Read Optimized File System Designs: A Performance Evaluation[1]

Margo Seltzer, Michael Stonebraker

Department of Electrical Engineering and Computer Science
University of California, Berkeley

Abstract

This paper presents a performance comparison of several file system allocation policies. The file systems are designed to provide high bandwidth between disks and main memory by taking advantage of parallelism in an underlying disk array, catering to large units of transfer, and minimizing the bandwidth dedicated to the transfer of meta data. All of the file systems described use a multiblock allocation strategy which allows both large and small files to be allocated efficiently. Simulation results show that these multiblock policies result in systems that are able to utilize a large percentage of the underlying disk bandwidth; more than 90% in sequential cases. As general purpose systems are called upon to support more data intensive applications such as databases and supercomputing, these policies offer an opportunity to provide superior performance to a larger class of users.

1. Introduction

Most current file systems can be divided into two distinct categories: fixed block systems and extent based systems. Fixed block systems allocate files as a collection of identically sized blocks while extent based systems allocate files as a collection of a few large extents whose sizes may vary from file to file. Traditionally, time-sharing oriented systems (e.g. UNIX) have used fixed block systems, while database oriented operating systems (e.g. MVS) have chosen extent based systems. Fixed block file systems have received much criticism from the database community. The most frequently cited criticisms are discontiguous allocation and excessive amounts of meta data [STON81]. Extent based file systems are often criticized for being too brittle with regard to fragmentation and too complicated in terms of allocation.

IBM's MVS system provides extent based allocation allowing users to specify extent sizes for each file [IBM]. If the user specifies sizes wisely, the file system allocates most files in a few large contiguous extents, and there is little wasted space on the disk. However, if extent sizes are chosen poorly, both external and internal disk fragmentation can greatly reduce the efficiency of the disk system. In addition, managing free space and finding extents of suitable size can become increasingly complex as free space becomes fragmented. Frequently, background disk rearrangers must be run (during off peak hours) to coalesce free blocks.

Fixed block systems, such as the original UNIX V7 file system [THOM78] solve the problems of keeping allocation simple and fragmentation to a minimum, but they do so at the expense of efficient read and write performance. In this system, files are composed of some number of 512 byte blocks. Free blocks are maintained on a free list and allocated off the head of this list. Unfortunately, as file systems age, logically sequential blocks within a file get distributed across the entire disk, and a disk seek is required to retrieve each 512 byte block.

The BSD Fast File System (FFS) is an evolutionary step from the simple fixed block system. Files are composed of a number of fixed sized ''blocks'' and a few smaller ''fragments''. In this way, tiny files may be composed of fragments, thus avoiding excessive internal fragmentation. At the same time, the larger block size (usually on the order of 4K or 8K) allows more data to be transferred per disk seek. Furthermore, allocation is performed in a rotationally optimal fashion so that successive blocks of the same file are retrieved during a single rotation [MCKU84].

Even on systems such as the FFS, commercial database vendors usually choose to implement their own file system on a raw disk partition [SYB87]. In this way, they can guarantee physical contiguity within blocks of a file. The drawback to such a mechanism is that it requires a static partitioning between the database files

[1] Appeared in the *Proceedings of the Seventh Internation Conference on Data Engineering*, Kobe, Japan, April 1991.

and all other files on the system. If either grows unexpectedly, this partitioning will prove unacceptable.

Another optimized UNIX file system design presented in [STON89] is designed to provide large transfers from an array of disks. Files are allocated to sequential blocks within striped tracks (a track on each disk of an array) and read-ahead and write-behind are used to achieve full stripe reads and writes. The resulting allocation for large files is similar to that in an extent based system where a file is composed of a few large contiguous extents. This is the extent based policy simulated in this study.

In an attempt to merge the fixed block and extent based policies, Koch designed a multiblock file system using binary buddy allocation [KOCH87]. Files are composed of a fixed number of extents, each of whose size is a power of two (measured in sectors). Files grow by doubling their size at each allocation. Periodically (daily in the DTSS system described) a reallocation algorithm runs. This reallocator rearranges extents to reduce both the internal and external fragmentation. Using this combination, most files are allocated in 3 extents and average under 4% internal fragmentation. This policy (without the reallocator) is also simulated in this study.

The goal of this study is to analyze how well different allocation policies perform on an array of disks without the use of an external reallocation process. In designing a file system for a disk array, the utility of large blocks increases. Not only does a larger block size provide more data transferred per disk seek, but it allows the file system to stripe across the disks, exploiting parallelism in the underlying disk system. Since we wish to support both large and small files, the file system must support a range of block sizes. Small blocks are required to provide reasonable fragmentation for small files, and large blocks or contiguous allocation are required to support high data throughput for large files. In this paper, we compare the performance of three read optimized file systems in terms of fragmentation and disk system throughput. We call these read optimized file systems in that they are optimized for reading entire files sequentially from disk as opposed to write optimized file systems which use logging techniques to optimize writes to one or more files [FINL87] [HAGG87] [HAS88] [ROSE90].

2. The Simulation Model

These allocation policies were analyzed by means of an event driven, stochastic workload simulator. There are three primary components to the simulation model: the disk system, the workload characterization and the allocation policies. The disk system and workload characterization are described in Sections 2.1 and 2.2, while the allocation policies are described in detail in Section 4.

