READ OPTIMIZED FILE SYSTEM DESIGNS: A PERFORMANCE EVALUATION

by

Margo Seltzer and Michael Stonebraker

Memorandum No. UCB/ERL M92/64

5 June 1992

READ OPTIMIZED FILE SYSTEM DESIGNS: A PERFORMANCE EVALUATION

by

Margo Seltzer and Michael Stonebraker

Memorandum No. UCB/ERL M92/64

5 June 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

READ OPTIMIZED FILE SYSTEM DESIGNS: A PERFORMANCE EVALUATION

by

Margo Seltzer and Michael Stonebraker

Memorandum No. UCB/ERL M92/64

5 June 1992

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Read Optimized File System Designs: A Performance Evaluation

Margo Seltzer, Michael Stonebraker

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

This paper presents a performance comparison of several allocation policies for producing read optimized file systems. 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. Our goal is to provide efficient support for large files without sacrificing small file efficiency, either in terms of disk capacity or read/write performance. All of the file systems described use a multiblock allocation strategy. That is, rather than supporting a fixed block size, the file systems support multiple block sizes. In this way, both large and small files may be allocated efficiently.

Simulation results show that such multiblock policies result in systems that are able to utilize a large percentage of the underlying disk bandwidth; more than 90% in sequential cases and 20-80% under application specific workloads. Furthermore, this high throughput is attainable without suffering poor disk utilization due to internal and external disk fragmentation. As general purpose systems, which have traditionally catered to small files and a time sharing environment, are called upon to satisfy larger and more data intensive applications such as databases and supercomputing, these multiblock policies offer an opportunity to provide superior performance to a larger class of users.

1. Introduction

Most current file systems may be divided into two distinct categories: fixed block systems and extent based systems. Traditionally, database oriented operating systems (e.g. MVS) have chosen extent based systems while time sharing oriented systems (e.g.UNIX) have used fixed block systems. Fixed block size 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]. On the other side of these debates, extent based file systems are often considered too brittle with regard to fragmentation and too complicated in terms of ease 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, their file systems allow most files to be stored 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 systems. In addition, managing the free space and finding extents of suitable size can become increasingly complex as free space becomes more and more fragmented. Frequently, background disk rearrangers need to 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 bytes 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 spread across the entire disk, and file system ends up requiring a disk seek to retrieve every 512 bytes of data.

The BSD Fast File System 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 8K or 16K) used for most files, allows more data to be transferred for each seek of the disk. Furthermore, allocation is performed in a rotationally optimal fashion so that successive blocks of the same file may be retrieved during a single rotation. [MCKU84]

Even on optimized systems such as this, 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 and all other files on the system. If either grows

unexpectedly, this partitioning may 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 disk 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 multi-block 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 (once every day in the DTSS system described) a reallocation algorithm runs. This reallocator shuffles extents around 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 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 also allows the file system to force striping across the disks, guaranteeing the ability to exploit parallelism in the underlying disk system. Since we wish to support both large and small files, it becomes apparent that the file system must support a range of block sizes. Small blocks are required to provide reasonable fragmentation for small files, but large blocks or contiguous allocation are required to support high data throughput for large files. Our allocation policies strive to provide contiguous allocation as well as disk striping.

This study introduces three allocation policies. Each of the policies optimizes for reading over writing in that the emphasis is placed on allocating files contiguously so that maximum per-

formance is attained on sequential operations. We call such systems, read optimized, in contrast to log structured file systems which optimize for writes [ROSE90]. In this paper, we compare the performance of these three read optimized file systems in terms of fragmentation and disk system throughput. The rest of this paper is organized as follows. First we present the simulation model and establish the evaluation criteria. Next, we present the different allocation policies, and the simulation results that characterize each. Finally, we present a comparison of the different policies.

2. The Simulation Model

We analyze these allocation policies 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 below while the allocation policies are described in detail in Section 4.

2.1. The Disk System

The disk system is designed to allow multiple heterogeneous devices. The disks may be configured as an array of disks (where multiple disks are addressed a single logical disk allowing data to be striped across multiple disks), a set of mirrored disks (where all data is stored on two identical disks), a RAID (an array of disks where for each N blocks, there is one block containing parity information for the remaining N blocks [PATT88]), or a parity striped configuration (an array of disks containing parity information across multiple disks, but files are allocated to single disks [GRAY90]). The results described in this study assume no parity information in the disk system and merely stripe the data across an array of 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 single 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 in the system. 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 size. The disks 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 a seek over more than 1 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 one contains a listing of the parameters which describe one common disk and their default values in these simulations. Note that the configuration simulated contains 8 disks and a total system capacity of 2.8G.

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. Each file type defines the size characteristics, access patterns, and growth characteristics of a set of files. Table two summarizes those parameters which define a file type.

