File Layout and File System Performance

Keith Smith
Margo Seltzer
Harvard University
TR–35–94

Abstract

Most contemporary implementations of the Berkeley Fast File System optimize file system throughput by allocating logically sequential data to physically contiguous disk blocks. This clustering is effective when there are many contiguous free blocks on the file system. But the repeated creation and deletion of files of varying sizes that occurs over time on active file systems is likely to cause fragmentation of free space, limiting the ability of the file system to allocate data contiguously and therefore degrading performance.

This paper presents empirical data and the analysis of allocation and fragmentation in the SunOS 4.1.3 file system (a derivative of the Berkeley Fast File System). We have collected data from forty-eight file systems on four file servers over a period of ten months. Our data show that small files are more fragmented than large files, with fewer than 35% of the blocks in two block files being allocated optimally, but more than 80% of the blocks in files larger than 256 kilobytes being allocated optimally. Two factors are responsible for this difference in fragmentation, an uneven distribution of free space within file system cylinder groups and a disk allocation algorithm which frequently allocates the last block of a file discontiguously from the rest of the file.

Performance measurements on replicas of active file systems show that they seldom perform as well as comparable empty file systems but that this performance degradation is rarely more than 10–15%. This decline in performance is directly correlated to the amount of fragmentation in the files used by the benchmark programs. Both file system utilization and the amount of fragmentation in existing files on the file system influence the amount of fragmentation in newly created files. Characteristics of the file system workload also have a significant impact of file system fragmentation and performance, with typical news server workloads causing extreme fragmentation.

1 Introduction

The single most important factor in determining the performance of a file system is the number of seeks and rotational delays that it incurs. A file system that makes many seeks will spend a disproportionate amount of time waiting for the disk head to reach its destination and correspondingly little time transferring data to or from the disk. This results in poor file system performance. Similarly, a system that performs few seeks will spend a larger portion of its time transferring data to and from the disk and will have correspondingly higher performance.

A common technique for reducing the number of seeks required by the file system is to increase the amount of data transferred in each disk request. Assuming that a separate seek is required by each disk request, a larger request size amortizes the latency of the seeks across more bytes of data. There are a variety of methods for increasing the size of disk requests. The simplest is to increase the file system block size, the minimum unit of allocation and transfer. This guarantees that all disk requests transfer more data, but has the drawback of increasing internal fragmentation in the file system.

Another way to increase the amount of data transferred in each disk request is to cluster blocks from the same file together on the disk. Using this technique, logically sequential data are allocated to physically contiguous disk blocks. This technique has been applied to the UNIX System V file system [3], the

Berkeley Fast File System [2], and is used in many contemporary UNIX file system implementations (e.g., BSD 4.4 [5], Solaris, SVR4, et al.).

Performance analyses [2][5] have shown that this clustering technique can provide a substantial improvement in file system performance. Intuitively, it seems that the performance of such systems must vary according to the success of the disk allocation algorithm in clustering files on the disk. When the file system is mostly empty, there should be large extents of free space, making it easier to perform optimal allocation. Similarly, when the free space is highly fragmented, poor file layout should result.

This observation implies that there is a major drawback to file system performance measurements as they are typically conducted. File system performance is usually not measured over the life of a file system. The typical scenario for measuring the performance of a file system is to create a new file system on a free disk partition and then to run the desired benchmark programs. This means that performance is typically measured on a new (and empty) file system—the optimal conditions for achieving good file layout.

This paper examines the long term behavior of the SunOS implementation of the Berkeley Fast File System (FFS). We examine file systems on several file servers that are in full-time use serving the needs of a diverse set of users. We analyze trends in file layout and their impact on file system performance over time.

The rest of this paper is structured as follows: Section 2 describes the implementation of FFS, focusing on the disk allocation algorithm, Section 3 describes the data we have collected, Section 4 describes our observations about file layout on the systems we examined, Section 5 studies the impact of file layout on file system performance, and Section 6 provides a summary of our results.

2 Disk Allocation in FFS

The Berkeley Fast File System (FFS) [1] divides the physical disk into regions called *cylinder groups*. A cylinder group consists of one or more consecutive cylinders and contains a bitmap of its free space. Disk space is managed in terms of three units: blocks, partial blocks, called *fragments*, and contiguous sets of blocks called *clusters*. Each file is represented by an on-disk data structure called an *inode,* which contains information about the file's owner, size, access permissions, etc.

The inode also contains a list of the blocks allocated to the file. In order to allow (nearly) unlimited file sizes, several levels of indirection are used in the block list. The inode only contains space for fifteen block addresses. The first twelve of these are used to point to the first twelve blocks of the file. The thirteenth address points to an *indirect block*, a block which does not contain file data, but rather contains pointers to additional data blocks. The final disk addresses in the inode point to a *double indirect block* and a *triple indirect block*, respectively. The double indirect block is a block of disk addresses, each of which points to an indirect block. The triple indirect block extents this indirection one more level. Inodes are allocated statically when the file system is created and are distributed evenly among the file system's cylinder groups.

Disk space is allocated to files one block at a time. To reduce internal fragmentation, a cluster of one or more fragments may be used instead of the final block of a file. When allocating a new block to a file, FFS first computes the *ideal block* to allocate. For the first block of a file, the ideal block is always the first data block in the cylinder group in which the file's inode is located. For other blocks, the ideal block is selected based on the location of the previous blocks of the file and the values of two file system parameters, *maxcontig* and *rotdelay. maxcontig* specifies the maximum cluster size allowed by the file system and *rotdelay* specifies the minimum rotational gap to be placed between clusters.

If the ideal block is not free, FFS attempts to allocate a nearby block in the same cylinder group in order to minimize the seek time required to reach the new block. First, FFS attempts to allocate a rotationally equivalent block on a different platter of the cylinder group. A file system parameter called *cpc* (for cylinders per cycle) specifies how many rotationally equivalent positions are considered. If no rotationally equivalent blocks are available, FFS uses a brute force search to find a free block in the cylinder group, first scanning from the position of the ideal block to the end of the cylinder group, then from the beginning of the cylinder group to the ideal block. FFS allocates the first free block encountered during this search.

Indirect blocks (of any type) are allocated in a similar manner. Whenever a file grows large enough to require an additional indirect block, FFS selects a new cylinder group for the file. The ideal block for the indirect block is the first data block in this cylinder group. FFS follows the algorithm described above if this block is already allocated. The data blocks referenced from an indirect block are allocated in the same cylinder group as the indirect block. When the first data block is allocated, its ideal block is also the first data block in the cylinder group. As subsequent blocks referenced by the indirect block are allocated, ideal blocks are selected based on the values of *maxcontig* and *rotdelay*.

In older implementations of FFS [1], all I/O requests were for a single block of data. Since the disk typically rotated at least one block between successive I/O requests from the operating system, *maxcontig* was set to one, and *rotdelay* was set to the distance the disk rotated between successive disk requests. On disks that had track buffers, higher read throughput could be attained by setting *rotdelay* to zero, forcing FFS to attempt to allocate files completely contiguously on the disk. In practice, this was seldom done because of its high cost in write throughput. Write requests to successive disk blocks incurred a delay of a full disk rotation.

Newer versions of FFS [2][5] have been modified to opportunistically cluster adjacent disk blocks into a single I/O operation during read-ahead and cache flushes. To maximize the opportunities for exploiting clustering, *maxcontig* is set to the maximum transfer size permitted by the hardware. As in the non-clustering implementations of FFS described above, *rotdelay* may be set to zero to optimize read throughput. Successive write requests may still incur a full rotation of overhead. With clustering, however, each write request writes many disk blocks amortizing this cost across more data, and in practice many clustering FFSs are tuned this way.

To demonstrate the way the block allocation algorithm works, we present two examples. First, consider a forty block file that is being created on an empty disk with *maxcontig* set to eight and *rotdelay* set to one. Because the disk is empty, FFS will always be able to allocate the ideal block. For the first block of the file, the ideal block will be the first block of the file's cylinder group. As subsequent blocks are allocated, FFS looks at the number of blocks that have been allocated contiguously prior to the block that is being allocated. If this number is less then *maxcontig* (eight), then the ideal block is one block after the last block of the file. Otherwise the ideal block is computed by skipping *rotdelay* blocks after the previous block of the file[1]. Following this algorithm, the first eight blocks of the file will be allocated in a single cluster, occupying the first eight data blocks of the cylinder group. When the ninth block is allocated, FFS notes that the previous eight blocks are contiguous, and therefore inserts a rotational gap before the new block. Since *rotdelay* is one, the ideal block for the ninth block will be two blocks after the eight block of the file. FFS would normally allocate the next seven blocks contiguously, in one cluster. Because only twelve blocks are directly referenced from the inode, however, the thirteenth block will require the allocation of an indirect block. This block, and the data blocks it references, is allocated in a different cylinder group. The indirect block is allocated in the first data block of new cylinder group. The thirteenth block of the file occupies the next block. FFS continues allocating subsequent blocks using *maxcontig* and *rotdelay*. Thus, blocks 13–20 of the file comprise a full cluster, as do blocks 21–28 and blocks 29–36. The final four blocks of the file form another cluster. Each of these clusters is separated by a single block as specified by *rotdelay*.

---

1. The *rotdelay* parameter is actually specified in milliseconds, rather than in disk blocks. This specifies the expected latency between disk requests. When computing the address of the ideal block, this latency is converted to disk blocks based on the block size and the rotational speed of the disk.

