Disk Cache -
Miss Ratio Analysis and Design Considerations*

Alan Jay Smith**
University of California

Abstract

The current trend of computer system technology is towards cpus with rapidly increasing processing power and towards disk drives of rapidly increasing density, but with disk performance increasing very slowly if at all. The implication of these trends is that at some point the processing power of computer systems will be limited by the throughput of the input/output system.

The solution to this problem described and evaluated in this paper is Disk Cache. The idea is to buffer recently used portions of the disk address space in electronic storage. Empirically, it is shown that a large (e.g. 80% to 90%) fraction of all I/O requests are captured by a cache of reasonable (e.g. 8 Mbyte) size. This paper considers a number of design parameters for such a cache (called Cache Disk or Disk Cache), including those that can be examined experimentally (cache location, cache size, migration algorithms, block sizes, etc.) and others (access time, bandwidth, multipathing, technology, consistency, error recovery, etc.) for which we have no relevant data or experiments. We find that disk cache is a powerful means of extending the performance limits of high end computer systems.

---

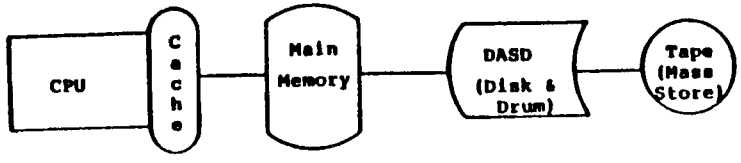
*Partial support for the research presented here has been provided by the National Science Foundation under grants MCS77-28429 and MCS82-02591, and by the Department of Energy under Contract DE-AC03-76SF-00515 (to the Stanford Linear Accelerator Center).

**Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, California, 94720.

I. Introduction

Computer systems have traditionally relied on a memory hierarchy (such as that in figure 1a), in which large amounts of less expensive storage (disk, tape) have been used to retain the bulk of the stored information, while small amounts of fast storage (main memory, cpu cache memory) have been employed to hold information while it is in active use. The problem in such systems has always been the large ratio in access times (the "access gap") between the slowest electronic storage (main memory, at less than one microsecond) and the fastest bulk storage (drum, at 5-10 milliseconds). The difficulty is that frequent accesses to bulk storage (either through impicit (paging) or explicit input/ output) may leave the cpu idle while the I/O request(s) complete. Multiprogramming is used in large and medium scale computer systems to overlap processing and I/O delays, but if all active programs are awaiting I/O, no processing can take place.

There exists a chain of reasoning which suggests that multiprogramming is limited in its ability to overlap input/output and cpu activity. The reasoning is as follows: (a) The speed of high end computer systems will continue to increase at a rate comparable to the recent past; i.e. doubling every 3-6 years. (b) Disk density will also continue to increase at a rate similar to the recent past: doubling every three or so years [Hark81]. (c) Disk access time will continue to improve only very slowly [Hoag79]. (d) The I/O rate associated with computer systems will remain roughly proportional to the instruction execution rate of the CPU [Amda70]. (e) The cost of large, high end disk spindles will not decrease significantly over the next few years, although sharp decreases in the cost per byte will occur. (f) The physical space required by large disk spindles will also not decrease significantly. (g) Therefore, the number of disk spindles in a large computer system will not increase (due to cost and space limits) as quickly as the I/O rate from the CPU(s). (h) The throughput of the I/O system is limited by the access time of the disks and the number of independent I/O paths. (i) The number of independent I/O paths is at most proportional to the number of disk



Typical Memory Hierarchy

Figure 1a

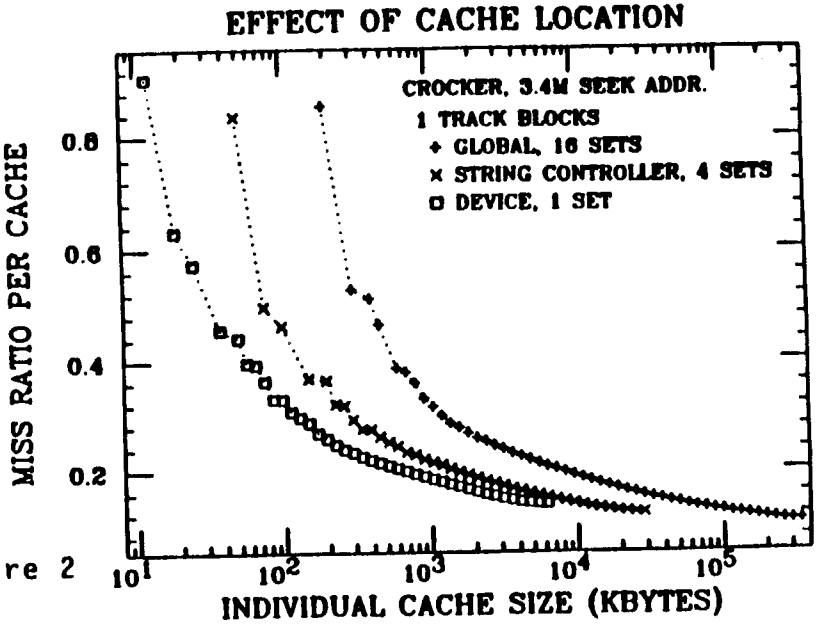
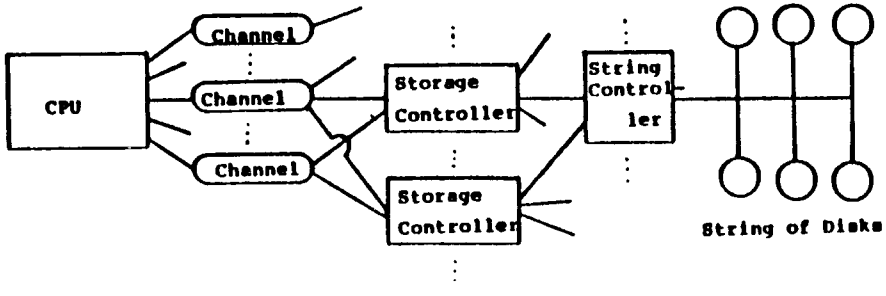


Figure 2



Part of Typical I/O Configuration

Figure 1b

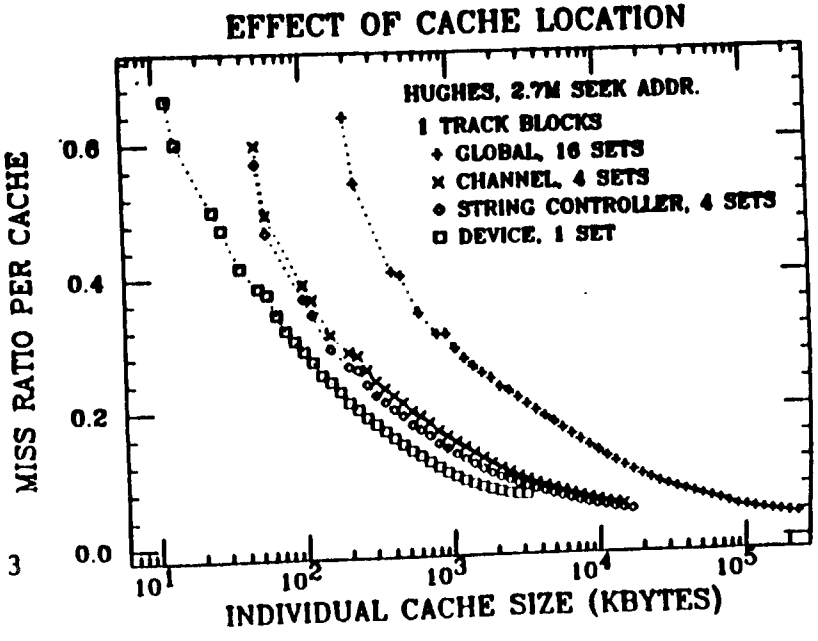


Figure 3

spindles. (j) Therefore, at some point the I/O system, no matter how carefully tuned, will be unable to service I/O requests as quickly as they can be generated by a fully utilized CPU.

This chain of reasoning, while indicating an eventual bottleneck, does not rule out changes in the software or hardware that postpone the problem. Ways of postponing the problem are discussed in section VI.

The mechanism proposed in this paper for coping with the I/O bottleneck is Cache Disk or Disk Cache. Disk Cache is a cache or buffer used to hold portions of the disk address space contents. If such a buffer can: (a) capture a significant fraction of the I/O operations, (b) without being too expensive, and (c) can provide access times and transfer rates significantly better than disk, then it can greatly improve I/O system performance and thereby postpone or eliminate the predicted I/O system bottleneck.

There is a reason to think that disk cache will be effective, and that is the long established Principle of Locality [Denn72], which describes most program reference behavior. This principle has two components: (a) Information which has been used recently is likely to be reused (or conversely, the information to be used in the near future is likely to consist primarily of information used in the recent past), and (b) information "near" the information in current use is likely to be used in the near future. The principle of locality accounts for the success of cache memories [Smit82] and main memory paging [Smit78a]. Because disk files are frequently reused (data bases, indexes, directories, etc.) and/or are read sequentially (most user files), we believe that the principle of locality also describes access patterns to the disk.

Locality in disk reference patterns has been previously observed (e.g. [Smit75], [Smit76]). It was suggested that multiple arms be used to access each of several open data sets on a given spindle in [Smit75] and the idea of a cache was proposed in [Smit78b]. A brief discussion of cache disk appears in [Smit81c]. The topic is also discussed in [Welc79a,b]. More recently, a number of papers have discussed and/or evaluated aspects of disk cache: [Bast82], [Buze82] and [Dods82]. None

of these, however, presents any miss ratio studies. Research and various industrial (unpublished) studies have been persuasive enough to lead a number of vendors to develop their own cache disk systems; included among them are IBM [IBM81a,b], Nippon Electric [Toku80a], Storage Technology [Cote82], and Memorex [Memo78].

In this paper, we will be concerned with the proper design, implementation, and operation of disk cache. There are a number of design considerations which merit attention: Where in the system shall the disk cache be placed? How large should/need the cache be, based on cost and performance? What migration algorithms (when to fetch or replace information) should be used? What block size is best? Should all or only some files/devices be cached? Which? Are the results time varying? Should the cache be turned on and off dynamically? What technology should be used for implementation? What are the error recovery considerations? Is there any impact on the rest of the system software? Each of these is discussed and/or evaluated later in this paper.

As explained in the next section, many of the aspects of cache disk design are sensitive to program behavior and file access patterns, and are therefore best evaluated empirically. We will discuss in that section (II) our evaluation methodology and the data that we have. Section III presents the results of our experiments, and in section IV we discuss those design considerations which are either not suitable for experiment or for which we have no relevant measurements. Section V considers briefly a related topic, disk arm buffers. Alternatives to disk cache are examined in section VI and current commercial products are described in section VII. An overview is provided in the conclusions section (VIII).

II. Methodology and Data

A. Cache Disk Effectiveness

The final measure of cache disk effectiveness is the change in the appropriate system performance measure (usually either response time or throughput) for a given cache disk system. The worth and desirability of cache disk must then be determined by taking into account the performance

improvement (if any) and the cost(s) involved. We shall not try to calculate in this paper the overall system performance impact of cache disk for several reasons. The primary reason is that translating the local effect of the disk cache on I/O times to the global effect on system performance is extremely sensitive to the detailed assumptions about the system configuration (number of disks, drums, string and storage controllers, their interconnections, etc.) and workload (I/O rate, distribution of I/Os to files and devices, etc.). Further, any given system can be tuned to some extent if a performance bottleneck is found. The appropriate performance measure is also an arbitrary one; the two that are used frequently are throughput and response time, but those two are not the same. Finally, an appropriate design point is heavily influenced by technology, which is rapidly changing. We do note, however, NEC [Toku80a], which has published some performance figures on an operating cache disk system.

It is possible to make some estimates of the effect of disk cache on mean disk I/O access times. That is done in [Buze82] where it is noted that disk cache may or may not yield any performance benefits, depending on the hit ratio and the design.

We shall instead evaluate the effectiveness of disk cache designs in two ways. First, we shall measure the miss ratio, which is the fraction of I/Os which are not captured by the disk cache, given certain disk cache parameters. As noted below, miss ratio measurements will be made using trace driven simulation. Low miss ratios can be expected to translate into higher system performance, since every hit to the disk cache results in an I/O time that is substantially less (e.g. 1-4 ms.) than would otherwise be required (10-100 ms.).

Our miss ratio measurements are limited in a number of aspects. Some information is not available from the trace data, as noted below, and some information depends on more than the sequence of I/Os; e.g. I/O path contention depends on the path configuration. Finally, some aspects, such as error recovery, are not suitable for miss ratio analysis. Thus, for topics for which miss ratios cannot be generated, discussions will be presented instead.

B. Trace Driven Analysis

Trace driven analysis is a powerful technique for evaluating aspects of computer systems. The idea is to trace a computer system, recording a sequence of events along with their relevant parameters (e.g. seek address, memory address, time, etc.). A variation of that computer system can be evaluated by using an event driven simulation in which the events are drawn from the trace rather than from random number generators. If the variation to be evaluated is such that the trace can be considered to be a valid sequence of events, then the simulation will indicate the behavior of the variant system. This technique has been used quite sucessfully to evaluate virtual memory systems [Bela66] and cache memories [Smit82] using virtual address traces, and cpu scheduling [Sher72] using cpu interval traces.

