Disk Cache—Miss Ratio Analysis and Design Considerations

ALAN JAY SMITH
University of California, Berkeley

The current trend of computer system technology is toward CPUs with rapidly increasing processing power and toward 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 (I/O) system.

A solution to this problem, which is 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., 80–90 percent) fraction of all I/O requests are captured by a cache of an 8-Mbyte order-of-magnitude size for our workload sample. 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. Consideration is given to both caches located in the I/O system, as with the storage controller, and those located in the CPU main memory. Experimental results are based on extensive trace-driven simulations using traces taken from three large IBM or IBM-compatible mainframe data processing installations. We find that disk cache is a powerful means of extending the performance limits of high-end computer systems.

Categories and Subject Descriptors: B.3 [**Hardware**]: Memory Structures—design styles; performance analysis and design aids; B.4 [**Hardware**]: Input/Output and Data Communications—*input/output devices; reliability, testing, and fault tolerance*; D.4 [**Software**]: Operating Systems—*storage management; performance.*

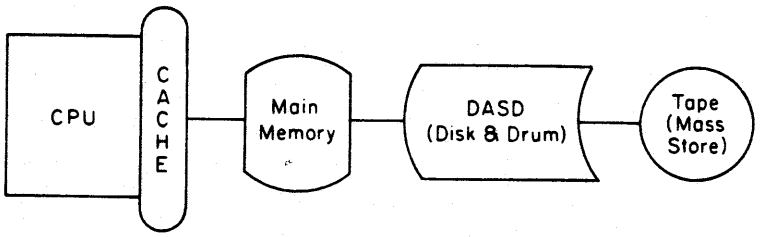
General Terms: Design, Performance

Additional Key Words and Phrases: Cache controller, disk, I/O buffer

1. INTRODUCTION

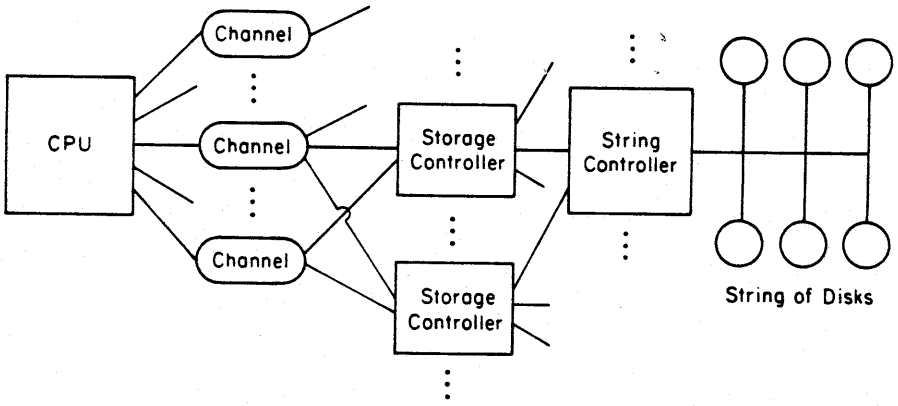
Computer systems have traditionally relied on a memory hierarchy (such as that in Figure 1a), in which large amounts of less expensive storage (disk, tape) have been used to retain the bulk of the stored information, while small amounts of fast storage (main memory, CPU cache memory) have been employed 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

This research was supported in part by the National Science Foundation grants MCS77-28429 and MCS82-02591, and by the Department of Energy contract DE-AC03-76SF-00515.
Author's address: Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720.



Typical Memory Heirarchy

(a)



Part of Typical I/O Configuration

(b)

Fig. 1. (a) Typical memory hierarchy. (b) Part of typical I/O configuration.

electronic storage (main memory, at less than 1 μs) and the fastest bulk storage (drum, at 5–10 ms). The difficulty is that frequent accesses to bulk storage, either through implicit (paging) or explicit input/output (I/O), 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 that suggests that multiprogramming is limited in its ability to overlap I/O 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, that is, doubling every 3–6 years. (b) Disk density will also continue to increase at a rate similar to the recent past, doubling every 3 or so years [35]. (c) Disk access time will continue to improve only very slowly [36]. (d) The I/O rate associated with computer systems will remain roughly proportional to the instruction execution rate of the CPU [1]. To a certain extent, large memories will cause some reduction in the amount of I/O traffic, as is

discussed later, but the effect will be small compared with the other factors considered. (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, per spindle, will also not decrease significantly. (g) Therefore, the number of disk spindles in a large computer system will not increase (owing 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 best proportional to the number of disk 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 or CPUs. This same argument applies to a distributed system, using a file server, and to a multicomputer or multiprocessor system.

This chain of reasoning, although indicating an eventual bottleneck, does not rule out changes in the software or hardware that postpone the problem. Means of postponing the problem are discussed in Section 6.

The mechanism proposed in this paper for coping with I/O bottleneck is *disk cache*, also sometimes called *cache disk*. Disk cache is a cache or buffer used to hold portions of the disk address space contents. If such a buffer (a) can 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 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 reason is the long established *principle of locality* [21], which describes most program reference behavior. This principle has two components: (a) Information that 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); (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 [74] and main memory paging [67]. Because disk files are frequently reused (databases, 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.

The function of disk cache is specifically as an I/O cache and not just as a way to add additional memory to the computer system. As discussed below, particularly in Section 6, the use of additional memory should reduce the I/O load, but unless that memory is used for the functions we assign to disk cache, it will not produce the performance improvements claimed here.

Locality in disk reference patterns has been previously observed (e.g., [65], [66]). It was suggested that multiple arms be used to access each of several open data sets on a given spindle in [65], and the idea of a cache was proposed in [68]. A brief discussion of disk cache appears in [73]. The topic is also discussed in [78, 79]. More recently, a number of papers have discussed and/or evaluated aspects of disk cache: [4], [11], and [23]. 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 disk

cache systems; included among them are IBM [51–54], Nippon Electric [77], Storage Technology [18], and Memorex [61].

In this paper we are concerned with the proper design, implementation, and operation of disk cache. There are a number of design considerations that merit attention: Where in the system shall the disk cache be placed? How large does the cache need to be on the basis of 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.

