CLUSTERING ACTIVE DISK DATA TO IMPROVE DISK PERFORMANCE

Carl Staelin
Hector Garcia-Molina

CS-TR-283-90

September 1990

Clustering Active Disk Data to Improve Disk Performance

Carl Staelin
Hector Garcia-Molina

Department of Computer Science
Princeton University
Princeton, New Jersey 08544

September 20, 1990

Abstract

We show that clustering the active data of a file system in the center of the disk is a viable and effective means of improving system I/O performance. We demonstrate that it can reduce the average seek delay and in the presence of disk queues it can also reduce the average rotational delay. We also present experimental results which show that file access patterns are strongly skewed, and that file activity levels are relatively stable over time, making them a good predictor of future file activity levels. Using simulations, we investigate two techniques for reorganizing the disk data, and we measure sensitivity to imperfect predictions of future file activity due to drifting file activity levels. We demonstrate significant performance improvements over a full spectrum of file system use, from lightly loaded systems through heavily loaded systems with large disk queues, with performance benefits increasing as the system load increases.

1 Introduction and Motivation

iPcress is an experimental, "next generation" file system under development at Princeton University. Its main goal is to provide improved performance, effectively utilizing larger memories and faster processors now available, as well as parallel I/O paths. It retains the popular UNIX file system interface, while at the same time providing more functionality: improved reliability, transaction facilities, integrated access to electronic disks and video jukeboxes, and so on. It runs as a user process under the MACH operating system, communicating with clients via ports, and hence is very portable.

One of the most salient features of iPcress is that it keeps file usage statistics and will use these to improve performance. The decision to utilize statistics was motivated by an earlier joint study with Amdahl Corporation. In that study we looked at I/O operations on very large file systems (on the order of one tera byte) at several Amdahl customer locations. We observed that even for such huge file systems, there was very high locality of reference. Some files had a much higher *temperature*, i.e., probability of access, than others. This observation has been made in experiments by other researchers. However, we also observed that the temperature of a file changed slowly over the days. This suggests that a file system that tracked recent temperatures could use this information to decide what files to keep in memory or where to locate the files on secondary storage.

In this paper we focus on this last optimization: placing high temperature files close to each other and in the center of the disk, with colder files placed on either side of them. The hope is that this will significantly reduce the seek time between accesses: If it is likely that consecutive accesses are to hot files, then it is likely that they will be to blocks in the center of the disk, reducing the expected seek time. The rotational delays may also be reduced, if it is likelier that accesses will be to the same cylinder. As we will see, the improvements due to both effects can indeed be very significant.

The idea of placing hot files near the center of a disk is not new. For years some system administrators have been placing a few of their hottest files in the center. However, such optimization of file placement

has been done on an ad-hoc basis by hand [1]. We are proposing to automate this process, and we evaluate expected performance gains. We also stress that this reorganization will be done frequently, possibly even on a continuous basis.

The development of iPcress consists of two concurrent efforts. One is the actual implementation of the system [2]. A current prototype is running, implementing a large file cache with variable sized blocks both in memory and on secondary storage. It allows multiple devices within a single file system. The second effort is a performance evaluation of the various planned optimizations, in order to guide the implementation effort. In this paper we report on the evaluation of the file placement optimization. Our goal is to decide if, given the observed temperature distributions, the gains due to disk placement will be significant and worth the effort of keeping file statistics. We also need to evaluate the impact of changing temperatures. As we observed above, temperatures drift slowly over time and it will take the system some time to reorganize the data. How sensitive are the expected improvements to this drift? Will the system have enough time to reorganize the data? A third set of issues have to do with implementation strategies. There are several ways to implement disk placement (on-line vs. off-line reorganization; perfect placement according to temperature vs. approximate placement; etc.) Our objective is to understand the tradeoffs involved in these decisions.

We start by presenting some of our experimental results from the Amdahl study (in Section 2). Based on this data, we develop a model for file accesses (Section 4) and study the expected performance (Sections 5.1 and 5.2). Finally, in Section 5.2 we discuss implementation strategies and their performance.

2 Data Access Experiments

Before we designed iPcress, we conducted experiments to determine if there were any useful file access patterns which would allow us to predict future behavior. Amdahl Corporation generously allowed the authors access to trace data and computing resources necessary to conduct this research during the spring of 1988. In this section we summarize some of the relevant results; additional data may be found in [3].

We were primarily interested in dynamic access patterns. The utility which generated the data was IBM's System Management Facility (SMF), which provided us with trace data for each file open and close in IBM's MVS/XA operating system.

The data came from two installations which we will call customer "S" and customer "Z". The only information we received from the shops was SMF trace data. The first trace, from customer "S", contained a week's worth of data and 170,437 file open-close sessions. The second trace, from customer "Z", contained three days of data with 142,795 sessions. Most of the analysis was done using SAS and Merrill's MXG Package. Essentially, SMF records file activity information for each active process and open file. This data is transformed into SAS data sets by Merrill's MXG package.

The SMF traces contain tremendous quantities of data, a lot of which was not relevant to our current work. Primarily, we can generate a table of file open-close sessions. For each session we know:

- which file was opened.
- roughly how many I/O's were done during the session.
- file open mode (read only, write only, read/write).
- when the file was opened.
- when the file was closed.
- how many tracks were allocated to the file when closed.
- name of the job which opened the file.
- logical name assigned to the file by the job.

There were several limitations with the SMF traces. For example, SMF collects summary statistics for each open. There is no record of the timing between various reads, nor even which blocks or tracks of the file were accessed. A second problem is that SMF only records the number of tracks allocated to a file,

not the actual space used by the file. Since most of the small files have allocations of one cylinder or a few hundred thousand bytes, and since the smallest unit of allocation is one track (thirteen thousand to forty-seven thousand bytes), the problem seems to be more severe with small files. Another problem with the SMF data is that we only have data on those files accessed during the measurement period. This means that we cannot tell precisely how large the entire file system is. Additional problems are detailed in [3].

Since our data was coming from a "production" environment, we were not able to modify or install an operating system that could give us more accurate or detailed information. Instead we had to be satisfied with trace data from the safe and stable performance monitoring facility provided by IBM. However, we believe the problems were minor and the data sufficient to answer our key questions.

One interesting feature of the data is that there are thousands of temporary files which consume ten to twenty five percent of the total I/O's and whose cumulative size if fifty to eighty percent of the total space consumed by all files seen in the trace data. However, the space used by temporary files at any given time is relatively small. This fact produces an interesting accounting problem from the standpoint of file system size since there is no easy way to account for the space used by temporary files. The primary problem is that the space is re-used by the system, and it is not obvious how to attribute the I/O's to particular locations. As a result we ignored temporary files, under the assumption that they require little space. (A second argument is that a good file system with a large memory buffer will keep these temporary files in memory, and hence eliminate their I/O activity.)

In order to quantify file activity we define the concept of file temperature. In cache related literature the terms "hot" and "cold" denote objects which are accessed heavily and lightly respectively. We extend this concept to files and give it an initial definition as the number of I/O's a file receives. Since files don't all have the same size, this simple definition is inadequate. For example, if two files each receive a hundred 512-byte I/O's, and one file is a thousand bytes long while the other is a million bytes long, then the thousand byte file is obviously "hotter". Thus, we define file temperature as the number of bytes transferred in a particular time period divided by the number of bytes in the file.

$$file\ temperature\ =\ \frac{number\ of\ I/O's}{file\ size} \tag{1}$$

Given this definition, the two files from the example above would have vastly different temperatures. As we have noted, SMF records file size in terms of allocated tracks. Hence, "file size" in the temperature definition will be interpreted in this fashion. The number of I/O's is recorded over a particular measurement period.

Our results are shown using the following technique. In order to detect a correlation between some criteria, such as file temperature during a given interval or the number of open-close sessions, and file activity, we rank the files according to that criteria and then we plot the cumulative I/O versus the cumulative disk space consumption, both expressed as percentages. By ranking the files and then plotting cumulative activity according to that ranking we can see whether the first files receive more or less activity than the rest of the files.