In this paper we use traces of I/O events taken from large computer systems to drive simulations of disk caches. The sequence of I/O events generated should be only slightly sensitive to the actions of the disk cache (which itself only changes the time for an I/O to complete), and therefore we believe that the miss ratio analysis presented is valid and accurate.

C. Data Sources

Three large IBM (or compatible) computer systems were traced for periods of 17 to 23 hours. The operating systems, as noted below, were variations of OS (OS/MVT, SVS and MVS); the primary data collection tool was GTF [IBM76], IBM's Generalized Trace Facility. GTF can be activated on the occurrence of most system interrupts, including supervisor calls (SVCs), I/O starts (SIOs), and I/O completions (I/O interrupts). The results presented in this paper are based mainly on a GTF generated trace of data references (seek addresses) as derived from the SIO events. Each trace record includes the device address, and the physical location of the block, consisting of the cylinder and track, for a direct address device (DASD = disk and drum). The record number on the track is also available, but since the block sizes are not always known, record numbers have not been used. Therefore, the smallest unit of storage in the cache disk

designs studied is the track. In all cases, only those input/output events directed to disk or drum devices were considered for caching. Our analysis (with the exception of some summary tables) uses only those DASD I/Os.

The three systems traced were located at the Stanford Linear Accelerator Center (SLAC), Crocker Bank and Hughes Aircraft. Each is briefly described below.

The SLAC computer installation at the time of tracing consisted of two IBM 370/168s and one IBM 360/91, connected via channel-to-channel adapters. The entire system was controlled by ASP version 3.1, and the CPU measured, one of the 370/168s, ran OS/VS2 release 1.6, otherwise known as SVS. The processor measured was the "support" processor, and was responsible for all unit record devices, spooling, the text editing and job entry system (Wylbur [Fajm73]), the time sharing system (Orvyl), and some portion of the batch workload; the other two machines were used as batch worker machines. The I/O configuration consisted of 16 IBM 3330 disks (@100 Mb), 27 IBM 2314 disks (@29 megabytes), two IBM 2305 drums, many tape drives, and numerous unit record and low activity devices.

The Crocker Bank computer system had two IBM 370/168s, which were connected only in that they shared all of the I/O devices. The processor traced ran TSO and small batch jobs during the day; at night it was mostly used for batch production work including bank transaction processing, business data processing and reporting. The operating system was OS/VS2 release 3.7, otherwise known as MVS, with JES2. The time sharing and text editing system was TSO using IBM 3277 terminals and SPF (a full screen editor). Cobol was used heavily with some PL/I and assembler use as well. The I/O configuration consisted of 25 3350 disks (@317 megabytes), 16 3330-11 disks (@200 megabytes), 7 3330 disks (@100 megabytes), and numerous tape drives, unit record devices and telecommunications controllers.

The third system traced was at the Hughes Aircraft Company, and consisted of an Amdahl 470V/6 and an IBM 370/165, loosely coupled via the ASP system. The installation was the central corporate computer center

for Hughes and ran a variety of work; the machine traced (the 470V/6) ran TSO, business data processing and production scientific, representing administrative, scientific, development and engineering support applications workloads. The operating system was OS/MVT release 21.8. The I/O configuration consists of 49 IBM 3330s, 9 3330-11s, STC Superdisks equivalent to 16 3330-11s, one IBM 2305-2, 24 tape drives, plus communications lines and numerous unit record devices.

D. Data Reduction

As noted above, the primary data collection tool was GTF. Each GTF record (after, in one case, a modification to GTF) contained the "seek address" for that I/O; i.e. the device address and track and cylinder location. Also used, with some modification, was IBM's System Management Facility (SMF) [IBM77b] which generates a record for every open and close of every data set. By combining SMF and GTF data and some partial device maps it was possible to tag each I/O as to the type of file (system, paging, or other) and the type of user (system, batch program, interactive system (TSO or Wylbur)), where the "user" is the cause of the I/O. This data reduction effort (described here so briefly) was immense and required man years of effort (see Acknowledgements). Further, the amount of data gathered is very large; a one day trace consists of about 1.5 gigabytes of data, or about 10 full reels of 6250 bpi tape.

The data generated had two important omissions: first, the location of the block referenced by each I/O within the track was not known; therefore the smallest block size used in any cache disk simulation is one track. The track size, of course, varies with the device. Second, I/O events were not tagged as to read or write; therefore, measurements or studies which depend on knowing whether an event is a read or write were not possible and were not done. (Much more extensive system modifications would have been required to obtain this information.)

We also note here that the large amount of data meant that not all experiments were run on all complete traces. In particular, for many of the SLAC and Crocker data analysis runs, a trace of one million I/Os (to all devices), from a daytime period, was selected for analysis. Since

non-DASD I/Os were discarded, the number of I/Os used was respectively 673K and 835K; see table I.

E. LRU Stack Analysis and Set Associative Mapping

Almost all of our disk cache simulations use a technique known as LRU stack analysis [Matt70]; we assume that the reader is familiar with that technique. A couple of special aspects of our analysis are worth pointing out, however. First, almost all simulations used set associative mapping (see e.g. [Smit82]), to reduce the mean stack depth; this technique is actually used in some commercial implementations of cache disk with which the author is familiar. The number of sets used is shown in all plots and tables. Experiments were run (but are not presented here for brevity) that showed that the effect of the number of sets was very small for realistic disk cache sizes. Second, we note that in many cases, multiple caches were used. For example, simulations were run showing separate caches in each device, string controller or channel. In other cases, separate caches were simulated for each user and file type combination.

F. Simple Data Characterization

In table I we present a number of simple statistics and figures that characterize the measurements from the three sites. Most of the numbers presented in table I are self explanatory. We note, however, the row labeled "fraction seeks." That set of figures gives the fraction of all SIOs which resulted in a seek taking place. As has been previously noted (e.g. [Smit75], [Lync72]), half or less of all I/Os tend to require disk arm movement.

Table II shows the fraction of all DASD I/O events that can be attributed to various File type and User type combinations. We note the following points. First, the system (operating system and associated system software, including communications, job entry system, etc.) accounts for the largest fraction of the I/Os. This tends to surprise many people, who perceive system operation as a reflection of the workload that they directly generate, and not that induced indirectly. We also note the small number of paging events at SLAC. The SLAC system is generally run with a large enough memory and a small enough degree of

Systems Studied

Site: / Type of Data	SLAC	Crocker	Hughes
Operating System	SVS	MVS	MVT
CPU	370/168	370/168	470V/6
Trace Period	5pm-4pm	5pm-12noon	6pm-11am
I/Os(total)	5491891	5367267	3800459
I/Os(DASD)	3671121	3387641	2731973
I/Os(DASD-short period)	673307	835236	-
I/Os(total)/second	66.3	78.5	62.1
I/Os(DASD)/second	44.3	49.5	44.6
CPU MIPS	2.8	2.8	4.2
fraction seeks	.540	.355	.411

Table I

Fraction of I/Os by File and User Type

File/User Type	SLAC	Crocker
Temporary/All	.0311	.130
System/Batch	.0564	.357
Other/Batch	.0367	.192
System/TSO|Wylbur	.4054	.068
Other/TSO|Wylbur	---	.015
System/System	.4563	.167
Paging/System	.0136	.072
Other/System	.0005	---

Table II

Distribution of I/Os Among Device Types

Device Type&Site	Fraction DASD I/Os	Fraction Total I/Os
Crocker Bank		
3330-1	.053	.034
3330-11	.278	.175
3350	.669	.422
tape	---	.310
other	---	.059
Hughes		
3330-1	.562	.404
3330-11	.154	.111
2305	.172	.124
superdisk	.111	.080
tape	---	.108
other	---	.173
Slac		
3330-1	.524	.350
2314	.190	.127
2305	.284	.190
tape	---	.0518
other	---	.281

Table III

multiprogramming that paging is relatively infrequent; further, the use of the SVS operating system, which allows only a total address space of 16 megabytes among all processes, restricts the overcommittment of memory.

Table III shows the distribution of activity among the device types. It is of interest for two reasons. First, we note that in all cases, the bulk of the activity is directed toward the newest and fastest disks. Second, the proportion of I/O directed toward the tapes varies a great deal and varies from 5% to 30%.

III. Disk Cache Miss Ratio Analysis

As noted earlier, there are a number of aspects of disk cache design which can be usefully examined by trace driven miss ratio analysis. In this section we present the results of our analysis, with attention to parameters and issues such as: how large should the cache be? Where should it be placed (cpu, channel, controller, spindle)? What should the block size be? What migration algorithm(s) are best? Should all or only some devices be cached? Should caching be restricted to only some types of files or users? Are there time of day aspects to the effectiveness of disk cache? Each of these items, and others, are considered below.

A. Cache Capacity

Perhaps the most basic aspect of cache design is the cache size; how large it should be in order to obtain a given hit ratio. In figures 2, 3, and 4 we show the miss ratio for caches located globally, in each string controller, in each disk spindle and (for Hughes only) in each channel connected to DASD. In each case the block size is one track, the results are based on the seek address trace for the full measurement period, the write algorithm is copy back, and the number of sets (using set associative mapping for the LRU stack simulation) is as indicated. The curves for the channel, controller and device caches are the miss ratio per cache, and the capacities must be multiplied by the number of caches to get comparable figures.

As may be seen from the data presented in figures 2, 3 and 4, the miss ratios for the global cache are already 26% or less at 2 megabytes.

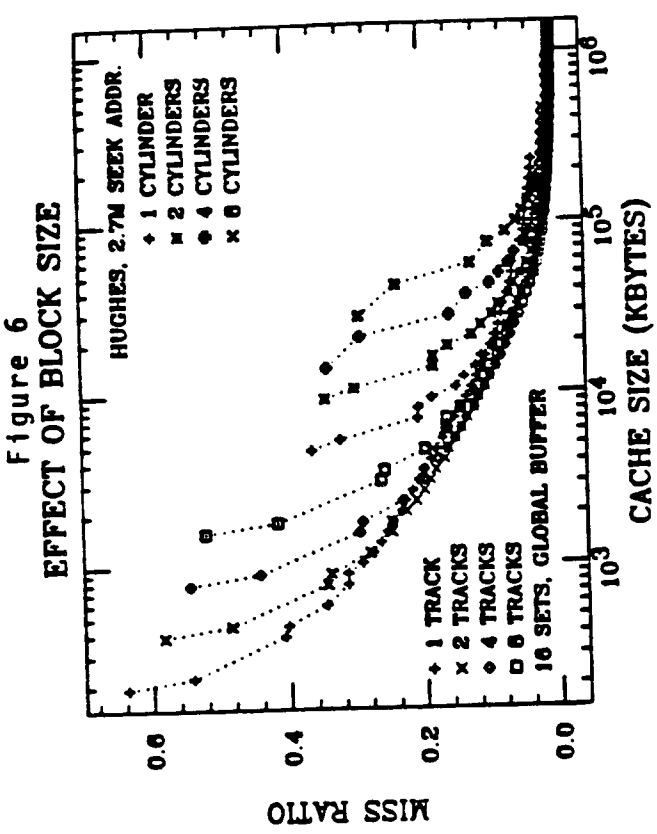


Figure 6
EFFECT OF BLOCK SIZE
HUGHES, 2.7M SEEK ADDR.
+ 1 CYLINDER
× 2 CYLINDERS
* 4 CYLINDERS
× 8 CYLINDERS
+ 1 TRACK
× 2 TRACKS
◇ 4 TRACKS
□ 8 TRACKS
16 SETS, GLOBAL BUFFER
MISS RATIO
CACHE SIZE (KBYTES)

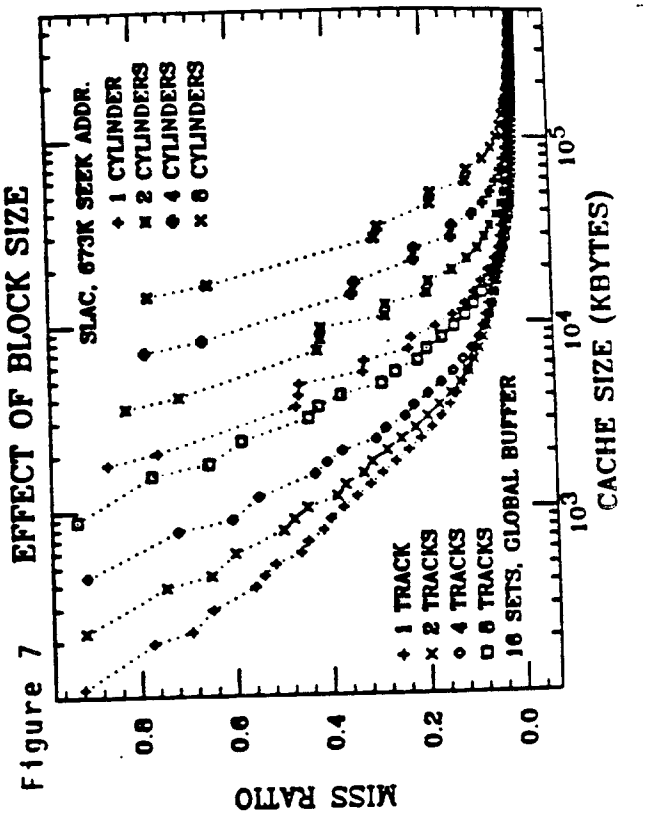


Figure 7 EFFECT OF BLOCK SIZE
SLAC, 673K SEEK ADDR.
+ 1 CYLINDER
× 2 CYLINDERS
* 4 CYLINDERS
× 8 CYLINDERS
+ 1 TRACK
× 2 TRACKS
◇ 4 TRACKS
□ 8 TRACKS
16 SETS, GLOBAL BUFFER
MISS RATIO
CACHE SIZE (KBYTES)