It should be noted that because our data are taken from large systems using IBM software and IBM (or compatible) hardware, the data analysis sections of this paper incorporate discussions of the peculiarities of such IBM systems. Further, because we are stressing the direct practical implications of our results, we consider, in detail, issues that might not be relevant to systems with a different architecture. Therefore, this paper presents not only "abstract" research results, but also engineering solutions to real problems.

As explained in the next section, many of the aspects of disk cache design are sensitive to program behavior and file access patterns, and are therefore best evaluated empirically. In Section 2 we discuss our evaluation methodology and the data that we have. In Section 3 we present the results of our experiments, and in Section 4 we discuss those design considerations that are either not suitable for experiment or for which we have no relevant measurements. In Section 5 we consider briefly a related topic, disk arm buffers. Alternatives to disk cache are examined in Section 6, and current commercial products are described in Section 7. An overview is provided in the conclusion.

2. METHODOLOGY AND DATA

2.1 Disk Cache Effectiveness

The final measure of disk cache effectiveness is the change in the appropriate system performance measure (usually either response time or throughput) for a given disk cache system. The worth and desirability of disk cache must then be determined by taking into account the performance improvement (if any) and the cost(s) involved. In this paper we do not try to calculate the overall system performance impact of disk cache 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 [77], which has published some performance figures on an operating disk cache system. Additional performance figures appear in [19], [33], and [57].

It is possible to make some estimates of the effect of disk cache on mean disk I/O access times. That is done in [11], where it is noted that disk cache may or may not yield any performance benefits, depending on the hit ratio and the design.

We, instead, evaluate the effectiveness of disk cache designs in two ways. First, we measure the *miss ratio*, which is the fraction of I/Os that are not captured by the disk cache, given certain disk cache parameters. As noted below, miss ratio measurements are 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; for example, I/O path contention depends on the path configuration. Second, some aspects, such as error recovery, are not suitable for miss ratio analysis. Thus, for topics for which miss ratios cannot be generated, discussions are presented instead.

2.2 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.). 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 successfully to evaluate virtual memory systems [6] and cache memories [74] using virtual address traces, and CPU scheduling [64] 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). Therefore, we believe that the miss ratio analysis presented is valid and accurate, to the extent that our workload sample is adequate.

2.3 Data Sources

Three large IBM System/370 (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 IBM's Generalized Trace Facility (GTF) [46]. 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 disk cache designs studied is the track. In all cases, only those I/O 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. We note that these three systems constitute a limited workload sample; each system is large and they use the same architecture and similar operating systems. Although we are confident that our results are representative of a broad class of systems, the data presented apply only to the systems and the period traced. In particular, it should be expected that some aspects of systems that magnify the effectiveness of disk cache, such as the use of small physical block sizes and the frequent use of temporary files, will diminish over time. We do feel, however, that large systems based on software from vendors other than IBM will have many similar characteristics. It is also important to note, however, the importance of having measured three *real* systems, rather than having modeled one hypothetical one.

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 [32]), 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 [45] disks (@100 Mbytes), 27 IBM 2314 disks (@29 Mbytes), two IBM 2305 drums [44], 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 twenty-five 3350 [47] disks (@317 Mbytes), sixteen 3330-11 disks (@200 Mbytes), seven 3330 disks, and numerous tape drives, unit record devices (printers, card readers, etc.), and telecommunications controllers.

The third system traced was at the Hughest 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 forty-nine IBM 3330s, nine 3330-11s, STC Superdisks equivalent to sixteen 3330-11s, one IBM 2305-2, twenty-four tape drives, communications lines, and numerous unit record devices.

2.4 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; that is, the device address and track and cylinder location. Also used, with some modification, was IBM's System Management Facility (SMF) [48], 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 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 Acknowledgments). Further, the amount of data gathered is very large; a one day trace consists of about 1.5 Gbytes of data, or about 10 full reels of 6250-bpi tape. (We therefore believe that it will be some time before a more varied and larger workload is available.)

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 disk cache simulation is one track. The track size, of course, varies with the device; track and cylinder sizes are shown in Section 3.3. Second, I/O events were not tagged as to read or write; therefore, measurements or studies that 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).

2.5 LRU Stack Analysis and Set-Associative Mapping

Almost all of our disk cache simulations use a technique known as least recently used (LRU) stack analysis [60]; 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., [74]) to reduce the mean stack depth; this technique is actually used in some commercial implementations of disk cache with which the author is familiar. The number of sets used is shown in all plots and tables. Experiments that showed that the effect of the number of sets was very small for realistic disk cache sizes were run (but are not presented here for brevity). (Realistic cache sizes mean more than 64 kbytes per device, 256 kbytes per controller, and 1 Mbyte per overall system.) 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.

Table I. Systems Studied

Type of data	Site		
	SLAC	Crocker	Hughes
Operating System	SVS	MVS	MVT
CPU	370/168	370/168	470V/6
Trace period	5 pm–4 pm	5 pm–12 noon	6 pm–11 am
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 II. 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 or Wylbur	.4054	.068
Other/TSO or Wylbur	—	.015
System/system	.4563	.167
Paging/system	.0136	.072
Other/system	.0005	—

2.6 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 that resulted in a seek taking place. As has been previously noted (e.g., [65] and [58]), 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, and job entry system) 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 multiprogramming that paging is relatively infrequent; further, the use of the SVS operating system, which allows only a total address space of 16 Mbytes among all processes, restricts the overcommitment 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

Table III. Distribution of I/Os among Device Types

Device type and 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

is directed toward the newest and fastest disks. Second, the proportion of I/O directed toward the tapes varies a great deal and ranges from 5–30 percent.

3. DISK CACHE MISS RATIO ANALYSIS

As noted earlier, there are a number of aspects of disk cache design that 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 such issues 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) is 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.

3.1 Cache Capacity

Perhaps the most basic aspect of cache design is the cache size, that is, how large it should be in order to obtain a given hit ratio. In Figures 2–4 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, which has been done and presented in Figures 5–7.

