reduce seek times. Clustering is achieved by copying hot blocks onto the reserved cylinders. The block layout outside of the reserved cylinders is left undisturbed. Incoming requests are redirected to the reserved cylinders if the requested data resides there. If the reserved cylinders accommodate the heavily referenced data, we expect that the disk head will tend to linger over those cylinders and seek distances will be reduced.

Normally, it is difficult to know in advance (e.g., when a file system is created), which data will be frequently accessed. Furthermore, the set of hot data blocks may change over time. For these reasons, our block rearrangement system is designed to monitor and adapt to changing data access patterns. The system monitors the stream of requests directed to the disk and periodically produces a list of hot (frequently-referenced) blocks, ordered by frequency of reference. The hot block list is used to determine which blocks should be placed in the reserved cylinders. The rearrangement system copies the selected hot blocks from their original locations to the reserved cylinders. These blocks remain in the reserved space until the next hot block list is produced. Blocks which are not hot anymore are copied back to their original locations.

The block rearrangement system assigns hot blocks to the reserved cylinders according to their rank in the list, i.e., their frequency of reference, using the organ-pipe heuristic. Assuming that C blocks fit on a cylinder, the C hottest blocks are placed on the middle cylinder of the reserved cylinder group. The next hottest are placed on an adjacent cylinder, and so on so that the final cylinder reference distribution across the reserved cylinders forms an organ pipe distribution. The reserved cylinders themselves are located in the middle of the disk.

Adaptive block rearrangement can be implemented in an intelligent disk controller or in a device-driver. No changes are needed in the file system implementation. In the rest of the paper we will describe the implementation of an adaptive block rearrangement system under the UNIX operating system, and present some performance results. The block rearrangement system was implemented in a disk driver. Before discussing the implementation, we will give a brief overview of some of the important features of the file system and the disk driver in the next section.

3 UNIX File System and Device Driver Overview

The adaptive block rearrangement system was implemented in a UNIX disk driver. The driver is used by the file system to read and write blocks of data. Some of the features of the file system have an impact on the performance of the rearrangement system. In the following sections, we give brief overviews of the file system and of the disk driver that we modified.

3.1 The File System

Our experiments were carried out using the UFS file system of SunOS 4.1.1.[1] This file system is closely related to the Berkeley UNIX Fast File System.

The operating system stores data in files, which are organized using a hierarchy of directories. A file system includes ordinary data files and meta-data used by the operating system to manage data storage and retrieval. Disks are divided into one or more logical devices, which are called partitions. Each file system is assigned to a disk partition and each partition can store only one file system.

In UNIX, a user views a file as an array of bytes. The operating system is responsible for mapping user files onto logical devices. Each disk partition can be thought of as an array of logical blocks. The operating system requests disk I/O operations in terms of these blocks. The block size is fixed when a file system is initialized in the partition.

Internally, files are are mapped to logical blocks using an index structure called an i-node. An i-node contains a set of pointers which point, either directly or indirectly, to the logical blocks that contain the file's data. The i-nodes themselves are also stored on the logical device, together with the file data blocks.

The operating system also manages the caching of file blocks in main memory buffers. All file I/O goes through the buffer cache. When a block is required, the file system first checks for it in the buffer cache. A read request is forwarded to the disk only in case the block is not found in the cache. Data in the buffer cache may be updated. As a performance optimization, the system does not immediately write modified blocks back to the disk. Instead, the updated blocks simply remain in the buffer cache. Periodically, all dirty blocks are copied back to the disk.

When a file system is mounted (made accessible to users), it may be mounted in either read-only mode, or in read/write mode. Users may not create, delete or modify files in a file system that is mounted read-only. However, the operating system itself may generate write requests to the logical device that holds a read-only file system. Such requests normally represent updates to bookkeeping information (e.g., time stamps) in the i-nodes.

3.2 The Disk Driver

Device drivers are software modules which provide clean interfaces to peripheral devices. They hide the device-specific details and provide an abstract and uniform view of the devices for the operating system and the user programs.

A disk driver fills the gap between the file system and the actual disk by mapping logical disk devices to physical disk. The file system makes requests to the driver in terms of logical device blocks. The driver converts the logical block numbers into physical block numbers and carries out the desired operations.

UNIX provides two types of I/O device interfaces: block-oriented and character-oriented. The file system uses the driver's block-oriented interface. The driver also provides a character-oriented interface, called the *raw* interface. It is actually implemented using the block-oriented interface. The raw interface can be used to access the device directly, bypassing the file system and the buffer cache.

To conform with the block-oriented interface, a disk driver implements a set of routines to carry out its tasks. These routines are responsible for autoconfiguration and initialization of the device, for servicing I/O requests, and for handling interrupts from the device. There are also routines for performing special operations specific to each device.

The `attach` routine of the driver is called at start-up time, or when the device is first recognized by the system, to initialize the device and its state within the operating system kernel.

The `strategy` routine is responsible for servicing the I/O requests to or from the device. The file system or the raw I/O interface routines call the `strategy` routine, supplying the logical device number and

[1] SunOS is a trademark of Sun Microsystems, Inc..

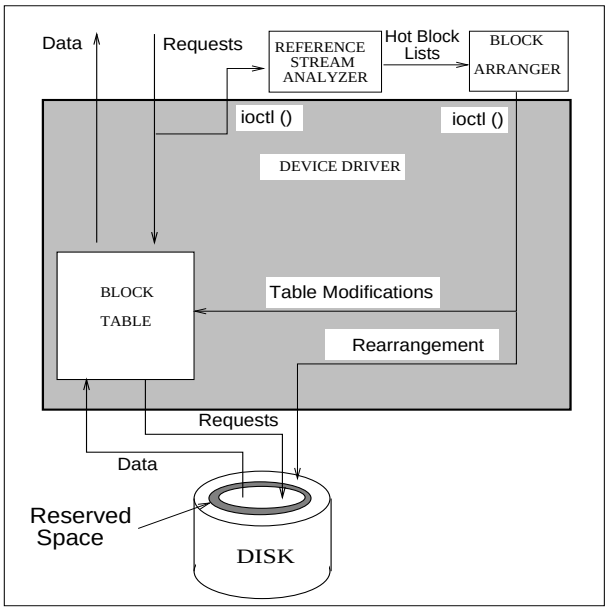

Figure 1: Adaptive Device Driver

logical block numbers on which the I/O operation is to be performed. `Strategy` translates the logical block numbers to physical block numbers and enqueues the operation. It maintains a queue of outstanding requests for each physical device, managed using a disk queueing policy. If the new operation is the only operation in the queue, `strategy` calls a routine called `start` to initiate the operation, and returns. Otherwise, it simply returns. The queued operations are initiated by the interrupt routines after each active operation is completed.