The shape of the resulting curve gives some indication whether the criteria used to rank the files is correlated with file activity. For example, a curve that starts at the origin, heads nearly straight up to the top of the graph, flattens out and then goes across to the upper right end of the graph would indicate that the criteria is highly correlated with file activity. In Figure 1 we show the cumulative I/O versus the cumulative disk space for customer "Z" for the whole three day period, with files ranked by descending temperatures. Note that the curve shows eighty-five percent of the I/O going to ten percent of the disk space. As we shall see in Figure 2 this result becomes even more strongly skewed over shorter time periods. As mentioned earlier, our graphs are obtained from traces of file system activity and only files accessed during the monitoring of the file system appear in the trace. In addition, disk space not containing files is not counted in the total disk space. Consequently, the total disk space is probably dramatically under-reported. This implies that eighty percent of the I/O's are probably going to *less* than ten percent of the *total* disk space.

Using the trace data we discovered that the hottest files tend to stay hot over a time frame which can be measured in days. We can show this by measuring file temperature over a sample period, say one to three days, and we can order the files by descending temperature. We can then measure the cumulative I/O over that *same* ranking for the next few days to see if the hottest files still get most of the I/O. In Figure 2 we

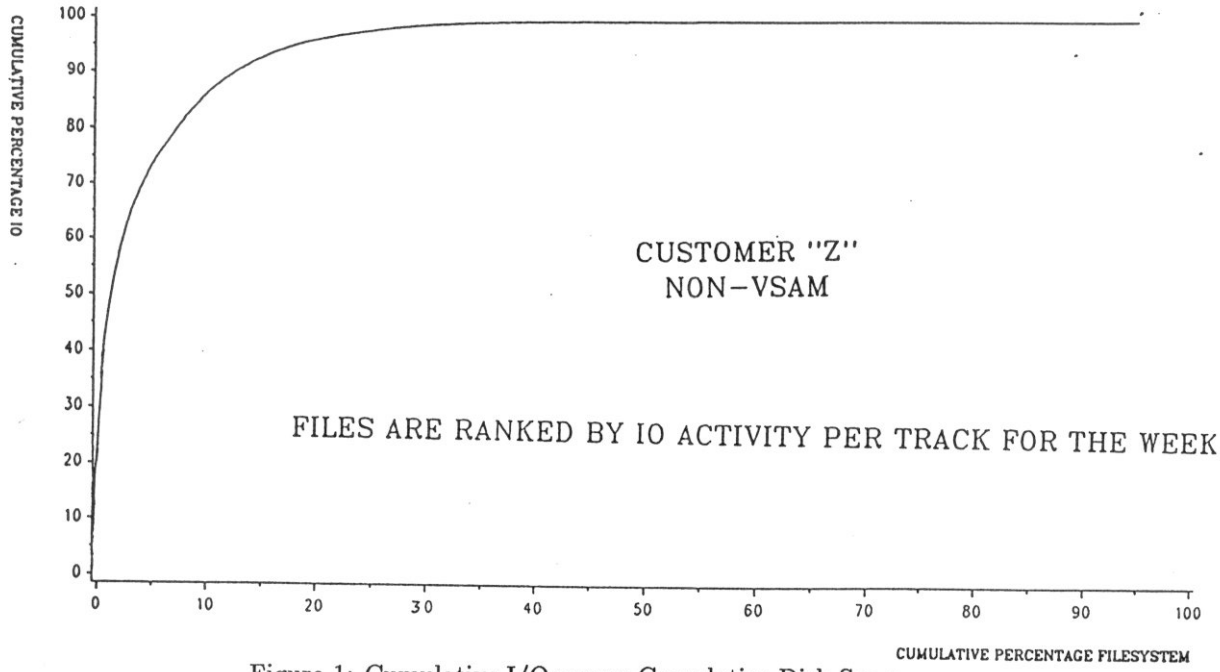

Figure 1: Cumulative I/O versus Cumulative Disk Space

show the cumulative I/O for the hottest one thousand files (out of roughly twenty thousand permanent files that were accessed during the week) from a one day measurement period; we then show the cumulative I/O for those same files during the next day. The percent of the file system used is relative to the collective size of the files accessed during each day, not the total size of the file system. (Note that the horizontal axis in Figure 2 only goes up to seven tenths of a percent of the total space accessed during each day.) It is evident that seventy percent of the I/O is still directed towards these hot files, while twenty percent of the I/O that was directed to these files has "drifted" to other files not shown. Measurements (not shown here) over longer periods (e.g. three days) show more drift (e.g. after three days fifty percent of the I/O is still directed to the hot files). This result means that current file temperature can be used as a predictor for future file temperature.

3 Prior Work

Recently there has been a resurgence of research interest in file system design and analysis. [3,4,5,6] have looked at file access patterns in the IBM MVS system, while [7,8,9] have looked at the UNIX system. [9] shows that roughly seventy percent of read-only and eighty percent of write-only accesses are whole file transfers; most opens are to files which are opened hundreds or thousands of times, and most files were opened less than 10 times a week. [3,4,5,6] have demonstrated that file accesses are highly skewed, and that most file system activity is concentrated on a relatively small fraction of files. [7] shows that in UNIX most files accesses are whole file transfers and the read/write ratio is between 80/20 and 65/35. As far as we know, none of the studies, except our work in [3], looked at how file temperatures varied over long periods of time and whether they could be used as good predictors of future activity.

There has been a great deal of work done over the years on optimization of disk accesses. [10] evaluates disk performance with large disk queues (up to 1,000 requests) for various scheduling algorithms. [11] discusses the performance advantages of mirrored disks. [12] proposes to move idle disk heads to the center of the disk to minimize the expected seek delay. [13] proposes an optimal means for balancing disk load over several devices based on the system I/O configuration and traces of file activity. Other means of improving disk performance include reducing the fraction of disk space used on devices to reduce the average load on the device, and restricting the available disk space to a small fraction of the disk surface to reduce the average seek time.

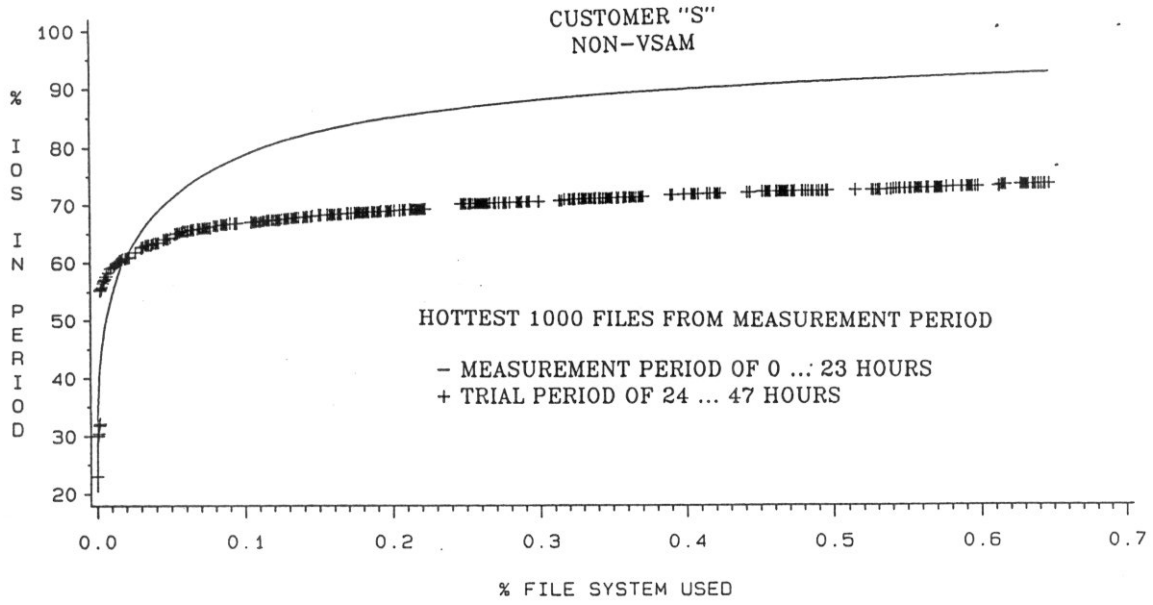

Figure 2: Cumulative I/O versus Cumulative Disk Space, Showing Persistence

4 Access Model

Our next step is to model the access patterns observed in our experiments. In particular, given a disk with cylinders 1, 2, 3, ..., N, we would like to have a function that gives us the probability of accessing each cylinder:[1]

$$Prob\,[\text{next access is to cylinder } x] \;=\; f(x) \quad x \;\in\; \{1, 2, \ldots, N\}$$

In a system where data is placed at random on the disk, without regards to its temperature, we would expect

$$f_r(x) \;=\; \frac{1}{N} \tag{2}$$

If data is placed according to temperature, we would place the hottest data in cylinder $N/2$, the next hottest in cylinders $(N/2)-1$ and $(N/2)+1$, and so on, with the coldest data in cylinders 1 and N.