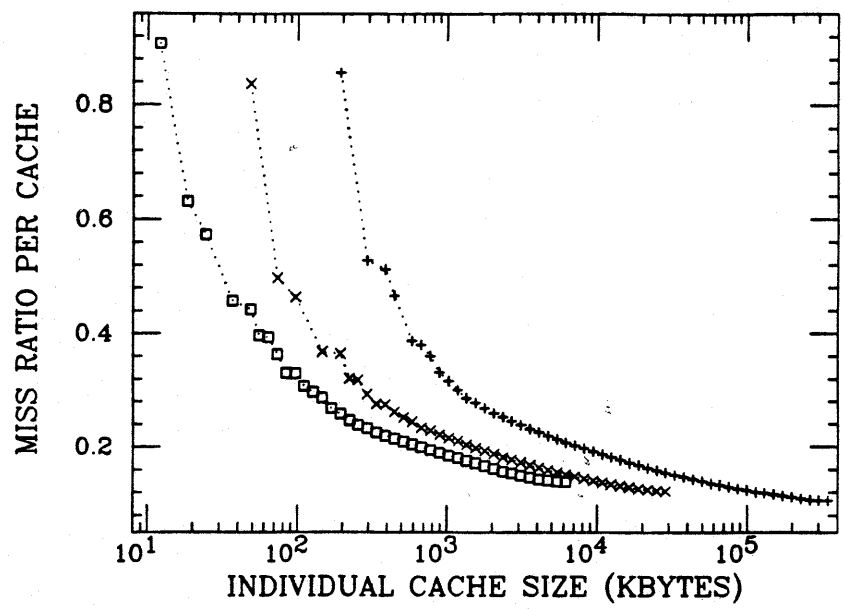


Fig. 2. Effect of cache location. Crocker: 3.4 million seek address; 1 track blocks. + = global, 16 sets; X = string controller, 4 sets; □ = device, 1 set.

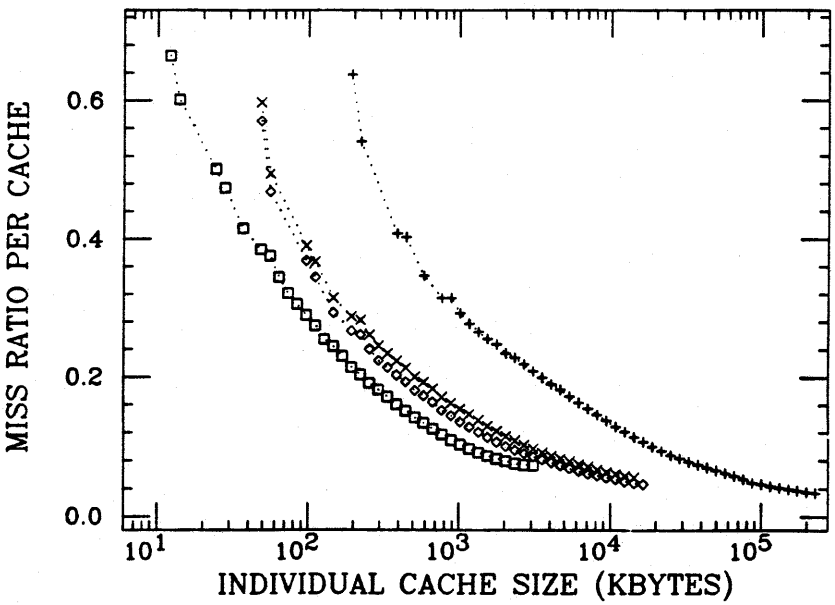


Fig. 3. Effect of cache location. Hughes: 2.7 million seek address; 1 track blocks. + = global, 16 sets; X = channel, 4 sets; ◇ = string controller, 4 sets; □ = device, 1 set.

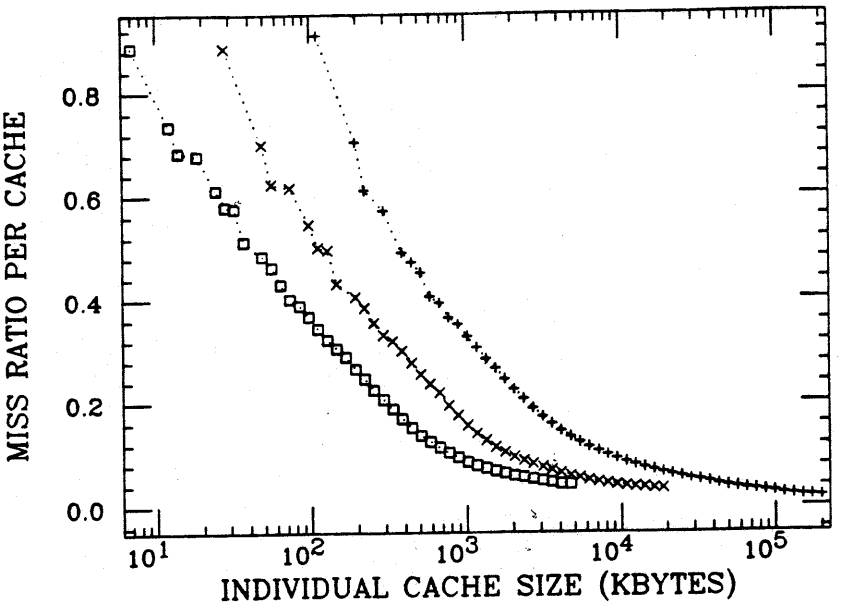


Fig. 4. Effect of cache location. SLAC: 3.7 million seek address; 1 track blocks. + = global, 16 sets; X = string controller; 4 sets; □ = device, 1 set.