We now consider a similar file allocated on a similar file system that is not empty. In this case, the results of a file allocation are far less predictable. Since it is impossible to specify exactly which blocks will be allocated to the file without providing the free space bitmaps for our hypothetical file system, we will make some generalizations about the resulting file layout. The ideal block for the first block of the file will again be the first data block of the cylinder group. However, since this is the ideal block for the first block of every file in the cylinder group, it is probably already allocated. In this case FFS tries to find a nearby alternate block, first attempting to allocate a rotationally equivalent block on a different platter in the same cylinder. Again, because all file in the cylinder group attempt to allocate the same block as their first block, all of the rotationally equivalent blocks have probably been allocated. So FFS scans the cylinder group, starting at the location of the ideal block, and allocates the first free block that it finds. Thus, every time a new file is created, there is a high probability that the first free block in the cylinder group will be allocated as the file's first block. Note that whenever the idea block is unavailable, we hope that the alternate block selected by FFS is part of a cluster of free space, so that several blocks can be allocated contiguously on the disk.

Subsequent blocks of the file are allocated in a similar manner. This means that the first few blocks (or more) of a file are likely to be allocated in the first cylinder of the cylinder group. Since the ideal block is always close to the previous block of the file, it is likely to be in the same cylinder. If it is not available, a rotationally equivalent block (also in the same cylinder) is allocated. If no such blocks are available, the next free block in the cylinder group is allocated. Depending on the location of the ideal block, and the distribution of free space, there is a chance that this too will be in the same cylinder. When a block is eventually allocated in the second cylinder of the cylinder group, this same allocation pattern will take place in that cylinder. This continues as the file grows into subsequent cylinders of the cylinder group. This same allocation behavior will occur when the file requires an indirect block and switches cylinder groups.

Thus, when allocating a file on a partially full disk, we expect the file to start at the beginning of a cylinder group, and to extend towards the end of the cylinder group, with some blocks allocated from the first cylinder of the cylinder group, followed by some blocks in the second cylinder, and so on.

3 Data Collection

To evaluate the success of the FFS disk allocation algorithm in allocating contiguous disk blocks, we collected data form forty-eight file systems on four different file servers in the Harvard Division of Applied Sciences. These systems are primarily used to satisfy the daily computing needs of computer science researchers and graduate students in the Division. The file systems contained user home directories, news and mail spools, source trees under active development, and a variety of installed software. All of the file servers were SparcStations running SunOS 4.1.3.

The data consists of daily snapshots recorded for every file system under study. A snapshot is a summary of a file system's meta-data, including information about the size and configuration of the file system, the age, size, and location of each file, and a map of the free blocks on the file system.

Examining a single file system snapshot provides information about the file system at a single point in time and allows the file system to be reconstructed on a test machine for performance analysis in a controlled environment. A succession of snapshots allows us to study changes in file layout and file system performance over time.

Tables 1–4 describe the file systems on each of the four file servers in this study. All of the file systems studied had *maxcontig* set to seven and *rotdelay* set to zero (i.e., they were optimized for read throughput). All of the file systems had a block size of eight kilobytes except for the *newsspool* and *services* file systems on *das-news*, which had a block size of four kilobytes. Exact data about the ages of some of the file systems were unavailable. In these cases, the age was estimated by using the oldest modification time for any inode on the file system.

DAS-NEWS FILE SYSTEMS					
File System Name	**Size (MB)**	**Age (months)**	**NCG**	**BPG**	**Description**
/	11	36	3	720	Server's root file system.
cnews	565	34	50	1539	News articles.
export	108	36	21	720	Root file system used by diskless clients.
newsspool	898	15	141	1729	News articles and news software.
services	900	15	142	1729	WWW pages and software.
staff	353	32	32	1539	Accounts for system administrators.
usr	90	36	18	720	UNIX utilities and libraries.
usr15	240	27	22	1539	Sources for miscellaneous utilities.
var	13	36	3	720	System configuration and log files.

Table 1: das-news File Systems. This table summarizes the file systems on the file server *das-news*. The name is the local name used for the file system. The size specifies the total capacity of the file system in megabytes. The age is the approximate age of the file system in months as of December 1, 1994. Subtract ten months to obtain the age at the beginning of the study. NCG is the number of cylinder groups used by the file system, and BPG is the number of eight kilobyte blocks per cylinder group. A brief description explains how each file system is used.

SPEED FILE SYSTEMS					
File System Name	**Size (MB)**	**Age (months)**	**NCG**	**BPG**	**Description**
/	15	19	3	720	Server's root file system.
bespin-new	823	15	74	1520	On-line Sun documentation.
local	953	17	86	1520	Locally installed software.
speed	574	5	52	1520	Scratch area used for installations.
usr	220	19	42	720	UNIX utilities and libraries.
usr3	953	18	86	1520	Experimental source code management system.
usr4	953	18	86	1520	Accounts for students and faculty.
usr5	953	18	86	1520	Accounts for post-docs.
usr12	953	18	86	1520	Graduate student accounts.
usr16	509	15	46	1520	xterm support utilities.
var	29	19	6	720	System configuration and log files.

Table 2: speed File Systems. This table summarizes the file systems on the file server *speed*. The name is the local name used for the file system. The size specifies the total capacity of the file system in megabytes. The age is the approximate age of the file system in months as of December 1, 1994. Subtract ten months to obtain the age at the beginning of the study. NCG is the number of cylinder groups used by the file system, and BPG is the number of eight kilobyte blocks per cylinder group. A brief description explains how each file system is used.

ENDOR FILE SYSTEMS					
File System Name	**Size (MB)**	**Age (months)**	**NCG**	**BPG**	**Description**
/	15	15	2	1360	Server's root file system.
adm	90	15	9	1360	Administrative files.
backupdatabase	1907	15	171	1520	Database for backup system.
ftp	90	15	9	1360	Files available for anonymous ftp.
local	953	15	86	1520	Locally installed software.
mail	180	15	18	1360	Mail files.
rabin	450	15	41	1520	Accounts for faculty and grad. students. Sun-3 binaries.
root	15	15	2	1360	Backup of / file system.
software	953	15	86	1520	Commercial software packages.
software2	1006	15	91	1520	Commercial software packages.
tmp	90	15	9	1360	Storage for temporary files.
usr	224	15	23	1360	UNIX utilities, libraries, and man pages.
usr0	224	9	23	1360	Backup of *usr* file system
usr1	620	15	63	1520	Accounts for administrative staff.
usr2	509	15	51	1520	Accounts for engineering students and faculty.
usr6	953	15	86	1520	Accounts for computer science courses and first year graduate students.
usr8	953	15	86	1520	Undergraduate accounts.
var	68	15	7	1360	System configuration and log files.
var/tmp	90	15	9	1360	Storage for temporary files.
white	450	15	41	1520	User accounts for CS theory faculty and students.

Table 3: endor File Systems. This table summarizes the file systems on the file server *endor*. The name is the local name used for the file system. The size specifies the total capacity of the file system in megabytes. The age is the approximate age of the file system in months as of December 1, 1994. Subtract ten months to obtain the age at the beginning of the study. NCG is the number of cylinder groups used by the file system, and BPG is the number of eight kilobyte blocks per cylinder group. A brief description explains how each file system is used.

VIRTUAL12 FILE SYSTEMS					
File System Name	**Size (MB)**	**Age (months)**	**NCG**	**BPG**	**Description**
/	20	26	6	456	Server's root file system.
glan5	1704	25	90	2596	Commercial and locally installed software packages.
glan6	502	19	27	2596	Accounts for systems students and faculty.
glan7	401	19	21	2596	Accounts for systems students.
glan8	401	19	21	2596	Accounts for systems students and faculty.
glan9	401	19	21	2596	Accounts for systems students and faculty.
usr	84	26	26	456	UNIX utilities and libraries.
var	66	26	20	456	System configuration and log files.

Table 4: virtual12 File Systems. This table summarizes the file systems on the file server virtual12. The name is the local name used for the file system. The size specifies the total capacity of the file system in megabytes. The age is the approximate age of the file system in months as of December 1, 1994. Subtract ten months to obtain the age at the beginning of the study. NCG is the number of cylinder groups used by the file system, and BPG is the number of eight kilobyte blocks per cylinder group. A brief description explains how each file system is used.

4 Characteristics of File Layout

It has been shown that any allocation scheme that allocates variable-sized pieces of memory, and does not relocate allocated memory, cannot be guaranteed to use memory efficiently [4]. In such a system, memory will become fragmented, and it is possible that a request for less than the total amount of free memory will not be satisfiable because there is not a free piece of memory of the desired size. This result applies even if there are a limited number of sizes that can be allocated.

This is the problem faced by a clustering file system such as FFS. FFS attempts to allocate disk space in clusters of *maxcontig* blocks. When contiguous blocks are not available, discontiguous ones are allocated instead, resulting in a fragmented file that takes longer to read and write. This problem is compounded by FFS's disk allocation algorithm, a simple heuristic that is not guaranteed to find the appropriate number of contiguous disk blocks, even if they are available.

The clustering enhancements to FFS have been shown to offer significantly improved file system performance [2][5] for empty file systems where it is easy to allocate clusters of contiguous disk blocks. The amount of fragmentation that occurs in a file system determines whether these performance gains can be maintained over the life of the file system. We use the data described in Section 3 to examine the types of fragmentation that occur on active file systems.

4.1 Free Space Distribution

When a new file is created on an FFS, its layout is determined by the availability of free space on the file system. If the free space in the file's cylinder group is clustered together in one place, then the FFS block allocation algorithm will generate a good file layout. If the free space is scattered throughout the cylinder group, then it will be impossible for FFS to achieve a good layout, and a fragmented file will result.