For each file type, initialization consists of two phases. In the first phase, the indicated number of events are created, and each is assigned a start time (uniformly distributed in the range

Disk Parameters Based on the CDC 5 1/4" Wren IV Drives (94171-344)		
	actual	simulated
Number of disks	NA	8
Total Capacity	NA	2.8 G
Maximum Throughput	NA	10.8 M/sec
Number of platters	9	9
Number of cylinders	1549	1600
Number of bytes per track	24 K	24 K
Single Track Seek Time	5.5 ms	5.5 ms
Seek Incremental Time	0.0320 ms	0.0320 ms
Single Rotation Time	16.67 ms	16.67

Table 1: Disk Drive Parameters and Simulator Default Values

File Parameters	
Number of Files	How many files of this type should be created
Number of Users	How many parallel events access this file type
Process Time	Number of milliseconds between successive requests from a single user.
Hit Frequency	Number of milliseconds between requests from different users
Read/Write Size	Mean number of bytes per read/write operation
RW Deviation	Standard deviation in Read/Write Size.
Allocation Size	For extent based systems, mean extent size.
Truncate Size	How many bytes to deallocate for a deallocate request.
Initial Size	Mean size of the file at initialization time.
Initial Deviation	Deviation in the mean file size.
Read Ratio	Percent operations which are reads.
Write Ratio	Percent operations which are writes.
Extend Ratio	Percent operations which are extends.
Delete Ratio	Of the deallocate operations, percent which are file deletes.

Table 2: File Parameters and Description

[0, (*number of users* * *hit frequency*)]). The events are maintained in a heap, sorted by their scheduled time. For the second phase, the files are created. For each file a size is selected from a uniform distribution with mean equal to *initial size* and deviation of *initial deviation*. Allocation requests are made until the allocation length of the file is greater than or equal to this size.

The simulation runs by selecting the first event from the heap. Since each event corresponds to a file and therefore a file type, an operation may be selected based on the *read*, *write*, *extend*, and *delete* ratios. Then the *rw size*, *rw deviation*, and *truncate size* are used to generate a size parameter. Based on the operation, a call is made to one or both of the allocation and disk subsystems. After completion of an operation, the operation completion time is added to an exponentially distributed value with mean equal to *process time* and an event is scheduled at that newly calculated time. This event is returned to the heap and the events are reheaped.

If an allocation request cannot be satisfied, a disk full condition is logged, and the current event is rescheduled (according to the *process time* parameter as above). If the test being run is an allocation test, the simulation ends. For non allocation 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 how full the disk system is allowed to become. Any extend operation occurring when the disk utilization is greater than M is converted into a truncate operation. In this way, the disk utilization is kept between N and M while measurements are being taken.

As mentioned above, 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 disks sytem, is considered stabilized when the throughput calculation for 3 consecutive 10 second intervals are within .1 % of each other.

This study uses 3 simulated workloads to represent a time sharing or software development environment (TS), a large transaction processing environment (TP), and a super computer 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 these files. In addition there are larger files (mean size 96K) which get the remaining requests. These 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 randomly read 60% of the time, written 30% of the time, extended 7% of the time, and truncated 3% of the time. The log files receive mostly extend operations (93% and 94% respectively) with a periodic read request (2% and 5%) and an infrequent truncate (5% and 1%). The system log receives a slightly higher read percentage to simulate periodic transaction aborts.

The super computer 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 also read and written in 32K bursts, but are periodically deleted and recreated as well as being read and written (60% reads, 30% writes, 5% extends, 5% deletes).

3. Evaluation Criteria

We will examine three allocation policies in terms of how efficiently they make use of the available space on the disk, the application performance presented to each of the workloads described above, and the sequential performance of each workload.

The metrics for measuring how efficiently the disk space is being used are the external and internal fragmentation present when the disk system initially fills. External fragmentation is the amount of space still available in the disk system when a request cannot be serviced. This will be expressed as a percentage of the total available disk space. Internal fragmentation is the amount of space allocated to files, but not being used by the file. For example a 1K file stored in a 4K block suffers internal fragmentation of 75% because 75% of the allocated space is not being used. This will be expressed as a percentage of the total allocated space. The allocation tests are run by performing only the extend, truncate, delete, and create operations in the proportion as expressed by the file type parameters. As soon as the first allocation request fails, the external and internal fragmentation are computed.

The performance metrics for reading and writing will be expressed as a percent of the sustained sequential performance the disk system is capable of providing. For example, the configuration described in section 2.1 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.

We perform two tests to evaluate the system throughput. They are the application performance test and the sequential performance test. For both tests the lower and upper bounds on

disk utilization are set at 90% and 95% respectively, thus insuring that the disks are at least 90% full and no more than 95% full during the test. For the application performance test, the application workloads as described in section 2.2 are applied. When the throughput has stabilized the throughput numbers are recorded and the sequential test begins. For this test, only read and write operations are performed and each read or write is to an entire file. Stabilization for both the application and sequential tests occurs when the reported throughput at 3 consecutive 10 second intervals are within .1% of each other.