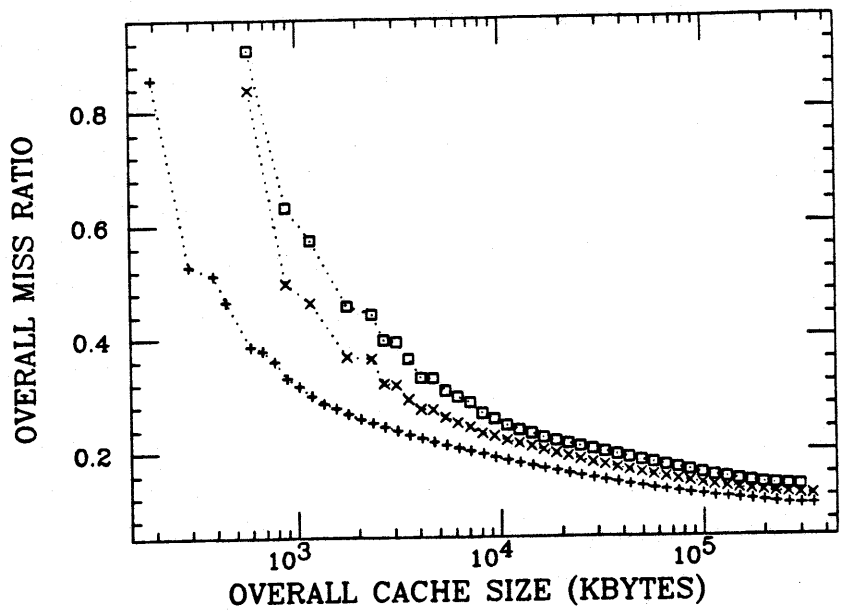


Fig. 5. Effect of cache location. Crocker: 3.4 million seek address; 1 track blocks. + = global, 16 sets; X = string controller, 4 sets; □ = device, 1 set.

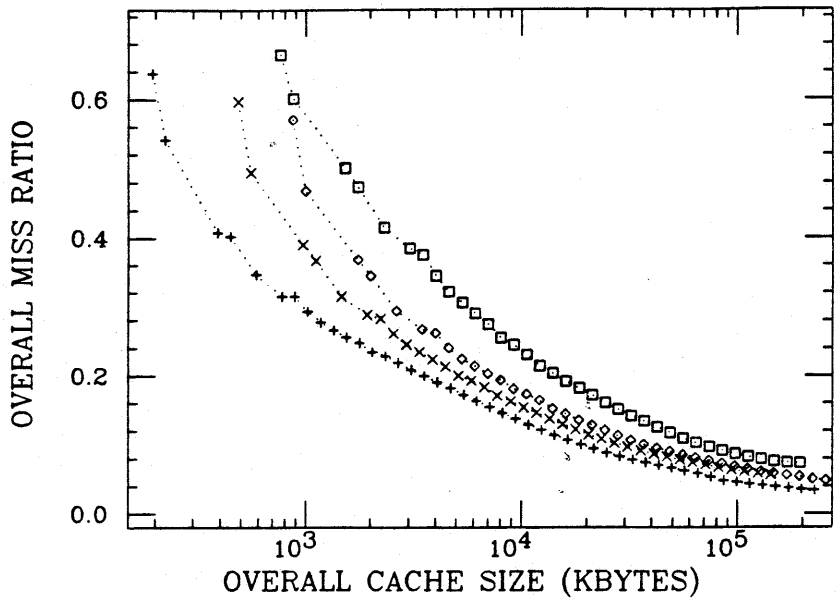


Fig. 6. Effect of cache location. Hughes: 2.7 million seek address; 1 track blocks. + = global, 16 sets; X = channel, 4 sets; ◇ = string controller, 4 sets; □ = device, 1 set.

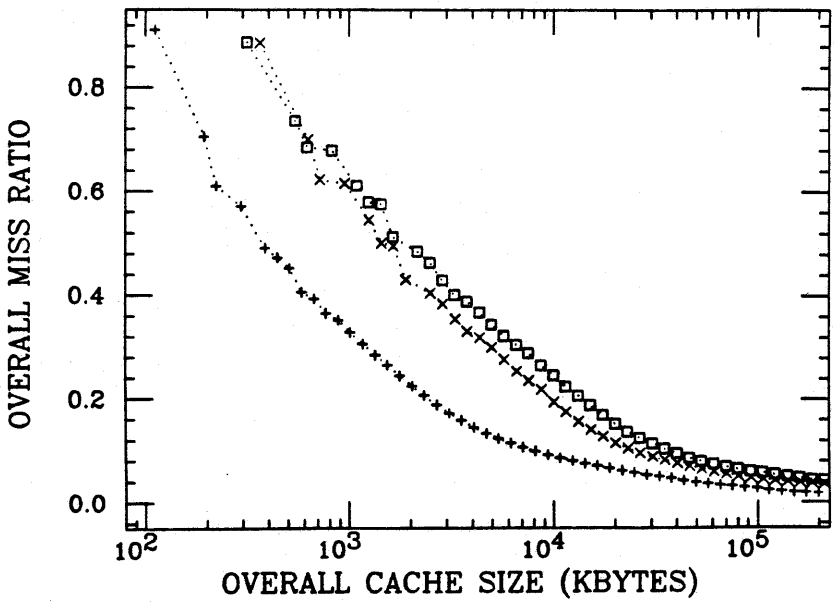


Fig. 7. Effect of cache location. SLAC: 3.7 million seek address; 1 track blocks. + = global, 16 sets; X = string controller, 4 sets; □ = device, 1 set.

As may be seen from the data presented in Figures 2–7, the miss ratios for the global cache are already 26 percent or less at 2 Mbytes. 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 Mbytes of total cache capacity appears to capture the "knee of the curve" 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 might need a larger cache.) We note that the subsequent mapping of hit ratio into overall system performance is not considered here.

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 2–7. Examining the controller miss ratios, we again see significant differences between the systems. It appears, as a generalization, that capacities of from 512 kbytes to 2 Mbytes per string controller seem to again include the "knee of the curve." At the device level, 256–512 kbytes seem to be needed.

3.2 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 among 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 3.1, 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 and Figures 5–7. 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 imbal-

Table IV. System Configuration

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

Table V. Miss Ratios

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

ance) and over long periods of time (days, weeks) some devices are still much more busy. For example, in [3] it is stated that over moderate to short periods of time 60 percent 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, if there are multiple CPUs, then a cache near the device will be shared by the CPUs, with corresponding efficiency considerations.) In addition, it is worth noting that one is likely to choose not to cache all of the devices, channels, or strings, omitting those with poor hit ratios or low traffic. Thus, the penalty for device, string, or channel caching is not likely to be as high as Table V and Figures 5–7 suggest.