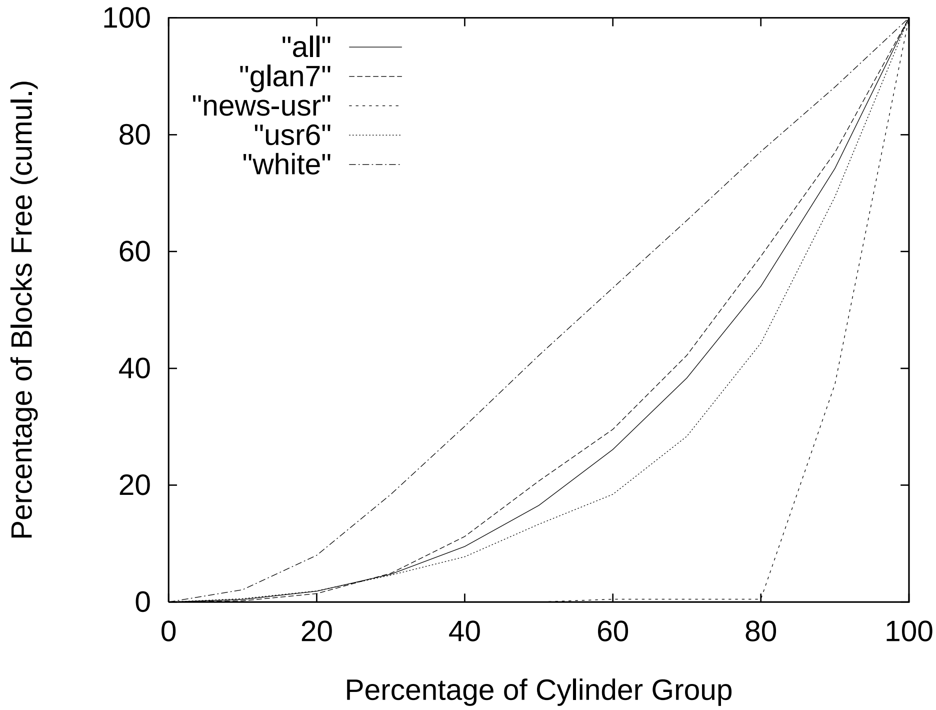

Figure 1: Distribution of free space within cylinder groups. This graph plots the cumulative percentage of a file system's free space as a function of position within the cylinder groups. The curve labeled "all" uses the combined data from all of the file systems in the study, with each file system weighted equally. The other curves provide data from a selection of individual file systems. "news-usr" is the *usr* file system from *das-news*. All data in this chart is from file system snapshots taken on October 15, 1994. Data from other dates is similar.

To examine the distribution of free space in an FFS snapshot, we divided the cylinder groups into ten equal-sized pieces called *segments*. The data for the first segment of each cylinder group were summed together to derive statistics for the free space usage in the first 10% of all cylinder groups. The statistics for other segment positions were similarly totaled.

Figure 1 shows the distribution of free space within a file system's cylinder groups. On a typical file system almost 50% of the free blocks are located within the last 20% of the cylinder groups. This skewed distribution of free space within a file system's cylinder groups is the direct result of the disk allocation algorithm. Recall that the basic pattern of allocation is for a file to start a the beginning of a cylinder group, and to extend towards the end of the cylinder group. Thus, every file will allocate some space at the beginning of the cylinder group, but only the larger files will allocate space from later in the cylinder group. The result is that the space at the beginning of a cylinder group is consumed more rapidly than the space at the end.

The sample file systems shown in Figure 1 exhibit a range of free space distributions. *glan7* and *usr6* are typical of most file systems, with most of their free space skewed toward the ends of cylinder groups. *white* and *news-usr* (the *usr* file system from *das-news*) represent the extremes seen in our file systems. The free space distribution in the news-usr file system is especially strongly skewed, with almost all of the free space located in the last 80% of the cylinder groups. This file system contains various UNIX binaries and libraries, and is probably unchanged from the day it was first installed. Recall that each newly created file starts at the beginning of its cylinder group, extending towards the end of the group as it grows. If many files have been created, but none removed (as we would expect on this file system) we would expect all free space at the beginning of the cylinder groups to be used, but little from the end of the cylinder groups.

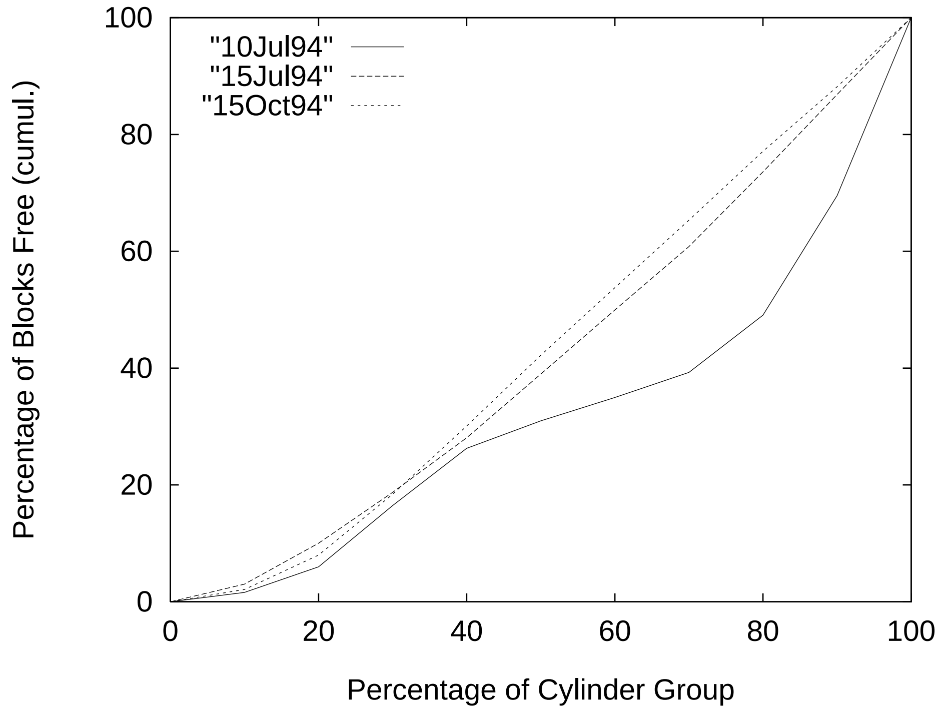

Figure 2: Distribution of free space on *white*. This graph plots the cumulative percentage of free space on the *white* file system as a function of position within the file system's cylinder groups. The curves labeled "10Jul94" and "15Jul94" provide data from before and after 290 megabytes of data were removed from the file system. "15Oct94" is the distribution of free space on October 15, 1994, also shown in Figure 1.

The *white* file system, in contrast, has an extremely even distribution of free space. This is the result of some housecleaning that was performed on the file system three months prior to this snapshot. On July 13, 1994, a large amount of data was removed from *white*, and it went from 90% full to 25% full. The removed data was fairly evenly distributed across *white*'s cylinder groups, resulting in an even distribution of free space afterwards. By October 15, 1994, the date of the snapshot used in Figure 1, *white* was still only 36% utilized. Figure 2 demonstrates this effect by comparing the free space distribution on *white* both before and after this "housecleaning."

The skewed distribution of free space exhibited by most file systems may have a significant impact of the layout of newly created files. Blocks are allocated to a new file starting from the beginning of the file's cylinder group. Since there is very little free space at the beginning of cylinder groups, FFS's clustering algorithm will often allocate poorly clustered blocks to new files, despite the fact that there may be large clusters of free space available at the end of the cylinder group. This issue is examined in more depth in the next section.

4.2 File Layout

We define a *layout score* to quantify the fragmentation of individual files. This score is defined to be the percentage of blocks in the file that are optimally allocated. A block is optimally allocated if there is no gap between it and the previous block of the same file. The first block of a file is not included in this calculation since it is impossible for it to have a "previous block." To illustrate the calculation of a file's layout score, consider a three block file occupying blocks 100, 101, and 200 on the disk. Since the first block is ignored, there are only two blocks that are considered in computing the layout score. The second block of the file (block 101) is contiguous with its predecessor (block 100). The third block (block 200) is not. Thus the layout score for this file is 0.5, indicating that half of the blocks in the file are located optimally on the disk.