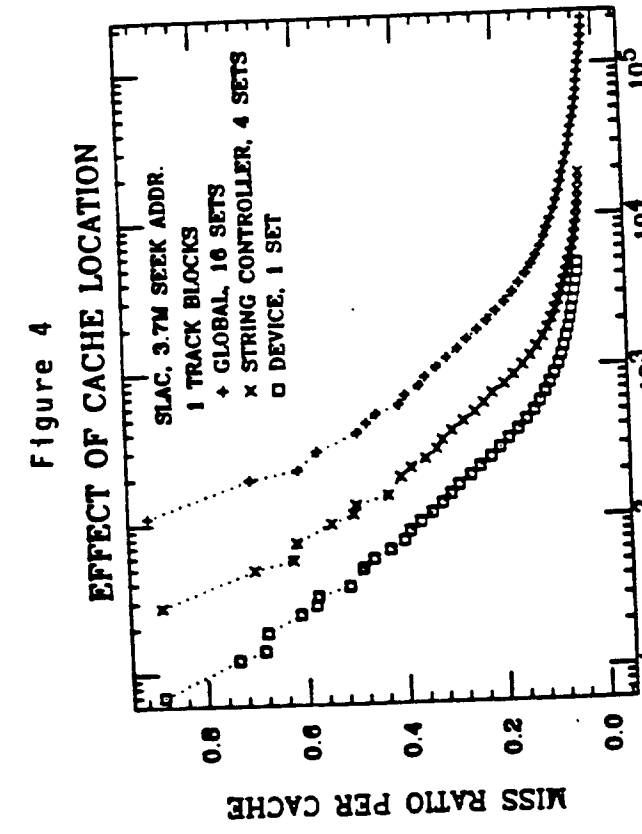
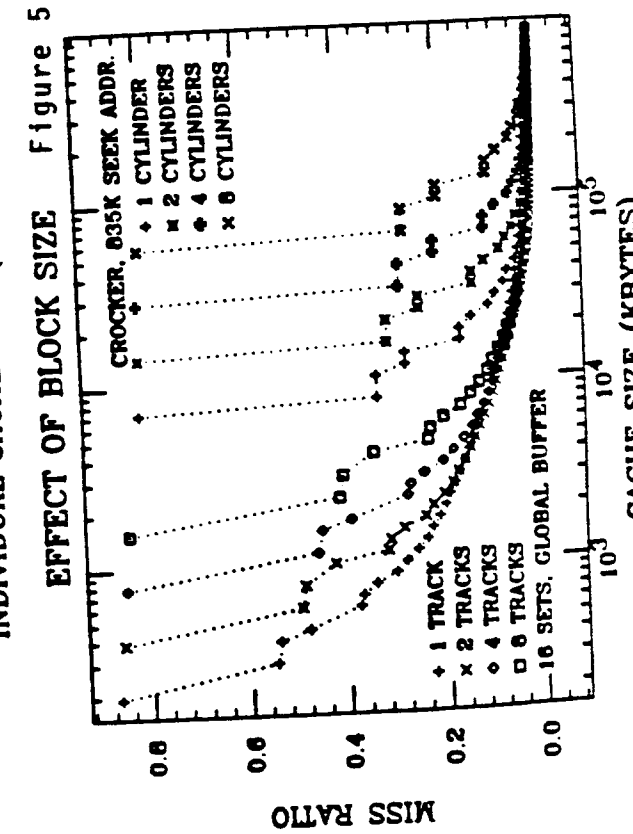


Figure 4
EFFECT OF CACHE LOCATION
SLAC, 3.7M SEEK ADDR.
1 TRACK BLOCKS
+ GLOBAL, 16 SETS
× STRING CONTROLLER, 4 SETS
□ DEVICE, 1 SET
MISS RATIO PER CACHE
INDIVIDUAL CACHE SIZE (KBYTES)


EFFECT OF BLOCK SIZE Figure 5
CROCKER, 835K SEEK ADDR.
+ 1 CYLINDER
× 2 CYLINDERS
* 4 CYLINDERS
× 8 CYLINDERS
+ 1 TRACK
× 2 TRACKS
◇ 4 TRACKS
□ 8 TRACKS
16 SETS, GLOBAL BUFFER
MISS RATIO
CACHE SIZE (KBYTES)

Improvement is apparent beyond this point, but each successive doubling of the overall global cache size yields only a small improvement. On the basis of these figures, from 2 to 8 megabytes of total cache capacity appears to be adequate when there is one global cache, and the machine is an IBM 370/168 or similar. (Faster processors, of course, would access more data per unit time and would need a larger cache.)

It should be pointed out that the miss ratios observed in the three sites are significantly different; SLAC has a relatively small disk system and shows high cache effectiveness. The Crocker Bank system shows the worst performance, perhaps because of the large data bases kept on- line.

An alternative to buffering globally is to place the buffer in the channel, string controller or disk spindle. Those measurements are also shown in figures 2, 3 and 4. Examining the controller miss ratios, we again see significant differences between the systems. It appears, as a generalization, that capacities of from 512K to 2Mbytes per string controller seem to give good results. At the device level, 256K to 512K seem to be needed.

B. Cache Location: hit ratio, consistency and cost.

A disk cache can be placed in any convenient location along the data path between the CPU and the disk surface (see figure 1b). In an IBM (or similar) system, that suggests a number of reasonable locations: (a) A global cache, at the cpu, either in main memory or outboard. (b) A cache associated with each channel or with a group of channels (e.g. with the storage director). (c) In/with a storage controller or group of storage controllers. (d) In/with the string controller. (e) In/with each device. There are a number of considerations in choosing between these possibilities including miss ratio, data consistency, and cost.

The closer a cache is to the cpu, the more it may be shared by a number of I/O devices. That is, if disk x is active at one time, and disk y at another, they can use the same cache if it is along each of their data paths to the cpu. The data paths from the cpu resemble for the most part a tree, so sharing is enhanced by buffering near the cpu. (Exceptions include the fact that storage controllers can connect to more

than one channel and disks to more than one string controller (for some vendors).)

In table IV we show the number of DASD devices, string controllers (for DASD) and channels (connected to DASD) for each system measured. Multiplying those numbers by the suggested capacities noted in section IV.A above, we find that the total capacity required for a given miss ratio is much larger for device caches than for string controller caches, and larger for controller caches than for a global cache. Those global figures appear in table V. The reason for this phenomenon is that I/O loads are unbalanced both statically and dynamically between devices and strings. By that, we mean that over short periods of time (e.g. minutes) some devices are much more active than others (dynamic imbalance) and over long periods of time (days, weeks), some devices are still much more busy. For example, in [Bast81] it is stated that over moderate to short periods of time 60% of the I/Os may go to two devices. It is impossible to balance devices over short periods of time and even over long periods, only very approximate balance is possible. Thus, much lower miss ratios can be expected for caches closer to the cpu than near the periphery. (Of course, is there are multiple cpus, then a cache near the device will be shared by the cpus, with corresponding efficiency considerations.)

Consistency is a second important issue. If there can be more than one copy of a given piece of information, all copies must be kept consistent. The easiest way of doing this is to make sure that all accesses to a given disk spindle pass through the same cache buffer. Since a given device can be reached via more than one channel, storage or string controller (if the system is so wired), a unique path is guaranteed only if the cache is at the device; otherwise explicit steps must be taken to maintain consistency. A survey of methods for maintaining cache consistency appears in [Smit82] where CPU caches are discussed. See also section IV.F.

The third issue, cost, would almost certainly be minimized by minimizing the number of different caches in the system, given the same hit ratio or total storage capacity. That is, the increased size and

Site:	System Configuration Crocker	Hughes	SLAC
System Aspect			
DASD	48	63	45
Strings	12	18	13
Channels to DASD	4	10	7

Table IV

Total Capacity	Miss Ratios Crocker Global	Crocker Controller	Crocker Device	SLAC Global	SLAC Controller	SLAC Device
1Mb	.316	.475	.61	.330	.601	.630
2Mb	.259	.365	.45	.226	.414	.496
4Mb	.225	.275	.330	.146	.326	.370
8Mb	.197	.233	.266	.099	.227	.271
16Mb	.172	.203	.224	.070	.136	.174
32Mb.	.150	.177	.199	.050	.085	.109
64Mb	.139	.155	.175	.033	.061	.071

Table V

Lowest Miss Ratio Block Size in Tracks
Global Set Associative Buffering, 16 Sets

Capacity	Crocker	Hughes	SLAC
1MB	1	1	1
2MB	1	2	1
4MB	2	2	1
8MB	4	2	2
16MB	4	4	4
32MB	4	4	4
64MB	8	8	8

Table VI

Effect of Time Period on Miss Ratio
Crocker Bank, 1 Track Blocks,
Global Buffering, 16 Sets

Trace Section: Cache Size(Megabytes)	0-1M	1-2M	2-3M	3-4M	4-5M
1	.313	.353	.313	.319	.303
2	.240	.298	.281	.292	.219
4	.195	.271	.262	.273	.167
8	.158	.258	.249	.260	.123
16	.126	.247	.238	.249	.085
32	.096	.236	.227	.237	.056
64	.076	.217	.212	.230	.038
128	.062	.183	.195	.224	.026
256	.057	.164	.163	.216	.020
512	.056	.163	.159	.215	.019
Number of Disk I/Os	635984	517691	452527	646051	830398

Table VII

complexity of a shared cache would not likely be as costly as replicating a simpler cache. This suggests that placing the cache in the disk spindle might not be cost effective.

The placement of the cache affects the amount of I/O overhead. If a cache is placed in the main memory, for example, a cache hit can bypass the entire I/O system, including some operating system routines, the channel and storage controller, etc. Placing the controller beyond the channel means that no operating system or channel time is eliminated; the latter is quite significant [Hunt80] and can typically be twice as large as the data transfer time.

Based on the above discussion, it is not possible to specify a uniquely best location for a cache. Some vendors have chosen to place it in the storage controller (IBM [IBM81a,b], Memorex [Memo78]), outboard at the cpu (NEC [Toku80a]), or inboard, with the cpu main memory [Smit81d].

C. Block Size

An important aspect of any cache is the size of the block used. For reasons noted earlier, we have examined block sizes of 1 track and its multiples, specifically 1, 2, 4 and 8 tracks, and 1, 2, 4 and 8 cylinders. (Cylinder and track sizes vary between devices, of course.) Miss ratios for all three sites and for those 8 different block sizes are shown in figures 5, 6 and 7. It can be seen in those figures that for small cache capacities, small block sizes give the lowest miss ratio. For larger cache sizes, lower miss ratios are obtained with larger block sizes. In table VI we show the block sizes that give the minimum miss ratios at a given total (global) cache capacity.

The reason for the behavior displayed in figures 5, 6 and 7 is as follows: For small cache sizes, small blocks permit several pieces of information to be in the cache at once. If the blocks are large, the several active blocks must be swapped in and out frequently, rather than being coresident. If the cache becomes larger, then several large blocks can be resident at once. In this case, fetching a lot of information with a large block results in a smaller miss ratio than fetching that information in pieces using smaller blocks.

Given, as noted before, that the appropriate buffering capacity is from 2 to 8 megabytes, block sizes of from 1 to 4 tracks yield minimum miss ratios. This observation, however, is somewhat misleading. Larger block sizes involve some other costs, the most important of which is that the very large transfer time for a multitrack block will tie up the device and data path for a very long time. If the block must be cached before it is accessed, then the latency for a large block can be a significant penalty. (See e.g. [Buze82].) Also, physical disk blocks need not end on a cylinder boundary, which will either result in a block which is partially full, an odd size block, or a block that must wait for a seek in the middle of being read or written. Since larger block sizes can be effectively simulated by prefetching, and since the miss ratio differences involved are small, it seems clear that a 1 track block size is best. (See [Duke82b] for a discussion of the problem of multitrack physical blocks.)