2.1. The Disk System

The disk system is an array of disks, viewed as a single logical disk, with no redundancy or parity information. Blocks are numbered so that data written in a large contiguous unit to the logical disk will be striped across the physical disks. When data is striped across disks, there are two parameters which characterize the layout of disk blocks, the **stripe unit** and the **disk unit**. The stripe unit is the number of bytes allocated on a disk before allocation is performed on the next disk. This unit must be greater than or equal to the sector sizes of all the disks. The disk unit is the minimum unit of transfer between a disk and memory. This is the smaller of the smallest block size supported by the file system and the **stripe unit**. Disk blocks are addressed by disk units.

Each disk is described in terms of its physical layout (track size, number of cylinders, number of platters) and its performance characteristics (rotational speed and seek parameters). The seek performance is described by two parameters, the one track seek time and the incremental seek time for each additional track. If ST is the single track seek time and SI is the incremental seek time, then an N track seek takes $ST + N*SI$ ms. Table 1 contains a listing of the parameters which describe one common disk and its default values in these simulations.

2.2. Workload Characterization

The workload is characterized in terms of file types and their reference patterns. A simulation configuration may consist of any number of file types, as defined by their size characteristics, access patterns, and growth characteristics. Table 2 summarizes those parameters which define a file type.

For each file type, initialization consists of two phases. In the first phase, *nusers* events are created, and each is assigned a start time uniformly distributed in the range [0, (*nusers* * *hfreq*)]. The events are maintained

Disk Parameters For the CDC 5 ¼" Wren IV Drives (94171-344)		
	actual	simulated
N Disks	NA	8
Total Capacity	NA	2.8 G
Max Throughput	NA	10.8 M/sec
N Platters	9	9
N Cylinders	1549	1600
Bytes/Track	24 K	24 K
1 Track Seek Time	5.5 ms	5.5 ms
Inc. Seek Time	0.0320 ms	0.0320 ms
Rotation Time	16.67 ms	16.67

Table 1: Parameters and Default Values

sorted by their scheduled time. During the second phase, the files are created. For each file a size is selected from a normal distribution with mean *i_size* and deviation *i_dev*. Allocation requests are issued until the allocation length of the file is greater than or equal to this size.

The simulation runs by selecting the first event, processing that request, and scheduling a new request. Since each event corresponds to a file type, the type of operation can be derived from the *r_ratio*, *w_ratio*, *e_ratio*, and *d_ratio*. The size of the operation is determined by the *rw_size*, *rw_dev*, and *t_size* parameters. After the operation is completed, an exponentially distributed value with mean equal to *ptime* is added to the time at which the operation completed and an event is scheduled at that newly calculated time.

If an allocation request cannot be satisfied, a disk full condition is logged, and the current event is rescheduled (according to the *ptime* parameter as above). If the test being run is an allocation test, the simulation ends. For throughput simulations, two parameters are used to maintain a level of disk utilization. The lower bound, N, indicates how full the disk system should be before measurements begin. The upper bound, M, indicates the maximum utilization allowed during the simulation. Any extend operation occurring when the disk utilization is greater than M is converted to a truncate operation. In this way, the disk utilization is kept between N and M while measurements are being taken.

A simulation may be terminated by a disk full condition. It may also be terminated by two other conditions, either a specified number of milliseconds have been simulated or the throughput of the system has stabilized. The throughput, measured as a percentage of the maximum possible sequential throughput of the disk system, is considered stabilized when the throughput calculations for 3 consecutive 10 second intervals are within .1% of each other.

File Parameters	
nfiles	Number of files created
nusers	Number of parallel events
ptime	Milliseconds between requests from a single user
hfreq	Milliseconds between requests from different users
rw_size	Mean size of each read/write operation
rw_dev	Standard deviation in read/write size
a_size	For extent based systems, mean extent size
t_size	Size of deallocate requests
i_size	Mean initial file size
I_dev	Deviation in the mean file size.
r_ratio	Percent operations which are reads.
w_ratio	Percent operations which are writes.
e_ratio	Percent operations which are extends.
d_ratio	Percent deallocates which are file deletes.

Table 2: File Parameters and Description

This study uses 3 workloads to simulate a time-sharing or software development environment (TS), a large transaction processing environment (TP), and a supercomputer or complex query processing environment (SC).

The time-sharing workload is characterized by an abundance of small files (mean size 8K bytes) which are created, read, and deleted. Two-thirds of all requests are to such files, while the remaining one-third are to larger files (mean size 96K). The large files are usually read (60% of all requests) and occasionally extended, written or truncated (15% writes, 15% extends, 5% deletes and 5% truncates).

The transaction processing environment is characterized by 10 large files (210M) representing data files or relations, 5 small application logs (5M), and one transaction log (10M). The relations are read and written randomly (60% reads, 30% writes), and infrequently extended and truncated (7% extends, 3% truncates). The log files are usually extended (93% and 94% for application and system logs respectively), and infrequently read and truncated (2% and 5% read, 5% and 1% truncated). The system log receives a slightly higher read percentage to simulate transaction aborts.