The driver also provides routines for special-purpose operations specific to its device. For example, disk drivers provide routines for reading or reseting disk geometry entries. These routines can be invoked from user processes through the `ioctl` system call.

The raw I/O interface works through the `physio` routine, which calls the `strategy` routine one or more times to satisfy a raw request.

4 Implementation

In this section we present the implementation of the block rearrangement system. A diagram of the system is shown in Figure 1. The block rearrangement system is implemented through modifications to the SCSI disk driver of SunOS 4.1.1. In addition, several user level programs are used to control the modified driver.

4.1 Modifications in the Device Driver

The device driver modifications implement the reserved space on the disk and the mapping of hot blocks to their new positions in the reserved space. They also provide entry points to the kernel for controlling the movement of blocks to and from the reserved area and for monitoring block accesses. Another function of the modified driver is to gather performance statistics.

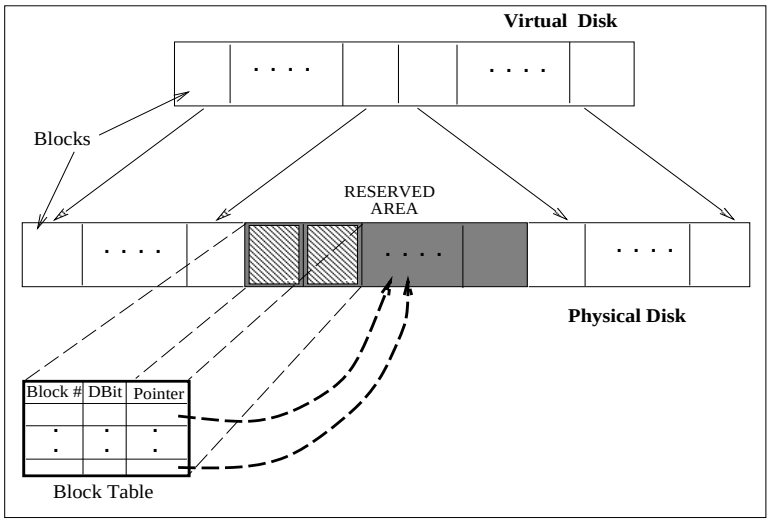

Figure 2: Block Mapping on a Rearranged Disk

4.1.1 Reserved Space

UNIX disk labels contain information about the size of the disk and the sizes and positions of disk partitions (logical disk devices). This information is used by the UNIX `newfs` utility to initialize a file system on a given partition on the disk.

To make space for the rearranged blocks, the target disk is made to look smaller than it really is by changing the disk geometry information on the disk label. Disk partitioning information is changed accordingly so that the file system thinks that the disk has fewer cylinders. The hidden cylinders implement the reserved space. The modified driver implements the mapping between the *virtual* (smaller) disk and the actual disk (Fig. 2)[2].

Block rearrangement is implemented on a physical device basis. A disk may have several partitions and consequently several file systems on it. However, only a single reserved region will be implemented by the driver, and blocks from any of the file systems may be copied there. The technique requires that all of the file systems on the disk have the same block size.

When a target disk is initialized for rearrangement, the number of the first sector and the length of the reserved space are recorded in its label. During initialization a special value is also recorded in the label to mark it as a "rearranged" disk. At the time of system start-up, the `attach` routine of the driver checks if the disk is a rearranged disk. If it is, then the information about the reserved area is read in to be used for block mapping.

4.1.2 Block Mapping

The `strategy` routine of a disk driver is responsible for converting file system's block addresses to physical block addresses. In the modified driver, it also implements the mapping from the *virtual* disk to the actual disk and reroutes requests for rearranged blocks when necessary.

When a request arrives, the `strategy` routine first converts the logical block address to a physical block address. It then determines whether that block has been repositioned into the reserved area. To implement this check the driver maintains a data structure called the *block table*. When a block is copied into the reserved space, its old and new physical block addresses are entered into the table. If an entry for

[2] A SCSI disk presents itself as a sequence of logical sectors. The actual physical locations of the sectors are not known. We rely on an implicit assumption that most SCSI sector numbers are close to their true physical numbers.

the requested block is found in the block table, its new physical address is used to retrieve (or update) the data.

A copy of the block table is also stored on the disk (at the beginning of the reserved area) to be used at start-up and for recovery purposes. The `attach` routine of the driver reads in the block table at start-up. The copy of the block table on the disk always correctly reflects the rearranged blocks and their positions in the reserved area (unless media failure occurs). However, the table also contains a dirty bit for each block entry, and these bits may not always be up-to-date in the disk-resident copy. For this reason, all blocks are marked as dirty when memory-resident copy of the table is recreated after a failure. This conservative strategy ensures that updates to repositioned blocks will not be lost because of a system crash.

The size of a "block" in the rearrangement system is the size of a file system block. The file system requests at most one block of data in any single request to the driver. Thus, it is not possible that part, but not all, of the requested data will have been rearranged. Normally, most (if not all) of the driver's workload is generated by the file system. However, requests through the raw interface are also possible. Through the raw interface, it is possible that requests larger than the block size will be forwarded to the driver. This raises the possibility that part of the requested data may have been rearranged and part may not. To accommodate such requests, the driver's `physio` routine was modified to break large requests into block-sized subrequests.

4.1.3 Block Movement

The driver provides kernel entry points for controlling the movement of blocks into and out of the reserved area. These entry points can be used by user-level processes via the `ioctl` system call. The driver implements two `ioctl` calls for block movement :

`DKIOCBCOPY` instructs the driver to copy a specified block to a specified location in the reserved area. The driver also places an entry for the block in the block table and forces the block table to be written to disk.

`DKIOCCLEAN` instructs the driver to clean out the reserved area by removing the blocks listed in the block table. If a block's dirty bit is set in the block table, the block is first copied to its original disk position. After each block is moved out, the block table is updated and the updated version is written to the disk.

Copying a block into the reserved area requires three I/O operations. Moving a block out of the reserved area requires at least one I/O operation and two extra operations if the block is dirty. Although this may appear to represent a significant overhead, these `ioctl` calls are normally issued very infrequently, perhaps once per day. Furthermore, other requests can interleave with these operations. Requests for a block that is being moved are delayed temporarily by the driver.

4.1.4 Request Monitoring

For the purpose of monitoring the device request stream, the driver records information about each I/O request in a small internal table. The information recorded includes the block number and the request size. An `ioctl` call enables user processes to read the contents of the table and to clear it. In the event that the table fills completely before being cleared, request recording is temporarily suspended.

A user process periodically reads the contents of the request table and uses the them to analyze the block accesses. The frequency of request table reads can be changed. We have used a period of two minutes in our experiments. This was short enough to ensure that request recording was almost never suspended.