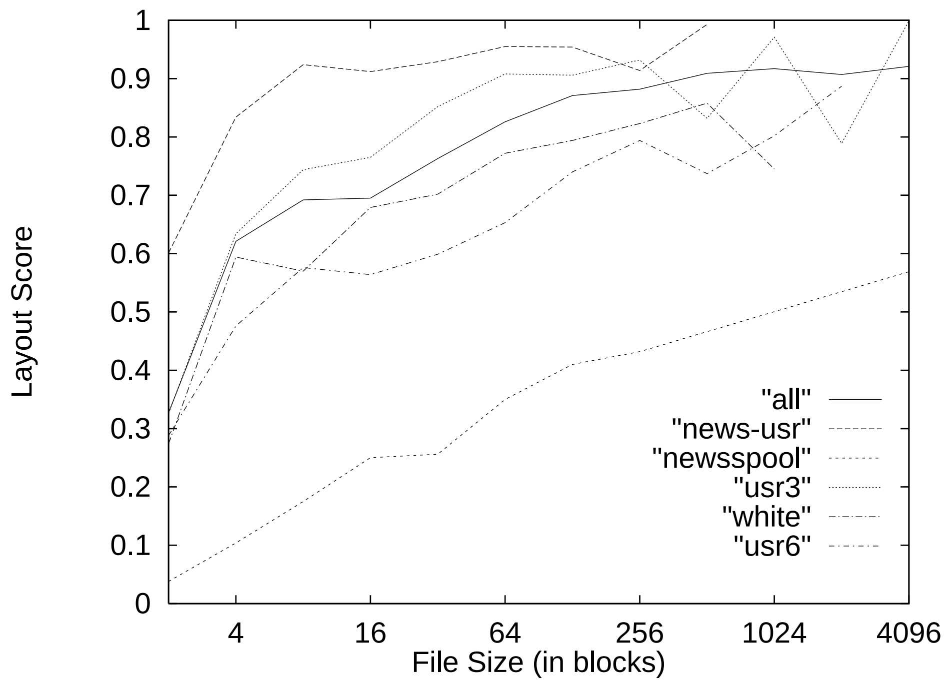

Figure 3: Layout score as a function of file size. This chart plots layout score as a function of file size (expressed in blocks). The files on each file system are grouped according to the number of blocks they use. The group boundaries are powers of two. A file is assigned to a group by rounding its size (in blocks) up to the nearest power of two. The layout score for a group is computed as the percentage of blocks, for all files in the group, that are optimally allocated. The curve labeled "all" uses the combined data from all of the file systems studied. Other curves are data from representative file systems. "news-usr" is the *usr* file system on *das-news*. The data in this chart is from file system snapshots taken on October 15, 1994.

A layout score of 1.0 indicates that a file's layout is completely contiguous. Similarly, a layout score of 0.0 indicates that no two successive blocks of the file are contiguous on the disk. On file systems with a non-zero *rotdelay*, it is impossible for large files to achieve a layout score of 1.0, since *rotdelay* forces the disk allocation algorithm to leave gaps in the file. In all of the file systems used in this study the *rotdelay* parameter is zero, theoretically allowing perfect layout scores.

In order to characterize the types of files that achieve good layout we analyzed the effect of file size and file system age on file layout. In Section 5 we also consider the effect of file system utilization. Figure 3 shows the relation between file size and file layout. The general trend on all file systems is that small files are more fragmented (i.e., have lower layout scores) than large files. Across all of the file systems that we studied, two block files have an average layout score of 0.327. In other words, only 32.7% of the two block files (files between 8193 and 16384 bytes) are laid out on two contiguous disk blocks. Larger files, in contrast, have better layout scores. Files with 33–64 blocks (256KB–512KB) have an average layout score of 0.826. In other words, slightly less than one seek is required for every five blocks read from these files. Larger files have higher layout scores.

The sample file systems shown in Figure 3 exhibit the range of relationships between file size and layout score seen in the file systems studied. The cumulative numbers across all file systems are indicative of most file systems, such as *usr3* and *white*. *usr6* and the *news-usr* (the *usr* file system from *das-news*) are at the extremes seen for most file systems. As discussed earlier, *news-usr* is a static file system, which has barely changed since the day it was first installed. With no file turn over, there is little opportunity for file fragmentation to occur. In contrast, *usr6*, the home to accounts of first year graduate students and the course accounts used by instructors and teaching fellows in computer science, is heavily used and exhibits

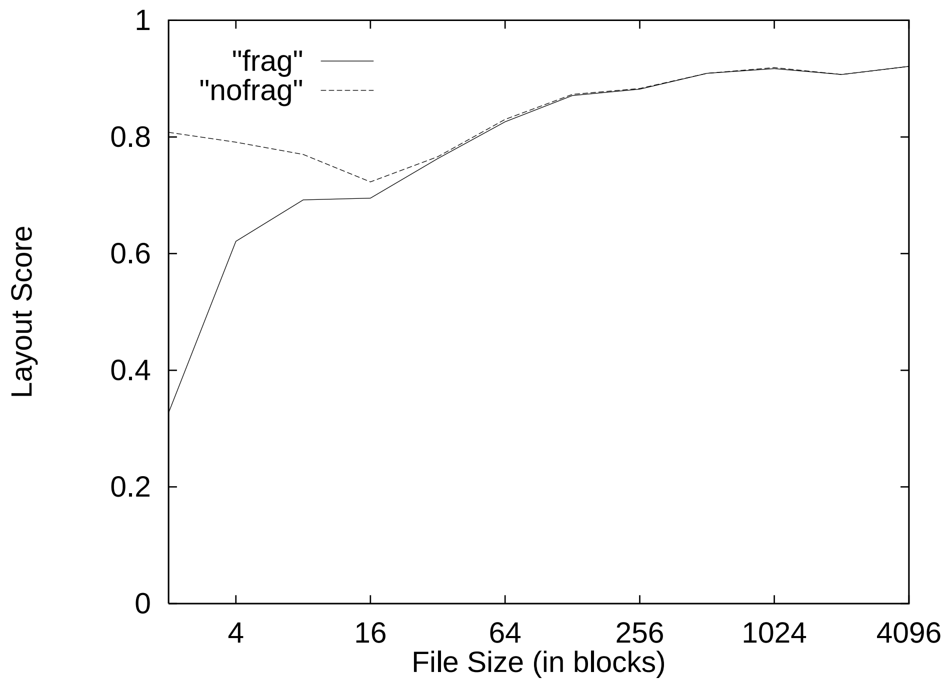

Figure 4: Effect of fragments on layout score. This graph plots layout score as a function of file size (expressed in blocks). Both lines are computed using data from all of the file systems studied. For the "frag" line, layout score is computed as the percentage of blocks that are optimally allocated. The "nofrag" line is computed the same way, except that a fragment at the end of a file is always considered to be optimally allocated, regardless of its location on the disk. All data is from file system snapshots taken on October 15, 1994. Data from other dates is similar.

greater fragmentation than most other file systems. The *newsspool* file system is a unique outlier. As a repository for net-news articles, it is subjected to a load consisting of the frequent creation and deletion of many small files, resulting in the extreme fragmentation shown in Figure 3. The other news file system in the study (*cnews*) exhibits similar fragmentation.

The lower layout scores exhibited by smaller files are the result of several factors. Recall from Section 4.1 that free space is skewed towards the end of a file system's cylinder groups. Regardless of this, the allocation of every file starts at the beginning of the file's cylinder group. Thus, small files are allocated near the beginning of a cylinder group where free space is unlikely to be well clustered. Large files are also allocated starting at the beginning of their cylinder group, but as a large file grows, it uses well-clustered free space from later in the cylinder group, raising the file's layout score.

Another factor which lowers the layout scores for small files is the allocation of the fragments at the end of the files. A different algorithm is used to allocate fragments than is used to allocate full blocks. As a result, the final fragment of a file is seldom allocated contiguously with the previous full block of the file. Because layout score is weighted by the size of a file, this extra seek has a much bigger impact on the layout scores of small files than on the layout scores of larger files. Figure 4 demonstrates this effect by recomputing the layout scores ignoring a seek required to reach the final block of a file when that block is a fragment. For large files, the extra seek required by the final fragment has no discernible effect on layout score. For small files, however, a final fragment can have a large impact on layout score.

4.3 File Layout Over Time

Intuitively, we expect that over time, an active file system will become increasingly fragmented. As a simple test of this intuition, we examined file fragmentation on the test file systems as a function of time. Figure 5 shows the combined fragmentation of all file systems at four month intervals during the period of the study. There is a small, but noticeable downward trend in layout scores over time.

To gain a better understanding of the change in file fragmentation over time, we modified the layout score metric to express the amount of fragmentation in an entire file system. The layout score for a file system is the percentage of file data blocks in the file system that are optimally allocated. The total layout score was computed daily for each file system in the study (static file systems, such as the *usr* file system on *das-news*, discussed above, were ignored.), and a linear regression analysis was performed to determine whether age can be used to predict the amount of fragmentation for a given file system. Table 5 shows the correlation between age and layout score on a variety of active file systems. In this data we see that many file systems display the expected inverse relationship between layout score and file system age (as the file systems get older, the layout score decreases and fragmentation increases). There are also some file systems (e.g., *cnews*, *glan7*, *usr3*, *usr8*) that show little correlation between age and layout score, and a few file systems (e.g., *bespin-new* and *mail*) that exhibit a positive correlation.

Combining the data from the file systems in Table 5, we find that there is not statistical evidence to support a generalized statement about the relationship between age and layout score. (The resulting correlation coefficient is –0.0426.) This is explained by the different types of correlation between age and layout score found in the various file systems studied and by the fact that different file systems age at substantially different rates. The *usr6* file system, for example, is the home for twenty different user accounts. In contrast, *usr12*, a file system of comparable size, has half as many users.

5 File System Performance

We have seen that file fragmentation frequently occurs in real world instances of FFS. This fragmentation may degrade the performance of an FFS. To assess this risk, we performed benchmarks on the file systems using the snapshots to reproduce the source file systems on a test machine. Because the snapshots span many months, we were able to examine changes in performance as the source file systems aged. The performance of these facsimile file systems was compared to the performance of comparable empty file systems to determine the amount of performance degradation experience by real file systems.

5.1 Performance Tests

Because the SunOS file systems we studied were in active use, it was not feasible to run benchmarks on them. Instead, the meta-data in the snapshots were used to reconstruct the file systems on the disk of a test machine. This not only allowed the file systems to be analyzed in an otherwise quiescent environment, but also made it easier to study the performance of comparable empty file systems. In the following discussion, the term “original file system” is used to refer to the actual file systems on the Division of Applied Sciences file servers, and “test file system” or “copied file system” is used for the facsimile file systems produced on the test machine. The benchmarks described in this section were run on a SparcStation I with 32 megabytes of memory, running 4.4BSD. Note that both SunOS 4.1.3, which is run on the servers where the snapshot data were collected, and 4.4.BSD implement a clustering FFS. Because both file systems use the same disk allocation algorithm, as described in Section 2, the change in operating system does not effect our results. Details of the test system are summarized in Table 6.