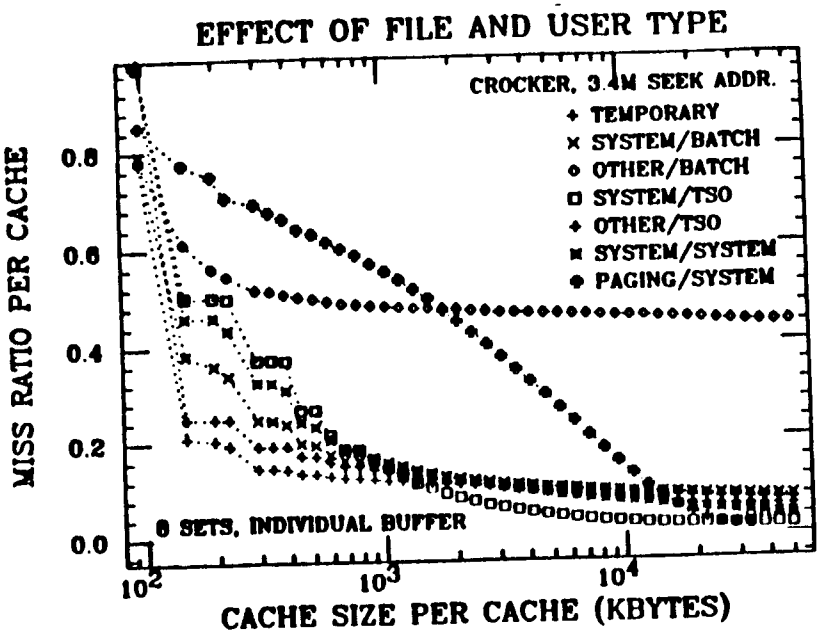
D. Migration Algorithms

1. Selective Fetch - User Type and File Type

It is very easy to imagine situations in which disk cache would be very effective, and other situations in which it would be totally ineffective; some such have been previously mentioned. For that reason, we have classified the "users" (source of an I/O request) by type (system, interactive (TSO or Wylbur), batch job) and the files by type (temporary, system, paging, other). Miss ratios were collected separately for the following classes: temporary files, and then all combinations of user and file types (excluding temporary). Each such class was buffered (globally) in its own separate cache (which means that if a given track was accessed by two different types of user, it appeared in two different caches). The miss ratios for each class for one track blocks appear in figures 8 and 9; data for the Hughes system was not classified by user and file type. From those figures (and other tables, not shown), we make the following observations:

(a) The paging data set(s) appear to show very little locality; i.e. the miss ratio doesn't drop significantly until most of the paging data

Figure 8



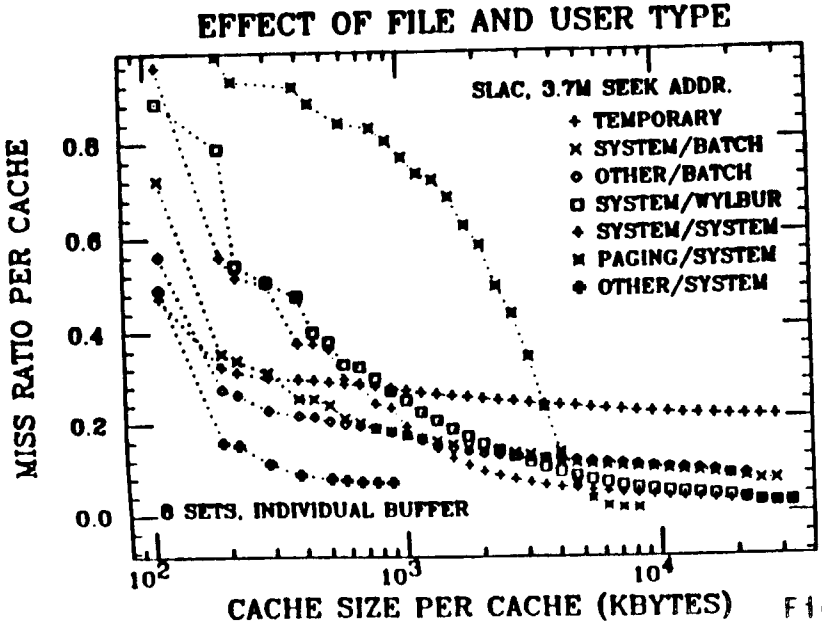
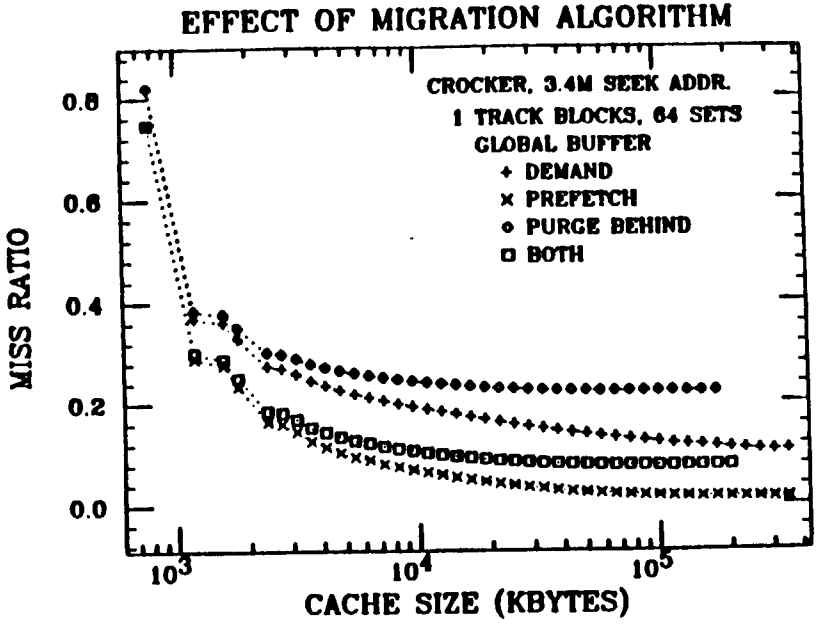


Figure 9

Figure 10



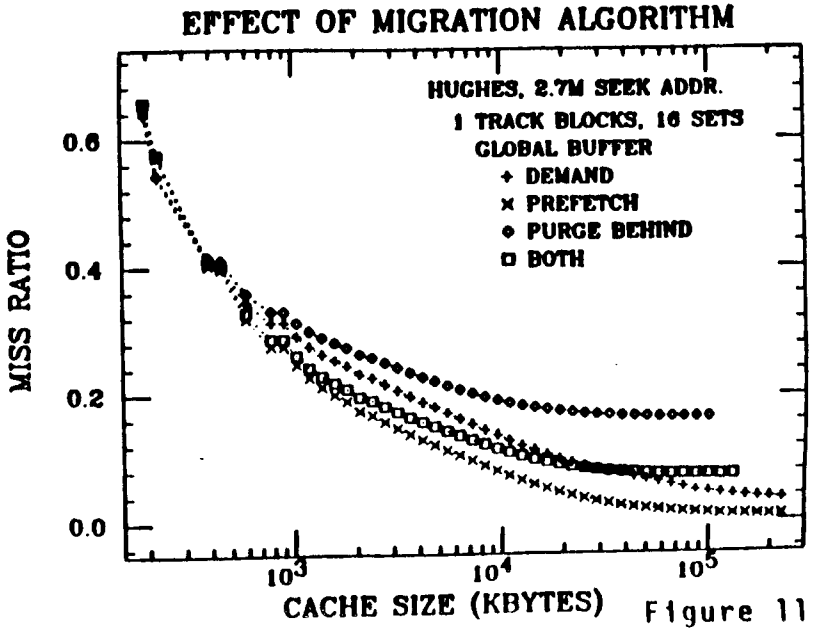


Figure 11

set is in the buffer. Since we are buffering using the physical disk addrss, we know nothing about the logical page name. Therefore, we are able to say very little about the use of gap filler technology for paging store based on this data. (It is possible that the logical addresses would show either more or less locality, but we have no way of knowing.) From our other data (not presented), it is also notable that increasing the block size does not seem to help reduce the paging data set miss ratio.

(b) Slac and Crocker show different comparative results for the effectiveness of buffering batch vs. buffering system data sets. The Slac data suggests that temporary data sets show poor locality compared to batch data sets. The Crocker data shows that the batch data set locality is poorer than the system and temporary data set locality. The first shift results at Crocker, however, suggest that batch and system data sets have roughly equal hit ratios when cached. (The workload at Crocker varies widely with the time of day; that is much less true at Slac.) The interactive system miss ratios are not markedly different from the other system miss ratios. On the basis of these measurements, neither batch or system data sets merit a consistent preference for buffering. The large variance between SLAC and Crocker, however, suggests that on some systems, a preference might be useful.

(c) It is worth noting that the batch and temporary file miss ratios drop rapidly with increasing block size (not shown), whereas the same phenomenon is not found for the other types of data sets. This suggests, as one would expect, that such files are accessed sequentially. Therefore, prefetching might be a good strategy for batch and temporary files, possibly instead of more buffering.

2. Prefetching

The standard method of moving data into a cache is called "demand fetch", by which data is moved into the cache at the time it is first referenced; thus demand misses result in a latency period during which the desired information is not available. If it were possible to accurately guess which disk blocks would be needed in the immediate future, those

blocks could be prefetched (i.e. fetched in advance) and they would therefore be available when referenced. Some quite sophisticated methods of prefetching have been devised; see for example [Smit78d] for prefetching in database systems, and [Smit78e] for prefetching to cache and main memory. (See also [Benn81].) For this study, we have examined only the prefetching method known as "one block lookahead" or OBL.

One block lookahead prefetching can be defined as follows: Consider each disk (or drum) to consist of a linear address space of tracks, numbered sequentially in the obvious way. Then when a block i is referenced, OBL prefetching checks to see if block i+1 is resident in the disk cache. If so, it is moved to the head of the (its) LRU stack; if not, block i+1 is (pre)fetched (asychronously) and is placed at the top of its LRU stack. The advantage to OBL prefetching is that if blocks are being accessed sequentially, it ensures that the next block is either in the cache or on the way in at the time it is first used. A number of costs are associated with this prefetch, however. If references are not sequential, then it does a lot of useless fetches. These useless fetches tie up the data paths to/from the cache, tie up the spindle, busy the cache controller, and cause "memory pollution", which is the phenomenon by which cache is polluted with blocks which are not in use, at the cost of removing those that may be reused.

The effects of prefetching, for a global cache for each system (with no user/file type breakdown) are shown in figures 10, 11 and 12; in each case, prefetching produces a very significant drop in the demand fetch miss ratio, usually on the order of 10% to 50% for reasonable cache sizes. We have also tabulated the effect of prefetching based on user and file type, but omit the tables for brevity; we do comment on them here, though: For both Crocker and Slac, we find that for batch files, batch users and temporary files, prefetching yields a dramatic drop (up to 80%) in miss ratio. This is clearly due to the fact that most batch and temporary files are sequentially allocated and are read and written sequentially. Conversely, paging data sets show no improvement from prefetching, which is consistent with our earlier observation that they show little locality.

Figure 12

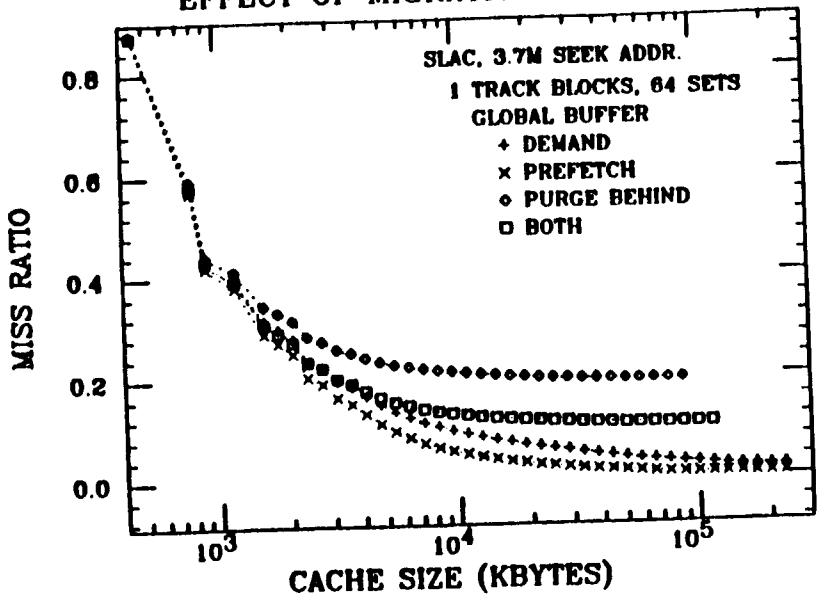


Figure 13

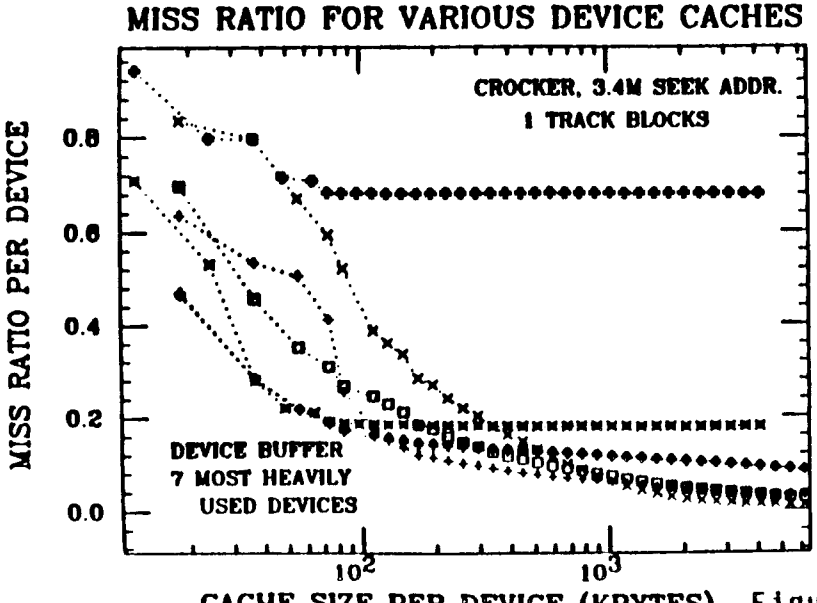
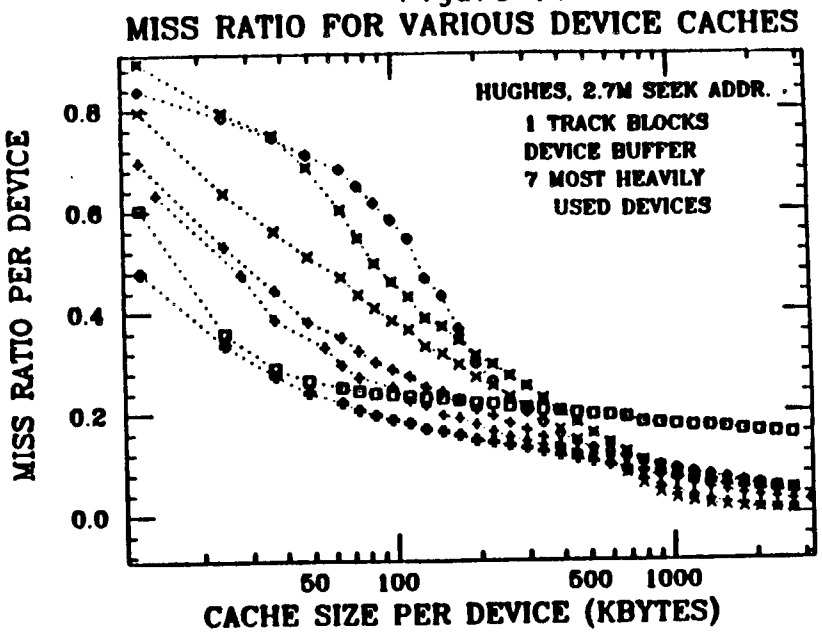