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 in that it supports only a few different block sizes. The last is an extent based policy as described in [STON89]. Each design is described in more detail and includes 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 at night rearranging the disk system. In this simulation, we consider only the allocation and deallocation algorithm (i.e. not the background reallocation). A file may be composed of some number of extents. The size of each extent is a

Workload	Disk Usage		Throughput	
	Internal Fragmentation (% allocated space)	External Fragmentation (% total space)	Application Performance (% max throughput)	Sequential Performance (% max throughput)
SC	43.1%	13.4%	88.0%	94.4%
TP	15.2%	9.0%	27.7%	93.9%
TS	18.4%	2.3%	8.4%	12.0%

Table 3: Results for Buddy Allocation.

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. As previous work suggests [KNOW65][KNUT69] such policies are prone to severe internal fragmentation, and our simulation results bear this out as shown in table three. 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 three for the supercomputer workload (SC).

4.2. Restricted Buddy System

As in the buddy system, the restricted buddy system applies the principal that as a file's size grows, so does its block size. In this way, small files are allocated from small blocks and don't suffer high fragmentation. As files grow, they are allocated in larger and larger chunks providing the ability to make large sequential transfers between the disk system and main memory. In addition, logically sequential disk blocks within a file are allocated contiguously in the disk system whenever possible. Therefore, even though files may start out with a small allocation, if they are laid out contiguously, we can still transfer a large quantity of data with only a single seek. Finally, the disk may be divided into regions and blocks within a single file may be clustered within these regions 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 N. If a system supports block sizes of 1K and 8K, the 1K blocks located at addresses 0 through 8 are considered buddies in that, together they form a block of the next larger size. These allocation policies attempt to coalesce buddies whenever possible, both when allocating blocks as well as when freeing them.

Free space is managed both by bit maps and free lists. A bit map is used to record the state (free or used) of every maximum sized block in the system. For smaller blocks, a circular doubly

linked list of free blocks is maintained in sorted order. Within each of these lists, blocks are arranged sequentially, and the allocator attempts to allocate logically sequential blocks of a file to physically contiguous regions. In this way, the free map is kept very compact. Maximum sized blocks which are completely unused require one bit. Smaller blocks are represented only if one of their buddies is in use.

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. In order to pick a robust configuration for this policy, 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.

We consider four different block size configurations:

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 block size, we consider both a clustered configuration and an unclustered configuration. In the clustered configuration, the disk system is broken up into 32M bookkeeping regions. Free lists and file descriptors are maintained per bookkeeping region. 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). When an allocation request is issued an attempt is made to satisfy that request from the "optimal" bookkeeping region as defined below.

The definition of the optimal region depends on the type of request. If the request is for a block of a file, the optimal region is that region which contains the most recently allocated block for that file. If no blocks have been allocated, the optimal region is that region in which the file descriptor was allocated. If the allocation request is for a file descriptor, the optimal region is the region after the region in which the last request was satisfied.

If a request is made to a specific region, and there is adequate contiguous space, but no block of the appropriate size, then a larger block (preferably the next sequential block) is split. The larger block is removed from its free list (or bit map). A block of the desired size is allocated, and the remaining space is linked into the free lists for the appropriate sized blocks. If the request fails in the desired region, a block of the appropriate size in any region is sought out. Only if no blocks of the appropriate size are found in any region does a block become split. The following summarizes the region selection algorithm.

Region Selection

1. Select Optimal Region
   - same as last
   - same as file descriptor
   - "next" region
2. Select region with a block of the correct size
3. Select next region with available space

In this way, the system attempts to keep large contiguous regions available for larger allocations.

The grow policy determines when we change the size of the block being allocated. The grow policy is expressed in terms of a multiplier. If g is the grow policy multiplier, then the unit of allocation increases from a_i to a_{i+1} when the total size 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.

The allocation and throughput tests were run on all the configurations described above. Figure one (a-f) show the fragmentation results. The most striking observation is that the attempts to coalesce free space and maintain large regions for contiguous allocation are