The supercomputer environment is characterized by 1 large file (500M), 15 medium sized files (100M), and 10 small files (10M). The large and medium files are all read and written in large, contiguous bursts (32K or 512K) with a predominance of reads (60% reads, 30% writes, 8% extends, and 2% truncates). The small files are read and written in 32K bursts, but are periodically deleted and recreated (60% reads, 30% writes, 5% extends, 5% deletes).

3. Evaluation Criteria

The two evaluation criteria for each policy are disk utilization and throughput. The metrics for measuring disk utilization are the external fragmentation (amount of space available when a request cannot be satisfied) and internal fragmentation (the fraction of allocated space which does not contain data). The allocation tests are run by performing only the extend, truncate, delete, and create operations in the proportion expressed by the file type parameters. As soon as the first allocation request fails, the external and internal fragmentation are computed.

The metrics for throughput are expressed as a percent of the sustained sequential performance of the disk system. For example, the configuration shown in Table 1

is capable of providing a sustained throughput of 10.8 Mbytes/sec. A throughput of 1.1 Mbytes/sec is expressed as 10% of the maximum available capacity.

Throughput is calculated for two sets of tests, the application performance test and the sequential performance test. For both tests, lower and upper bounds on disk utilization are set at 90% and 95% respectively. For the application performance test, the application workloads described in Section 2.2 are applied. For the sequential test, files are read and written in their entirety.

4. The Allocation Policies

This analysis considers three different allocation policies. The first is a buddy system similar to that described in [KOCH87]. The next is a restricted buddy system which supports only a few different block sizes. The last is the extent based policy described in [STON89]. Each design is described in more detail, including a discussion of the selection of the parameters relevant to each model.

4.1. Buddy Allocation

The buddy allocation policy described in [KOCH87] includes both an allocation process and a background reallocation process that runs during off peak hours. In this simulation, we consider only the allocation and deallocation algorithm (i.e. not the background reallocation). A file is composed of some number of extents. The size of each extent is a power of two multiple of the sector size. Each time a new extent is required, the extent size is chosen to double the current size of the file.

Workload	Disk Usage	
	Internal Fragmentation (% allocated space)	External Fragmentation (% total space)
SC	43.1%	13.4%
TP	15.2%	9.0%
TS	18.4%	2.3%

Table 3a: Fragmentation for Buddy Allocation.

Workload	Throughput	
	Application Performance (% max throughput)	Sequential Performance (% max throughput)
SC	88.0%	94.4%
TP	27.7%	93.9%
TS	8.4%	12.0%

Table 3b: Performance for Buddy Allocation.

As previous work suggests [KNOW65][KNUT69], such policies are prone to severe internal fragmentation, and our simulation results confirm this (as shown in Table 3a). However, the small number of extents results in very high throughput in the presence of large files as is evidenced by the percent utilization (shown in Table 3b) for the supercomputer workload (SC).

4.2. Restricted Buddy System

As in the buddy system, the restricted buddy system uses the principal that a file's block size grows as the file's size grows. Small files are allocated from small blocks and don't exhibit severe internal fragmentation. Large files are composed of large blocks allowing large sequential transfers between the disk system and main memory. In addition, logically sequential disk blocks within a file are allocated contiguously whenever possible. Therefore, even though files may start out with small allocations, it is still possible to perform large transfers of data with a single disk seek when these small blocks are laid out contiguously. Finally, the disk can be divided into regions so that blocks within a single file may be clustered to reduce the seek time when sequential layout is not possible.

The disk system is addressed as a linear address space of disk units. Each block size is an integral multiple of the disk unit and of all the smaller block sizes. In order to keep allocation simple, a block of size N always starts at an address which is an integral multiple of N. If a system supports block sizes of 1K and 8K, the 1K blocks located at addresses 0 through 7 are considered buddies, together forming a block of size 8K. Whenever possible, buddies are allocated sequentially to the same file and are coalesced at deallocation.

The parameters which define a file system in the restricted buddy policy are the number of block sizes, the specific sizes, when to change block sizes (the grow policy), and whether or not to attempt to cluster allocations for the same file. We consider four different sets of block sizes, two different algorithms for choosing when to increase the block size, and both clustered and unclustered policies. The four block size configurations are:

Number of Block Sizes	Block Sizes
2	1K, 8K
3	1K, 8K, 64K
4	1K, 8K, 64K, 1M
5	1K, 8K, 64K, 1M, 16M

For each set of block sizes, we consider both a clustered configuration, where the disk system is broken up into 32M regions, and an unclustered configuration. The goal in selecting regions and blocks is to select a block that is conveniently close to related blocks (either meta data or

the previously allocated block within the same file).

The grow policy determines when we change the size of the allocation unit and is expressed in terms of a multiplier. If g is the grow policy multiplier and the block sizes are a_i, then the unit of allocation increases from a_i to a_{i+1} when the sum of the sizes of all blocks of size a_i is equal to $g * a_{i+1}$. For example, a system with block sizes 1K and 8K and a grow policy multiplier (grow factor) of 1 will allocate eight 1K blocks before allocating any 8K blocks. If the next larger block size were 64K, then eight 8K blocks would be allocated before growing the block size to 64K. Intuitively, we expect that a smaller grow factor will suffer worse internal fragmentation (since we use bigger blocks in smaller files), but might offer better performance (since fewer small block transfers are required). However, if the small blocks are allocated contiguously, then the performance should be similar.