Figure 14



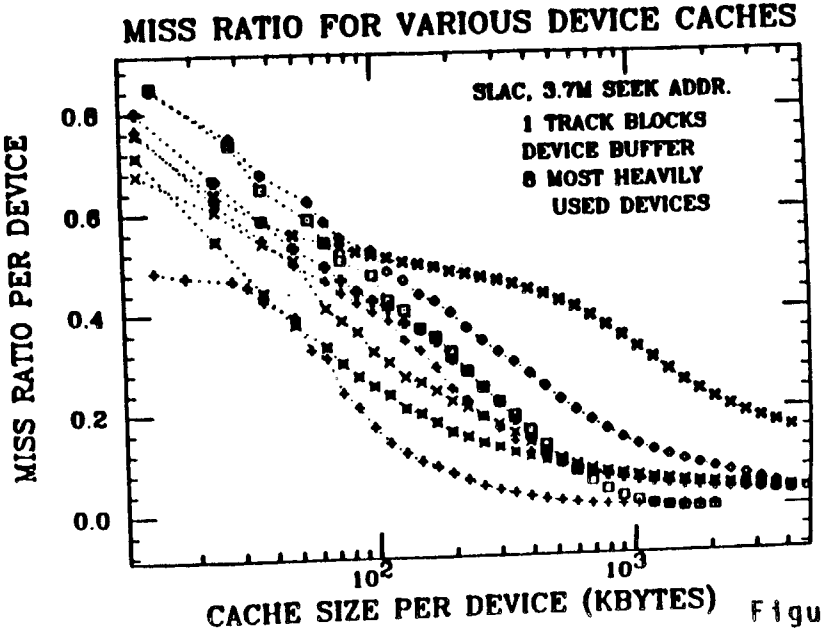


Figure 15

Intermediate between the two cases are system files and files used by the system which show some but not massive improvement.

Wide variations in the effectiveness of prefetching can also be seen by looking at a miss ratio breakdown by device and controller. The efficacy of prefetching seems to be highly correlated with the type of files on the device(s).

It is worth noting that our prefetching has used the crudest algorithm possible, OBL. A scheme such as that in [Smit78d], in which a variable number of blocks are prefetched depending on the observed degree of sequentiality should show marked improvement. Therefore, we believe that prefetching should be implemented, but made optional, and the implementation should be one which minimizes the costs and overhead associated with prefetching, as noted above.

3. Purge Behind

Many files, as noted earlier, are accessed sequentially. For batch and temporary files, it would be expected that after block i is referenced, block i-1 would not have a high probability of reuse. This leads to the idea that block i-1 could be removed from the cache, which would free up a cache storage location. If the block removed were indeed no longer active, then the effect should be beneficial. We define purge behind replacement as follows: whenever block i is referenced, remove block i-1 from the cache immediately.

Figures 10, 11 and 12 also show the effect of purge behind and the effect of purge behind combined with OBL prefetch. As may be seen, purge behind significantly increases the miss ratio. Breaking down the results by device, controller, user type and file type shows almost universal increases in the miss ratio from purge behind.

Because of these poor results, we recommend that purge behind not be used in disk caches.

E. Which Devices or Controllers to Cache

Illustrated in figures 13, 14 and 15 are miss ratios for the 7 or 8 most heavily used devices for the Crocker, Hughes and Slac computer systems. By referring to tables which show the uses of the various disk

drives, we note that of the 11 most heavily used devices at Crocker, the three packs that show the worst cache performance are paging, batch applications, and unknown. At Hughes, the worst results are for the three scratch packs; we speculate that these spindles contain user batch data sets with large block sizes, since all three showed excellent results from prefetching. At Slac, the worst results are for the packs with ASP spooling and job queue, and the Orvyl (time sharing) file system. (The next worst results are for user scratch packs.)

Based on these observations and those in section III.D.1 on user and file type, the following design principles seem appropriate: (1) A caching mechanism should have a provision for selecting only certain devices for caching. (2) Paging data sets/packs should not be cached. (3) User data set devices should be cached only if the block sizes are small enough that there are at least 2 or 3 blocks per track. (4) Prefetching should be available for sequential files, especially temporary files, which are almost always sequential. (5) System packs should be cached on an individual basis, depending on their contents.

F. Time of Day Effects

In tables VII, VIII and IX, we show the global cache miss ratio as a function of the segment (portion) of the seek address trace. That is, the miss ratio was recorded for each one million trace I/Os (of which only some were to DASD) throughout the trace period, using warm start (a full buffer) for all but the first segment. We can see that the three systems exhibit somewhat different behavior. The Slac system shows no significant time of day effects, excepting the initial transient while the cache fills up. Hughes shows a minor time of day effect, in that in the middle of the night, the miss ratio drops slightly. Crocker has a very marked effect, with a miss ratio at night much higher than during first shift. This latter behavior can be explained by the following argument: online TSO and IMS applications (during the day) have relatively small data working sets. The batch jobs and batch IMS applications that run at night (e.g. check processing) have poor locality. Conversely, both Slac and Hughes run similar workloads during the day and night.

Effect of Time Period on Miss Ratio
Hughes Aircraft, 1 Track Blocks,
Global Buffering, 16 Sets

Trace Section:	0-1M	1-2M	2-3M
Cache Size(Megabytes)			
1	.261	.217	.348
2	.221	.176	.287
4	.195	.141	.236
8	.172	.106	.177
16	.146	.078	.123
32	.113	.064	.084
64	.080	.056	.056
128	.064	.040	.035
Number of DASD I/Os	693101	706571	739077

Table VIII

Effect of Time Period on Miss Ratio
SLAC, 1 Track Blocks,
Global Buffering, 16 Sets

Trace Section:	0-1M	1-2M	2-3M	3-4M	4-5M
Cache Size(Megabytes)					
1	.326	.281	.333	.405	.406
2	.235	.205	.232	.276	.267
4	.172	.150	.157	.160	.151
8	.132	.107	.112	.091	.094
16	.106	.075	.084	.057	.060
32	.088	.051	.063	.036	.036
64	.052	.034	.049	.022	.024
128	.035	.021	.036	.014	.014
256	.032	.013	.031	.011	.010
Number of DASD I/Os	614087	681279	639404	678272	646961

Table IX

These time varying performance figures will reinforce our later discussion (section IV.D) about the advantages of dynamic on/off for the cache.

G. Static Device

One possible use for a level of storage between disk and main storage, the "gap filler", is as a statically allocated device, not for a dynamically managed cache such as we have discussed so far. The idea is to statically allocate to the gap filler device those data sets with the highest density (rate per byte) of reference. Such a use exactly reflects what this author calls the "lambda(i,j)" model, by which user i references file j as a Poisson process with rate lambda(i,j); if such a model is valid, then an optimal static allocation should give non-lookahead optimal results.

The method that we use to get a static allocation is lookahead optimal. For each DASD (disk and drum) cylinder, the entire trace was processed, and the number of references to that cylinder was counted; then the density of reference for each cylinder was computed, since the cylinders varied in size by device. Finally, the cylinders were sorted by decreasing density of reference and the miss ratio was computed.

In figures 16, 17 and 18, the static and dynamic miss ratios are compared. It can be seen that except for very small cache sizes (where the set associative mapping distorts the results) and very large cache sizes (where the entire DASD address space is in the cache), the miss ratios for dynamic management are dramatically better. Thus with respect to miss ratio, a static device is a very poor idea. This also suggests that fixed disk arms (as are available on the 3330 and 3350 disks) are not cost effective.

An implication of the results presented in this section is that the lambda(i,j) model, like the independent reference model for programs, is not valid; reference probabilities are clearly time varying. Mathematical models which are sensitive to the time invariance of the lambda(i,j) assumption are also therefore not valid.

Figure 16

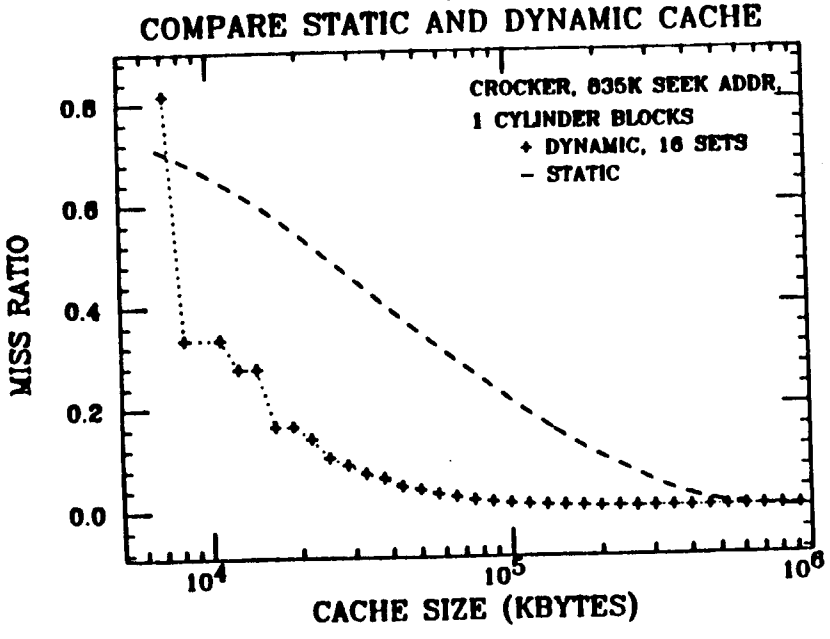
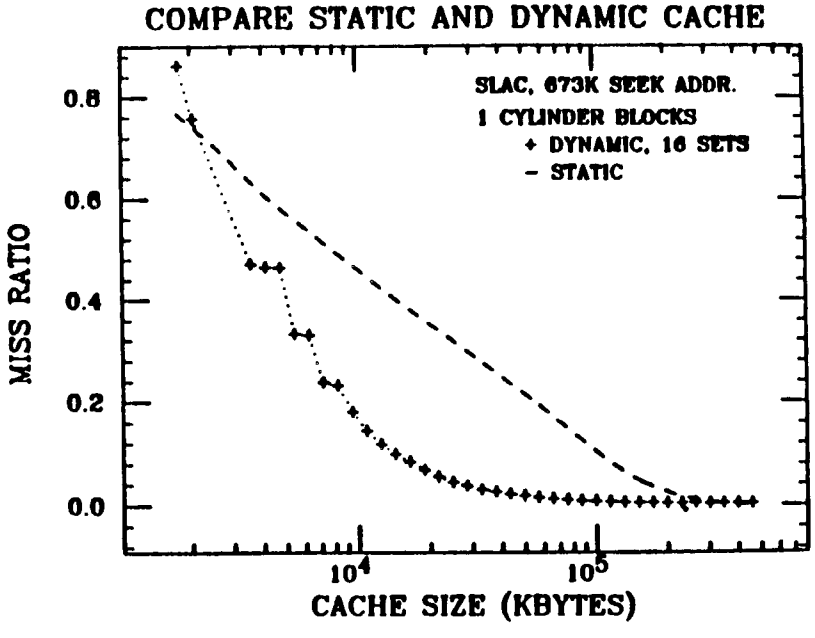


Figure 18



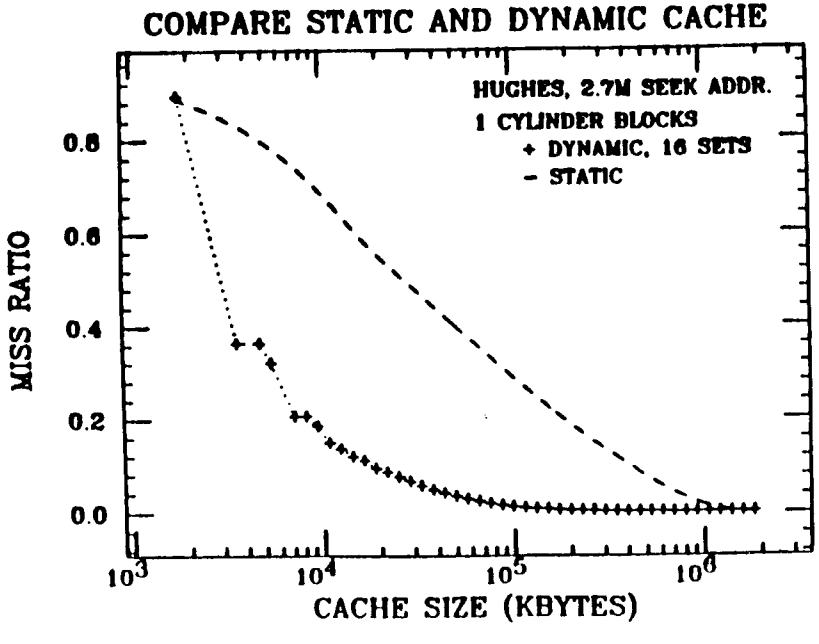


Figure 17

IV. Design Considerations

It was noted earlier that there are a number of design considerations which we have not evaluated by means of a miss ratio analysis. There are three reasons why we haven't done so: first, our data is limited (e.g. no read/write differentiation) and therefore, some studies are not possible. Second, some items, while performance related (e.g. path contention), are not usefully examined by that means. Finally some topics, such as error recovery or data consistency, are not primarily performance related. Therefore in this section we consider a number of design considerations, but without the use of trace data analysis.