What would be a good function $f(x)$ to describe this situation? The answer is a function that would yield a cumulative distribution similar to the one observed in Figure 1. That is, say we consider a fraction d of the disk that contains the hottest data. The fraction of the total I/O's that we expect to access this data would be

$$G(d) \;=\; \sum_{i=\lfloor(1-d)\frac{N}{2}\rfloor+1}^{\lfloor(1+d)\frac{N}{2}\rfloor} f(i) \tag{3}$$

For instance, if $d = 1$ we are considering the entire disk and the probability that we access something in the disk should be 1. If a single disk contains all of the system data, this cumulative distribution should have the same shape as the cumulative distribution reported in Figure 1. If the data is spread over several disks, we should still have the same shape, assuming first the hottest data is spread over all disks, then the next hottest is spread, and so on. This is a reasonable assumption, as placing the hottest data on a single disk would create a bottleneck at that disk.

[1] We assume that accesses are independent

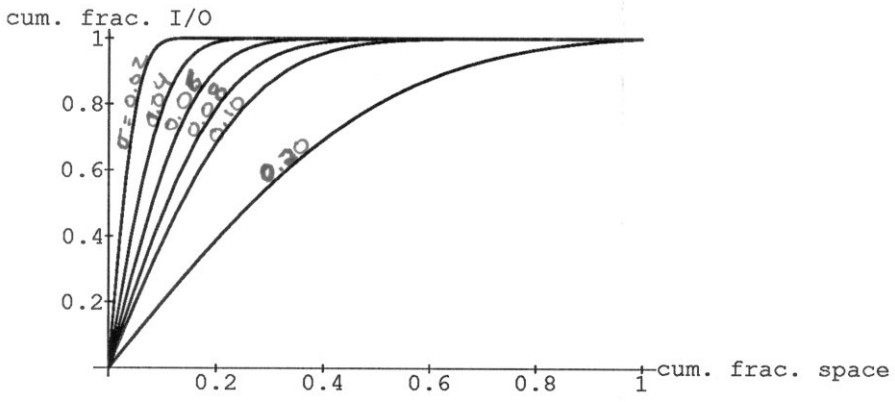

Figure 3: Normal Distributions: Cum. Prob. of Access vs. Cum. Frac. Disk Space

Intuitively, it seems that a normal distribution, centered at $N/2$ would be a candidate for $f(x)$. So let us start with this distribution and then check its cumulative distribution. For simplicity, we use a continuous distribution centered at $1/2$ and with standard deviation σ:

$$h'(t;\sigma) \;=\; \frac{1}{\sqrt{2\pi}\;\sigma}\; e^{-\frac{(t\,-\,0.5)^2}{2\sigma^2}}$$

Since we want to limit the function to values in $[0,1]$ we actually use:

$$h(t;\sigma) \;=\; \frac{h'(t;\sigma)}{\int_0^1\; h'(s;\sigma)\; ds}$$

The probability of access is then

$$f_n(x;\sigma) \;=\; \int_{(x-1)/N}^{x/N} h(t;\sigma)\; dt \tag{4}$$

Figure 3 shows the resulting cumulative distribution $G(d)$ (equation 3) for various values of σ from 0.02 to 0.2. These curves approximately have the shape of those in Figure 1 (note that in Figure 1 the maximum d is about 0.01, not 1). By comparing our function to other experimental results (not shown here due to space limitations), we empirically observe that a better function to use is a combination of normal distributions:

$$h_p(x) \;=\; 0.3\,h(x\,;\,0.001) \;+\; 0.3\,h(x\,;\,0.01) \;+\; 0.2\,h(x\,;\,0.05) \;+\; 0.2\,h(x\,;\,0.1)$$

The access distribution for this perfect layout case is

$$f_p(x) \;=\; \int_{(x-1)/N}^{x/N} h_p(t)\; dt \tag{5}$$

The cumulative distribution $G(d)$ (equation 3) for this f_p closely matches our experimental results (such as those in Figure 1) and is the function that we use in our analysis for the case where the data is perfectly organized by temperature.

In addition to distributions f_r and f_p (random and perfect placement), there are some others distributions we will study. The next one we introduce is motivated by the observation that our measurements only refer to files that were actually accessed during the test period. In addition to these files, there were many other files that were simply not touched. There was also empty disk space that was not used, and consequently is not reported in the experimental trace data. This implies that some fraction of the disk is rarely used, and such data can be concentrated at the edges of the disk. Performance will be even better since the disk arm will only move among the used cylinders (except for very rare accesses to the rest of the disk).

Let us say that only $M = \lfloor kN \rfloor$ cylinders in the disk are used during a given test period, $0 < k \leq 1$. Thus, the range of active cylinders is $a = \lceil \frac{N}{2} \rceil - \lfloor \frac{M-1}{2} \rfloor$ to $b = a + M - 1$. Within this range, the probability of access is as if we only had M cylinders, and cylinder a was actually the first cylinder. Thus, the probability of access with "concentrated" data is

$$f_c(x;\ k) \ = \ \begin{cases} \displaystyle\int_{(x-a)/M}^{(x-a+1)/M} h_p(t)\ dt & \text{for } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The file system will attempt to place files by their temperature. However, when the temperature of a file changes, it will take the system some time to recognize this, and it will take additional time to actually move the file to its correct new position. To model the effect of drifting temperatures, let us say that with probability $(1 - \delta)$ a file that has not drifted will be accessed; with probability δ the accessed file has drifted and is hence located in a random spot on the disk ($0 \leq \delta \leq 1$). This can be modeled with the distribution

$$f_d(x\,;\,\delta) \ = \ (1-\delta)\, f_p(x) \ + \ \delta\, f_r(x) \tag{7}$$

(Functions f_p and f_r are defined in equations 5 and 2.) If δ is 0, all files are in their correct place by temperature. As δ increases, the system is having a more and more difficult time placing files.