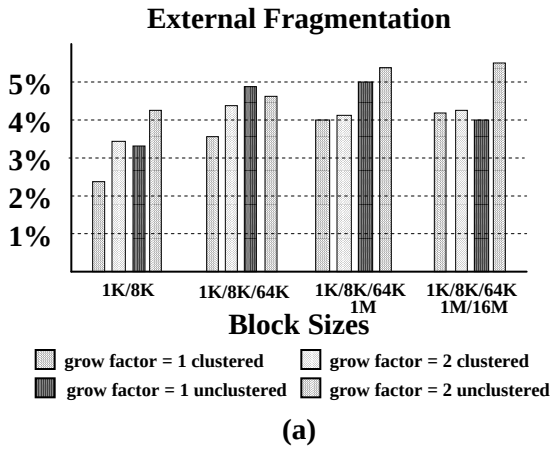


Figure 1a: External Fragmentation for the time-sharing workload using restricted buddy allocation.

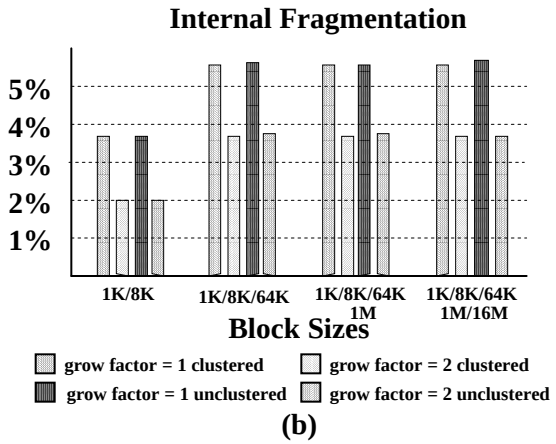


Figure 1b: Internal Fragmentation for the time-sharing workload using restricted buddy allocation.

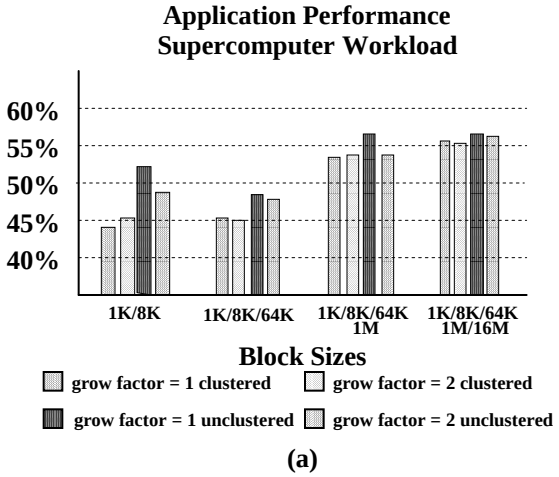
The allocation and throughput tests were run on all the configurations described in Section 2.2. The most striking result is that the attempt to coalesce free space and maintain large regions for contiguous allocation are successful. None of the polices produce either internal or external fragmentation greater than 6%. As expected, the time-sharing workload which has the blend of large and small files exhibits the greatest fragmentation (Figures 1a and 1b), and fragmentation increases as the number of blocks sizes and the block sizes themselves increase. Increasing the grow factor from one to two reduces the internal fragmentation by approximately one-third (in Figure 1b, note the difference between each pair of adjacent bars). External fragmentation increases slightly as we go to an unclustered configuration since a larger selection of blocks are eligible for splitting (all blocks in the disk system instead of just those in a specific region).

Figures 2a-2f show the results of the application and sequential tests for the three workloads under each configuration of the restricted buddy policy. As expected, the configurations which support the larger block sizes provide the best throughput, particularly where large files are present (Figures 2a, 2b, 2c, and 2d). The supercomputer application in Figure 2a shows up to 25% improvement for configurations with large blocks while the transaction processing environment shows an improvement of 20%. These same workloads are relatively insensitive to either the grow policy or clustering. For the five block size configuration (the rightmost on each graph), most show slightly better performance with an unclustered configuration. The explanation of this phenomena lies in the movement of files between regions. In a clustered configuration, when a change of region is forced, the location of the next block is random with regard to the previous allocation. In an unclustered configuration, the attempt to keep subsequent allocations contiguous results in the improved performance.

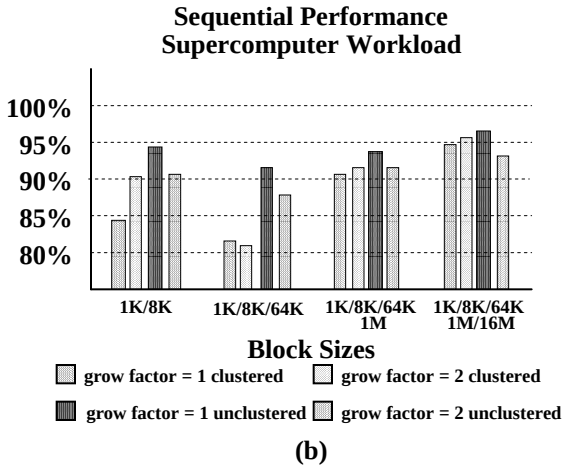
The time-sharing workload reflects the greatest sensitivity to the clustering and grow policy. Uniformly, clustering tends to aid performance, by as much as 20% in the sequential case (in Figure 2f, the first two bars of each set represent the clustered configuration and the third and fourth bars represent the unclustered configuration). Since this environment is characterized by a greater number of smaller files, data is being read from disk in fairly small blocks even with the larger block sizes. As a result, the seek time has a greater impact on performance, and the clustering policy which reduces seek time provides better throughput.