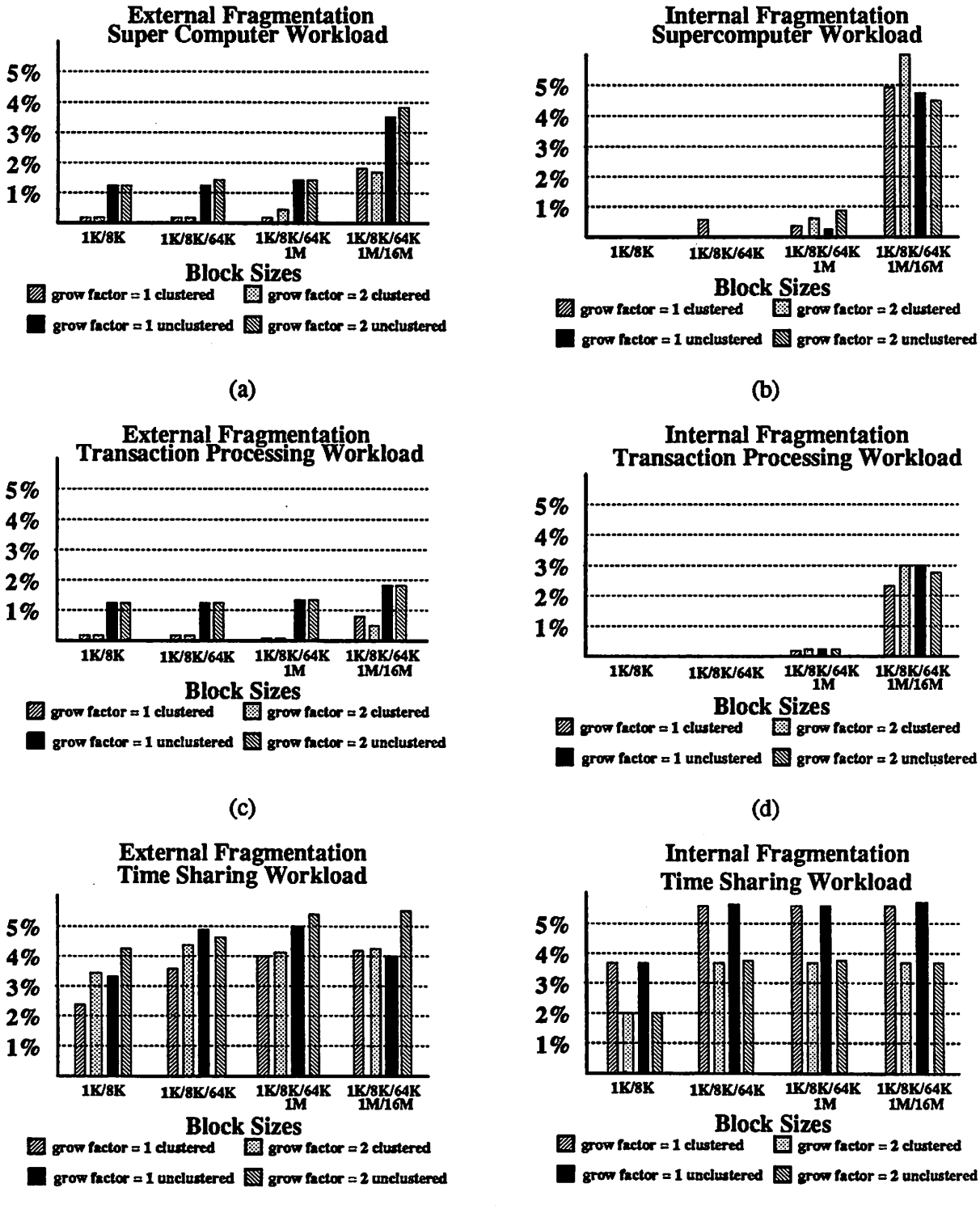

Figure 1: Internal and External Fragmentation Results for the restricted buddy allocation policy. Notice that even the worst fragmentation is under 6%. Since the supercomputer and transaction processing workloads consist of large files, fragmentation is rarely discernible.

successful. None of the polices show fragmentation greater than 6%. As expected, the time sharing workload which has the blend of large and small files exhibits the greatest fragmentation (figures 1e and 1f), 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 1f, note the difference between each pair of adjacent bars). External fragmentation increases slightly as we go to an unclustered configuration all blocks are eligible for splitting rather than just those in the "optimal" region.

Figure two (a-f) shows 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 show relatively little sensitivity to either the grow policy or clustering. For the five block size configurations (the rightmost on each graph), most show slightly improved performance with a non clustered 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 a non clustered configuration, the attempt to keep subsequent allocations contiguous results in the slightly 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 represent higher throughput numbers than the third and fourth bars). Since this environment is characterized by a greater number of smaller files, even with the larger block sizes, data is being read from disk in fairly small blocks. As a result, the seek time has a greater impact on performance, and the clustering policy which reduces seek time provides the better throughput.