4.1.5 Performance Monitoring

Another function of the driver is to monitor its own performance. This functionality was provided for the purpose of evaluation only and is not a part of the block rearrangement system. Performance monitoring is implemented much like request monitoring. Statistics are recorded in a table inside the driver. User-level processes can read the contents of this table through an `ioctl` call which also clears the table.

All statistics are recorded separately for read operations and write operations. The driver records seek distance distributions (in arrival order and in scheduled order) and service time and queueing time distributions. Times are measured with microsecond resolution. However, time distributions are recorded with a resolution of one millisecond. Cumulative service times and queueing times are recorded as well, using the full resolution of the measurements. The queueing time for a request is the period between the time the driver first receives the request and the time the request is submitted to the disk. The service time for a request is from the end of the queueing time to the time the request is returned from the disk.

4.2 User Level Programs

User level programs use the entry points provided by the modified kernel to monitor the block request stream. Based on the request stream, they decide which blocks should be rearranged and where they should be placed in the reserved area. One user-level process, the *reference stream analyzer*, monitors the block requests and tries to determine which blocks are most frequently requested. Another process, which is called the *block arranger*, selects the most frequently requested blocks for rearrangement and controls their placement in the reserved area.

The functions implemented by the user-level processes can easily be incorporated in to the device driver itself. We implemented them at the user level to facilitate experimentation with different analysis and rearrangement techniques.

The reference stream analyzer maintains a list of *block number/reference count* pairs. In the worst case, the length of the reference stream analyzer's list will be proportional to the number of blocks on the disk. If the reference stream analysis were to be performed in the device driver itself, this list could take up a substantial amount of kernel memory. However, the analyzer can *guess* at the hottest blocks using a much smaller amount of memory. This can be accomplished by limiting the size of the list. In case a block that does not appear on the list is referenced, a replacement heuristic is used to make room for it on the list. In our experiments, the analyzer maintained a list of several thousand reference counts, enough so that replacement was rarely necessary. However, experiments reported in [Salem 92, Akyurek 93, Salem 93] indicate that if space is limited, this technique can still generate very accurate guesses using much shorter lists.

Given a list of block reference frequencies, the block arranger selects the hottest blocks from the list and determines where to place them in the reserved region. For purposes of comparison, the block arranger implements three placement policies:

Organ-pipe placement This policy arranges the blocks in an organ-pipe pattern according to their access frequencies. The blocks with highest access frequencies in the center cylinder of the reserved area. One of the adjacent cylinders is then filled with the next-hottest blocks, followed by the other adjacent cylinder. This process continues on alternating sides of the center cylinder until all of the selected blocks have been placed.

Interleaved placement The SunOS UNIX file system (UFS), which is based on FFS, tries to place successive blocks of a file interleaved by gaps. The size of the gaps is called the interleaving factor. This policy is intended to reduce rotational delays when a file is accessed sequentially. Our interleaved placement policy attempts to preserve this interleaving within the reserved region.

The driver has no knowledge of files. However, it can attempt to guess whether one block is the successor of another in a file. Suppose that X and Y are blocks, and that Y's location is greater than

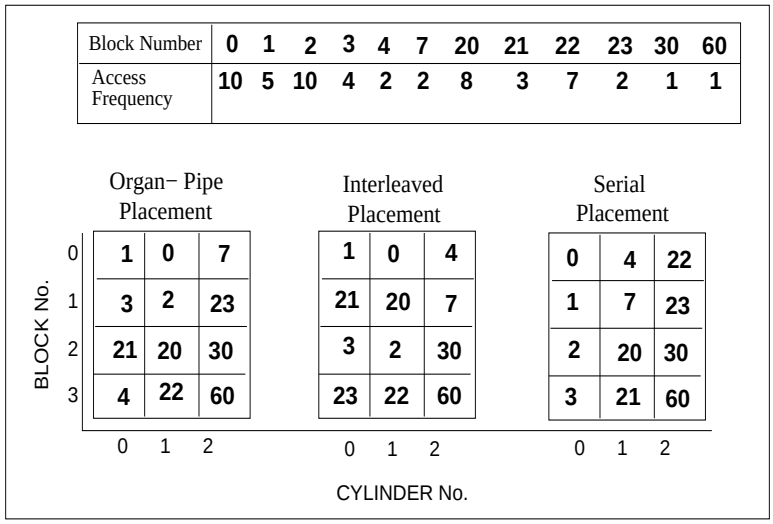

Figure 3: Placement Policies. The block placement of different policies are shown. The set of blocks to be rearranged and their estimated access frequencies are also given. The reserved area has three cylinders with four blocks in each cylinder. The interleaving factor (rotational delay) of the file system is assumed to be one block. The interleaved placement policy considers the access frequency of a block "close" to another block's, if the first block's frequency is at least half of the second block's access frequency.

X's by the interleaving factor. We define Y to be the successor of X if Y's reference frequency is close to X's. We define Y's frequency to be close if it is at least 50% of X's. The 50% figure was chosen arbitrarily.

The interleaved policy works as follows. The block arranger starts by choosing the hottest block and placing it in the center cylinder. It then determines whether the hottest block has a successor in the hot block list. If so, that block is placed in the center cylinder, separated from the first block by the interleaving factor. The block arranger continues by looking for and attempting to place the successor's successor. A chain of successors is followed either until a successor cannot be placed or until a block is found to have no successor. At that point, the block arranger selects the hottest remaining block and attempts to begin a new chain. Cylinders are filled in the same order used by the organ-pipe policy.

Serial placement This is the simplest placement policy. Blocks are placed in the reserved space in ascending order of their original block numbers. Block access frequencies are used to determine which blocks will be rearranged, but they are not used to control the placement of those blocks within the reserved region.

An example illustrating the three policies is given in Figure 3.

5 Performance

The adaptive block rearrangement system is operational on a Sun Sparcstation[3] 2, called Sakarya. The operating system on the machine is SunOS 4.1.1. The file system block size is 8 kilobytes and the fragment size is 1 kilobyte.

Sakarya has a main memory of 32 megabytes. The amount of memory used for buffer cache is not fixed. Under SunOS, the amount of physical memory used for the buffer cache is determined dynamically. Potentially, all of the available memory can be used for the buffer cache [McVoy 91]. Sakarya is usually

[3]Sun and Sparc are trademarks of Sun Microsystems, Inc.