In order to approximate the original file systems’ configurations as closely as possible, the test file systems were created using the same values of *maxcontig* and *rotdelay.* The test file systems were also configured with the same number of cylinder groups as the corresponding original file systems. When different sized cylinders on the original and test disks made it impossible to precisely duplicate the size of the cylinder groups, slightly larger cylinder groups were created on the test file system. The extra blocks in each

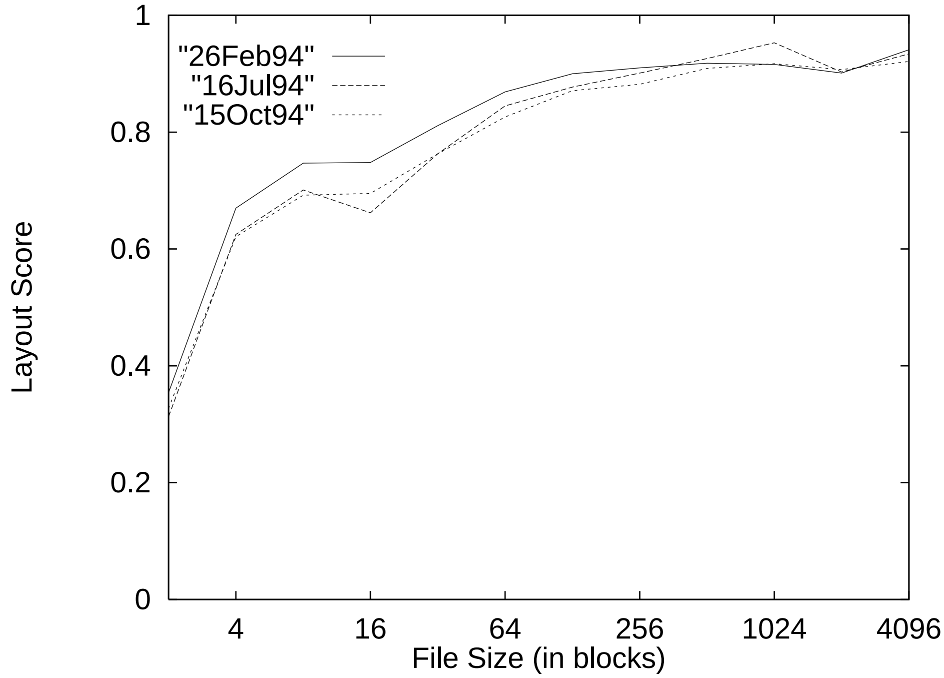

Figure 5: Change in layout scores over time. This chart plots the layout score versus the file size for all of the file systems in the study at three different dates over an eight month period.

File System	Correlation Coefficient	P-value	File System	Correlation Coefficient	P-value
bespin-new	0.8297	0.0000	newsspool	–0.2137	0.0000
cnews	–0.0252	0.0000	services	0.2986	0.0000
ftp	–0.8873	0.0000	software	0.1995	0.0058
glan5	–0.7166	0.0000	staff	–0.9059	0.0000
glan6	–0.5206	0.0000	usr3	0.0885	0.0000
glan7	–0.0933	0.0194	usr4	–0.4479	0.0000
glan8	–0.7829	0.0000	usr6	–0.6077	0.0000
glan9	–0.5658	0.0000	usr8	0.0753	0.0578
local (endor)	–0.9541	0.0000	usr12	–0.9734	0.0000
mail	0.6419	0.0000	white	–0.7620	0.0000

Table 5: Correlation between file system age and file system layout score. This table presents the correlation between file age and file system layout score for twenty active file systems over the duration of the study. The correlation coefficient indicates the strength of the relationship for each file system. The square of this coefficient is the proportion of the variation in layout score that can be explained by a linear relationship to the date. Negative values indicate an inverse relationship. The P-value expresses the probability that there is no relationship.

CPU Parameters	
CPU	SparcStation I
Mhz	20
Disk Parameters	
Disk Type	Fujitsu M2694EXA
RPM	5400
Sector Size	512 bytes
Sectors per Track	68-111
Cylinders	1818
Tracks per Cylinder	15
Track Buffer	512 KB
Average Seek	9.5 ms

Table 6: Benchmark Hardware Configuration

cylinder group were marked as allocated, so the test file system could not use them. Because the test disk has variable sized cylinders, FFS's notion of rotationally equivalent blocks is meaningless (FFS assumes all cylinder are the same size), and *cpc* was set to zero on all test file systems. With both *rotdelay* and *cpc* set to zero (all of the original file systems had a *rotdelay* of zero), FFS allocates disk blocks in a simple and predictable manner. When a new file is created, its first block will use the first free block in the file's cylinder group; its next block will use the next free block in the cylinder group, and so on.

Because all of our file system benchmarks rely on the creation and use of new files, the most important characteristic of the original file systems to reproduce is the arrangement of free space. Other details, such as the precise mapping of allocated blocks to files and the directory hierarchy, are less important because they have little or no impact on the layout or performance of newly created files. Therefore the only meta-data copied from the snapshots of the original file systems were the free space bitmaps. The resulting test file system contained an empty root directory and no other data but could only utilize the same free blocks as the original file system.

We compared the test file systems using a benchmark of sequential read and write performance. The data set consisted of 32 megabytes of data, decomposed into the appropriate number of files for the file size being measured. Because FFS allocates all of the files in a single directory in the same cylinder group, the data was divided into subdirectories, each containing no more than twenty-five files. This allowed the benchmark to use most of the cylinder groups on the test file systems. There were two phases to this benchmark:

1. **Create/write:** All of the files are created by issuing single *write* system call for all of the data in the file.
2. **Read:** The files are read in the same order in which they were created. A single *read* system call is issued to read all of the data in each file.

Performance was measured for a range of file sizes from eight kilobytes to one megabyte from test file systems reproduced from dates at four intervals evenly spread over the data collection period. Nine different file systems were used for these measurements. Active file systems used for a variety of purposes (user accounts, news, serving binaries) were selected.

5.2 Test Results

For each file size tested, the throughput for reading and writing on the copied file systems was compared to the throughput for the corresponding empty file system. Figure 6 and Figure 7 show the performance of each of the test file systems as a percentage of the throughput of the corresponding empty file system.

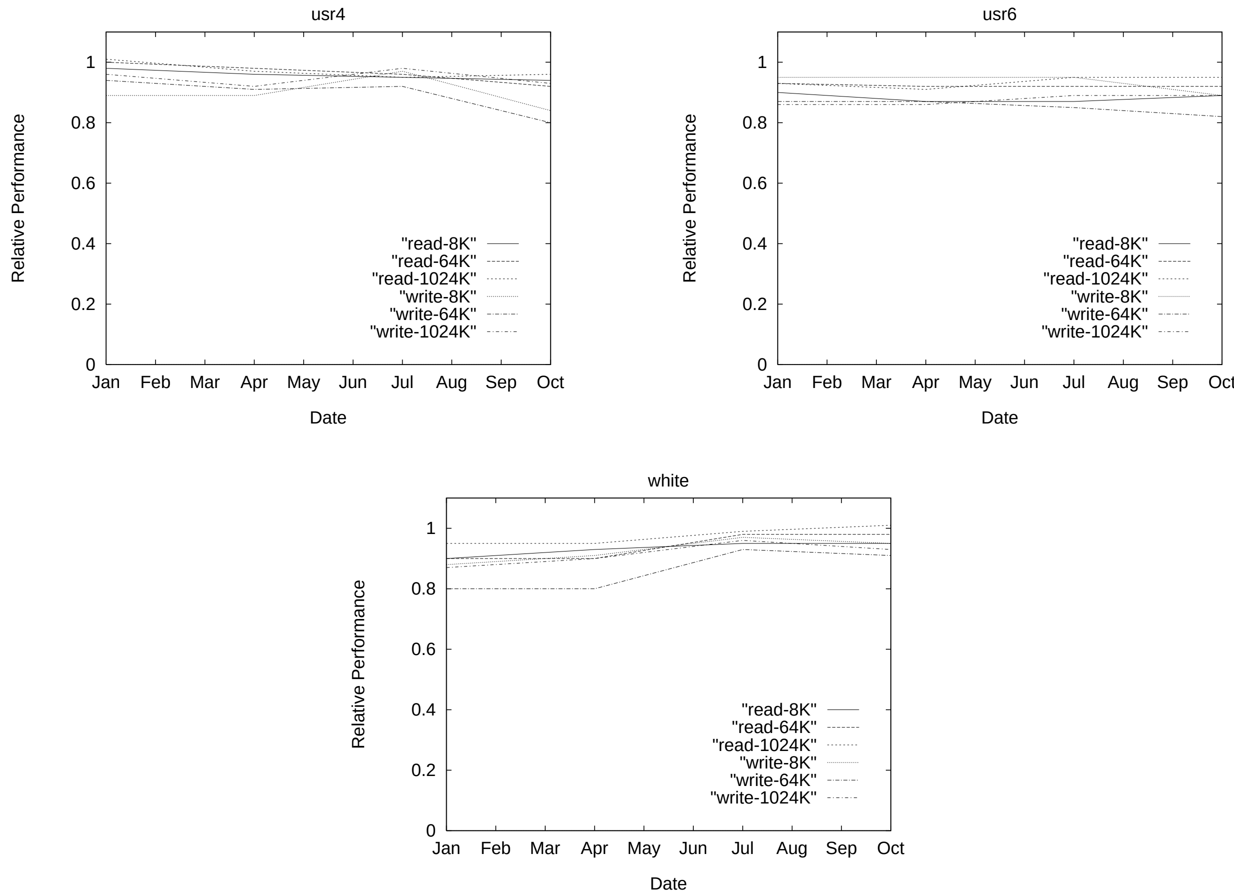

Figure 7: Relative File System Performance (II). Relative performance is the file system throughput measured on a copied file system and expressed as a fraction of the throughput on a comparable empty file system. For each file system, relative performance was measured reading and writing files of three different sizes, 8 KB, 64 KB, and 1024 KB, and is plotted as a function of time. Tests were performed on copied file systems from four dates, the 26th of January, April, July, and October of 1994.