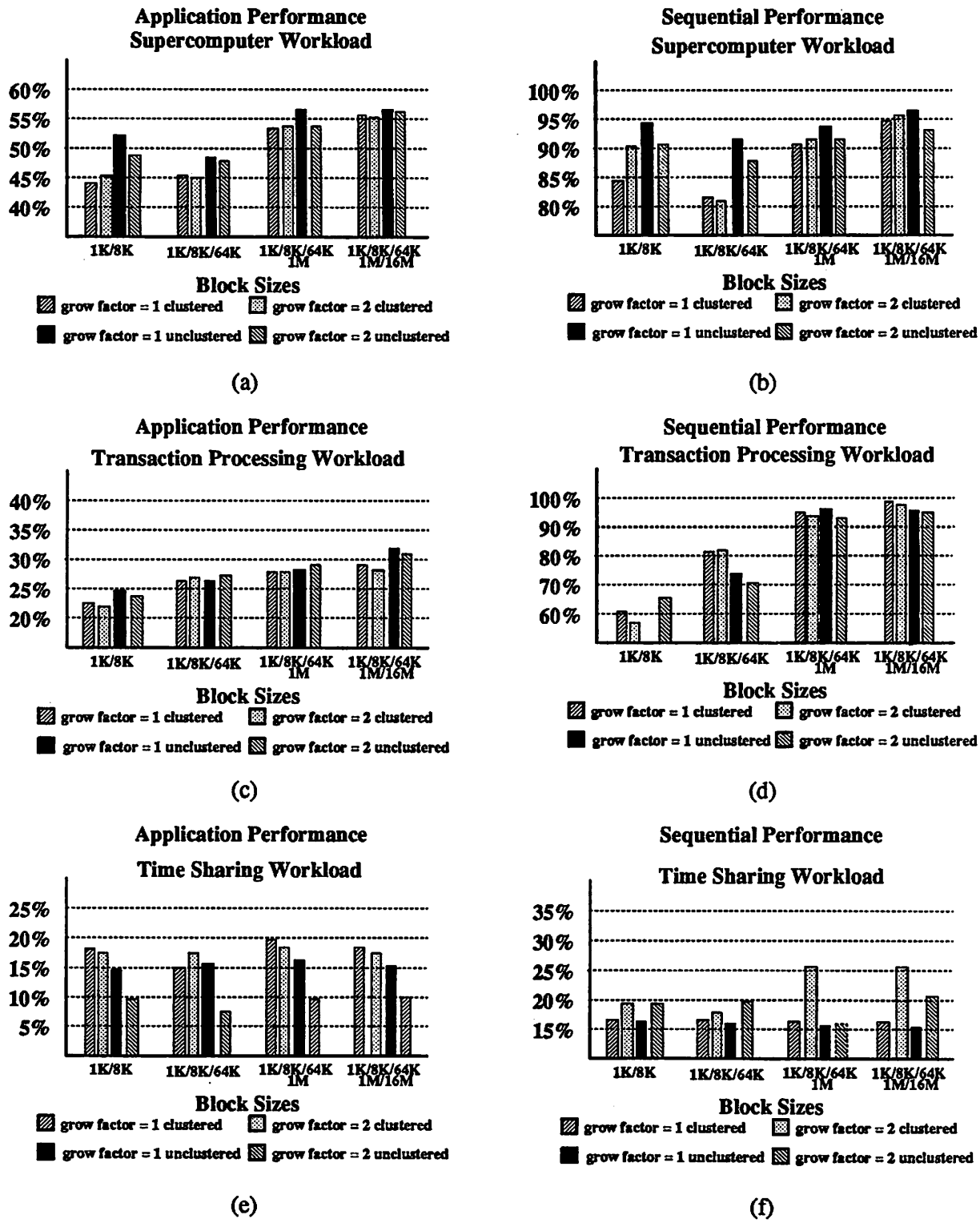

Figure 2: Application and Sequential Performance for the Restricted Buddy Policy.

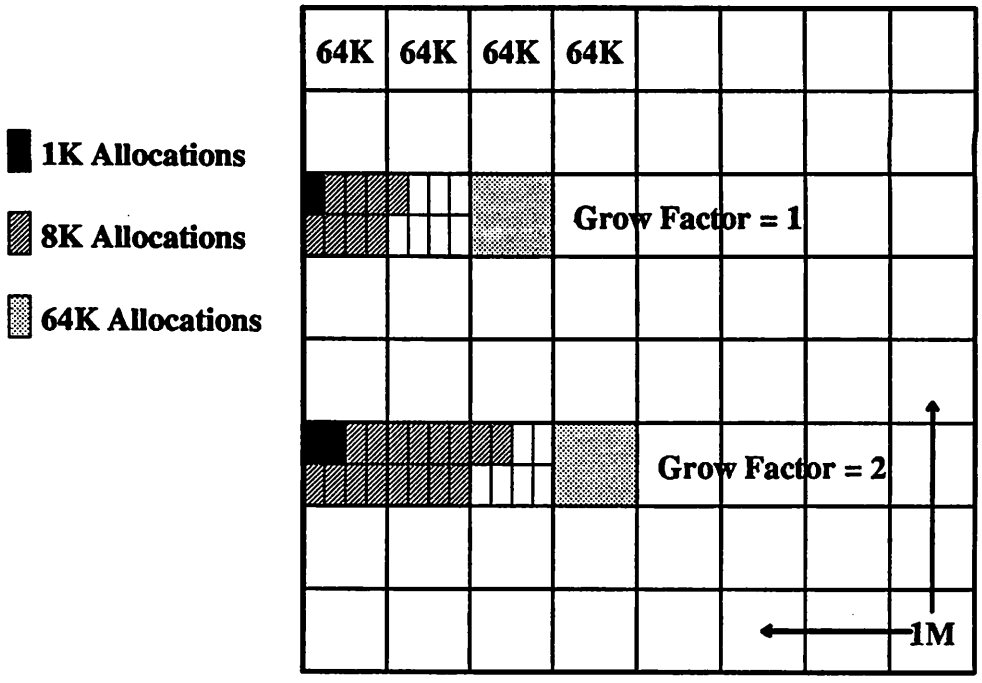

Figure 3: How contiguous allocation and grow factors interact. Because the total file length is not a multiple of the new block size, we are required to pay a seek when the block size grows.