A. Access Time and Bandwidth

The access time of a disk cache depends on both the access time desirable for performance reasons and that achievable from technology considerations. Because current disks are accessible in 20-30 ms., average, [Hunt80], any access time less than 10 ms. should yield significant performance gains. Therefore, the access time can be allowed to be primarily a function of the technology, as discussed below in section IV.E. MOS RAMs or CCD's are very fast and access times of 1 ms. or so ought to be easily achieved. Magnetic bubbles are a lot slower; their access time could be as much as 5 or 10 ms. depending on the implementation. These times are still acceptable in most cases. Note that the effective access time is the sum of the device access time, the transmission time (including any path contention delays), channel and storage controller "overhead" (command processing time), and the operating system (OS) overhead time. Therefore, the device access time may not be the limiting factor.

The bandwidth of a disk cache should be dictated by the desired access time. If one assumes that it should be possible to transmit a typical block (e.g. 4Kbytes) in less than 1 ms., this implies a data rate of at least 4 megabytes/second. This compares to a current maximum of 3 MB/sec. in IBM systems (to/from the 3380 disk) and up to 5 MB/sec. in Cray I systems. Thus, figures in this range are already available, and higher rates should be straightforward to obtain for this special case.

(High I/O rates are expensive to implement, and may require short cables. They are probably not justified for many other devices.)

B. Multipathing

There are several stages in the data path between the disk surface and the CPU and each stage is a point at which I/O congestion can occur. Consider a system in which the data path is {cpu <-> channel <-> storage controller <-> string controller <-> disk spindle}. Assume 3 Kbyte blocks on the disk, 16 ms. disk rotation time, and 3 megabytes/sec. transmission rate (i.e. approximately an IBM 3380 disk). Then the mean latency is 8 ms. and the transmission time for a block is 1 ms; thus excluding seek time, the disk I/O time is 9 ms. Assume k disks/string and j strings/controller. Let the utilization of the disk (excluding seek time and missed RPS delays; i.e. including only rotational latency and transmission time) be x. Then the disk transmits 1/9 of x time. The string controller is busy with data transmission k/9 of x. The storage controller is busy for data transmission jk/9 of x. (Assuming 1 string controller per string, and each string connected to only one storage controller.) For k=6 and j=3 (typical figures), the bottleneck is the storage controller, which is twice as busy as the disk and 3 times as busy as the string controller. (The channel is likely even busier than the storage controller, since there are usually fewer channels accessing the disk system than there are storage controllers.) Even allowing for seek time (approximately 40% of I/Os require seeks, at a mean of 30ms./seek. (see table I); and missed RPS delays (at 16ms. per miss for a full rotation time), the bottleneck is likely to be at the storage controller and channel. (It is also worth noting that the channel "overhead" (command processing time) might typically be twice the transmission time [Hunt80].)

The congestion at the storage controller is worse when the storage controller contains a cache. Assume a copy back and allocate on write strategy. A hit to the cache requires the transfer (to the CPU) of only the bytes accessed. A miss to the cache requires that the cache be loaded, and then the I/O takes place to the cache. (Fetch bypass can also

be used, whereby the information is transmitted at the same time as the cache is loaded.) If each record is read exactly once, then the traffic to the storage controller has doubled; i.e. each byte is loaded into the cache when a miss occurs, and is then read by the CPU. (The hit ratio may still be high in this case, since the blocks can be small and several may be in each track.) For a write, the track is loaded, the write occurs, and finally the track is written back, thus tripling the I/O traffic if each byte on the track is written exactly once. If only some of the records on each track are accessed (read or written), the factor by which the utilization of the storage controller data path has gone up is still larger, because the whole track still gets loaded.

The implication of the above argument is that a workable disk cache, if located at the storage controller, must have multiple data paths. In particular, it should be possible to read or write the cache from the CPU at the same time a cache to disk or disk to cache transfer is taking place. More than two data paths would be even better. The performance impact is discussed further in section IV.K.

C. Write Through Vs. Copy Back

A write through cache is one in which all writes go immediately to the disk surface. A write through cache can be implemented with three different allocation mechanisms: (1) Write allocate means that a copy is made in the cache on a write, even if the block has not been in the cache previously. (2) Write update means that (only) if a copy is already in the cache, is it updated. (3) Write purge means that if a copy is in the cache, it is purged. A copy back cache accepts writes and transfers them to the disk surface when that information is pushed from the cache. A copy back cache is usually implemented with write allocate, by first reading the block to be written to, if it is not already in the cache, and then modifying it.

A write, in a write through cache, has many of the undesirable characteristics of a read miss. Even though the CPU does not have to wait for a write to complete to continue work on a given process (unless the operating system so requires), the write still takes the time of a

physical disk access, and the relevant data paths are made busy for its duration. Since in many systems, writes are about one quarter to one third of all I/Os, it should be clear that the frequency of disk access will be significantly higher in many or most write through systems as opposed to copy back. Performance thus suggests that copy back should work better than write through.

Two aspects of system design favor write through: error recovery and consistency. Each is discussed in more detail below in its own section. We do note here, however, a modification to write through with performance advantages. It is suggested in [Duke82a] that two completions be signalled on a write. The first specifies that the write is complete to the cache (so that processing can continue); the second indicates that the write is complete to the disk surface (so that reliability and recovery are assured).

There is a peculiarity to some system architectures (IBM's in particular) which makes it difficult to implement a copy back write through cache. In IBM's count-key-data architecture [Smit81a], a sequence of data commands is followed by the command which indicates whether the transfer is actually a read or write. Therefore, the transfer can't be set up (to disk or to cache) until the last command is received. This problem does not occur in the newer fixed block architecture (FBA) of the 3310 and 3370 disks [Smit81a,IBM79a,b].

A minor complication which relates to copy back is the removable volume problem. If a disk can be dismounted, all buffered information must be written back first. This must be provided for.

The conflict between performance and reliability suggests that a mixture of modes may be desirable. It is worth noting that the design by NEC [Toku80a] has multiple modes; temporary files can be made write through with allocate on write, and files in "high speed mode" can be made cache resident only; the normal mode is write through with update on write.

D. Dynamic Cache On/Off

It is quite possible for a disk cache to lessen rather than improve performance (see e.g. [Buze82]). Consider, for example, a file which is stored with one large block per disk track, and which is read once. In that case, each block will be read to cache and then to the cpu, thus doubling the I/O traffic, and causing the I/O access time to increase. The hit ratio would be zero. Somewhat less realistic but even worse cases can be proposed. Further, it is likely that such poor performing situations will occur unpredictably, and will be intermixed with I/O in which high hit ratios are observed.

The possible unpredictable occurrence of situations in which the disk cache reduces the performance of the computer system suggests that it might be a good idea to permit a disk cache to turn itself on and off dynamically. That is, the cache would dynamically monitor its own miss ratio and write ratio (fraction of I/Os that are writes). When these were found to be too high, the cache would disable itself, with respect to the offending devices or strings. The cache could continue to simulate the miss ratio to be expected were it still active (by maintaining a "shadow directory"), and could turn itself on when performance had improved sufficiently.

This idea has not yet been tested, and algorithms need to be developed to specify when the cache should be enabled and disabled.

E. Technology

There are three technologies that have been suggested in the past for use as a level of storage between disk and main memory. These three are charge coupled devices (CCDs), magnetic bubbles and electron beam accessed memory (EBAM). EBAM has not proved to be a viable technology and there are currently no commercial EBAM projects. CCDs have not been available in sufficient supply in recent years and in all probability will not experience increased availability in the near future. It was generally predicted however [Hodg75], that they would be two to four times cheaper on a cost per bit basis should they ever be produced in comparable volume. Magnetic bubbles are also not yet available in adequate volumes and at

reasonable prices. Further, they are somewhat slow and might not be suitable for many cache designs.

A better choice, and one which has been made for all current commercial designs, is that of MOS RAM. At the time of this writing, the memory chips for a megabyte can be purchased for $500, and the performance and design advantages of RAM are obvious.

MOS RAM is the technology used for main memory, and the question arises as to why there is an advantage to disk cache, when the use of the storage in main memory is more general. There are several reasons, which are discussed elsewhere in this paper, but we note in particular: consistency problems may be worse if data is shared between independent cpus, and operating system modifications are certainly required for a main memory cache.

F. Consistency

One problem with disk cache is that multiple copies of data may exist; one copy on the disk, and one in each cache connected to it. The important point is that at any point in time there be a unique value for every byte in a file and any attempt to read or write that file should reference the uniquely correct value at that time.

There appear to be two general ways of ensuring that all cpus have consistent views of a given file. One is to force all accesses to a given disk to be via the same cache; this solution may not permit sufficient bandwidth, however. Further, it won't work if the cache is located in the channel or main memory, neither of which are shared between independent cpus.

The alternative is to provide explicit synchronization. The synchronization required would depend on the cache design. For example, consider a write through cache. In that case, it is sufficient that if a file is opened for writing, there be no other readers or writers. If the cache is copy-back, then it is necessary that all modified file blocks be rewritten to the disk before that file can be read along a different data path. See [Smit82] for a more thorough discussion, in the context of CPU caches.

It is possible for there to be inconsistent copies of a file even with only one cpu. Suppose that there are two (or more) data paths to a given file from a single cpu, with a copy back cache along each path. In that case, it is possible for each cache to have different values for blocks of the file; this situation must be prevented via either explicit synchronization or by allowing only a unique path to a given disk from a given cpu.

G. Error Recovery

Since disks are generally highly reliable, most operating systems assume that once something is written to disk it is permanently stored and will not be lost in a system failure. A disk cache design cannot afford to lower this level of reliability, otherwise it would not be commercially acceptable. The possible exception is that the loss of temporary files by jobs that would be restarted after a crash would be permissible; this is what is presumed by "high speed file mode" in the NEC disk cache [Toku80a].

There are therefore three features that should be part of a disk cache design: (1) Error correcting codes must be used in the cache to avoid I/O data errors, even though a correct copy of the data may exist on the disk. (2) A copy back cache must be nonvolatile, which implies either magnetic bubbles or battery back-up. The probability of a failure must be insignificant. (3) If the cache is disabled, there should be provision for access to the disks without the use of the cache.

H. Software vs. Hardware Management and Locus of Control

A disk cache will consist of several parts: (1) the storage array, which holds the data, (2) the directory, which specifies which blocks are in the cache, where they belong on disk, etc., (3) the control mechanism or logic, and (4) the data transfer logic. The question about items 3 and 4 is whether they should be implemented in software or hardware, and where.

The locus of control for the cache can be either in the same location as the cache (e.g. the storage controller) or the cache can be managed from the cpu by means of the system software. There are several reasons

not to do the latter: (1) The cpu would take on a new and significant processing task, resulting in increased supervisor overhead. (2) If there is more than one cpu sharing a cache, this could lead to either significant additional overhead to maintain consistency and error free operation of the shared cache, or to possible failure. (3) In the event of system failure, it seems easiest to preserve cache integrity if the control is local.

Because of the complexity of the cache control (with respect to synchronization, directory maintenance, dynamic on/off, etc.), it should be clear that the control should be either in software or microcode; it is not feasible to hardwire it. Further, the algorithms for cache operation will likely be changed and updated, which is much more easily done in software or microcode. Conversely, the data transfer logic, which must transfer data at high (>=3 megabytes/sec.) rates, will have to be hardwired; software isn't fast enough.

I. Operating System Implications of Disk Cache

The use of disk cache in a computer system has implications for the operating system, both as to correctness and system performance. The correctness aspects have already been discussed. The performance aspects of disk cache are somewhat more subtle, and each is discussed below.

Mechanisms for data set searches [Knut73] are typically reflective of the storage technology used. For large disk data sets, tree structures are often used to minimize the number of disk blocks read. Such tree structured indexes can be even more efficient when the highest level or levels of the indexes can be expected to stay cache resident. Conversely, linear searches, such as occur in reading catalogs and volume tables of contents, should not be permitted to go through a disk cache. Such a linear search (with large block sizes) is likely to flush the cache of other, more useful information, without achieving a high hit ratio itself.