These graphs indicate that there are performance differences for the different file sizes tested. Many of the largest differences between the copied file systems and the empty file systems occur in the 64 KB test.

For each of the nine file systems, we ran read and create tests for each of three file sizes on four different dates for a total of 216 tests. Of the seventy-two tests that used a 1024 KB file size, only two performed at less than 85% of the corresponding empty file system's bandwidth. These two points were for January on the *cnews* file system. Excluding the January tests on *cnews*, which were generally worse than any other test (for reasons we shall come to), of the seventy tests that used a file size of 8 KB, there was only one test case that achieved less than 85% performance (the April write test on *glan5*, 84.2%). Of the seventy-two tests using 64 KB files, fifteen performed at less than 85% of the empty file system's throughput. Of the remaining fifty-seven tests using 64 KB files, forty-one of them were over 90%.

There are a variety of reasons for the performance differences among the different file sizes. All of the file sizes suffer from the increased fragmentation of free space on the copied file systems. This is most noticeable in the 64 KB files because of ameliorating circumstances in the 8 KB and 1024 KB tests.

Recall that when FFS allocates an indirect block, it switches to a new cylinder group. Since the first indirect block is required for the thirteenth data block of a file, all of the files in the 1024 KB tests use

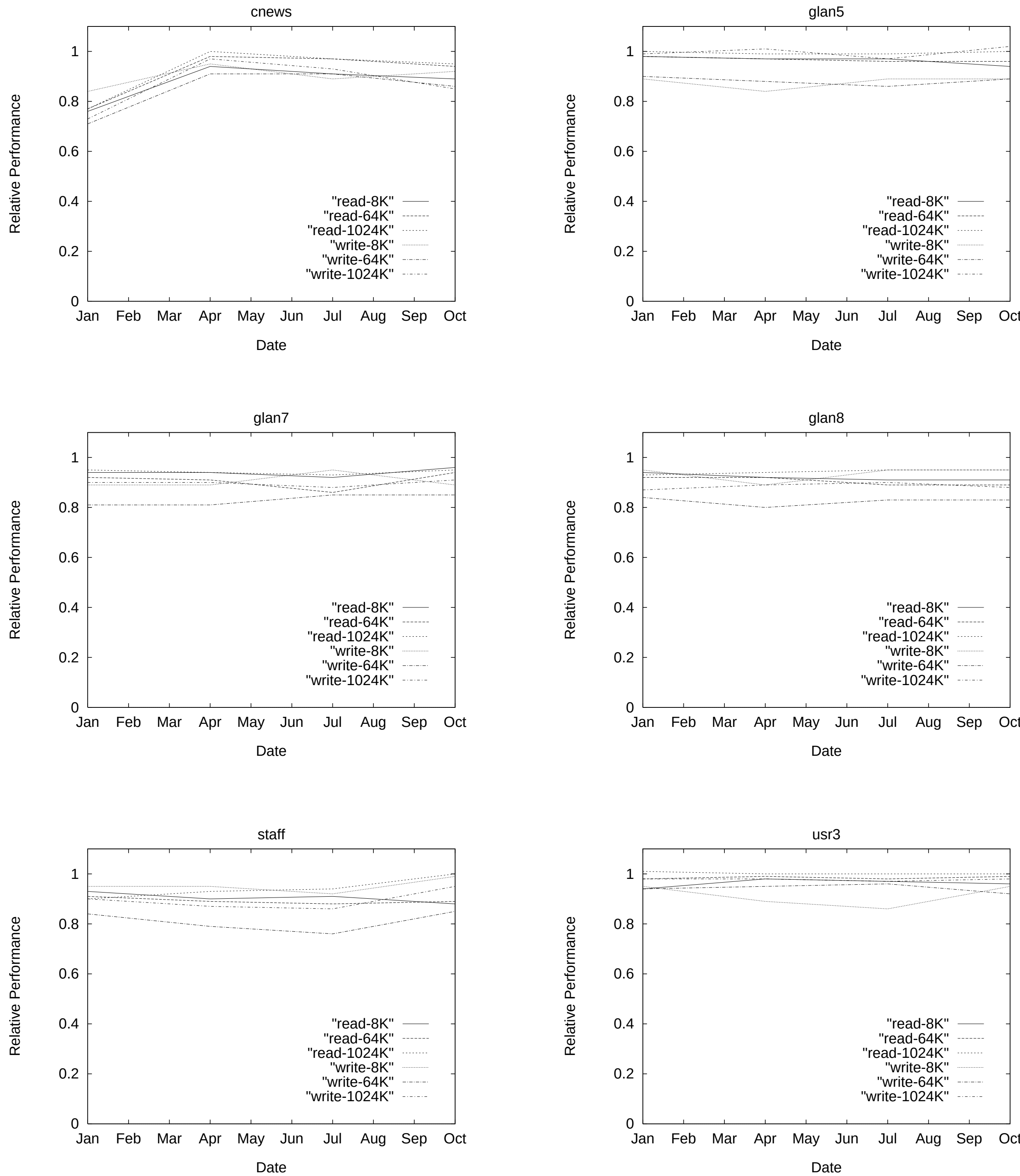

Figure 6: Relative File System Performance (I). Relative performance is the throughput measured on a copied file system and expressed as a fraction of the throughput on a comparable empty file system. For each file system, relative performance was measured reading and writing files of three different sizes, 8 KB, 64 KB, and 1024 KB, and is plotted as a function of time. Tests were performed on copied file systems from four dates, the 26th of January, April, July, and October of 1994. The final test date *cnews* and *staff* was October 20, 1994.

indirect blocks. The result is that reading or writing each of these files involves at least two long seeks (i.e., between cylinder groups). One seek is required to get to the cylinder group containing the indirect block and the data blocks that it references; another is required to return to the original cylinder group to read or write the next file. Because the benchmarks that use smaller file sizes do not incur these long seeks, performance differences caused by fragmentation within a cylinder group have a larger impact on the relative performance than they do in the 1024 KB tests.

The fragmentation of free space has little impact in the 8 KB test case. The benchmark creates directories containing twenty-five files. Each directory is placed in a different cylinder group. Since the block size is 8 KB for all of the file systems, the 8 KB test case will read or write twenty-five blocks of data to one cylinder group, then seek to a different cylinder group to start the next directory. Thus, the 8 KB test spends a larger portion of its time performing seeks than either of the other test sizes. Increased fragmentation in the copied file systems means that the files in a given directory are typically spread out more than on an empty file system. Because of the large number of seeks between cylinder groups, however, the amount of overhead introduced by this fragmentation has a smaller impact on overall performance than in the 64 KB test case.

In general, the write tests performed worse than the corresponding read tests. In the 64 KB tests, fourteen of the fifteen tests cases that had a relative performance of less than 85% were write tests. (The sole read test was for January on *cnews*.) The lower relative performance for writing than for reading was caused by the overhead of meta-data updates. When a file is created, its inode and directory must be updated. The inodes are located at or near the beginning of the cylinder group. Because the directory is small (one block) and created before the files, it is likely to occur one of the first free blocks in the cylinder group. When each file is created, FFS must allocate an inode and update the directory. These operations are synchronous, and require a seek from the data blocks of the previously written file to the beginning of the cylinder group. In the test file systems, which suffer from fragmentation, and in which little free space is available early in the cylinder groups, the file data tends to be allocated later in the cylinder groups than it is on empty file systems. Thus, these seeks are longer on the test file system than on the empty file system. During the read tests, the directory and an inode must be read for each file accessed, but most of the seeks issued in the write tests are avoided because the inodes are read of the disk a block at a time and cached in memory, as is the directory.

5.3 Fragmentation and Performance

An important factor in the performance of these benchmarks is the layout of the test files. Files that are laid out contiguously on the disk can be read and written quickly. In contrast, files whose blocks are scattered across the disk do not perform as well.

To validate the usefulness of the layout score as a predictor of performance we examined the correlation between the layout score of the test files and the throughput measured in the corresponding test. The results are presented in Table 7. In general there is a strong correlation between the layout score and performance. This correlation is better for 1024 KB files than for 64 KB files, and better for read performance than for write performance. In both cases, the stronger relationship is the result of meta-data overhead. In the 1024 KB tests, there is less meta-data to be read/written relative to the file data, so intra-file seeks, which are measured by layout score, play a larger role in determining performance than they do for smaller file sizes. Similarly, in the write tests, meta-data must be synchronously updated, whereas in the corresponding read tests, most meta-data requests can be satisfied out of the cache. As a result, more meta-data related seeks are required by the write tests, and the intra-file seeks measured by layout score have a smaller effect on total performance.