Figures 1e and 1f also indicate that the higher grow factor provides better throughput (the second and fourth bars in each set represent higher throughput than the first and third bars). 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 three 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. If we were not successfully allocating small blocks sequentially,

then the greater grow factor would show a decrease in performance.

Since none of the configurations suffered excessive fragmentation we use the application and sequential performance results to select a configuration for the final section. Since clustering had little effect on the large file environments and improved performance in the time sharing environment, we will select a clustered configuration. In four of the six cases a-f, the grow factor of 1 provided slightly improved throughput so we will select that, knowing that it will penalize sequential performance for the time sharing workload. This is the leftmost bar in each group of each graph. 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.

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 extent sized chunks. 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.

In such a system, the significant design parameters are how to select extents for allocation and how widely varied the extent sizes are. We consider two different allocation policies, first-fit and best-fit and five different extent configurations. We assume that the high bandwidth will be achieved by selecting large extent sizes for large files so we make no effort to allocate logically sequential extents contiguously in the disk system. 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. Thus, most extents would fall in the range 716K to 1.3M. As the number of extent ranges

increases, we expect to see increased fragmentation since we are allocating a more diverse set of extent sizes.

Figure four shows the fragmentation results for the extent based polices. The x axis indicates the number of extent ranges in each configuration. The extent ranges for the different workloads are listed below.

Workload	Number of Extent 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

With extent sizes ranging from 1K to 16M, we expect to see poor results for fragmentation, but the results to not support this. One possible explanation is that the ratio of large files to small files is fairly constant in these simulations. As a result, new extents are allocated to extents of the correct size. This would explain why best fit consistently resulted in less fragmentation.

We expect throughput to be fairly insensitive to the selection of best fit or first fit since in both cases, files are read in the same size unit. Figure five shows the application and sequential performance results for the extent based polices and validates this intuition. In general, we see slightly better performance from first fit, due to the slight clustering that results from 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 four. We expect 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 (figure 5b and 5d) while the time sharing