Toshiba MK156F SCSI DISK	
Capacity	135 MB
Cylinders	815
Tracks/Cyln	10
Sectors/Track	34
Speed	3600 RPM

$$seektime(d) = \begin{cases} 0 & \text{if } d = 0 \\ 6.248 + 1.393\sqrt{d} - 0.99\sqrt[3]{d} + 0.813 \ln d & \text{if } d < 315 \\ 17.503 + 0.03d & \text{if } d \geq 315 \end{cases}$$

Fujitsu M2266SA SCSI DISK	
Capacity	1 GB
Cylinders	1658
Tracks/Cyln	15
Sectors/Track	85
Speed	3600 RPM
Track Buffer	256 KB

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

Table 1: Specifications of the disks. Seek time is given in milliseconds as a function of seek distances (in cylinders).

very lightly loaded. As a result, a large portion of the memory is available for the a buffer cache most of the time.

Experiments were performed using two different types of disks. First we conducted experiments using a Toshiba Model MK156F disk. Later we repeated the experiments on a larger disk. Both disks provided SCSI (Small Computer System Interface) interfaces. Specifications and seek time functions for the disks are given in Table 1. The seek time function for the Toshiba disk is borrowed from [Jobalia], in which the disk delay parameters for this disk were measured and a precise seek time function was devised. We derived the seek time function for the Fujitsu disk ourselves, using a methodology similar to that used in [Jobalia].

The Fujitsu disk has a track buffer and supports 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.

Of the Toshiba disk's 815 cylinders, 48 cylinders in the middle of the disk have been used as reserved cylinders. This amounts to approximately 8 megabytes (6% of the total disk capacity). Approximately 1000 8Kbyte blocks can fit in this space. On the Fujitsu disk, 80 cylinders were reserved for rearrangement. This amounts to approximately 50 megabytes of space, about 5% of the disk's total capacity. The size of the reserved region puts an upper bound on the amount of data that can be rearranged. We chose the size in light of the results of our previous simulation studies [Akyurek 93]. These suggested that most of the benefits of block rearrangement were realized by rearranging 1%-2% of the total number of blocks.

Sakarya was used as a file server for two different file systems. The first consisted primarily of executable files and libraries. We will refer to it as the *system* file system. Using NFS, it was mounted read-only on 14 Sparcstations. The user population of these workstations consisted of about 40 faculty and graduate students in the Computer Science Department of the University of Maryland.

The second file system was mounted for both reading and writing on the client workstations. It held the user home directories, and we will refer to it as the *users* file system. In the case of the Toshiba disk, we were able to accommodate ten home directories. On the larger (Fujitsu) disk, we accommodated twenty.

Disk	On/Off	Daily Mean Seek Time			Daily Mean Service Time			Daily Mean Waiting Time		
		min	avg	max	min	avg	max	min	avg	max
Toshiba	Off	18.70	19.46	21.51	38.41	39.78	41.71	65.39	82.73	94.52
Toshiba	On	0.98	1.17	1.55	22.61	22.88	23.34	40.39	46.43	51.13
Fujitsu	Off	7.80	8.14	8.67	21.26	21.60	22.04	61.35	66.57	72.69
Fujitsu	On	0.70	0.91	1.16	13.83	14.18	14.41	35.65	45.31	52.52

Table 2: Summary of Results of On/Off Experiments (*system* file system). The values reported are the minimum and maximum daily mean times (in millisecons) observed over all "on" days or all "off" days of the experiment.

5.1 Experiments

We will present the results of several sets of experiments. In the first set we study the effectiveness of block rearrangement by switching it on and off over a period of several days. The remaining experiments investigate the effects of several factors on performance. These factors we considered are the number of blocks rearranged and the block placement policy in the reserved region.

In each of the experiments, block reference counts measured during one day were used (at the end of the day) to rearrange blocks for the next day's requests. Reference counts were measured between the hours of 7am and 10pm each day.

5.2 On/Off Experiments: the *system* file system

We applied block rearrangement on alternate days to compare the driver's performance with and without rearrangement. This experiment was run for a ten days (5 on, 5 off) for each of the two disks. On days when rearrangement was applied, 1018 blocks were placed in the reserved area on the Toshiba disk and 3500 blocks were rearranged on the Fuji disk. The organ-pipe placement heuristic was used for these experiments. Table 2 summarizes the results. All table entries are measured values except for seek times. These were computed using the measured seek distance distribution and the seek time functions shown in Table 1.

Seek times were reduced by approximately 90% on both disks when rearragement was used. This resulted in service time reductions of over 40% for the Toshiba disk, and about 35% for the Fujitsu. In Figure 4 service time distributions for an "on" day and an "off" day on the Fuji disk are plotted. These distributions suggest that rearrangement reduces the service time for many requests. For example, on the day without rearrangement only 50% of all the requests are completed in less than 20 milliseconds. On the day with rearrangement, 85% of the requests completed in that time.

Because of the lower service times, request queue waiting times were also reduced. Queueing time reductions were often much greater than the service time reductions. Although the disks were lightly utilized, the request arrival pattern was very bursty. Arrival bursts produce long queues. Small service time reductions for requests near the front of the queue accumulate to produce large waiting time reductions for requests nearer the queue's end.

Table 3 presents some of our measurements in more detail. The table shows results from two consecutive days of experiment with each disk. On Day 1, no blocks were rearranged and the reserved region was left unused. Block rearrangement was applied on Day 2. In the table two rows are highlighted. The first gives the seek times we would have observed had requests been served in first-come-first-served (FCFS) order, with no block rearrangement. The second row gives the actual seek times. On days without rearrangement, these two values differ because of request reordering performed by the driver, which implements a SCAN policy. On days with rearrangement, the differences are due to a combination of request reordering and block rearrangement.

Large reductions in seek distance do not necessarily translate into large reductions in seek time.

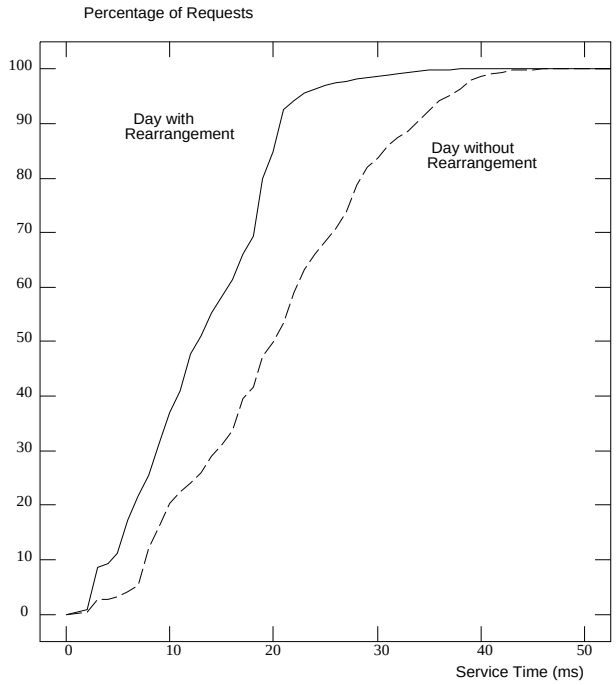

Figure 4: Service time distribution for *system* filesystem on Fuji disk.