Consistency is the 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 [74], where CPU caches are discussed. See also Section 4.6.

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 complexity of a shared cache is not likely to be as costly as replicating a simpler cache. This suggests that placing the cache in the disk spindle might not be cost effective, and a disk cache in main memory should be the lowest cost solution.

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. Placing the controller beyond the channel means that no operating system or channel time is eliminated; the latter is quite significant [39] and can typically be twice as large as the data transfer time. This performance advantage for placing the cache in main memory is substantial.

On the basis of the above discussion, it is not possible to specify a uniquely best location for a cache, but there are two that seem most suitable. *Placing the cache at the main memory should eliminate the most overhead and provide the best performance improvement, if operating system modifications and address space limits do not make this approach impractical. A cache at the storage controller has commercial and practical advantages, in that it should require few if any operating system modifications and can be sold independently of other components of the system.* Some vendors have chosen to place the cache in the storage controller (IBM [51, 53], Memorex [61]), outboard at the CPU (NEC [private communication with T. Tokunaga, Sept. 1980]), or inboard, with the CPU main memory [private communication with mainframe manufacturer], [63].

3.3 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. For the 2314, 3330, and 3350 the number of bytes per track and tracks per cylinder are respectively, 7294 and 20, 13030 and 19, and 19254 and 30. Miss ratios for all three sites and for those eight different block sizes are shown in Figures 8–10. It can be seen in those figures that for small cache capacities, small block sizes give the lowest miss ratios. 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 block sizes in bytes can be computed by taking the track sizes weighted by the relative frequency of device types, from Table III.

The reason for the behavior displayed in Figures 8–10 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, instead of 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 piecemeal using smaller blocks.

Given, as noted before, that the appropriate buffering capacity is from 2 to 8 Mbytes, 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., [11].) Also, physical disk blocks need not end on a cylinder boundary, which will either result in a block that is partially

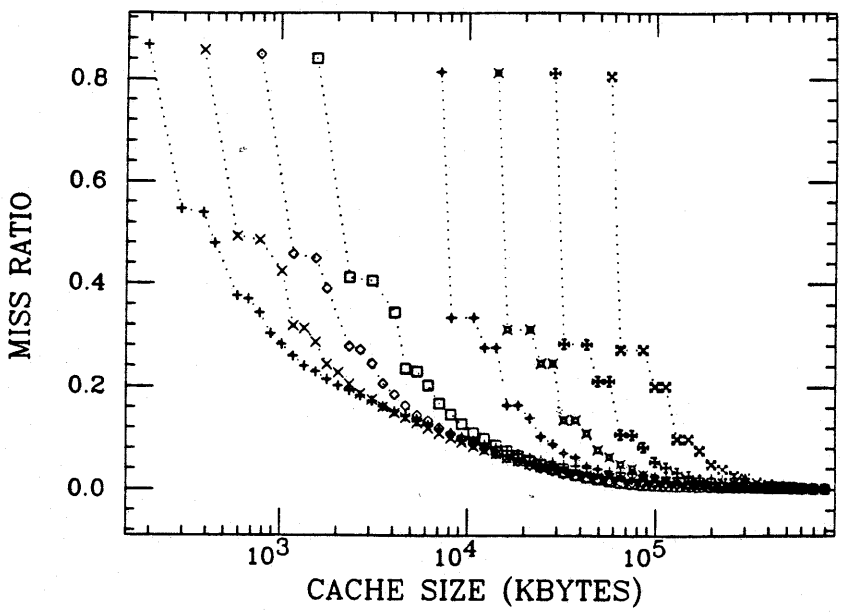


Fig. 8. Effect of block size. Crocker; 835 thousand seek address; 16 sets, global buffer. + = 1 track, X = 2 tracks, ◇ = 4 tracks, □ = 8 tracks, ✧ = 1 cylinder, ⌑ = 2 cylinders. ✢ = 4 cylinders, ⤫ = 8 cylinders.

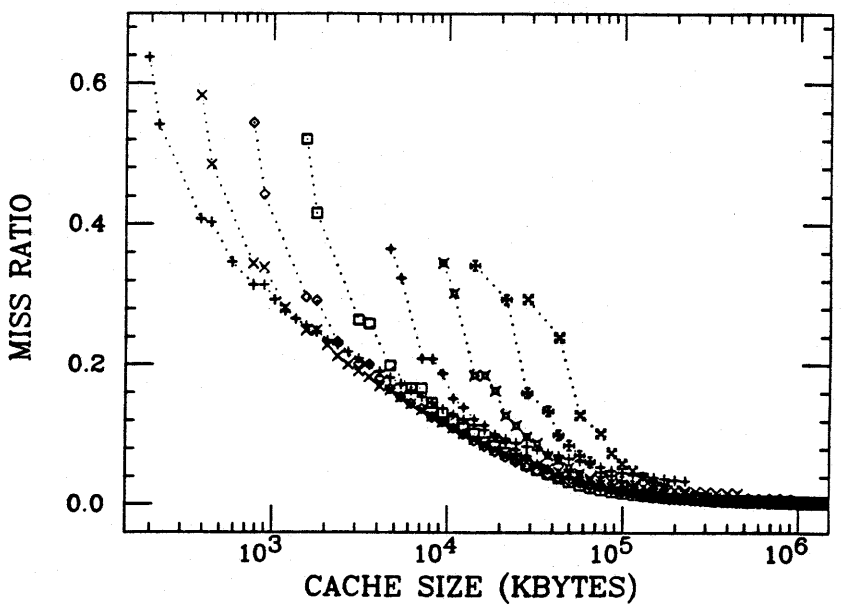


Fig. 9. Effect of block size. Hughes: 2.7 million seek address, 16 sets, global buffer. + = 1 track, X = 2 tracks, ◇ = 4 tracks, □ = 8 tracks, ✧ = 1 cylinder, ⌑ = 2 cylinders, ✢ = 4 cylinders, ⤫ = 8 cylinders.