So far we have assumed that if a file has a temperature corresponding to cylinder x, it will be placed in exactly that cylinder. In practice this may be hard to achieve, as that cylinder may be full and it may be too expensive to make room. A more effective approach may be to divide the cylinders into buckets, and to place a file anywhere in the bucket that corresponds to its temperature. To model this situation, we assume that there is a function $b(x)$ that maps a cylinder number to one of the buckets. For example, $b(1)$ will be the first bucket, i.e., the one for the hottest cylinders. Depending on the size of the bucket, cylinders 2 and 3 may be in this bucket ($b(2) = b(3) = 1$), and so on. Within a bucket, data is placed at random. Thus,

$$f_b(x\,;\,b) \ = \ \frac{\displaystyle\sum_{\{i \mid b(i)=b(x)\}} f_p(i)}{\displaystyle\sum_{\{i \mid b(i)=b(x)\}} 1} \tag{8}$$

A last case we consider is the following: Say the system focuses only on the hottest fraction k of the data. If the data is hot enough, it will be placed in the correct cylinder by temperature; otherwise it will be placed at random in the remaining $(1 - k)$ fraction of the disk. There will be a total of $M = \lfloor kN \rfloor$ cylinders in the organized area. The cylinders in this case will be $a = \lceil \frac{N}{2} \rceil - \lfloor \frac{M-1}{2} \rfloor$ through $b = a + M - 1$. Call this range R. This models a "lazy" system that only partially organizes data. The access distribution in this case is

$$f_l(x\,;\,k) \ = \ \begin{cases} f_p(x) & \text{for } x \in R \\ \dfrac{1 - \displaystyle\sum_{i \in R} f_p(i)}{N - \lfloor kN \rfloor} & \text{otherwise} \end{cases} \tag{9}$$

In summary, we have presented a family of distributions that represent various scenarios: randomly placed data (f_r), data perfectly ordered by temperature (f_p), accessed data concentrated over a fraction of the disk (f_c), drifting temperatures (f_d), data placement by buckets (f_b), and lazy organization of just a fraction of the data (f_l). The impact of these distributions on performance will be studied in the next sections. Note that one could also define distributions for combined effects, e.g., a system where data is organized into buckets and temperatures drift. In this paper we will not consider such combined effects,

mainly because of space limitations, but also because it is easier to understand the issues if one studies each case separately.

5 Analysis

Before we present our analyses, we first present some basic functions and definitions. If we define the number of cylinders to be N, the cost of moving d cylinders to be $C(d)$, and the probability of accessing cylinder i to be $f(i)$, then we can define the average seek time (T) to be:

$$T = \sum_{i=1}^{N} f(i) \left(\sum_{j=1}^{N} C(|i-j|) f(j) \right) \tag{10}$$

In any case, if we assume that $C(0) = 0$, then using a change of variable we may rewrite Equation 10 as:

$$T = \sum_{d=1}^{N-1} 2C(d) \left(\sum_{j=1}^{N-d} f(j) f(d+j) \right) \tag{11}$$

As a simpler model, if we assume that the disk head always starts at the center cylinder, then we get:

$$T = \sum_{i=1}^{N} C(\left| \frac{N}{2} - i \right|) f(i) \tag{12}$$

We may also use Equations 10 and 11 to determine the expected seek distance by letting $C(d) = d$. However, we are primarily interested in the average disk utilization, which is the fraction of time spent transferring data over the total time spent for an access.

$$U = \frac{T_{transfer}}{T_{transfer} + T_{rotation} + T_{seek}} \tag{13}$$

There are several equations describing seek times in terms of distance. (Note: unless otherwise noted, all times are in units of milliseconds). Most disks use voice coils which have non-linear behavior and so the seek cost can be modeled as [11]:

$$C(d) = \begin{cases} 0 & \text{if } d = 0 \\ 5 + 0.5\sqrt{d} & \text{otherwise} \end{cases} \tag{14}$$

5.1 Lightly Loaded Systems

We first analyze performance in a lightly loaded system. We define this to be the case where at all times there is either zero or one request on the disk queue. In this scenario, the disk head is always moved from the position of the last request to the position of the new request, hence our performance improvements from reorganization will be due to reduced seek times.

Using Equation 11 we can numerically compute the average seek times for the various cylinder access probability distributions of Section 5. We model an imaginary disk with 1000 cylinders and with the seek time described by equation 14.

We try to show how sensitive our result is to how well the system manages to cluster the data. We do this by plotting the average seek time over a variety of distributions. We vary the parameters for the three distributions (normal f_r, drifting f_d, and lazy f_l) from zero to twenty percent ($[0, .2]$). For comparison, we also show the average seek time for uniformly distributed data.

The drift distribution (f_d) shows the system's response to drifting file access patterns, and the parameter from Figure 4 represents the percent of the total I/O which is uniformly distributed over the disk. Consequently, the point at $percent = 0$ represents our "perfect" distribution. Note that with twenty percent drift we still realize a thirty percent improvement in seek times, while the "perfect" distribution gives a forty percent reduction in seek time.

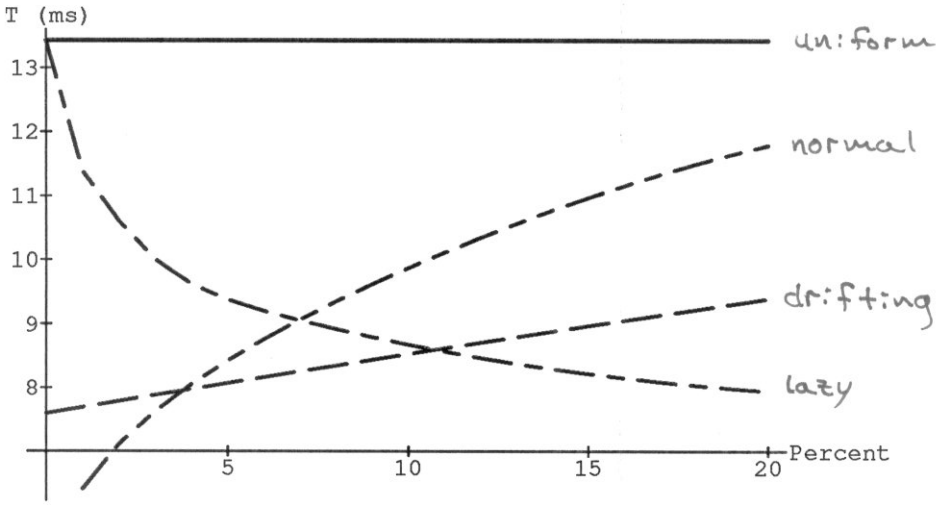

Figure 4: Expected Seek Time

The parameter for the normal distribution is σ, the standard deviation of the distribution. On this graph we show it as a percent of the number of the cylinders on the disk. For example, $percent = 10$ for our imaginary 1000 cylinder disk means that the normal distribution would have a standard deviation of 100 cylinders. For $percent = 3$ ($\sigma = 30cyls$), the performance of the normal distribution is equivalent to that of our "perfect" distribution, f_p ($T = 7.5ms$). The important point to note is that gains are significant over a wide range of σ values in this region. Thus, even if our access distribution is not *exactly* f_p in some application, we can still expect significant gains.

The "lazy" distribution represents the partial reorganization of the file system — the system reorganizes the hottest x percent of the file system, leaving the access probability for the remaining data uniformly distributed over the rest of the disk. With none of the data reorganized, we have no improvement, while with five percent of the file system reorganized we see a thirty percent improvement in seek time. Once we have reorganized twenty percent of the file system, we have acquired nearly all the benefits of a complete reorganization.

We are also interested in the fraction of time the system spends doing useful work (transferring data) versus overhead, such as seek and rotational delays. For this computation, we assume that as soon as the current request is satisfied, another request is generated so that the system is always busy. This situation models the case of a single I/O bound job. In Figure 5 we show the utilization (Equation 13) for the same distributions and the same parameters as in Figure 4.

5.2 Heavily Loaded Systems

A heavily loaded system will have many processes generating I/O requests. In this environment, each disk will have a queue of requests waiting to be served, and the system may serve those requests in any order it chooses. We model this environment for a range of queue lengths, and we evaluate disk performance under a variety of assumptions, primarily regarding data organization. We discuss the basic features common to all simulations, and then we present the results of our simulations.