Disk	Toshiba		Fujitsu	
	Day 1	Day 2	Day 1	Day 2
Rearrangement	Off	On	Off	On
FCFS Mean Seek Dist (cyln)	220	225	435	413
Mean Seek Distance (cyln)	173	8	315	27
Zero-length Seeks (%)	23	88	27	76
FCFS Mean Seek Time (ms)	20.92	21.46	10.31	9.73
Mean Seek Time (ms)	18.21	1.55	8.01	1.16
Mean Service Time (ms)	38.41	22.95	21.15	14.08
Mean Waiting Time (ms)	87.30	50.03	69.98	35.65

Table 3: Experimental results for *system* file system.

Disk	On/Off	Daily Mean Seek Time			Daily Mean Service Time			Daily Mean Waiting Time		
		min	avg	max	min	avg	max	min	avg	max
Toshiba	Off	12.46	14.31	16.60	30.50	32.80	35.32	4.48	5.80	6.86
Toshiba	On	3.54	3.89	4.49	22.57	23.59	24.03	4.46	4.97	5.47
Fujitsu	Off	7.52	7.79	8.02	19.69	20.29	21.48	3.21	4.72	7.59
Fujitsu	On	1.32	1.58	1.89	12.34	12.87	13.41	2.54	2.98	3.32

Table 4: Summary of Results of On/Off Experiments (read requests only). The values reported are the minimum and maximum daily mean times (in millisecons) observed over all "on" days or all "off" days of the experiment.

This is the nature of the relationship between seek distance and seek time. One of the reasons that block rearrangement is successful in reducing seek times is that it increases the number of very short seeks. In particular, the percentage of zero-length seeks is increased dramatically, as shown in Table 3. Zero-length seeks are more frequent because block rearrangement concentrates hot blocks onto a small number of cylinders. This produces cylinders in the reserved region that are much hotter than any cylinder produced by the file system's layout policy on the remainder of the disk.

As was discussed in Section 3, disk reads and writes are handled differently by the file system. We were interested in determining whether block rearrangement was having the same impact on both read and write requests. Table 4 shows results from the same experimental days as Table 2. However, in Table 4 only read requests are considered.

Block rearrangement results in substantial reductions in seek times for the read requests. However, the reductions were not as great as those we observed for the entire workload. Seek time reductions were approximately 75%, with service time reductions of about 30%. Waiting times for read requests were low even without block rearrangement. This left much less room for reductions from block rearrangement, although useful reductions were measured on the Fujitsu disk. Read waiting times tended to be shorter because the read arrival pattern is less bursty than the write pattern. This is because of the file system's periodic update policy.

We attribute the difference in performance between read and write requests to two factors. First, the read request distribution was normally less skewed than the write request distribution. Write requests were concentrated on a very small set of blocks. As a result, they benefit more from block rearrangement than do the reads. Figure 5 shows typical daily block request distributions for reads and for all requests, seperately for each of the disks.

Second, we believe that there is some synergy among block rearrangement, the driver's SCAN head scheduling, and the bursty arrival pattern of the write requests. Block rearrangement makes it likely that written blocks will reside on the same cylinder, and head scheduling ensures that a set of requests for blocks on the same cylinder will be serviced with no intervening requests to other cylinders. Together, these factors account for the very high percentage of zero-length seeks we observed in the read/write workload (Table 3). Although many read requests also resulted in zero-length seeks, write requests were much more likely to do so.

5.3 On/Off Experiments: the *users* file system

We applied block rearrangement on alternate days over a twelve day period for the Toshiba disk, and a ten day period for the Fujitsu disks. As in our experiments with the *system* file system, 1018 blocks were rearranged on the Toshiba disk and 3500 blocks were rearranged on the Fujitsu disk during the "on" days, and organ-pipe placement was used in the reserved region. Table 5 summarizes the results of these experiments.

Seek time reductions were not as great as those we observed for the *system* file system. On both

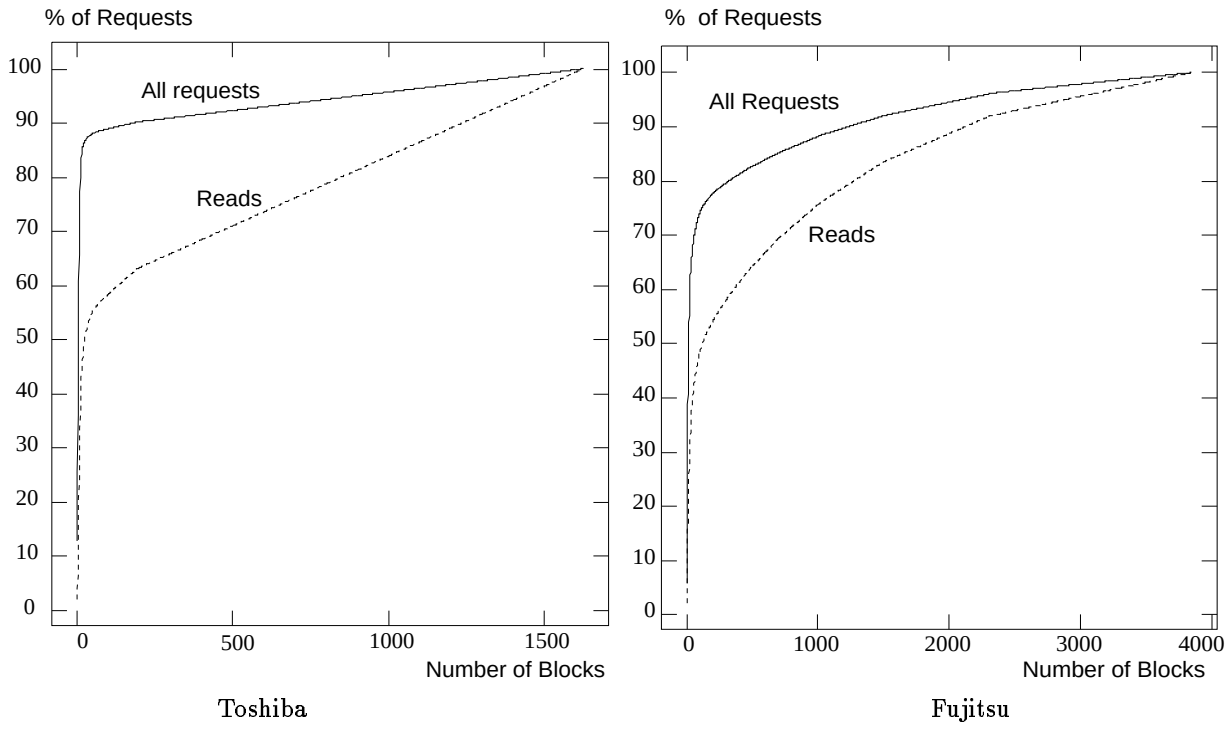

Figure 5: Distribution of block accesses for *system* file system on Toshiba and Fujitsu disks.