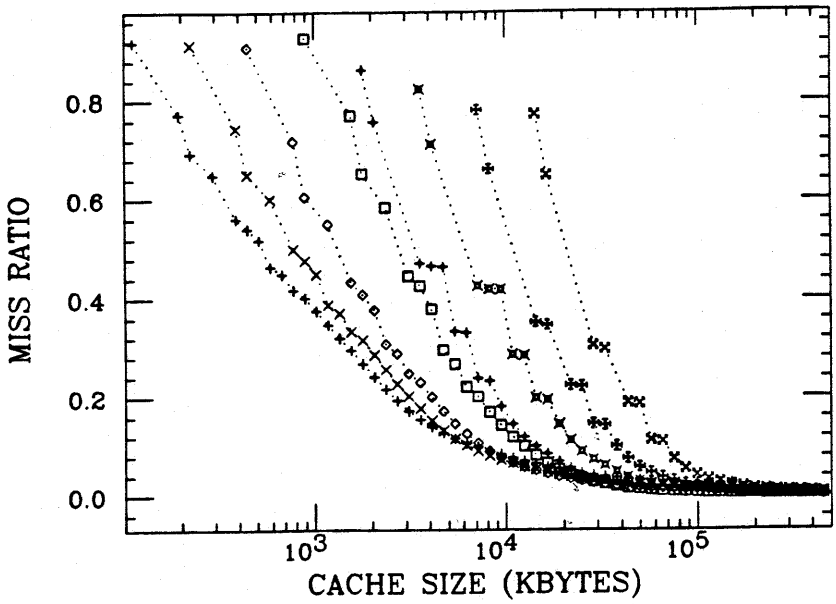


Fig. 10. Effect of block size. SLAC: 673 thousand seek address, 16 sets, global buffer. + = 1 track, X = 2 tracks, ◇ = 4 tracks, □ = 8 tracks, ✧ = 1 cylinder, ⌑ = 2 cylinders, ✢ = 4 cylinders, ⨯ = 8 cylinders.

Table VI. Lowest Miss Ratio Block Size in Tracks—Global Set Associative Buffering, 16 Sets

Capacity (Mbytes)	Crocker	Hughes	SLAC
1	1	1	1
2	1	2	1
4	2	2	1
8	4	2	2
16	4	4	4
32	4	4	4
64	8	8	8

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 [25] for a discussion of the problem of multitrack physical blocks.)

3.4 Migration Algorithms

3.4.1 *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.

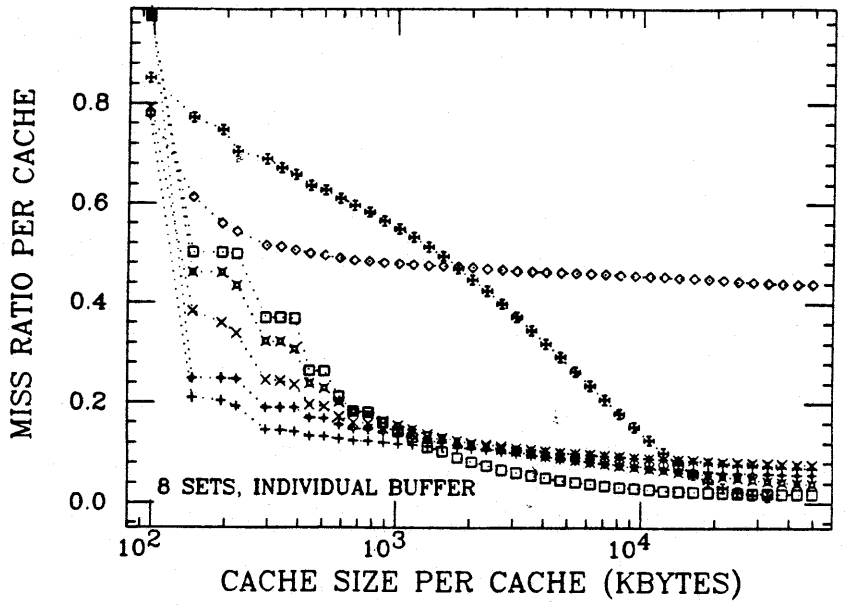


Fig. 11. Effect of file and user type. Crocker: 3.4 million seek address, 8 sets, individual buffer. + = temporary. X = system 1 batch, ◇ = other/batch. □ = system/TSO, ✧ = other/TSO, ♢ = system/system, ✢ = paging/system.

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 temporary files and then for 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 users, it appeared in two different caches). The miss ratios for each class for 1-track blocks appear in Figures 11 and 12; data for the Hughes system were 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) appears to show very little locality; that is, the miss ratio does not drop significantly until most of the paging data set is in the buffer. Since we are buffering using the physical disk address, 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 these 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. It is also possible that there is little locality remaining in the memory references constituting the stream of "misses."

(b) SLAC and Crocker show different comparative results for the effectiveness of buffering batch versus buffering system data sets. The SLAC data suggest that temporary data sets show poor locality compared with batch data sets. The

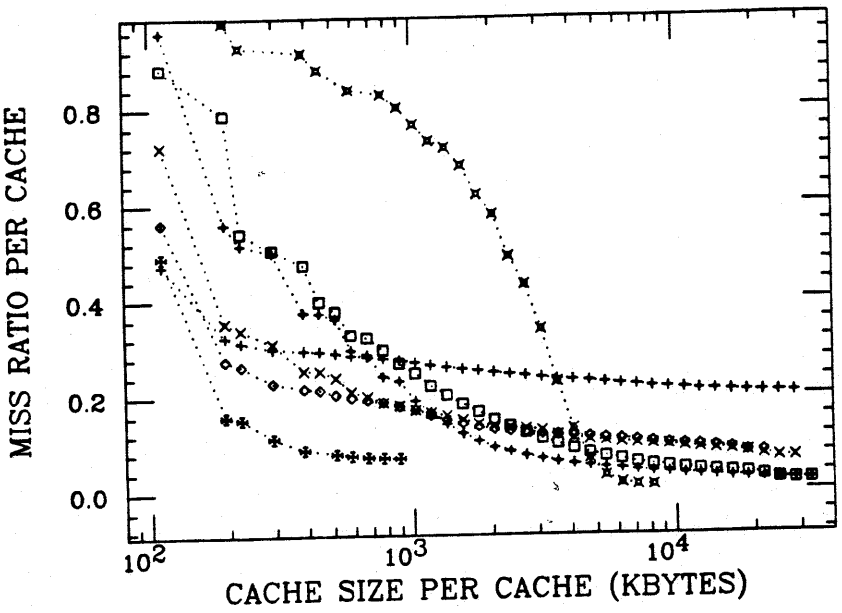


Fig. 12. Effect of file and user type. SLAC: 3.7 million seek address, 8 sets, individual buffer. + = temporary. X = system/batch, ◇ = other/batch, □ = system/wylbur, ✧ = system/system, ⌑ = paging/system, ✣ = other/system.