Figure 2f indicates that the higher grow factor provides better throughput (the second and fourth bars in

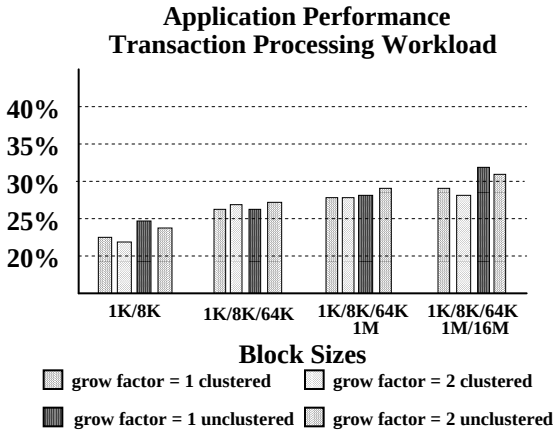
Figure 2



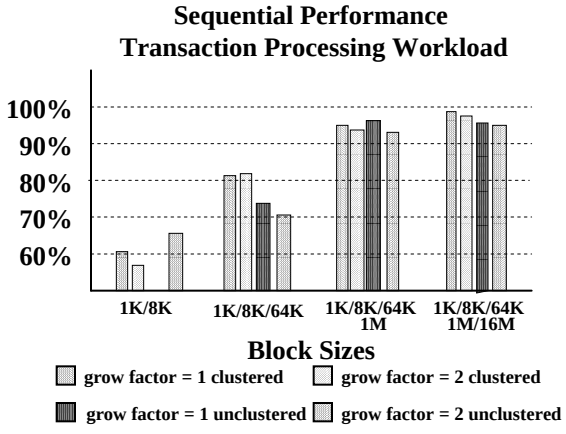
(a)



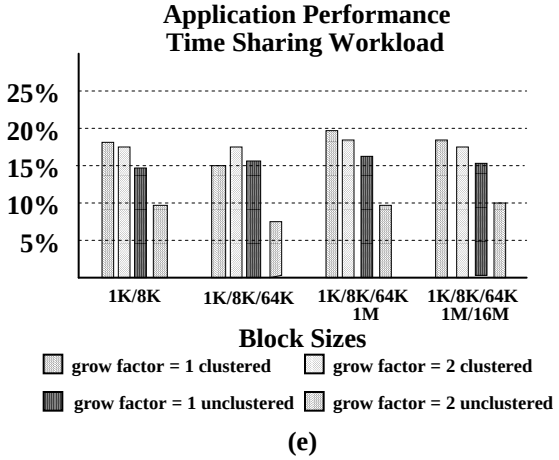
(b)



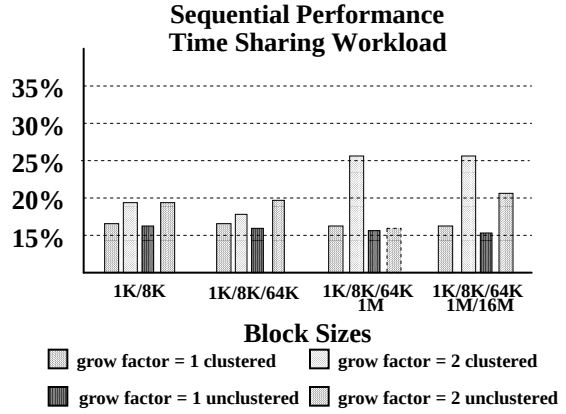
(c)



(d)



(e)



(f)

each set represent a grow factor of 2 while the first and third bars represent a grow factor of 1). This is counter intuitive since a higher grow factor means that more small blocks are allocated. To understand this phenomena, we need to analyze how the attempt to allocate blocks sequentially interacts with the grow policy. Figure 3 shows a 1M block that is subdivided into sixteen 64K blocks, each of which may be subdivided into eight 8K blocks. When the grow factor is 1, any file over 72K requires a 64K block. However, when it is time to acquire a 64K block, the next sequential 64K block is not contiguous to the blocks already allocated. In contrast, when the grow factor is two, the 64K block isn't required until the file is already 144K. Since most files in the timesharing workload are smaller than this, they never pay the penalty of performing the seek to retrieve the 64K block. Thus our grow policy and our attempts to lay out blocks sequentially are in conflict with one another, and the grow policy should be modified to allow contiguity between different sized blocks.

Using the results of this section, we select a configuration for comparison with the other allocation policies. Since the larger blocks sizes did not increase fragmentation significantly, we select the 5 block size configuration (1K, 8K, 64K, 1M, 16M) which is the rightmost group on each graph. Clustering had little effect on the large file environments and improved performance in the time-sharing environment, so we select a clustered configuration. In four of the six cases, the grow factor of 1 provided better throughput than the grow factor of 2, so we select that, with the understanding that it will penalize sequential performance for the time-sharing workload. This configuration is represented by the leftmost bar in the rightmost group of each graph.