Disk	On/Off	Daily Mean Seek Time			Daily Mean Service Time			Daily Mean Waiting Time		
		min	avg	max	min	avg	max	min	avg	max
Toshiba	Off	11.06	13.10	15.45	28.83	31.14	34.06	8.32	16.86	31.93
Toshiba	On	8.10	8.90	10.78	26.08	27.32	29.54	4.74	10.18	18.63
Fujitsu	Off	3.27	4.27	4.79	16.23	17.00	17.37	4.33	15.19	48.96
Fujitsu	On	1.76	2.73	3.92	14.04	15.12	16.13	3.53	5.83	8.75

Table 5: Summary of Results of On/Off Experiments (*users* file system). The values reported are the minimum and maximum daily mean times (in millisecons) observed over all "on" days or all "off" days of the experiment.

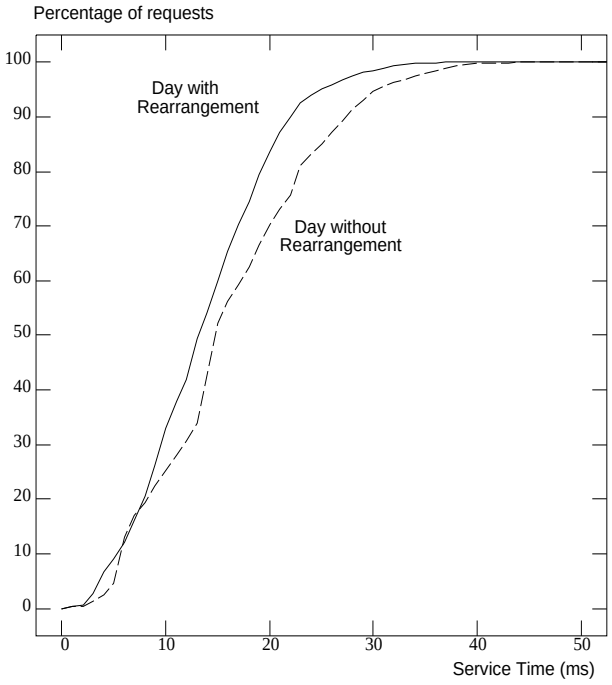

Figure 6: Service time distributions for *users* filesystem on Fuji disk.

disks, daily mean seek times were about 30%-35% lower on the "on" days than on the "off". As a result the service time reductions were also lower. In Figure 6, service time distributions for an "on" and an "off" day on the Fuji disk are plotted. Rearrangement is still beneficial to many requests but not as much as in the case of the *system* file system.

One reason that seek time reductions were not as great for the *users* file system is that the block request distributions tended to be less skewed. Figure 7 shows typical daily request distributions for reads and for all requests. The *users* workload includes write requests resulting from new file creation and file expansion operations. It is very unlikely that seek times for such requests will be reduced by block rearrangement.

Another factor that may have adversely affected performance is a greater daily workload variation for the *users* file system. The accuracy of the block rearrangement system's predictions depends on day-to-day access patterns that change only slowly. In the *users* filesystem experiments, we had a smaller population of users and little sharing of data between users. Thus, changes in the day-to-day behavior of a small number of users are likely to have a greater impact on performance.

Because write requests were less predictable, we found that block rearrangement worked better for read requests than for writes under the *users* file system. The opposite was true for the *system* file system. Table 6 summarizes our results for the read requests only. It should be compared to Table 5.

5.4 Varying the Number of Rearranged Blocks

In our next set of experiments we varied the number of rearranged blocks in the reserved area. Block rearrangement was applied for several weeks (on weekdays only), with a different number of blocks being rearranged each day. The experiments were performed using the Toshiba disk and the *system* file system.

The results of these experiments are summarized in Figure 8. The graph shows the percentage reduction in daily mean seek distance and time as a function of the number of blocks rearranged. The percentage reduction is computed relative to the seek distance that would have been observed had the

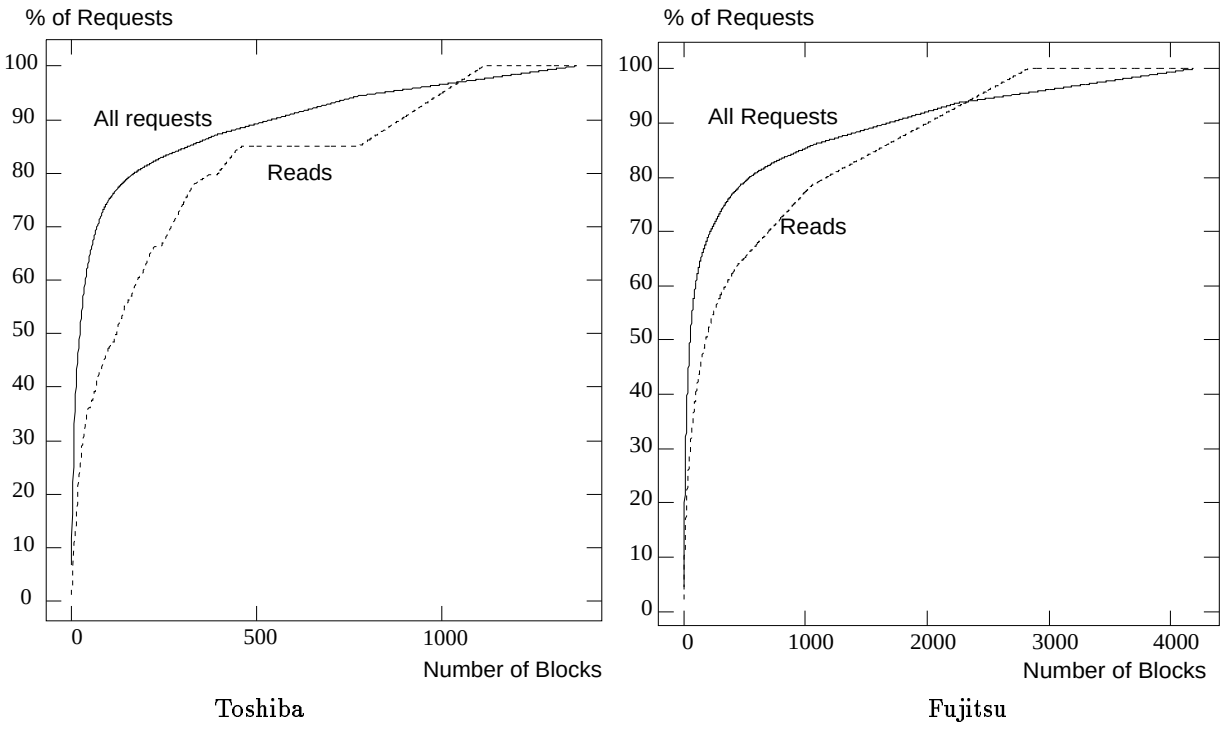

Figure 7: Distribution of block accesses for *users* file system on Toshiba and Fujitsu disks.