Crocker data show 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 nor 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.

For comparison with our results, we note suggestions in [19] for what should and should not be cached in IBM (software) systems. Dahman and Grossman recommend the following as strong candidates for caching: program load libraries, message format services library, application control block library, VSAM KSDS indices, the system catalog, and sort and program product libraries. Poor candidates for caching include databases with poor locality of reference, long message queues, the dynamic log, disk scratch pad areas, and restart and recon data sets. Intermediate candidates include OSAM databases, VSAM ESDA databases, and short message queues.

3.4.2 *Prefetching.* The standard method of moving data into a cache is called *demand fetch*, by which data are moved into the cache at the time they are first referenced; thus demand misses result in a latency period during which the desired information is not available. If it were possible to guess accurately 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, [69] for prefetching in database systems and [70] for prefetching to cache and main memory. (See also [7].) For this study we have examined only the prefetching method known as *one-block lookahead* (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 (asychonously) 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*, the phenomenon by which the cache is polluted with blocks that 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 13–15; in each case, prefetching produces a very significant drop in the demand fetch miss ratio, usually on the order of 10 to 50 percent for reasonable cache sizes. We have also tabulated the effect of prefetching on the basis of 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 percent) 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. Intermediate between the two cases are system files and files used by the system that 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 file on the device(s).

It is worth noting that our prefetching has used the crudest algorithm possible, OBL. A scheme such as that in [69], in which a variable number of blocks are prefetched depending on the observed degree of sequentiality, could be expected to perform better. In the article cited, variable prefetching performed significantly better than prefetching by a fixed number of blocks. Therefore, we believe that prefetching should be implemented, but made optional, and the implementation should be one that minimizes the costs and overhead associated with prefetching, as noted above.

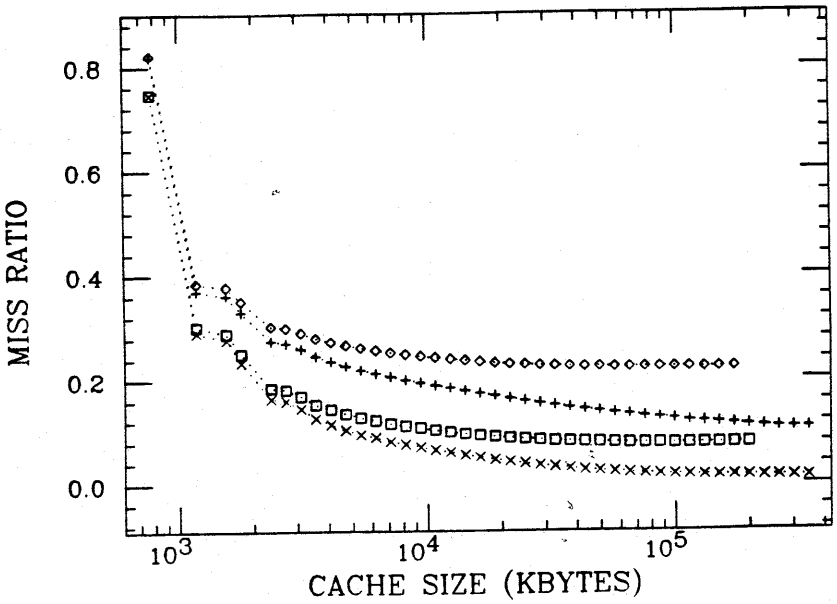


Fig. 13. Effect of migration algorithm. Crocker: 3.4 million seek address, 1 track blocks, 64 sets, global buffer. + = demand, X = prefetch, ◇ = purge behind, □ = both.

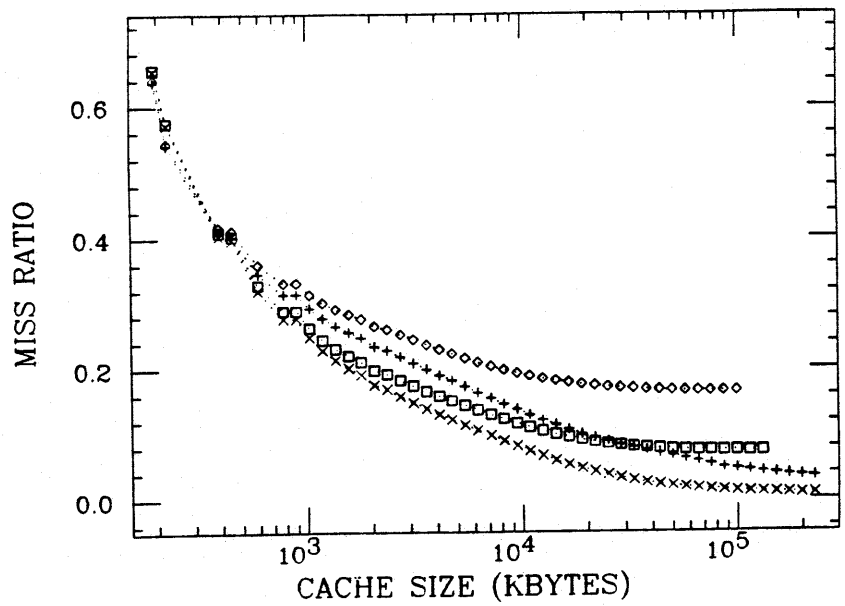


Fig. 14. Effect of migration algorithm. Hughes: 2.7 million seek address, 1 track blocks. 16 sets, global buffer. + = demand, X = prefetch, ◇ = purge behind, □ = both.