While we do not expect systems to have disk queues as large as three hundred *on the average*, we believe it is important to study performance during extreme bursts of activity. These bursts may occur because of unusual application circumstances (e.g., the stock market is tumbling) or because of an earlier failure (e.g., a database system is redoing updates from a log). Performance during these critical periods may be even

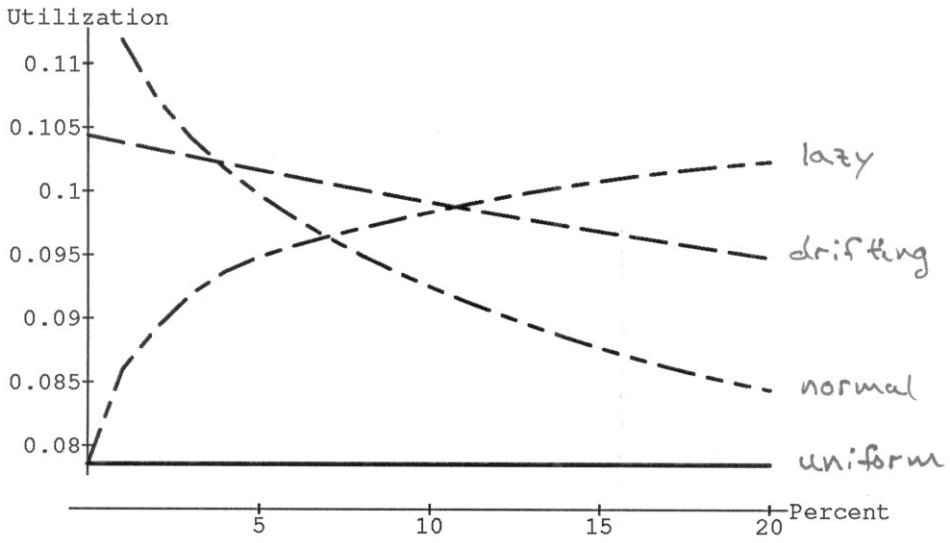

Figure 5: Expected Utilization

more important than "average" performance. Other researchers like [10] have looked at queues as large as one thousand requests arguing that although average queue lengths may be short in current systems, bursts of activity may generate lots of dirty buffers which can be viewed as part of the disk queue. As systems have larger buffer pools and faster processors, longer disk queues will become more common.

We present results for only one disk scheduling algorithm, the "elevator" algorithm, although we have analyzed several others. The "elevator" (or cyclical scan [10]) algorithm scans the disk in one direction servicing requests, and on reaching the edge of the disk it jumps back to the start to begin servicing requests again. In [10] this algorithm provided the best performance among those algorithms modeled which did not include knowledge of the rotational position of the disk.

In all our simulations we modeled the DEC RZ55 disk, which has 1224 cylinders, 15 tracks per cylinder, and 36 sectors per track. Because we do not have the seek cost function for the RZ55, we used the seek cost function described in equation 14. The RZ55 rotates at 3600 rpm, so the time for a single revolution, $T_{revolution}$, is 16.6 ms. We used a 2048 byte block because it is a common size used in UNIX file systems, and because it takes 1.85 ms to transfer the block, which is similar to the 2 ms transfer time for a 4096 byte block on the Fujitsu Eagle drive used in [10].

By reorganizing the data so the active data is near the center of the disk, we can increase the average number of accesses per seek. This means that the rotational delay becomes a dominant factor in the total access time. (If the average seek time is about 12 to 16 ms., the average rotational latency is 8.3 ms. and there is more than one request to be serviced in the cylinder, then the rotational latency dominates the overall delay). We can reduce the average rotational delay by ordering the requests by rotational position. In Equation 15 we show the total time required to service k requests in a single cylinder:

$$T = T_{seek} + T_{rotation} + T_{transfer} + T_{other} \tag{15}$$

Where T_{other} includes CPU overhead, communication overhead, and other hidden costs. For our purposes, we set T_{other} equal to zero. If requests within a cylinder are processed in random order (order of arrival), then the average rotation cost is:

$$T_{rotation} = \frac{k}{2} T_{revolution} \tag{16}$$

If requests are reordered according to rotational position on the disk, then the rotation cost can be

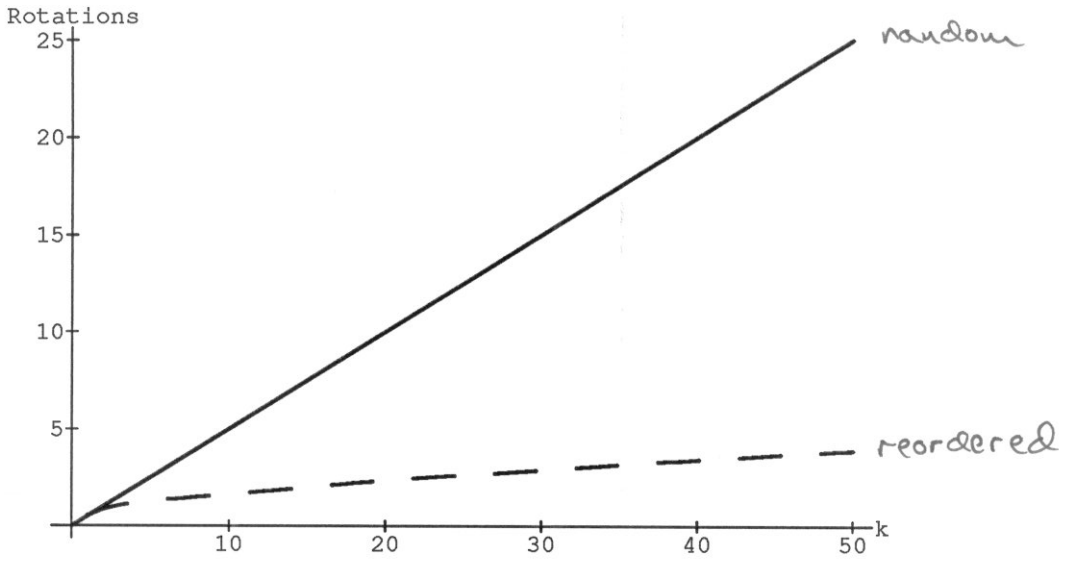

Figure 6: Expected Service Cost (in revolutions) for RZ55

reduced dramatically. The general idea is that the requests will be ordered according to their rotational position, regardless of their track. If there are two or more requests for a given rotational position, then there is a "collision". The maximum number of requests for any rotational position determines the number of full revolutions that will be required to process the requests. If there are two or more rotational positions with the maximum number of collisions, then their separation determines the last partial revolution needed to process all requests. In addition the system will pay an average $\frac{1}{2}$ rotations to read the first block.

Using the assumption that all blocks within a cylinder are equally likely to be accessed, we wrote a program to compute the expected rotational cost for processing k requests for a given disk geometry. We divide the problem into cases, counting the number of ways requests could be arranged (configurations) and the cost of each configuration. For example, to compute the average rotation cost for accessing two blocks we divide the problem in two cases: two blocks in the same rotational position but different tracks, and one block in each of two different rotational positions. The first case has rotation cost $R = 1 + \frac{1}{sectors}$ because the disk will get the first block, make a full revolution, and get the second block. There are $sectors \times \binom{tracks}{2}$ ways to arrange two requests in the same rotational position. For blocks at different rotational positions i and j such that $j - i = d$ for each d in $\{1, \ldots, sectors - 1\}$, the system would count $(sectors - d) \times \binom{tracks}{1}^2$ configurations with a rotation cost $R = \frac{d+1}{sectors}$. Also, there are a total of $\binom{tracks \times sectors}{2}$ ways of arranging two requests. From this we can compute the probability of each case, and hence the expected rotational delay. For more than two blocks, the program proceeds in a similar fashion, considering all cases. Incidentally, our program uses the *bignum* package developed at DEC PRL [14].

In Figure 6 we show the expected rotation cost (measured in revolutions) for servicing k requests to a particular cylinder for a DEC RZ55 disk. As the number of requests increases, the performance benefits of reordering the requests become dramatic, and with as few as four requests we see significant improvements.