Disk	On/Off	Daily Mean Seek Time			Daily Mean Service Time			Daily Mean Waiting Time		
		min	avg	max	min	avg	max	min	avg	max
Toshiba	Off	11.97	15.38	17.73	30.03	32.90	35.29	1.18	5.16	16.87
Toshiba	On	6.67	8.40	9.64	25.35	26.48	27.79	0.73	2.48	4.19
Fujitsu	Off	4.95	5.98	7.13	16.62	17.59	18.00	1.30	3.01	7.21
Fujitsu	On	2.05	2.44	2.74	13.12	13.84	14.51	0.99	2.04	4.05

Table 6: Summary of Results of On/Off Experiments (*users* file system, read requests only). The values reported are the minimum and maximum daily mean times (in millisecons) observed over all "on" days or all "off" days of the experiment.

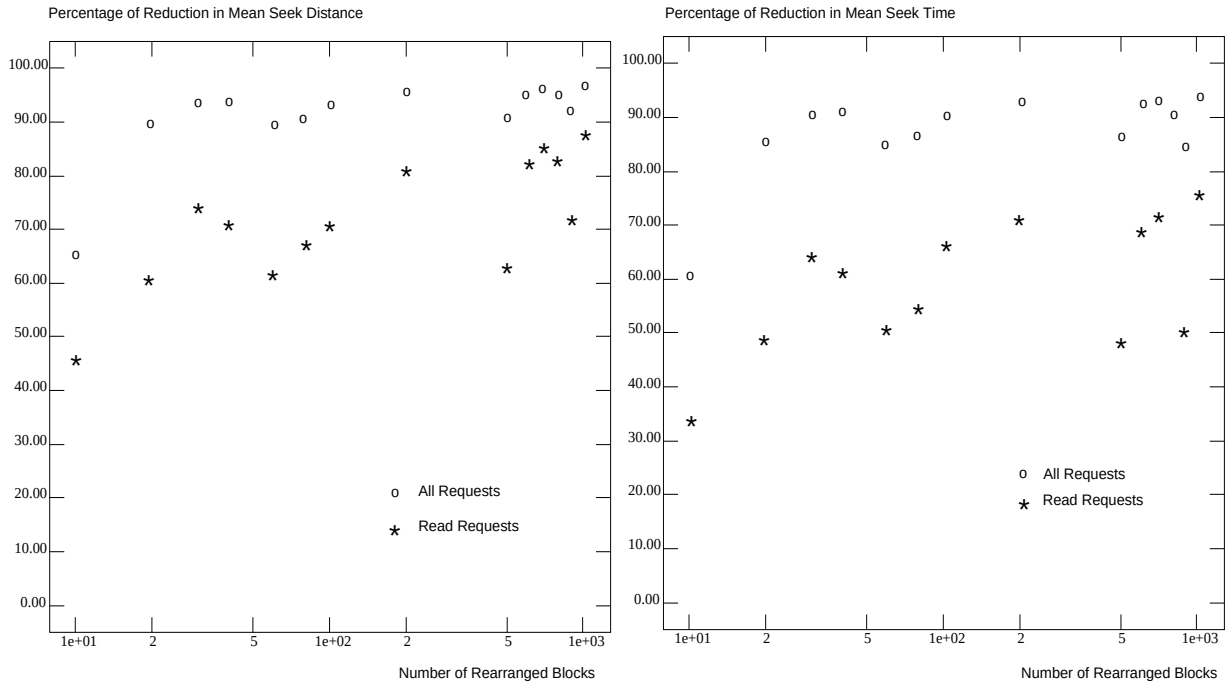

Figure 8: Results of experiments with different number of rearranged blocks on Toshiba disk.

	All Requests			Read Requests		
Disk	Organ-Pipe	Interleaved	Serial	Organ-Pipe	Interleaved	Serial
Toshiba	95	87	58	76	62	40
Fujitsu	90	88	76	78	77	65

Table 7: Summary of results of placement policy experiments (*system* file system). The values reported are the percentage reduction in daily mean seek time, as compared to the mean seek time that would have been observed had requests been served in arrival order, with no block rearrangement.

requests been serviced in arrival order. Two curves are plotted: one for all requests, and the other for read requests only.

The marginal benefit of rearranging blocks in excess of about 100 is quite small. This is because of the skew in the block request distribution (Figure 5). For the *system* file system, fewer than 2000 blocks absorbed all of the requests, and the 100 hottest blocks absorbed about 90%. The exact shape of the block request distribution varies from workload to workload. However, most data request distributions exhibit a great deal of skew. If this is true, then the reserved region can be made very small without severely impacting the performance of an adaptive driver.

5.5 The Placement Policy

The organ-pipe placement technique was used to obtain all of the results reported so far. We conducted a series of experiments in which the serial and interleaved policies were used. All of the experiments were performed using the *system* file system. On each day, 1018 blocks were rearranged on the Toshiba disk and 3500 blocks were rearranged on the Fujitsu disk.

Table 7 summarizes the results of these experiments. Results for the organ-pipe heuristic are included for purposes of comparison. More detailed results from several representative days are summarized in Tables 8 and 9.

	Organ-pipe		Interleaved		Serial	
	All Requests	Reads	All Requests	Reads	All Requests	Reads
FCFS Mean Seek Dist (cyln)	225	165	208	144	208	142
Mean Seek Distance (cyln)	8	23	15	24	22	39
Zero-length Seeks (%)	88	67	83	61	26	39
FCFS Mean Seek Time (ms)	21.46	16.14	20.02	14.39	20.02	14.23
Mean Seek Time (ms)	1.55	4.49	2.50	5.86	8.50	8.57
Mean Service Time (ms)	22.95	24.18	23.71	24.31	28.53	27.8
Mean Waiting Time (ms)	50.03	5.47	46.85	5.14	61.32	6.32

Table 8: Experiments with placement policies on Toshiba disk.