Some systems maintain main memory buffers which serve the same function as a disk cache. Unix [Rich74], for example, keeps a main memory I/O buffer. IMS [Date77] maintains a buffer of recently accessed blocks of the data base. It is preferable that multiple buffers serving the same

function not be used; if they are, the result is increased overhead, and possible side effects leading to worse performance. For example, if a main buffer captures many data rereferences, the reference stream to the disk cache can result in the wrong blocks being kept.

The existence of a disk cache adds a new parameter to disk I/Os: the probability of a miss. Therefore, for all I/O system optimizations, there is a new figure of merit to consider. For example, if the cache is write through, then writes count (more or less) as misses. Thus whenever there are tradeoffs between reads and writes, one would now prefer to a greater extent additional reads and fewer writes. Also, large block sizes are likely to lower the hit ratio, so much of the incentive for increasing the block size is gone.

Traditionally, data sets are placed on I/O devices so as to balance the load among spindles, controllers and channels. With a cache, the load must still be balanced, but the effective load depends on the hit ratio. In addition, one may want to segregate those data sets suitable for caching from those that are not; thus caching can be made selective on a device or string basis.

The use of an electronic drum and/or disk cache may suggest some changes in the scheduling algorithms used and in the way the dispatcher operates. For example, there is significant overhead in standard I/O processing, including handing I/O interrupts and executing the dispatcher at least twice. If the time to complete the I/O is short enough, it may be more effective to wait for the I/O to finish rather than to task switch. (If the cache is in main memory, then of course a real I/O has to be issued only on a miss.) Similarly, one might add as a factor in the dispatching algorithm information about which ready process has the most blocks in the cache; it would generally be advantageous to start a process which is using a lot of cache blocks.

I. Cache Visibility, User Control and Operating System Modifications

The purpose of a disk cache is to improve performance; it has no other user visible function. Thus, for example, if a cache is located in the string or storage controller, it should be possible to activate or

deactivate the cache, or for that matter, interchange a storage controller without a cache with one that has a cache, all without the user having to change either his programs or the way in which he uses the system. Further, in that case, it should be possible to add or delete the cache without even modifying the operating system.

The problem with a completely invisible cache is that it sacrifices possible performance benefits. For example, if it is known that a given file can be lost without ill effects in a system failure, then write through need not be used. Similarly, a temporary file can be written with write allocate, whereas a permanent file should be written without write allocate. If the cache is in main memory, which has the performance advantages noted above (section III.B), then the cache must be visible to the operating system, which must have been modified to manage it.

As one example of the performance advantages to be gained by a user visible cache, we note the cache built by NEC for the ACOS 1000 [Toku80a]. In that system, running real commercial workloads, performance improvements using all possible file modes (described further in section VII.A) are twice those available using only the basic (write through, no write allocate) mode.

To obtain the full possible performance benefits for a disk cache, it seems clear that there must exist a mechanism for the user (or the system, by default for certain classes of files) to specify how the cache should handle I/Os to a given file. This can be done in most systems by adding one or more parameters to the command that opens file. In, for example, an IBM OS based system, job control language (JCL) parameters can be added.

K. Performance Impact

As noted earlier, we have not attempted to directly calculate the performance impact of disk cache in this paper. There is one paper in the literature which does look at the effect of disk cache on the mean I/O access time. In [Buze82] it is observed that if the read/write ratio is high enough and the hit ratio is high enough, then the use of cache disk can significantly cut mean I/O time; conversely, if the miss ratio is low

and/or the read write ratio is low, I/O access time increases. That analysis, however, assumes a single data path through the cached controller; thus the disk cache has the potential of impeding performance. Analysis of a design with multiple data paths should show much better results.

V. Disk Spindle Buffers

The purpose of disk cache is to frequently eliminate the mechanical access time component of disk I/O. An examination of the components of that access time also suggests another idea. Most modern disk drives have a feature known as rotational position sensing (RPS), which works as follows. A command is given to the disk to search for a specific record; when that record is recognized, the disk attempts to reconnect to the storage controller and channel. If both are free, then the data transfer begins. If one or both are busy (the reconnect fails) then a full rotation time must elapse (until the record passes under the read heads) before transmission can again be attempted. This delay (known as an RPS miss) can add significantly to mean I/O times [Hunt80] and its frequency is obviously very sensitive to the utilization of the channel and storage controller.

A mechanism to eliminate RPS misses is called the Disk Spindle Buffer (or the DASD arm buffer [Hunt81]). The idea is to build into each disk spindle (or each disk arm controller - i.e. device address) a buffer capable of holding a disk track or some part thereof. A read from disk would then consist of the device accepting the command, loading the buffer with the track (or record), and then seizing the data path to the cpu as soon as it becomes available, without waiting for a specific angular position of the disk. In this way, RPS misses are eliminated, and further, the channel utilization can be increased greatly. Previously, if the channel utilization became too high (above 30 or 40%) [Bere78], RPS misses became quite frequent and costly. With a spindle buffer, queueing can take place in a useful way.

A spindle buffer also has another potential advantage. All transfers to/from a disk must run at exactly the transmission rate of the device, which itself is a function of the bit density along the disk track. If transfers are buffered in electronic storage, they can be made at arbitrary speeds, and also asynchronously. That is, they can be interrupted by other (unbuffered) transfers, or can be slowed down if the channel or storage controller is otherwise too busy to operate at the full data rate.

VI. Alternatives to Disk Cache

Disk cache has been proposed as a solution to a predicted I/O bottleneck. There are other ways to approach this problem and we discuss them in this section.

One of the reasons that disk cache is so effective is that it avoids repeated physical I/Os to read many small blocks; those blocks are read (or written) from disk to cache a track at a time, and are then sent to the cpu from electronic storage. It is well known (see e.g. [Berb78]) that block sizes are frequently small, and that larger block sizes improve performance [Bere78]. If system users begin to use relatively large block sizes (half or full track), then disk cache ceases to be very useful for sequential files. This is evident from an examination of our miss ratio data on a device by device basis. When the miss ratios for scratch volumes (not shown) are examined, miss ratios can be seen to be very high. The reason is that scratch volumes are often used by experienced users to hold large sequential data sets with large block sizes; in that case, rereferences to a given track are infrequent.

A certain amount of input/output occurs because of the limited size of main memory. For example, old compilers typically generate large numbers of temporary files so that they can run in small (50K, 100K) memory sizes. Each such temporary must be written and then reread. With large modern main memories, most such information can be kept in main storage; thus that fraction of I/O (see table II) going to temporary files can be almost eliminated.

Some systems already do a significant amount of internal buffering. For example, it was noted earlier that the IMS data base system keeps its own buffer of recently used blocks. Similarly, UNIX maintains its own main memory I/O buffer. To the extent that dedicated buffers are set up to manage I/O streams, the disk cache becomes redundant. In many cases, it can be expected that the author of a system or program will be better able to manage his buffers than a standardized, shared disk cache. We don't believe that this is a good approach, however, because of the time, effort and expense of writing many different buffering systems, some of which may not actually be effective.

The idea of disk cache is that it is a self managed cache which retains recently used blocks of data. An alternative is to create an explicitly managed electronic storage device, such as the electronic drums made by Intel (FAST-3805 [Inte79] and 3825) and Storage Technology (3805). Software already exists to manage drums and to maintain on them the most frequently used information. In some cases, that software may work fairly well.

From the above list of alternatives, we can see that each performs one or more functions of the disk cache. Internal buffers and main memory keep data in electronic storage just as does disk cache. Reblocking does explicitly what disk cache does automatically. Thus to the extent that one or more of these techniques are implemented, the projected I/O bottleneck is put off; if all are implemented, then disk cache becomes redundant. The appealing aspect of disk cache is that it is a simple and elegant solution which avoids the need for many small and costly system changes and improvements.

VII. Commercial Disk Cache Products

In the last two to four years, a number of cache disk products have been announced; some have actually been delivered. In this section, we mention some of them, with particular attention to the NEC and IBM designs.

A. The Nippon Electric (NEC) Integrated Disk Cache

NEC has designed and tested a disk cache for their ACOS 1000 system [Toku80a]. That machine is structured around a "system control unit" (SCU) through which all input/output passes. Attached to the SCU is the disk cache; thus the cache is global to the entire system and is accessible on all I/Os. Further, the operating system is aware of the disk cache, and the type of caching provided is determined by the operating system based on the file type. Performance is also improved by providing a 5.3 MByte/second I/O rate between the cache and main memory.

The disks in the NEC system are fixed sectored, and the cache holds fixed size blocks, equal to some (adjustable) multiple of the fixed block size. These blocks are arranged in a set associative manner (as with our simulations) with LRU replacement within each set.

One of the most interesting features of the NEC cache is that it provides four different modes of operation, depending on the file type:

1. Basic Mode- (i) A read hit is serviced by a read of one block from the cache. A read miss results in a load to the cache followed by a read from the cache. (ii) A write always goes directly to disk, with write update if the block is also in the cache. Completion is reported only after the physical disk update.

2. Sequential File Mode - This is the same as the Basic Mode, except: A read miss causes n records to be prefetched into the cache. A cache block whose last record has been accessed is placed at the head of the list for replacement. (Similar to "purge behind".)

3. Temporary File Mode - Same as basic mode, except with write allocate as well as write update.

4. High Speed File Mode - The file is allocated to the disk cache only and is never written to disk; space must be preallocated. This mode is used for temporary files only, and is vulnerable to cache failure.

Error correction is provided. Upon cache failure, only the use of high speed file mode can cause a job to fail; in all other cases, a failure of the cache will cause the cache to be bypassed.

Some performance figures are reported, with the use of all but high speed mode. Hit ratios of 60% to 95% are observed in practice, and the elapsed job times decrease by 10% to 35%. Response time decreases for interactive use by 15% to 25%. It has also been observed [Toku80b] that the use of basic mode alone yields only about half of the potential performance gains.

B. IBM 3880 Mod 11 and Mod 13.

The IBM 3880 Model 11 storage controller [IBM81a] is a storage controller expressly designed for paging and swapping. It has a capacity of 8 megabytes, and can transfer data to the channel at up to 3 megabytes/sec.. Its use is invisible to the user and operating system and no changes are required by either. It will support up to one string of 8 model 3350 disks, although two disks are recommended per cache. Access time on a page read hit is 2.4 ms. It appears to use copy back rather than write through, although the documentation is not explicit. It also appears that there is only one data path in the storage controller (to which the cache is connected); thus at most one I/O operation can be transferring data at a time.

There appear to be two problems with the 3880 mod 11: First, our earlier results suggest that paging data sets have low locality, and that therefore the hit ratio will be low. (But it may be possible to reorganize the paging data sets in such a way that they can be cached effectively.) Second, the existence of only one data path suggests that there may be path contention; the suggested restriction to only two disks implies that IBM is aware of the problem. No performance results on the 3880/11 have been published yet, but for the reasons noted, significant system performance improvements are unlikely.

The IBM 3880 mod 13 storage controller [IBM81b] uses a different cache design than the mod 11 and is explicitly designed for nonpaging I/O. It can have either 4 or 8 megabytes, and can attach to two storage directors. It supports only IBM 3380 disks. The I/O transfer rate is 3 Mbytes/sec. Use of the cache requires no changes in either the operating system or the user program. Individual devices attached to the controller

can be locked out of use of the cache. It is possible to lock certain data sets into the cache; prefetching appears to also be implemented for sequential data sets. Write through is used with write update but no write allocate. Read hit access time is an average of 3.5 ms. Read hit ratios (based on sample IBM benchmarks) are claimed to range from 50% to 85% at 4 megabytes, and 65% to 90% at 8 megabytes.

It also appears that only one data path to/from the 3880/13 cache can be active at any one time. This fact, plus the use of write through and the fact that I/O overhead has not been cut, suggests that only small performance improvements are possible with the current 3880 designs. This impression is reinforced by the analysis presented in [Buze82].

C. Other Cache Disk Systems

Some time ago, Memorex [Memo78] designed and build a disk cache (the model 3770) into one of their storage controllers. It contained 1 to 18 megabytes of storage which buffered recently used tracks using LRU replacement. The 3770 was never commercially successful, due among other reasons to the fact that since the cache had only one data path, with no fetch bypass (a miss caused a load, followed by a read from the cache). Performance improvements, if any were minor.

Storage Technology offers a disk cache system, the 8890 Intelligent Disk Controller, or Sybercache [Cote82] [Stor82]. It consists of from 1.5 to 12 megabytes of RAM storage associated with two storage directors. A "typical" configuration would be 6 megabytes of storage and 16 spindles. The design uses write through and full track blocks. No actual performance figures are presented in [Cote82].

Amperif manufactures a disk cache which can be used on Sperry Univac computers. Significant performance improvements [Comp82d] are reported from its use. Sperry Univac also manufactures its own cache disk system [Sper81]. Amperif also has a model of their cache [Comp82b] that is specifically designed to run with the Airline Control Program on IBM computers.

Computer Automation [Huge82] makes a disk cache to run with their SyFA line of minicomputers. It uses from .5 to 2 megabytes of RAM,

achieves an average access time for a hit of 4 ms. and a hit ratio average of 85% is claimed. Replacement is by a complex algorithm reflecting both the amount of reference and time since last reference. Write through is used.