Having computed the expected rotational delays, we used a simulator to measure expected system performance under a variety of circumstances. The simulator is given an initial cylinder access probability distribution f, a description of the disk geometry, and a queue length l. Since we wish to determine system performance for various queue lengths, we keep the queue lengths stable during each simulation run. The simulator generates l requests to initialize the queue, and as each cylinder is processed, it generates new requests to replace those requests processed, so there are again l requests on the queue. Individual requests are generated based on the input cylinder access probability distribution f (see Section 4) and subject to

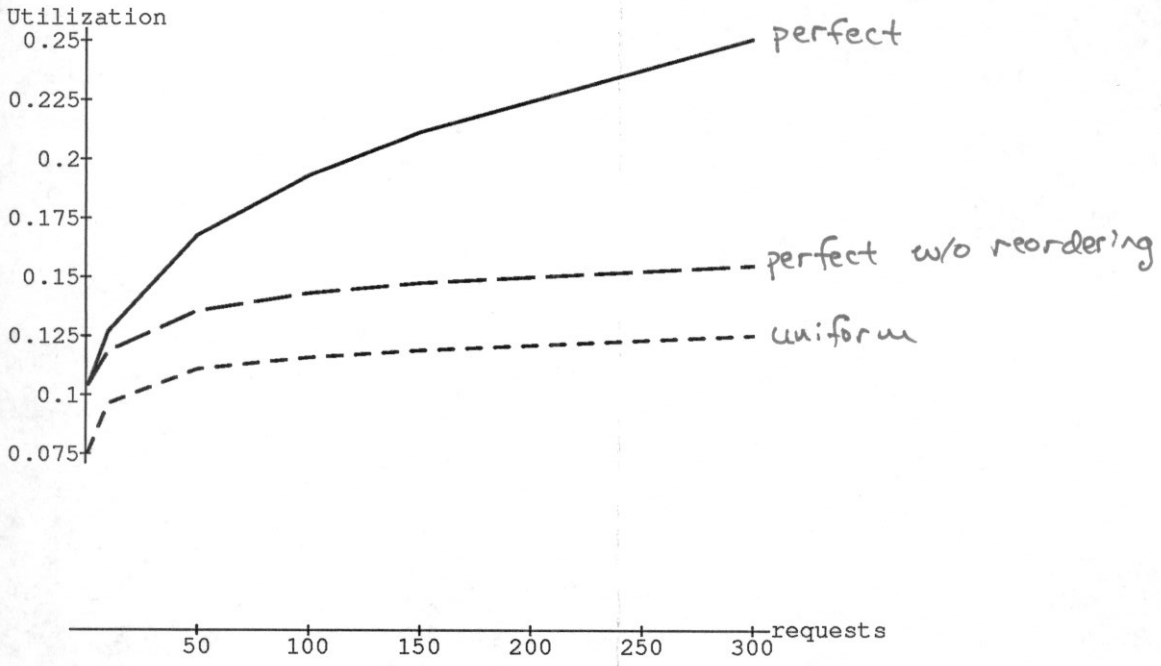

Figure 7: Average Disk Utilization

two constraints: each block within a cylinder is equally likely to be accessed, and each block can be in the queue at most once at any given time.

During operation the simulator chooses which cylinder to process ("elevator" algorithm), computes the time to service all requests to blocks in the cylinder (Figure 6), adds new requests to the queue, and goes on to the next cylinder. Once $100\,l$ requests have been added to the queue, the simulator stops generating new requests and finishes the simulation after processing the remainder of the queue. The disk utilization is computed using Equation 13, where the seek, rotation and transfer times are the total times to service all $100\,l$ requests. In addition, we repeated each experiment at least ten times (with different seeds) and averaged the resulting utilizations. (Standard deviations were quite small, less than three percent in most cases.)

In Figure 7 we show the simulated average disk utilization under three conditions for the "perfect" distribution (f_p) using various queue lengths. The middle curve shows system performance when the data has been clustered but requests within a cylinder are processed in the order of arrival. The benefits of data clustering alone provide roughly a twenty five percent improvement in system throughput. In addition, by reordering requests, we can roughly double the system throughput (with queue lengths of 300) over the base case of uniformly distributed data. However, with short queue lengths (less than 50 requests) we get throughput improvements between twenty five and fifty percent over the base case. While the benefits of clustering data together are evident at all queue lengths, the secondary effect of increasing the number of requests per cylinder dramatically improves performance as the system load increases. Note that in a heavily loaded system, relatively small improvements in utilization may dramatically reduce system response times because each of the jobs in the queue in front of a request is serviced faster, so the total benefit in response time is the sum of the reduction in service time of *all* the jobs in the queue before the request.

In Figure 8 we evaluate the impact of unused disk space on our results. We are interested in this because of the under-reporting on disk usage in the experimental results. We suspect that the active data comprises somewhere between ten and fifty percent of the total disk space available at the two sites "S" and "Z". We model this in f_c by concentrating all the I/O in a fraction ($k = 0.1, 0.5$) of the disk space, leaving essentially zero probability of accessing the remaining disk space. Figure 8 implies that if large fractions of the disk space are empty or unused, then we may obtain even more dramatic performance improvements than we indicate in the rest of this paper.

In Figure 9 we try to show the benefits that can be gained by reorganizing only part of the file system.

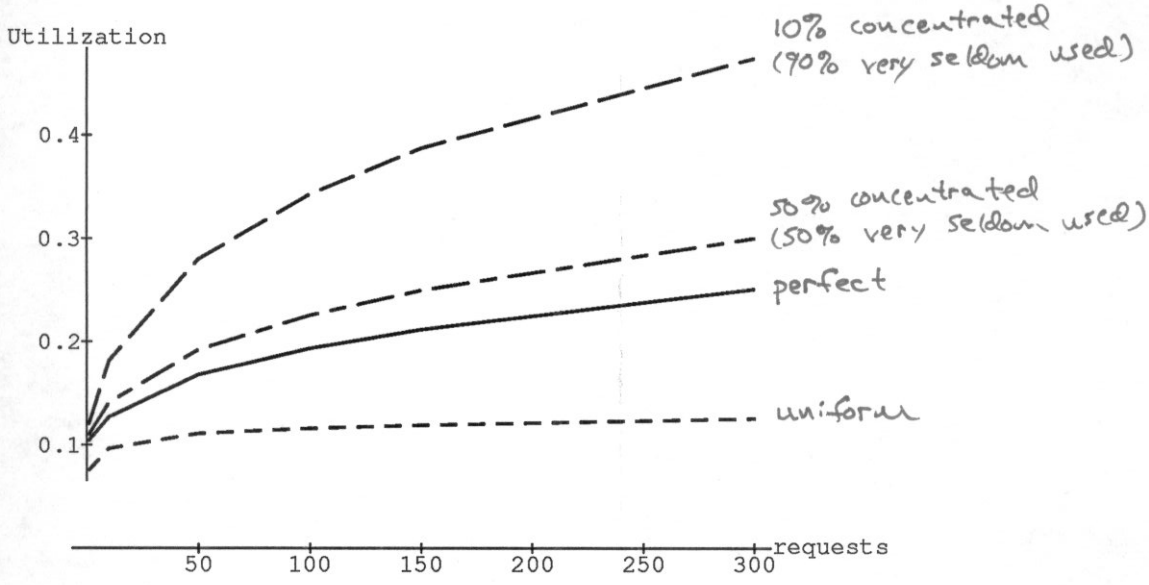

Figure 8: Concentrated Distribution: Average Disk Utilization

This is modeled in equation 9 as f_l. As one can see, by the time eight percent ($k = 0.08$) of the file system has been perfectly organized we have obtained most of the available performance benefits. But, almost as important, even reorganizing one percent of the file system can produce half of the available performance gain. This implies that only a small fraction of the total data in the file system need be reorganized to produce large performance improvements.