Achieving optimal file layout requires contiguous free space on the file system. Thus, anything that causes free space to become fragmented will degrade the performance of a file system. To better understand the causes of fragmentation we examined the layout of the file systems and of the files used in the performance

Tests	Correlation Coefficient	P-value
All 64 KB Tests	0.4871	0.0000
64 KB Read Tests	0.5117	0.0014
64 KB Write Tests	0.6821	0.0000

Table 8: Correlation between Relative Performance and File System Free Space. This table shows the correlation between the fraction of disk space free and the relative performance of the test file systems for the 64 KB tests on the file systems in Figures 8 and 9. The correlation coefficient indicates the strength of the relationship; the square of this coefficient is the portion of the variation in relative performance that can be explained by a linear relationship with the fraction of the file system that is free. The P-value expresses the probability that there is no relationship.

tests, focusing on the 64 KB test cases. For each test the layout scores of the files created by the benchmark were computed, as were the aggregate layout scores of the original file systems and the fraction of file system blocks that were free in each test. Figure 8 and Figure 9 present this data.

File system utilization, the amount of space that is used on a file system, may have an effect of file layout. A highly utilized file system has few free blocks. Not only may these be poorly clustered, but it is possible that they will be widely scattered on the disk, imposing longer seek times on fragmented files. In several of the file systems in Figures 8 and 9 there is a clear relationship between changes in relative performance and changes in utilization. In both *cnews* and *white* this is especially apparent. Table 8 shows the results of a linear regression analysis of the data from tests on all nine of the file systems. Note that the correlation is strongest for the write tests. This reflects the overhead imposed by the synchronous meta-data writes that occur when a file is created (see Section 5.2). Given the skewed distribution of free space on active FFSs, it is likely that many of the files created by the benchmark will be allocated in the later portions of cylinder groups, farther from the locations of the meta-data which must be modified. On highly utilized file systems, these distances are likely to be greater, requiring longer seeks, and thus decreasing the performance of the benchmark.

Another condition that may effect the performance of an FFS is the amount of fragmentation in existing files. Most, if not all, of the free space in an FFS was once part of a file. When a file is removed, it creates new free space that has the same fragmentation as the file. Because the total free space in an FFS is the aggregation of many removed files, it may not be as fragmented as the individual files. Nevertheless, the fragmentation of existing files may be useful as a predictor of the fragmentation of free space, and hence of the performance of newly created files. Table 9 shows the results of a statistical analysis of the relationship between the aggregate layout score of a test file system, and the relative performance of the benchmark on that file system. The correlation, although weak, does suggest that there is a some relationship between these elements. The relative performance of the read tests shows a stronger correlation to aggregate layout score than the relative performance of the write tests. This is because of the additional overhead of synchronous meta-data updates incurred by the write tests (as explained at the end of Section 5.2), a factor that is unrelated to the fragmentation of a particular file.

Characterizing the workload on a file system is a much more complex task than computing its free space. Two basic characteristics of a file system's workload are the size of files that get created and deleted, the frequency with which these operations occur. Most of the file systems in this study have very similar workloads, since they primarily serve the home directories of various users. Other files systems, such as the */* and *usr* partitions on each machine, seldom change, and therefore do not present a workload that will cause fragmentation to change over the lifetime of the file system.

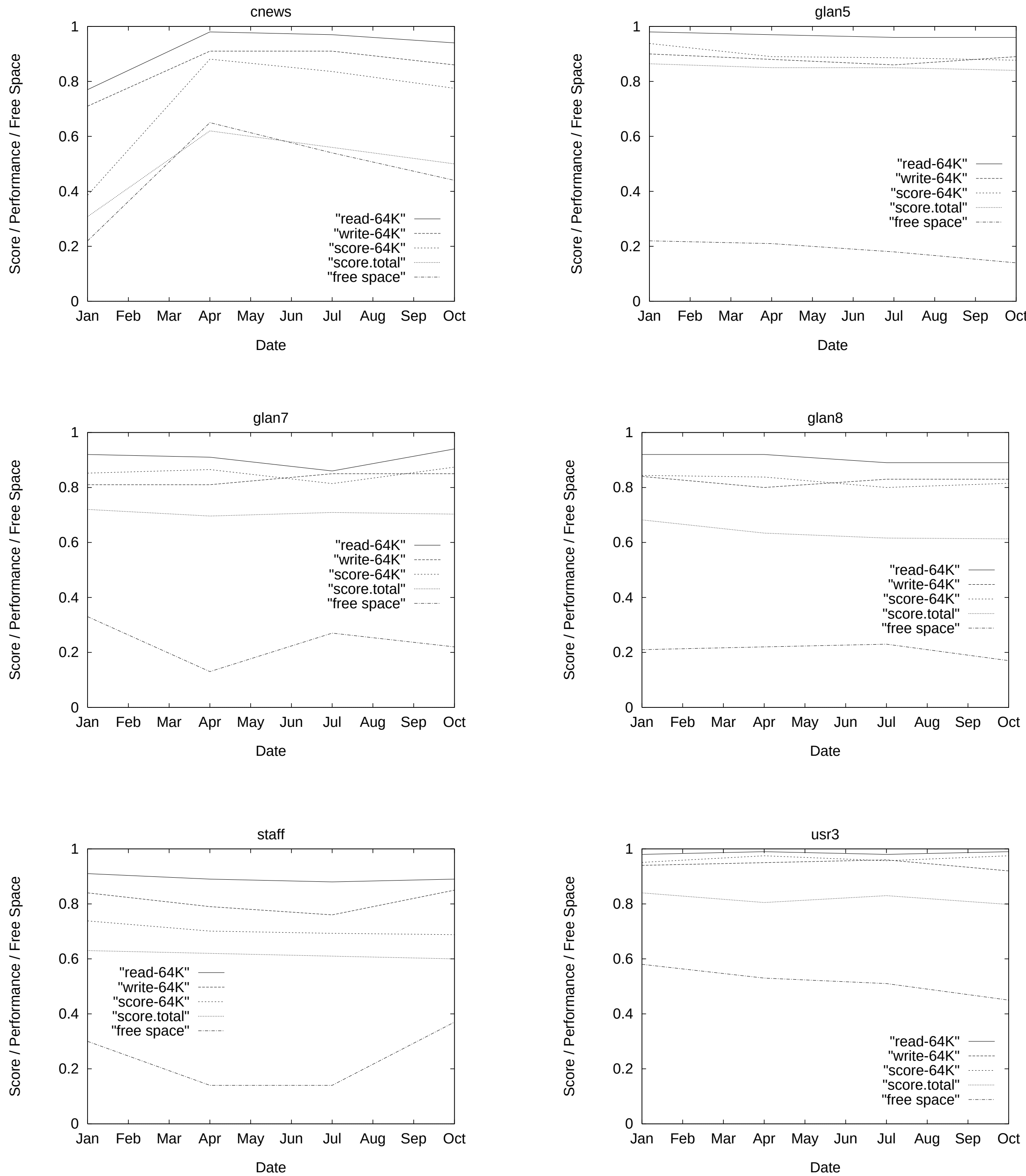

Figure 8: File Fragmentation and Performance (I). For each file system, the relative performance of the 64 KB read and write tests are plotted ("read-64K" and "write-64K") along with the layout score of the files created by the benchmark program ("score-64K"), the aggregate layout score for the entire file system ("score.total") and the fraction of the file system blocks that are free in the original file systems ("free space"). Relative performance is the ratio of throughput on the copied file systems to throughput on comparable empty file systems. Tests were performed on copied file systems from four dates, the 26th of January, April, July, and October of 1994. The final test data for *cnews* and *staff* was October 20, 1994.