	Organ-pipe		Interleaved		Serial	
	All Requests	Reads	All Requests	Reads	All Requests	Reads
FCFS Mean Seek Dist (cyln)	408	311	400	305	440	321
Mean Seek Distance (cyln)	22	35	26	44	26	41
Zero-length Seeks (%)	74	59	77	62	35	35
FCFS Mean Seek Time (ms)	9.62	7.63	9.79	7.78	10.36	8.02
Mean Seek Time (ms)	1.10	1.74	1.12	1.92	2.49	2.82
Mean Service Time (ms)	13.83	13.03	14.35	13.74	15.47	14.51
Mean Waiting Time (ms)	44.52	3.23	51.33	3.25	46.16	2.73

Table 9: Experiments with placement policies on Fuji disk.

The organ-pipe and interleaved heuristics had similar performance. Organ-pipe performed slightly better on the Toshiba disk. The serial policy performed worse than either of the others. Its poorer performance indicates that the placement policy does have an impact and that block reference counts should be taken into account when placement decisions are made. The percentage of zero-length seeks was much smaller when the serial policy was used, since it does not attempt to cluster together the hottest of the rearranged blocks.

Our original motivation for the interleaved heuristic was to preserve rotational optimizations performed by the file system. Our driver is unable to measure rotational delay directly. However, on the Toshiba disk the difference between the measured service time and the seek time represents the combination of rotational delays plus transfer time[4]. Since transfer time is unaffected by block rearrangement, differences in the combined delay should be attributable to rotational latency. Table 10 gives the mean daily combined seek and rotation times we observed for read requests on the Toshiba disk under each of the placement policies. For comparison, the table also shows the mean rotation plus transfer time when no block rearrangement was being used.

It appears that block rearrangement using the organ-pipe (or serial) placement policies adds about a millisecond to the average rotational latency of a disk access. This is presumably because organ-pipe policy does not attempt to preserve the file system's rotational placement decisions within the reserved region. However, Table 8 indicates that the total read request service times were about the same under either policy. Although the organ-pipe policy produces higher rotational delays, it produces shorter seek times. These two effects essentially cancel each other out. Given that the organ-pipe policy is simpler than the interleaved policy, it appears to be the best overall choice.

[4]This is not true for the Fujitsu disk because of its track buffer. If a read operation is satisfied from the track buffer, the measured service time is the time to read the data from the disk's buffer in to the host's memory.

	Mean Rotational Latency + Mean Transfer Time (ms)
Without Rearrangement	18.58
Organ-pipe Placement	19.42
Serial Placement	19.29
Interleaved Placement	18.47

Table 10: Effects of different placement policies on rotational delays. The values in the table are average daily mean rotational delay + mean transfer time for read requests in milliseconds, on the Toshiba disk.

6 Conclusion

We have described the implementation of an adaptive block rearrangement technique under a UNIX operating system. The technique is implemented in a disk device driver and it is transparent to the rest of the system.

Measurements taken during the operation of the system on a network file server showed that it can be very effective at reducing disk seek times. In some experiments, the reduction was greater than 90%. Waiting times were also reduced, especially for bursty arrivals. The disk space overhead of the block rearrangement technique is low. Most of the benefits of block rearrangement were obtained by rearranging about 1% of all of the blocks on a disk.

The benefits of block rearrangement vary with the workload. The technique works best when request distributions are highly skewed, and when they change slowly over time. In our experiments, the best results were obtained under read-only file system shared by many users. Performance improvements were smaller under a file system holding the home directories of ten or twenty users.

We also experimented with several candidate placement policies for the rearranged blocks. The two policies that took estimated reference frequencies performed comparably, and better than the one which did not. Of these two, the organ-pipe policy appears to be the better choice because it is the simpler of the two.

Acknowledgements

We would like to thank Ólafur Gudmundsson, James da Silva, Harry Mantakos and the technical staff at the University of Maryland's Computer Science Department for their help and suggestions in setting up the experimental environment.

References

[Akyurek 93] Akyurek, Sedat, Kenneth Salem, "Adaptive Block Rearrangement," Proceedings of Ninth International Conference on Data Engineering, Vienna, Austria, April 1993.

[English 92] English, Robert M., Alexander A. Stepanov, " Loge: A Self-Organizing Disk Controller," Proceedings of the Winter 1992 USENIX Conference, San Francisco, CA, 1992.

[Floyd 89] Floyd, Richard A., Carla Schlatter Ellis, "Directory Reference Patterns in Hierarchical File Systems," IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 2, June 1989.

[Ford 91] Ford, Daniel A., Stavros Christodoulakis, "Optimizing Random Retrievals from CLV format Optical Disks," Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona, Spain, September, 1991.

[Grossman 73] Grossman, David D., Harvey F. Silverman, "Placement of Records on a Secondary Storage Device to Minimize Access Time," JACM, Vol.20, No.3, July 1973.

[Jobalia] Jobalia, Meenal, "Precision Measurement of Disk Delay Characteristics," Master's Thesis, Electrical Engineering Department, University of Maryland at College Park, 1991.

[McKusick 84] McKusick, K. Marshall, et al, "A Fast File System for UNIX," ACM Transactions on Computer Systems 2(3), August 1984.

[McVoy 91] McVoy, L.W., S.R. Kleiman, "Extent-like Performance from a UNIX File System," USENIX Winter 1991 Conference Proceedings, Dallas, TX, 1991.

[Ouster 85] Ousterhout, John K., et al, "A Trace Driven Analysis of the UNIX 4.2 BSD File System," Proceedings of the 10th ACM Symposium on Operating System Principles, 1985.

[Rosenblum 91] Rosenblum, M., J. K. Ousterhout, "The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems, Vol.10, February 1992, 26-52.

[Ruemmler 91] Ruemmler, C., J. Wilkes, "Disk Shuffling", HPL-91-156, Hewlett-Packard Laboratories, Palo Alto, CA, October, 1991.

[Salem 92] Salem, K., D. Barbará, R. Lipton, "Probabilistic Diagnosis of Hot Spots," Proceedings of the Eighth International Conference on Data Engineering, February, 1992, pp. 30-39.

[Salem 93] Salem, K., "Space-Efficient Hot Spot Estimation," CS-TR-3115, Computer Science Dept., Univ. of Maryland, College Park, MD, August, 1993.

[Staelin 90] Staelin, Carl, Hector Garcia-Molina, "Clustering Active Disk Data To Improve Disk Performance," Technical Report CS-TR-283-90, Department of Computer Science, Princeton University, September 1990.

[Staelin 91] Staelin, Carl, Hector Garcia-Molina, "Smart Filesystems," Proceedings of the Winter 1991 USENIX Conference, Dallas, TX, 1991.

[Vongsath 90] Vongsathorn, Paul, Scott D. Carson, "A System for Adaptive Disk Rearrangement," Software-Practice and Experience, Vol. 20(3), March 1990.

[Wong 80] Wong, C. K., "Minimizing Expected Head Movement in One-Dimensional and Two-Dimensional Mass Storage Systems," Computing Surveys, Vol.12, No.2, June 1980.