Other disk caches are announced and/or manufactured by Minicomputer Technology [Elec81] for the DEC PDP-11 and VAX, by Point4 Data [Comp82c] for their Mark 5 or Mark 8 minicomputers, by Qualex Technology [Comp82a] for the HP 3000 computers, by Integrated Business Computers [IBC82], by IBM for the Series I [Elec83], and one vendor has a software package that simulates a disk cache in main memory [Info83].

VIII. Summary and Conclusions

As explained at the beginning of this paper, the rapid increase in cpu performance without a corresponding improvement in the performance of mechanical I/O devices is leading to an I/O bottleneck. The addition of a disk cache to a large computer system can result in significant performance improvements and elimination of the projected I/O bottleneck. Our trace driven experiments and the commercial benchmarks reported all show that a disk cache can capture 60-95% of I/Os, with access times for read operations of 2 to 4 ms.

Reflecting the clear desirability of disk cache, a number of commercial instances of this product have been produced in the last few years. Performance figures, where reported, confirm the utility of disk cache. In some cases, however, the designs are suspect, due to the lack of multiple data paths to/through the cache; those products may not be useful.

The experiments reported in this paper are only a fraction of the range of data we have gathered. Never the less, there are a number of aspects of disk cache design that have not yet been investigated, and need to be explored. Migration algorithms need to be studied further, and the

effectiveness of more sophisticated prefetch algorithms must be tested. Algorithms for managing a dynamic on/off cache must be developed. The overall system performance impact needs to be quantified.

The data that we have currently available is not sufficient, but there is the need to study information gathered from non-IBM systems, to look at the frequency of writes and compare write through with copy back, and to further refine the measurements with regard to user and file type.

Acknowledgements

The research reported in this paper consists of some of the author's portion of a larger study conducted in collaboration with George Rossman, David Rosetti and James Richardson. The work by Rosetti and Richardson in collecting and reducing the data to analyzable form was particularly heroic and this research would not have been possible without their help. The cooperation of the computer center staffs at Crocker Bank, Hughes Aircraft and SLAC is also appreciated. The computer time for the data reduction and data analysis was provided mainly by Amdahl Corporation, without which much of this research would not have been possible.

Bibliography

[Amda70] G. M. Amdahl, "Storage and IO Parameters and Systems Potential", Proc. IEEE Computer Group Conference, June 16-18, 1970, Washington, D. C., pp. 371-372.

[Best81] A. L. Bastian, J. S. Hyde and W. E. Langstroth, "Characteristics of DASD Use", Proc. CMG XII International Conference, December, 1981, New Orleans, La., pp. 107-109.

[Bast82] A. L. Bastian, "Cached DASD Performance Prediction and Validation", Proc. CMG XIII International Conference, December, 1982, pp. 174-177.

[Bela66] L. A. Belady, "A Study of Replacement Algorithms for a Virtual Storage Computer", IBM Sys. J., 5, 2, 1966, pp. 78-101.

[Benh82] M. T. Benhase, A. H. Duke, J. D. Huntley and F. J. Marschner, "Handling Defective Tracks in a Cached Storage System", IBM Tech. Disc. Bull., 25, 7B, December, 1982, pp. 3758-3759.

[Benn81] B. T. Bennett and C. May, "Improving Performance of Buffered DASD to Which Some References Are Sequential", IBM Tech. Disc. Bull., 24, 3, August, 1981, pp. 1559-1562.

[Berb78] Susan Berbec, A. Shibamiya, S. Togasaki and H. Yoshida, "Use of Direct Access Storage Devices by MVS Customers - Guide Survey Results", Proc. Guide 47 Conference, Chicago, Ill., November 10, 1978, pp. 1121-1138.

[Bere78] T. Beretvas, "Performance Tuning in OS/VS2 MVS", IBM Sys. J., 17, 3, 1978, pp. 290-313.

[Bran81] Alexandre Brandwajn, "Models of DASD Subsystems: Basic Model of Reconnection", Performance Evaluation, 1, 1981, pp. 263-281.

[Buze82] Jeffrey P. Buzen, "BEST/1 Analysis of the IBM 3880-13 Cached Storage Controller", Proc. CMG XIII International Conference, December, 1982, pp. 156-172.

[Comp82a] "HP 3000 Access Time Slashed 400%", Computerworld, February 8, 1982, p. 109.

[Comp82b] "Cache Disk System Out for IBM ACP", Computerworld, June 7, 1982, p. 101.

[Comp82c] "Cache Memory Reduces Data Transfer Time", IEEE Computer, February, 1982, p. 103

[Comp82d] "Cache Disk System Speeds Access for Power Firm", Computerworld, December 6, 1982, p. 34.

[Cote82] Henry J. Cote and Ben Duhl, "New Horizons for Cached Disk and Buffered Tape", Proc. CMG XIII International Conference, December, 1982, pp. 333-337.

[Date77] C. J. Date, "An Introduction to Database Systems", Second Edition, Addison-Wesley, Reading, Mass., 1977.

[Denn72] Peter J. Denning, "On Modelling Program Behavior", Proc. SJCC, 1972, pp. 937-944.

[Dods82] George W. Dodson, "Cached DASD Evaluations for Paging and Non-Paging Data", Proc. CMG XIII International Conference, December, 1982, pp. 338.

[Duke82a] A. H. Duke, M. H. Hartung, J. D. Huntley and F. J. Marschner, "Buffered Writing in a Peripheral Storage Hierarchy", IBM Tech. Disc. Bull., 25, 4, September, 1982, pp. 2075-2076.

[Duke82b] A. H. Duke and M. H. Hartung, "Controlling Multitrack References in a Cached Storage System", IBM Tech. Disc. Bull., 25, 7B, December, 1982, pp. 3756-3757.

[Elec81] "Cache Memories Catch on for Disks", Electronics, April 21, 1981, pp. 62-63.

[Elec83] Electronic News, "IBM Expands CPU for System 38; Adds Series 1 Processor, Disk Drive", Monday, April 11, 1983.

[Fajm73] Roger Fajman and John Borgelt, "Wylbur: An Interactive Text Editing and Remote Job Entry System", CACM, 16, 5, May, 1973, pp. 314-322.

[Hark81] J. M. Harker, D. W. Brede, R. E. Pattison, G. R. Santana, and L. G. Taft, "A Quarter Century of Disk File Innovation", IBM J. Res. Devel., 25, 5, September, 1981, pp. 677-689.

[Hoag79] A. S. Hoagland, "Storage Technology: Capabilities and Limitations", Computer, 12, 5, May, 1979, pp. 12-18.

[Hodg75] David A. Hodges, "A Review and Projection of Semiconductor Components for Digital Storage", Proc. IEEE, 63, 8, August, pp. 1136-1147.

[Huge82] Willi Hugelshofer and Bruce Shultz, "Cache Buffer for Disk Accelerates Minicomputer Performance", Electronics, February 10, 1982, pp. 155-159.

[Hunt80] David Hunter, "Modeling Real DASD Configurations", IBM Research Report RC 8606, December, 1980, Yorktown Heights, New York.

[Hunt81] D. W. Hunter, "DASD Arm Buffers", IBM Tech. Disc. Bull., 24, 4, September, 1981, p. 2035.

[Info83] Infoworld, "Cache/Q Disk Buffering Enhancement for CP/M", 5, 7, January, 14, 1983, pp. 50-54.

[Inte79] Intel Corp., "FAST-3805 Functional Description", PN 19-1619-006, August, 1979, Intel Commercial Systems Division, Phoenix, Az.

[IBC82] IBC/Integrated Business Computers, CADET/10 Cache Disk Memory Reference Manual, IBC, Chatsworth, Ca. 1982.

[IBM72] IBM Corp., "Reference Manual for the IBM 2835 Storage Control and the IBM 2305 Fixed Head Storage Module", GA26-1589, IBM Corp., 1972, San Jose, Ca.

[IBM73] IBM Corp., "Reference Manual for IBM 3830 Storage Control and IBM 3330 Disk Storage", GA26-1592, IBM Corp., Armonk, N. Y.

[IBM77a] IBM Corp., "Reference Manual for IBM 3350 Direct Access Storage", GA26-1638-2, IBM Corp, San Jose, Ca., 1977.

[IBM77b] International Business Machines Corp., "OS/VS MVS Systems Programming Library: System Management Facilities (SMF)", GC28-0706-1, IBM Corp., Poughkeepsie, N. Y., 1977.

[IBM79a] IBM Corp., "IBM 3310 Direct Access Storage Reference Manual", GA26-1660, IBM Corp., San Jose, Ca. 1979

[IBM76] IBM Corp., "OS/VS2 System Programming Library: Service Aids", GC28-0674-1, IBM Corp., Gaithersburg, Md. 1976.

[IBM79b] IBM Corp., "IBM 3370 Direct Access Storage Description", IBM, General Products Division, San Jose, Ca. GA26-1657-2. 1979.

[IBM81a] International Business Machines Corp., "Introduction to IBM 3880 Storage Control, Model 11", IBM Pub. No. GA32-0060, Sept., 1981, IBM Corp., Tucson, Ariz.

[IBM81b] International Business Machines Corp., "Introduction to IBM 3880 Storage Control Model 13", IBM Pub. No. GA32-0062, September, 1981, IBM Corp., Tucson, Ariz.

[Knut73] Donald E. Knuth, "The Art of Computer Programming, volume 3, Sorting and Searching", Addison Wesley, Reading, Mass., 1973.

[Lync72] William C. Lynch, "Do Disk Arms Move?", Performance Evaluation Review, 1, December, 1972, pp. 3-16.

[Matt70] R. L. Mattson, J. Gecsei, D. R. Slutz and I. L. Traiger, "Evaluation Techniques for Storage Hierarchies", IBM Sys. J., 9, 2, 1970, pp. 78-117.

[Memo78] Memorex Corp., "3770 Disc Cache Product Description Manual", Santa Clara, Ca. 1978.

[Ritc74] Dennis Ritchie and Ken Thompson, "The UNIX Time Sharing System", CACM, 17, 7, July, 1974, pp. 365-375.

[Schn77] Peter Schneider, "CPU-Utilization and Secondary Storage Performance - The Demand for a New Secondary Storage Technology", Proc. NCC, 1977, pp. 819-825.

[Sher72] S. Sherman, F. Baskett, and J. C. Browne, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System", CACM, 15, 12, December, 1972, pp. 1063-1069.

[Smit75] Alan Jay Smith, "A Locality Model for Disk Reference Patterns", Proc. IEEE Computer Society Conference, February, 1975, San Francisco, Ca., pp. 109-112.

[Smit76] Alan Jay Smith, "Analysis of a Locality Model for Disk Reference Patterns", Proc. Second Conference on Information Sciences and Systems, The Johns Hopkins University, Baltimore, Md., April, 1976, pp. 593-601.

[Smit78a] Alan Jay Smith, "Bibliography on Paging and Related Topics", Operating Systems Review, 12, 4, October, 1978, pp. 39-56.

[Smit78b] Alan Jay Smith, "On the Effectiveness of Buffered and Multiple Arm Disks", Proc. Fifth Computer Architecture Symposium, April, 1978, Palo Alto, Ca., pp. 242-248.

[Smit78d] Alan Jay Smith, "Sequentiality and Prefecting in Data Base Systems", ACM Transactions on Database Systems, 3, 3, September, 1978, pp. 223-247.

[Smit78e] Alan Jay Smith, "Sequential Program Prefetching in Memory Hierarchies", IEEE Computer, 11, 12, December, 1978, pp. 7-21.

[Smit81a] Alan Jay Smith, "Input/Output Optimization and Disk Architecture: A Survey", Performance Evaluation, 1, 2, 1981, pp. 104-117.

[Smit81b] Alan Jay Smith, "Bibliography on File System and Input/Output Optimization and Related Topics", Operating System Review, 15, 4, October, 1981, pp. 39-54.

[Smit81c] Alan Jay Smith, "Optimization of I/O Systems by Cache Disk and File Migration, A Summary", Performance Evaluation, 1, 3, 1981, pp. 249-262.

[Smit81d] Private communication with mainframe manufacturer.

[Smit81e] Brian J. Smith, "I/O Subsystem Workloads: Measurements and Modeling", Proc. CMG XII International Conference, December 1981, New Orleans, La., pp. 110-118.

[Smit82] Alan Jay Smith, "Cache Memories", Computing Surveys, 14, 3, September, 1982, pp. 473-530.

[Sper81] "Cache Disk System", Sperry Univac Product Announcement, for 5057 Cache Disk Processor and 7053 Storage Unit, 1981.

[Stor82] Storage Technology Corporation, "Sybercache 8890 Intelligent Disk Controller", Louisville, Colo. 1982.

[Toku80a] T. Tokunaga, Y. Hirai and S. Yamamoto, "Integrated Disk Cache System with File Adaptive Control", Proc. IEEE Computer Society Conference, September, 1980, Washington, D. C., pp. 412-416.

[Toku80b] T. Tokunaga, private communication, September, 1980.

[Welc79a] Terry Welch, "Effects of Sequential Data Access on Memory Hierarchy Design", Proc. IEEE Computer Society Conference, February, 1979, San Francisco, Ca., pp. 65-68.

[Welc79b] Terry A. Welch, "Analysis of Memory Hierarchies for Sequential Data Access", Computer, 12, 5, May, 1979, pp. 19-26.