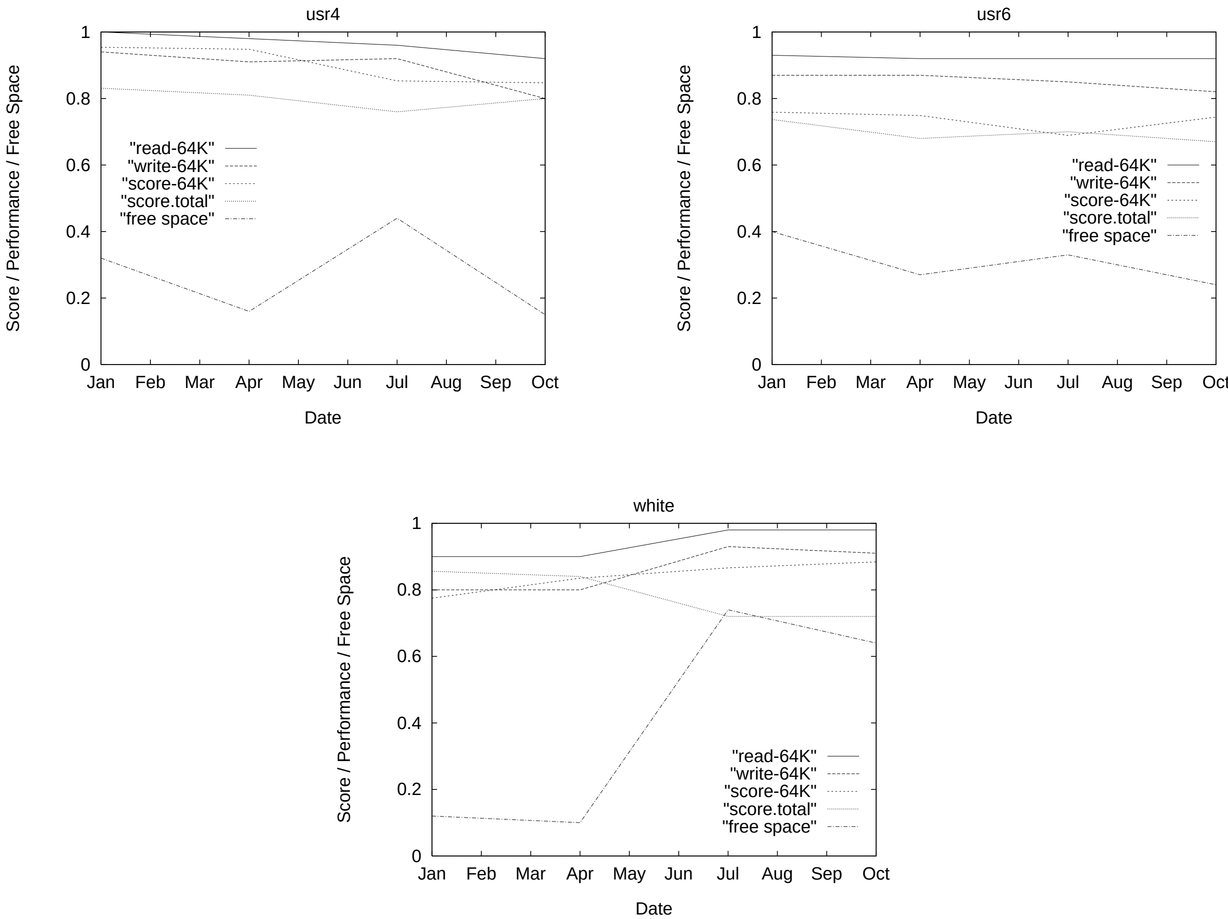

Figure 9: Fragmentation and Performance (II). For each file system, the relative performance of the 64 KB read and write tests are plotted ("read-64K" and "write-64K") along with the layout score of the files created by the benchmark program ("score-64K"), the aggregate layout score for the entire file system ("score.total") and the fraction of the file system blocks that are free in the original file systems ("free space"). Relative performance is the ratio of throughput on the copied file systems to throughput on comparable empty file systems. Tests were performed on copied file systems from four dates, the 26th of January, April, July, and October of 1994.

Test Set	Correlation Coefficient	P-value	Test Set	Correlation Coefficient	P-value
All Tests	0.7901	0.0000	64 KB Reads	0.8990	0.0000
64 KB Tests	0.7554	0.0000	64 KB Writes	0.8137	0.0000
1024 KB Tests	0.8490	0.0000	1024 KB Reads	0.9094	0.0000
Read Tests	0.9013	0.0000	1024 KB Writes	0.8989	0.0000
Write Tests	0.8357	0.0000			

Table 7: Correlation between Layout Score and Throughput. This table shows the correlation between the layout score of the files created by the tests discussed in Section 5.1 and the throughput measured by the same tests. Correlation was examined for all test cases, and for various subsets of the tests cases, as described in the "Test Set" column. The correlation coefficient indicates the strength of the relationship for each set of tests. The square of this coefficient is the portion of the variation in throughput that can be explained by a linear relationship to the layout score. The P-value expresses the probability that there is no relationship.

Tests	Correlation Coefficient	P-value
All 64 KB Tests	0.4682	0.0000
64 KB Read Tests	0.6435	0.0000
64 KB Write Tests	0.5118	0.0012

Table 9: Correlation between Relative Performance and Aggregate File System Layout Score. This table shows the correlation between the aggregate layout score of all of the files on a test file systems and the relative performance of the file system. This analysis was performed using data from the 64 KB tests on the files systems in Figures 8 and 9. The correlation coefficient indicates the strength of the relationship; the square of this coefficient is the portion of the variation in relative performance that can be explained by a linear relationship with the aggregate layout score. The P-value expresses the probability that there is no relationship.

Among the file systems used in the performance tests, *cnews* is an example of a file system with unique usage pattern. As a repository for net-news articles most of the files on this file system are extremely small, and there is a high rate of file turnover. This should cause its free space to rapidly become fragmented. In the snapshot of this file system from January 26, 1994, the file system had an aggregate layout score of 0.308. During the weeks following this snapshot, the layout score of *cnews* dropped to 0.201, at which point (February 8, 1994) *cnews* reported that it was out if disk space. An examination of the file system showed that although there were no free blocks, there were 92 megabytes of free space (sixteen percent of the file system)—all of it in fragments.

Not surprisingly, the January tests on *cnews* exhibited the greatest performance differences of any of the file systems. The single greatest performance difference in all of the tests was in the 64 KB write test on cnews. This test achieved only 71% of the empty file system performance. In February, after cnews ran out of disk space, a large amount of data was removed from it—hence the increase in free space (and the corresponding increase in layout score) on the later test dates.

The only other file system that had layout scores as low as *cnews* was the other news partition, *newsspool*. No other file system ever had a layout score lower than 0.6. It is reasonable to infer that the combination of high file turnover and small file size in the workloads on these two file systems causes this extreme fragmentation.

In contrast, *glan5* is a file system with very little file turnover. It is used to store the sources and binaries for various software packages. Files on this file system are created or deleted only when new software is installed or when existing software is upgraded. Not surprisingly, *glan5* performed better than the other file systems. Fifteen of the twenty-four tests on copies of *glan5* achieved 95–100% of the performance of the empty *glan5* file system.

5.4 Benchmark Shortcomings

Our benchmark program, like all other file system benchmarks, suffers from a variety of imperfections [6]. Consider the sequence of events when twenty-five eight block (64 kilobyte) files are allocated in the same directory. The first file will be allocated the first eight free blocks in the directory's cylinder group. The next file will be allocated the next eight free blocks, and so on. The net result is that the layout of these files will be the same as the layout of one 200 block file allocated in the same cylinder group.

A more "realistic" benchmark might intermingle file delete operations among the file creates, possible causing the newly created files to be more fragmented, as they fill in the holes left by deleted files instead of consuming less fragmented free space later in the cylinder group. It would certainly be possible to create an artificial benchmark that behaves in this manner. Such a benchmark could cause newly created files to by laid out as well, or as poorly as the designer wished. As there is not sufficient information about what the “right” amount of fragmentation would be, we felt that following this course would lead to a more complex benchmark that would not add a significant amount of verisimilitude to the testing.

Another approach to creating a more realistic benchmark would be to use the file system snapshots to more accurately reproduce the original file system. Instead of simply copying the free space bitmaps into each cylinder group, each file could have been reproduced, with the same blocks allocated to it as on the original file system. Performance could then be measured by reading and writing a selection of pre-existing files from the file system instead of creating new ones. There are a variety of complications in this approach. Which set of files would represent a realistic work load? How would a comparable set of file be selected on different file systems (or even from different snapshots of the same file system)? Even if a comparable set of files could be selected, what order would they be read in so that different inter-file seek distances would not dominate the performance differences caused by file fragmentation which we hoped to measure? Given these difficulties, we decided that this approach would add too much complexity without a compensating improvement in realism.

One useful aspect of our benchmark arises from the fact that the test file systems did not have any of their inodes allocated (except for the root directory’s inode). An empty FFS has exactly the same inodes allocated. Since the number of free inodes in a cylinder group is the major factor determining which cylinder group will be used for a new directory, both the empty file systems and the new file systems allocated directories in the same sequence of cylinder groups. Thus all tests on a given file system, whether empty or full, had a comparable set of seeks between cylinder groups, eliminating a possible source of unwanted performance difference.

We hope that in the future better technology will be available for performing file system benchmarks.

6 Summary

We have used data collected from FFSs that are in daily use to study the real world behavior of this widely used file system. We have made several observations about the layout of these file systems:

- The distribution of free space within cylinder groups is skewed towards the end of the cylinder groups.
- Small files are more fragmented than large files.
- Although some file systems show a tendency to become increasingly fragmented with the passage of time, there were a number of active file systems in our study which did not exhibit this increase in fragmentation, and some where fragmentation actually improved over time.

Studies of the performance on the file systems in our study indicate that performance is closely related to the fragmentation of the files being read and/or written. Both the amount of free space in a file system, and the fragmentation of other files in the file system have some value as predictors of the layout, and hence the performance, of newly created files.

Over the past ten months, we have collected a large amount of snapshot data from our file systems. This paper has presented a few simple conclusions and observations based on the analysis of that data. In the future we hope to conduct more extensive performance measurements and to examine a variety of other questions about file system layout.

7 References

[1] Marshall Kirk McKusick, William Joy, Sam Leffler, and R. S. Fabry, “A Fast File System for UNIX,” *ACM Transactions on Computer Systems,* Vol. 2, No. 3, August 1984, pp. 181–197.

[2] L. W. McVoy and S. R. Kleiman, “Extent-like Performance from a UNIX File System,” *Proceedings of the 1991 Winter Usenix Conference,* Dallas, TX, January 1991, pp. 33–44.

[3] J. Kent Peacock, “The Counterpoint Fast File System,” *Proceedings of the 1988 Winter Usenix Conference,* Dallas, TX, February 1988, pp. 243–249.

[4] J.M. Robson, “An Estimate of the Store Size Necessary for Dynamic Storage Allocation,” Journal of the ACM, Vol. 18, No. 3, July 1971, pp. 416–423.

[5] Margo Seltzer, Keith Bostic, Marshall Kirk McKusick, and Carl Staelin, “An Implementation of a Log-Structured File System for UNIX,” *Proceedings of the 1993 Winter Usenix Conference,* San Diego, CA, January 1993, pp. 307–326.

[6] Diane Tang and Margo Seltzer, “Lies, Damned Lies, and File System Benchmarks,” in *VINO: The 1994 Fall Harvest*, Harvard Division of Applied Sciences Technical Report tr-34-94, 1994.