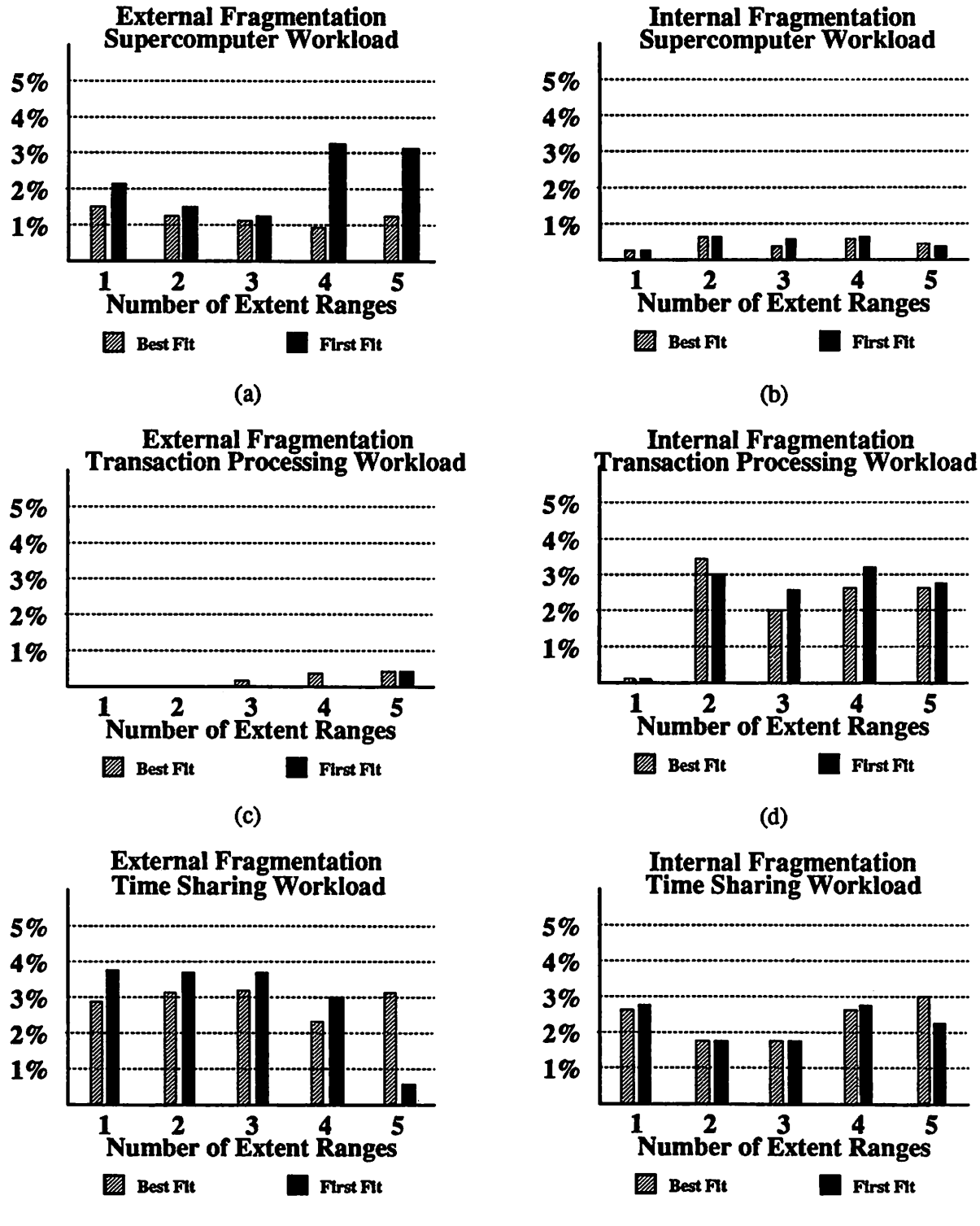

Figure 4: Internal and External Fragmentation Results for the Extent Based Allocation policies. Surprisingly, we find that even with a wide range of extent sizes, neither internal nor external fragmentation surpasses 5%.