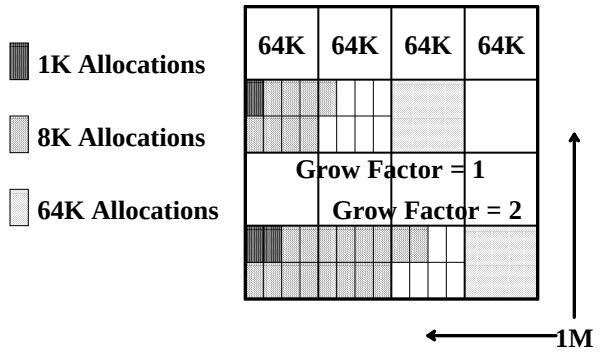


Figure 3: How contiguous allocation and grow factors interact. Because the total file length is not a multiple of the new block size, we are required to pay a seek when the block size grows.

4.3. Extent Based Systems

In the extent based models, every file has an extent size associated with it. Each time a file grows beyond its current allocation, additional disk storage is allocated in units of an extent. As in the restricted buddy policy, we view the disk system as a linear address space. However, in this model, an extent may begin at any address. When an extent is freed, it is coalesced with its adjoining extents if they are free.

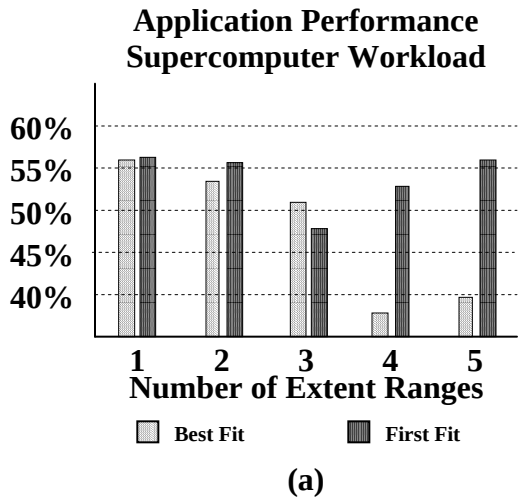
The parameters which define a file system in the extent-based system are the allocation policy and the variance in the sizes of the extents. We consider two different allocation policies, first-fit and best-fit with five different extent configurations. Each extent configuration is characterized by the number of extent size ranges. An extent size range is a normal distribution with a standard deviation of 10% of the mean. For example, an extent range around 1M with 1K disk units would produce a normal distribution of extent sizes with mean 1M and standard deviation of 102K. The extent ranges simulated are as follows:

Workload	Number of Ranges	Range Means
TS	1	4K
	2	1K, 8K
	3	1K, 8K, 1M
	4	1K, 4K, 8K, 1M
	5	1K, 4K, 8K, 16K, 1M
TP/SC	1	512K
	2	512K, 16M
	3	512K, 1M, 16M
	4	512K, 1M, 10M, 16M
	5	10K, 512K, 1M, 10, 16M

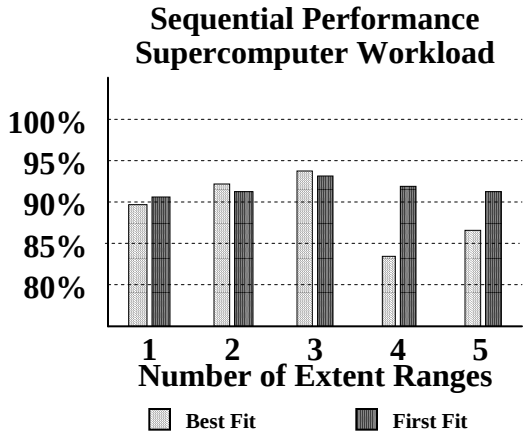
As the number of extent ranges increases, one expects to see increased fragmentation since we are allocating a more diverse set of extent sizes, but the results do not support this. Instead, across all extent ranges, both internal and external fragmentation is below 4%, independent of the number of extent ranges. One likely explanation is that the ratio of large files to small files is constant in these simulations. As a result, once large extents are allocated, they do not become fragmented later, because requests for small extents may be satisfied by already fragmented blocks. This also explains why best fit consistently resulted in less fragmentation.

We expect throughput to be insensitive to the selection of best fit or first fit since in both cases, files are read in the same size unit. Figures 4a-4f show the application and sequential performance results for the extent based polices and validate this intuition. In general, we