In Figure 10 we evaluate the *bucket* scheme described in Section 4 for placing files "near" their optimal location and modeled in equation 8 as f_b. We used variable size buckets, growing exponentially in size from the center to the edges of the disk. At the center there is a single bucket with 2^1 cylinders, one on each side of the center. The next bucket has 2^2 buckets, 2^1 on each side of the center. The number of buckets is a parameter b, and there are buckets with sizes $2^1, \ldots, 2^b$ and one bucket for the remaining space on the disk. The logic behind this approach is that it is far more important to get the hot files "exactly" right than it is to get the cold files "approximately" right. Note that for each disk there is a maximum number of buckets available due to the finite size of the disk, and in particular there are at most eight buckets for the RZ55 disk. In Figure 10 we see that with as few as three buckets on the RZ55, we gain roughly half of the available benefit derived from this reorganization, and that by the time we have six buckets we have obtained nearly all the available performance benefit. We conclude that using imperfect placement is a viable means for efficiently reorganizing the data, and that its performance is very close to that of perfect placement.

Next, we want to evaluate the impact of drifting file temperatures on disk performance. As we saw in Figure 2, while the majority of file I/O goes to files that were recently hot (over the last few days), some fraction of the I/O goes to files that were cold before. Since the reorganization schemes described use recent past file temperature to reorganize the file system, we must evaluate the impact of these drifting file temperatures. In Figure 11 we show the impact of drifting file temperatures using function f_d. We conclude that drifting file temperatures will reduce effectiveness of data reorganization, but not enough to nullify the performance benefits, since experimentally we observed from twenty to thirty percent drift over the course of a day. Also, we expect that the system will reorganize the data regularly, probably daily, so that the temperatures will not drift much more than that before the file system is reorganized again.

In all our simulations so far, we have used a block size of 2048 bytes, or four sectors. We want to show how our results change using different block sizes. We calculate the average speedup gained by clustering the active data in the center of the disk combined with reordering requests within a cylinder over the base case of data uniformly distributed over the disk and requests within a cylinder being processed in order of

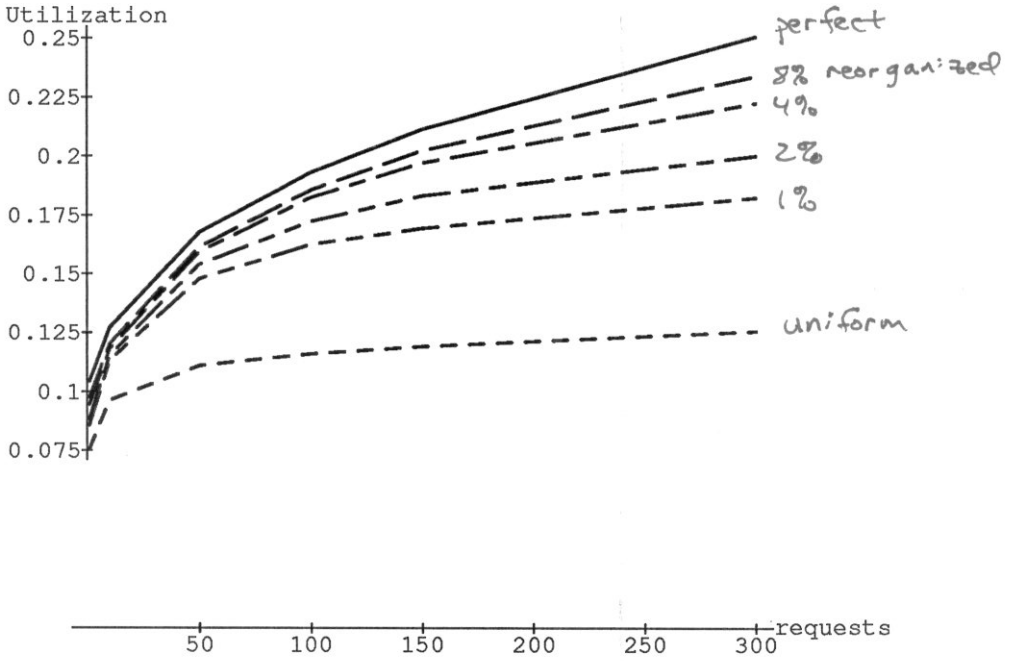

Figure 9: Lazy Reorganization: Average Disk Utilization

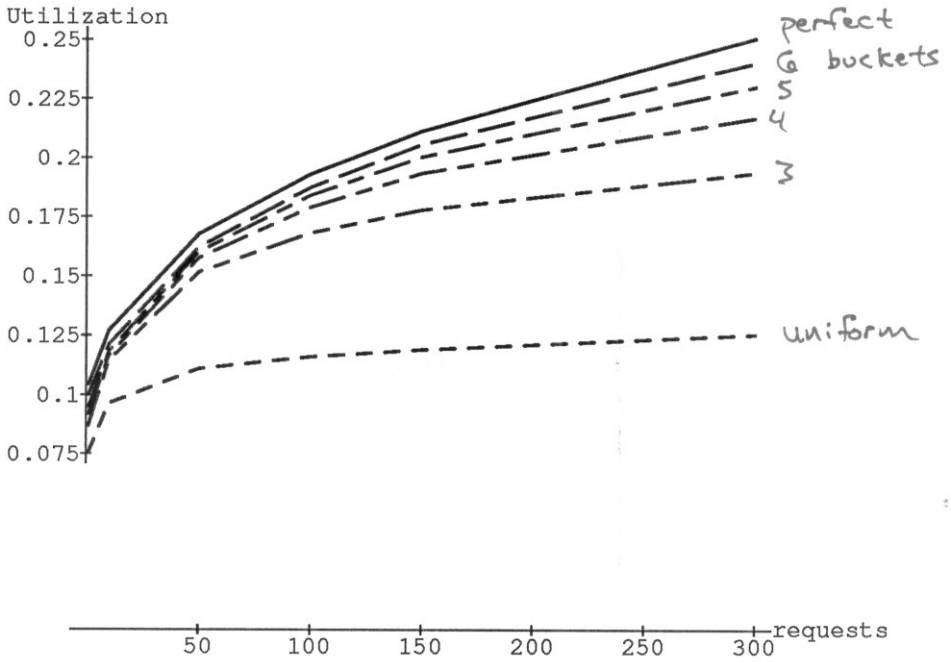

Figure 10: Bucket Reorganization: Average Disk Utilization

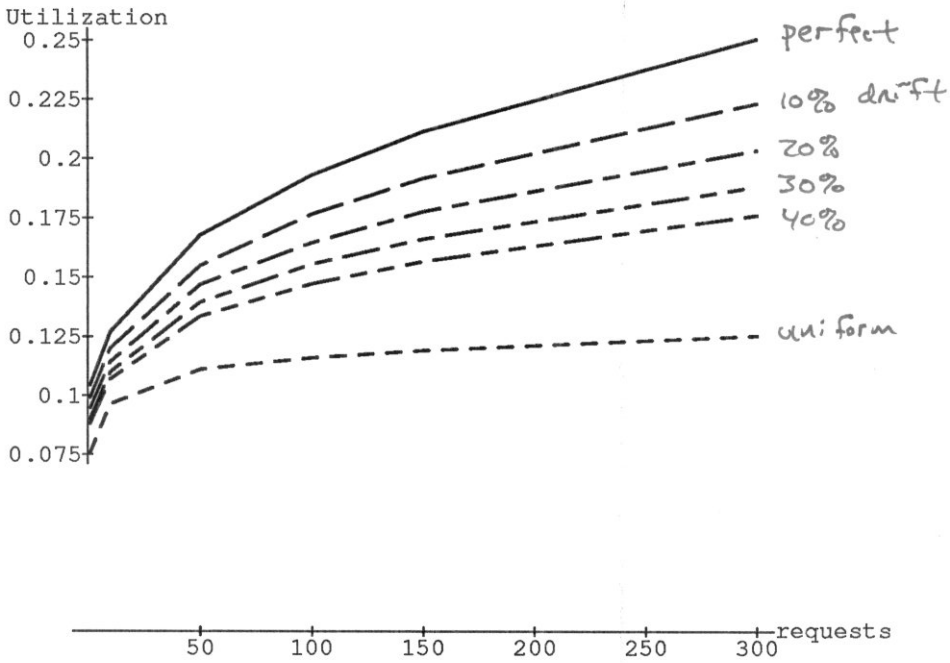

Figure 11: Average Disk Utilization, Drifting Temperature

arrival using:

$$Speedup = \frac{Utilization\,[clustering + reordering]}{Utilization\,[base\ case]}$$

In Figure 12 we show the speedup gained using the "perfect" distribution versus block sizes 512, 1024, 2048, 3072, 4608, and 9216 for several queue lengths. Note that the minimum speedup for block size 2048 is about 1.3. Also note that as the block size increases, the performance gains drop. This is primarily due to the reduced effectiveness of reordering requests within a cylinder because of the diminishing number of blocks (there are only two 9216 byte blocks within an RZ55 track). Consequently, as the number of sectors per track increases, performance for each block size will increase. For example, if we had a hypothetical new disk with 72 sectors per track, then a block size of 4096 on the new disk would correspond to a block size of 2048 on the RZ55, because they will both take 1.85 ms. to transfer a block. So one would find the expected speedup for this new disk with 4096 byte blocks by looking up the speedup of a 2048 byte block for the RZ55 disk in our results.