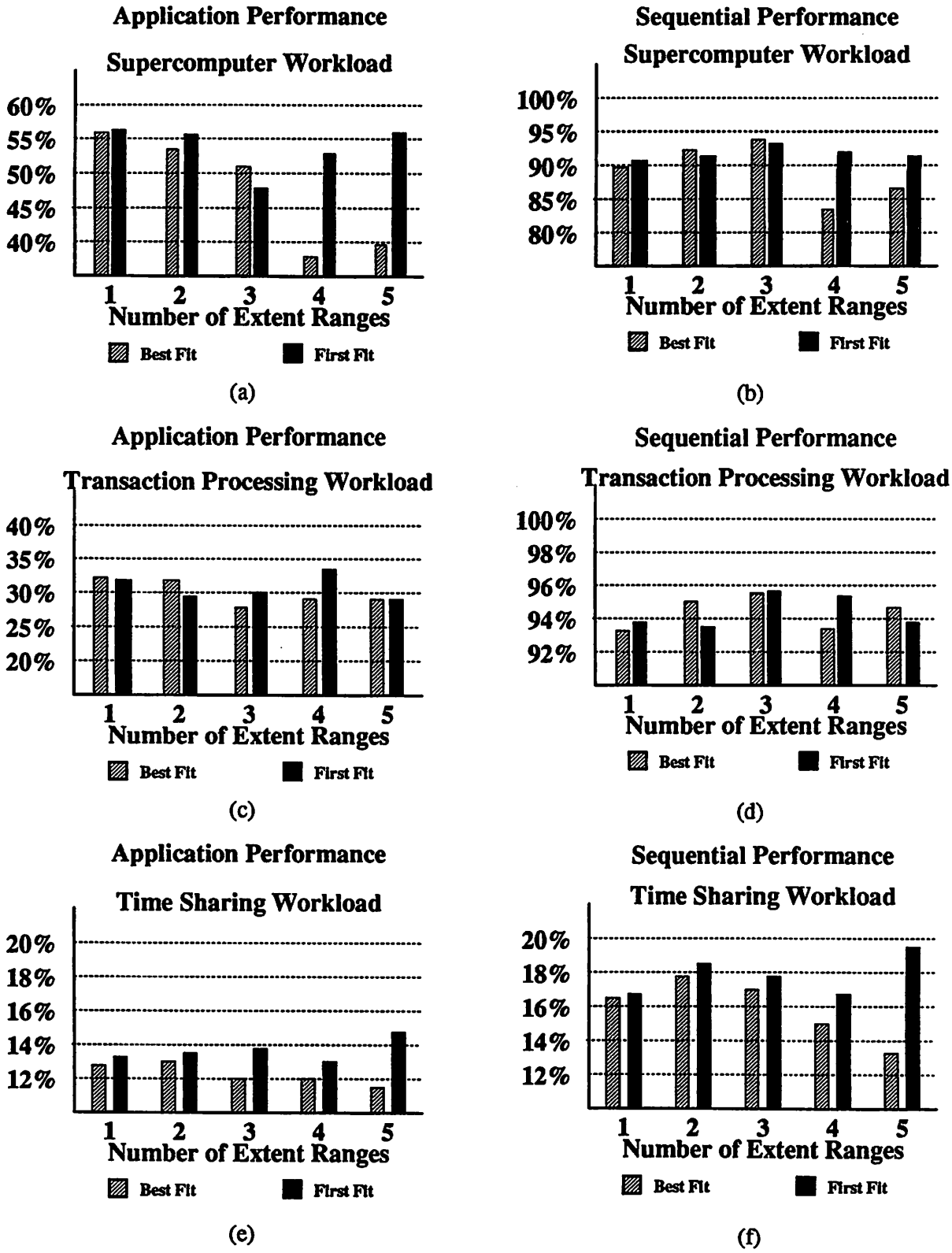

Figure 5: Application and Sequential performance for the extent based policies.

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.

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 are small in the TS environment, they can be allocated in one or two 4K extents. The larger files require 24 extents (96K file length/4K extent size). 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 slightly better performance than best fit. Next we must select a number of extent ranges. For the transaction processing and supercomputer workloads simulated, the 3 range sizes result 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.

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 number against a 4K and a 16K fixed block system which does not bias towards automatic striping or contiguous layout. The 4K system is more compared with the timesharing workload while the 16K is compared for the transaction processing and supercomputer workloads.

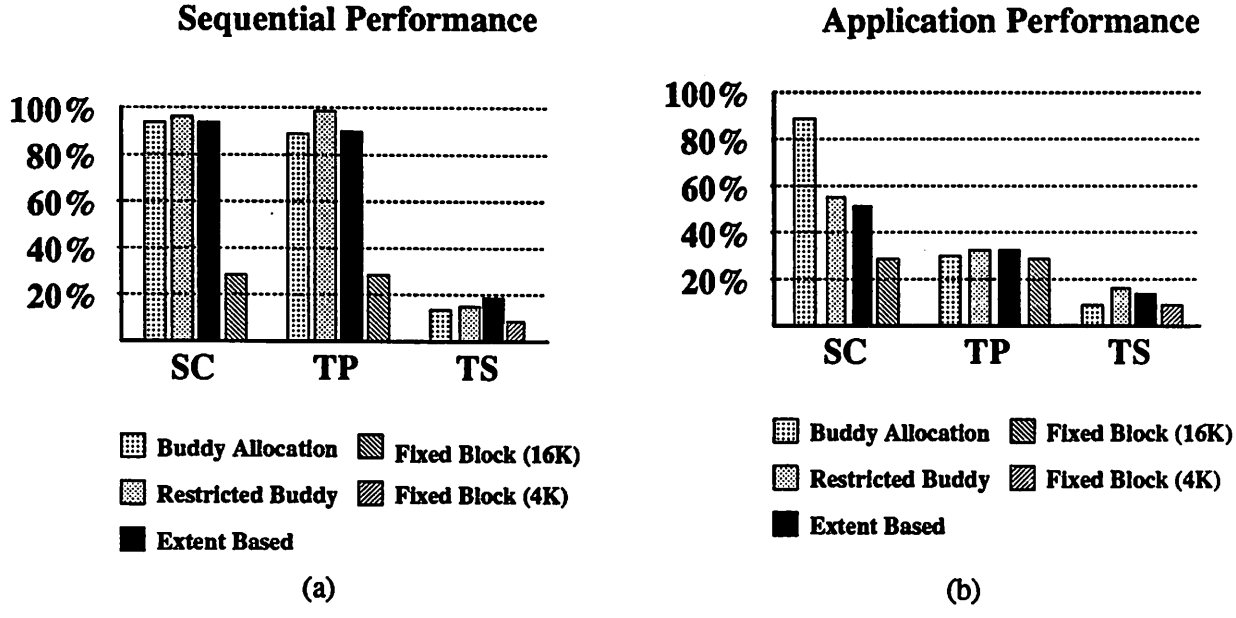

Figure 6: Comparative Performance of the different allocation policies.

Figure 6a shows the sequential performance of the four methods (the three discussed above 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 utilize nearly the complete bandwidth. In the time sharing environment, none of the policies succeed in pushing the system above 20% utilization due to the presence of so 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 6b), 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 25%. Even for workloads like the transaction processing environment, which are dominated by small reads and writes to large files, the improvement is as great as 10%. While the large blocks do not benefit the small file environment greatly, they don't 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 results suggests that time sharing environments could benefit significantly from these allocation techniques. Without hindering the small file performance, such systems could then effectively compete with larger systems designed with database or supercomputer applications in mind.

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 of 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 argument.

7. Bibliography

[GRAY90] Gray, J., Walker, M., "Parity Striping for Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput", to appear in *Proceedings of the Sixteenth International Conference on Very Large Databases*, Brisbane, Australia, August 1990.

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

[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] Operating System Support for Database Management, Michael Stonebraker, 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.