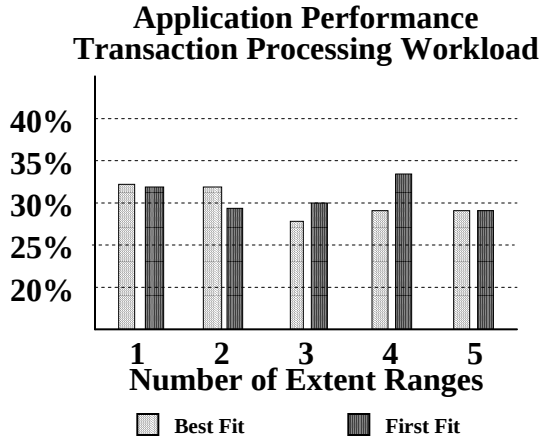
Figure 4



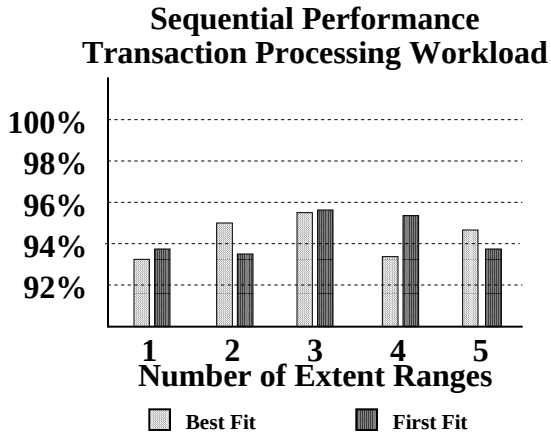
(a)



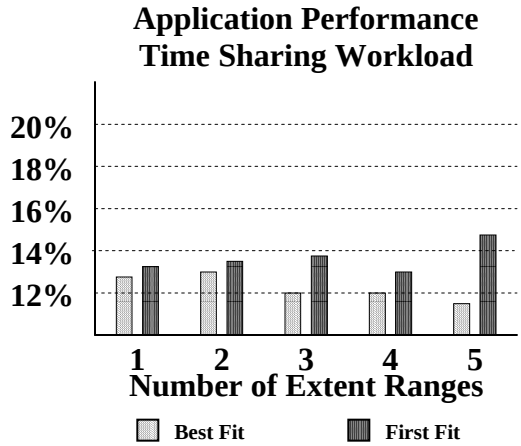
(b)



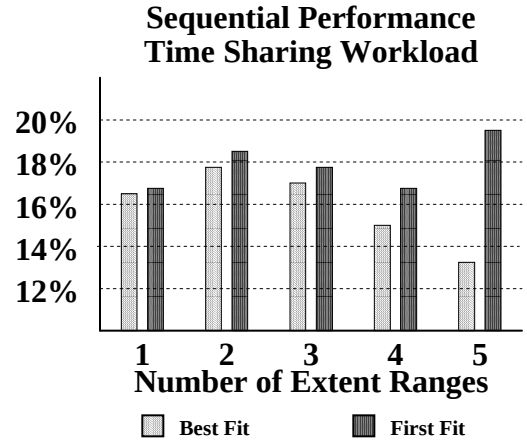
(c)



(d)



(e)



(f)

see better performance from first fit, due to the clustering that results from the tendency to allocate blocks toward the ‘‘beginning’’ of the disk system.

In order to understand the small changes in performance, we need to look at the average number of extents per file for the different workloads and extent ranges. These numbers are summarized in Table 4. We expected to see the best sequential performance when the average number of extents per file is a minimum since the fewest seeks are performed. The supercomputer and transaction processing workloads behave as expected (Figures 4b and 4d) while the time-sharing workload does not exhibit this tendency. Further inspection indicates that the ratio of small to large files alters this result. Since most of the files in the time-sharing environment are small, they can be allocated in one or two 4K extents. The larger files require 24 extents (96K files with 4K extents). However, the larger files consume more disk space and take longer to read and write. As a result, the time spent processing large files is greater than the time spent processing small files. Therefore, in the configurations where the large files have fewer extents (12 extents in the systems that use 8K extents for these files), the overall throughput is higher.

In selecting the configuration to compare in Section 5, we select the first fit allocation policy since it consistently provides better performance than best fit. For the transaction processing and supercomputer workloads simulated, the 3 range size configuration results in the highest sequential performance. Although it does not offer the best performance for the timesharing workload, it is within 10% of the best performance. This configuration is represented by the righthand bar in the middle group of each graph.

5. Comparison of Allocation Policies

As we’ve seen in the preceding sections, all the allocation policies except for the buddy system yield satisfactory fragmentation. As a result we focus on the application and sequential performance. We compare all the performance numbers to 4K and 16K fixed block systems which do not bias toward automatic striping or contiguous layout. The 4K system is compared with the timesharing workload while the 16K is compared to the transaction processing and supercomputer workloads.

Average Number of Extents Per File			
Number of Extent Ranges	SC	TP	TS
1	162	267	5
2	124	13	9
3	97	12	9
4	151	14	7
5	162	108	6

Table 4: Average number of extents per file for each extent based configuration.

Figure 5a shows the sequential performance of the four methods (the three discussed in Section 4 and a fixed block policy). As expected, all of the multiblock policies perform better than the fixed block policy due to the ability to read and write very large contiguous blocks. On the large file applications (SC and TP) we find that all the large block policies achieve close to the maximum

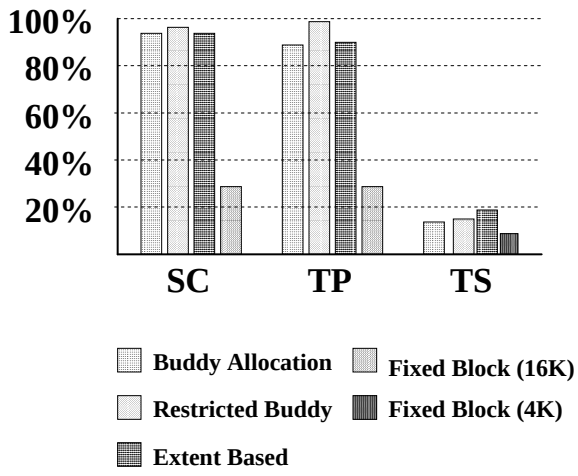


Figure 5a: Sequential Performance of the different allocation policies.