6 Conclusions

We have presented the analysis of a conceptually simple technique for dramatically improving disk performance. We have shown that the system need only reorganize a small percent of the file system to realize most of those performance gains, and that the reorganization may be imperfect — files need only be "near" their optimal location. We have also shown that thirty percent improvements in disk utilization are possible for lightly loaded systems, with improvements of over one hundred percent possible for heavily loaded systems. These predictions are based on the file access patterns measured in our experiments.

In our analysis we studied each factor (drift, buckets, ...) separately to understand its import. In reality, one may face combined factors. For instance, files may drift *and* they may be placed approximately into buckets. In this case, the losses in performance will be compounded. But in interpreting all this, one must keep in mind that our experimental data did not report unused files in the test period. As shown in Figure 8, this means all of our other results are pessimistic.

We have designed and are currently implementing a data reorganization facility, to be added to our

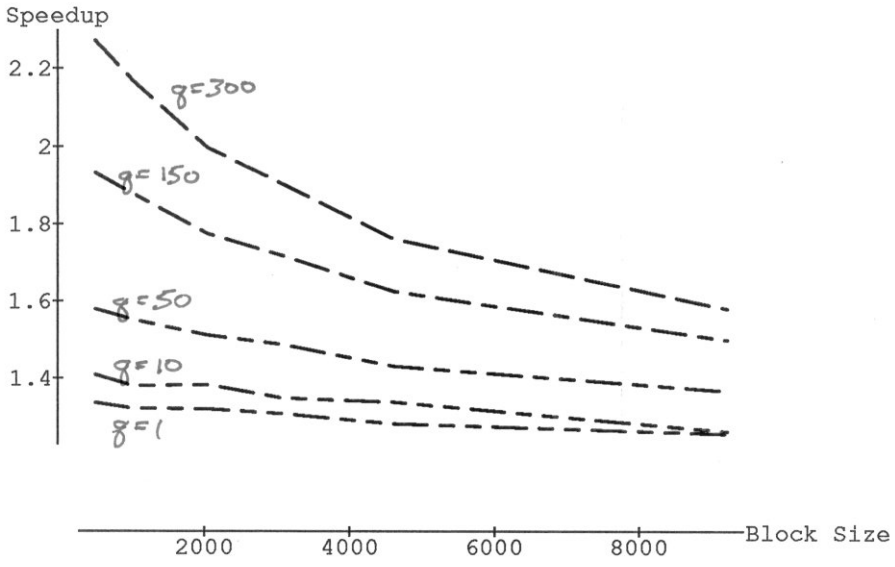

Figure 12: Average Speedup versus Block Size

iPcress File System prototype. There were several issues pertaining to a successful implementation of this technique: data reorganization must not affect normal use of the system, reorganization should be simple.

The underlying assumption of our strategy is that we may measure the activity for each file in the system, and have it at hand. In the iPcress File System, the file system keeps some simple statistics on file use for each file in the system. A good first statistic is the number of disk accesses generated "by" the file. Of course, this number must decay over time so that it will accurately reflect current file activity, so periodically the system divides all temperatures by two. In the future, we may evaluate the efficiency of various file temperature measures.

Our general strategy is to move data during idle periods, so as not to conflict with system use. Where appropriate, we also move data when it is going to be written to disk anyway. We try to place new files in the center of the disk, and as files cool over time, they will be gradually migrated towards the edge of the disk. In the event that a cold file gets hot, we may migrate it back to the center of the disk like a new file.

In order to simplify the task of reorganization we used the bucket system of Section 4. By using buckets we can reduce the time necessary to determine where files should be placed — we only have to determine a partial ordering of the files based on file temperature rather than a full ordering which requires sorting the files by temperature (on our departmental file server which supports about 150 users we had about 500,000 files). In addition, we can reorganize the data by having separate free lists for each bucket, so moving files simply entails allocating blocks from the appropriate free lists. Note that cold files in the center buckets should probably be moved out of the center first, freeing up space in the center.

7 Acknowledgements

We would like to thank Mordecai Golin, Christos Polyzois, Matthew Blaze, and Mark Greenstreet for their valuable comments. We would also like to thank Sigal Ar for her invaluable support and assistance. We would also like to thank Dick Wilmot, Dieter Gawlick and Amdahl Corporation for their time, assistance, and support.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by the Office of Naval Research under Contracts Nos. N00014-85-C-0456 and N00014-85-K-0465, and by the National Science Foundation under Cooperative Agreement No. DCR-8420948. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily

representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

References

[1] M. G. Baker, "DASD tuning — understanding the basics," in *Proceedings CMG*, Dec. 1989.

[2] C. Staelin and H. Garcia-Molina, "File system design using large memories," in *Proceedings Fifth Jerusalem Conference on Information Technology*, Oct. 1990.

[3] C. Staelin, "File access patterns," Tech. Rep. CS-TR-179-88, Department of Computer Science, Princeton University, Princeton, NJ 08540, Sept. 1988.

[4] R. Wilmot, "File usage patterns from SMF data: Highly skewed usage," in *Proceedings CMG*, Dec. 1989.

[5] K. Baclawski, S. H. Contractor, and R. Wilmot, "File access patterns: Self-similarity and time evolution." College of Computer Science, Northeaster University, 1989.

[6] M. Henley and I. Y. Liu, "Static vs dynamic management of consistently very active data sets," in *International Conference on Management and Performance Evaluation of Computer Systems*, CMG, Dec. 1987.

[7] J. K. Ousterhout, H. Da Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson, "A trace-driven analysis of the UNIX 4.2 BSD file system," in *Proceedings of the 10th ACM Symposium on Operating System Principles*, (New York, New York), pp. 15–24, ACM, 1985.

[8] R. Floyd, "Directory reference patterns in a UNIX environment," Tech. Rep. TR-179, Computer Science Department, University of Rochester, Rochester, NY 14627, Aug. 1986.

[9] R. Floyd, "Short-term file reference patterns in a UNIX environment," Tech. Rep. TR-177, Conputer Science Department, University of Rochester, Rochester, NY 14627, Mar. 1986.

[10] M. Seltzer, P. Chen, and J. Ousterhout, "Disk scheduling revisited," in *Proceedings Winter 1990 USENIX*, 1990.

[11] D. Bitton and J. Gray, "Disk shadowing," in *Proceedings of the 14th VLDB Conference*, (Los Angeles, California), 1988.

[12] R. P. King, "Disk arm movement in anticipation of future requests," Research Report, I.B.M. Thomas J. Watson Research Center, Dec. 1987.

[13] J. Wolf, "The placement optimization program: a practical solution to the DASD file assignment problem," technical report, I.B.M. Thomas J. Watson Research Center, 1987.

[14] B. Serpette, J. Vuillemin, and J.-C. Hervé, "BigNum: A portable and efficient package for arbitrary-precision arithmetic," Research Report 2, DEC Paris Research Laboratory, May 1989.