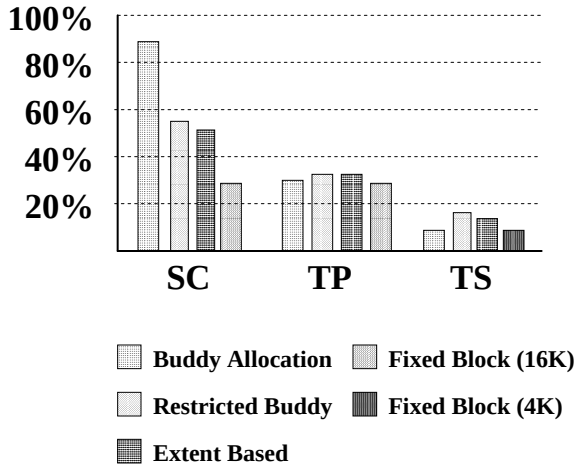


Figure 5b: Application Performance of the different allocation policies.

throughput. In the time-sharing environment, none of the policies succeed in pushing the system above 20% utilization due to the presence of many small files. However, the extent based policy can respond to this burden most effectively since each file is limited to a small number of extents.

In the application performance (Figure 5b), we find similar results. However, there are two points to note. First, in the supercomputer environment, the buddy system performs substantially better since, for large files (over 100M), it is using substantially larger block sizes (64M). In the transaction processing environment, all the policies are limited by the random reads and writes to the large data files.

6. Conclusions

We can see that allocation policies which force striping across disks and contiguous allocation provide improved performance over those which do not. In the large file environments such as supercomputer applications, this improvement is on the order of 250%. Even for workloads like the transaction processing environment, which are dominated by small reads and writes to large files, there is some improvement (10%). While the large blocks do not benefit the small file environment greatly, they do not hinder it either in terms of performance or fragmentation. Therefore in systems with both extremely large and extremely small files we are likely to be able to derive this improved performance without handicapping the small file efficiency.

This result suggests that time-sharing environments could benefit significantly from these allocation techniques. Such systems could then effectively compete with larger systems designed with database or supercomputer applications in mind without hindering the small file performance.

There are several more areas that warrant further investigation. First, varying the file distributions so that the proportion of large and small files is not constant may affect fragmentation results. Secondly, the impact of a RAID in the underlying disk system will reduce the small write performance. In order to correct for this, block sizes and extent sizes may need to be selected based on the configuration of the RAID. In the small file environment we might want to incorporate policies from a log structured file system to allocate blocks [ROSE90]. The different policies may show different sensitivities to the stripe size parameter. And as always, applying the allocation policies to genuine workloads will yield a much more convincing result.

7. Bibliography

[IBM] *MVS/XA JCL User's Guide*, International Business Machines Corporation, chapter 15, pp. 15-29.

[FINL87] Finlayson, R., Cheriton, D., "Log Files: An Extended File Service Exploiting Write-Once Storage", *Proceedings 11th Symposium on Operating System Principles*, November 1987.

[HAGG87] Haggman, R., "Reimplementing the Cedar File System Using Logging and Group Commit," *Proceedings 11th Symposium on Operating System Principles*, 1987.

[HAS88] Haskin, R., Malachi, Y., Sawdon, W., Chan, G., "Recovery Management in Quicksilver", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February 1988.

[KNOW65] Knowlton, K.D., "A Fast Storage Allocator," *Communications of the ACM*, October 1965, pp. 623-625.

[KNUT69] Knuth, D., *The Art of Computer Programming*,Vol 1, *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1969, pp. 442-445.

[KOCH87] Philip D. L. Koch, "Disk File Allocation Based on the Buddy System", ACM Transactions on Computer Systems, Vol. 5, No. 4, November 1987, pp. 352-370.

[MCKU84] 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.

[PATT88] Patterson, D. et. al., "RAID: Redundant Arrays of Inexpensive Disks," Proc. 1988 ACM-SIGMOD Conference on Management of Data, Chicago, IL, June 1988.

[POST88] Poston, Alan, "A High Performance File System for UNIX," NASA Ames, June 1988.

[ROSE90] Rosenblum, M., Ousterhout, J. K., "The LFS Storage Manager", *Proceedings of the 1990 Summer Usenix*, Anaheim, CA, June, 1990.

[STON81] Stonebraker, M., "Operating System Support for Database Management", *Communications of the ACM*, July, 1981, pp. 412-418.

[STON89] Stonebraker, M., Aoki, P., Seltzer, M., "Parallelism in XPRS," Electronics Research Laboratory, University of California, Berkeley, CA, Report M89/16, February 1989.

[SYB87] *Sybase Administration Guide*, Sybase Corporation, 1987.

[THOM78] Thompson, K., "Unix Implementation," *Bell Systems Technical Journal*, 57, 6, part 2, July-August 1978, pp. 1931-1946.