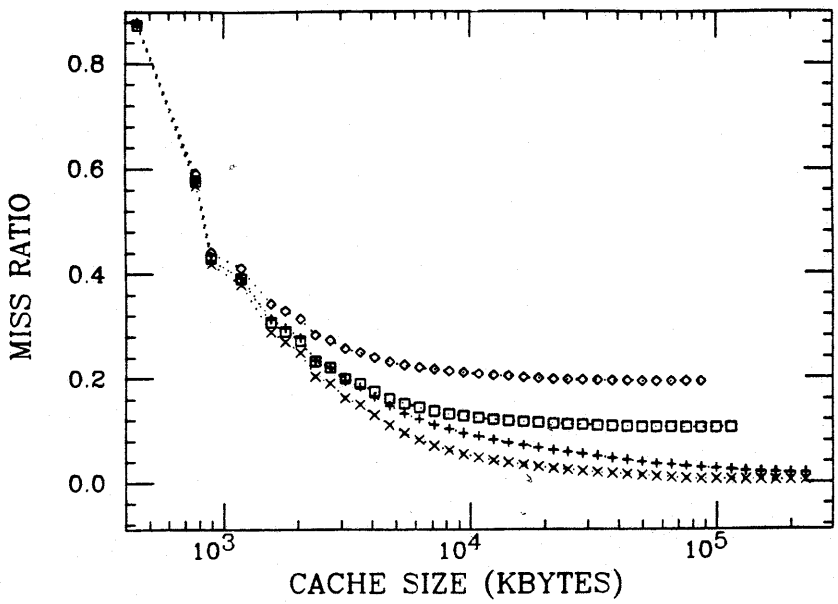


Fig. 15. Effect of migration algorithm. SLAC: 3.7 million seek address, 1 track blocks, 64 sets, global buffer. + = demand, X = prefetch, ◇ = purge behind, □ = both.

3.4.3 *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 13–15 also show the effect of purge behind alone and 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.

3.5 Which Devices or Controllers to Cache

Illustrated in Figures 16–18 are miss ratios for the 7 or 8 most heavily used devices for the Crocker, Hughes, and SLAC computer systems. By referring to tables that 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

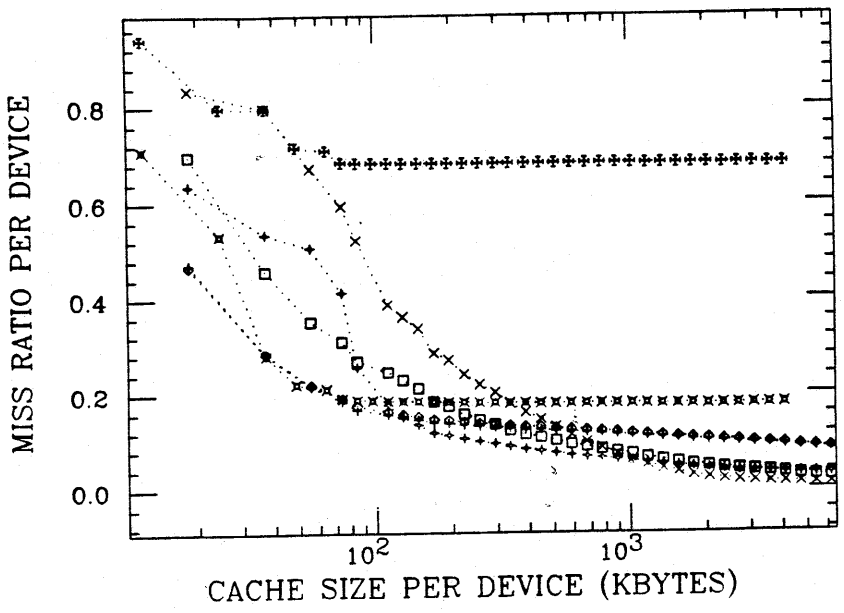


Fig. 16. Miss ratio for various device caches. Crocker: 3.4 million seek address, 1 track blocks, device buffer, 7 most heavily used devices.

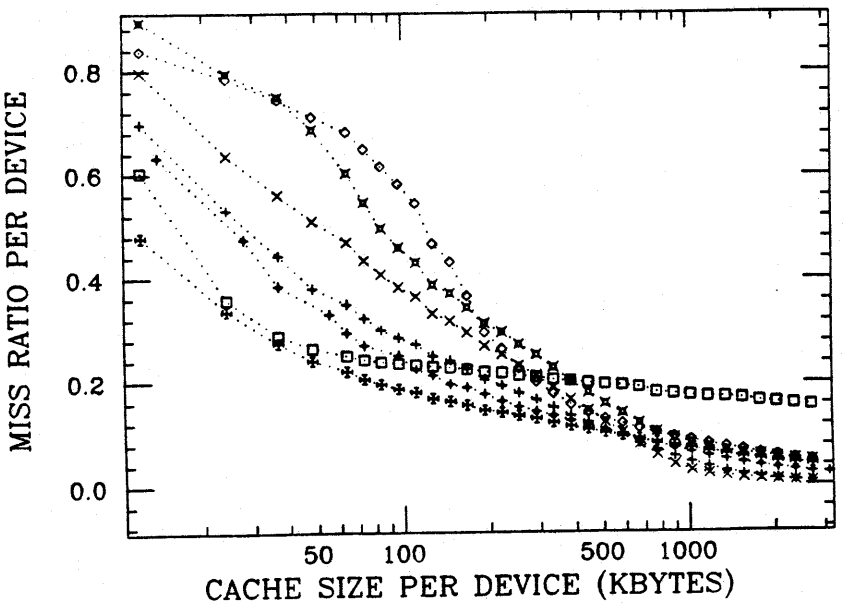


Fig. 17. Miss ratio for various device caches. Hughes: 2.7 million seek address, 1 track blocks, device buffer, 7 most heavily used devices.

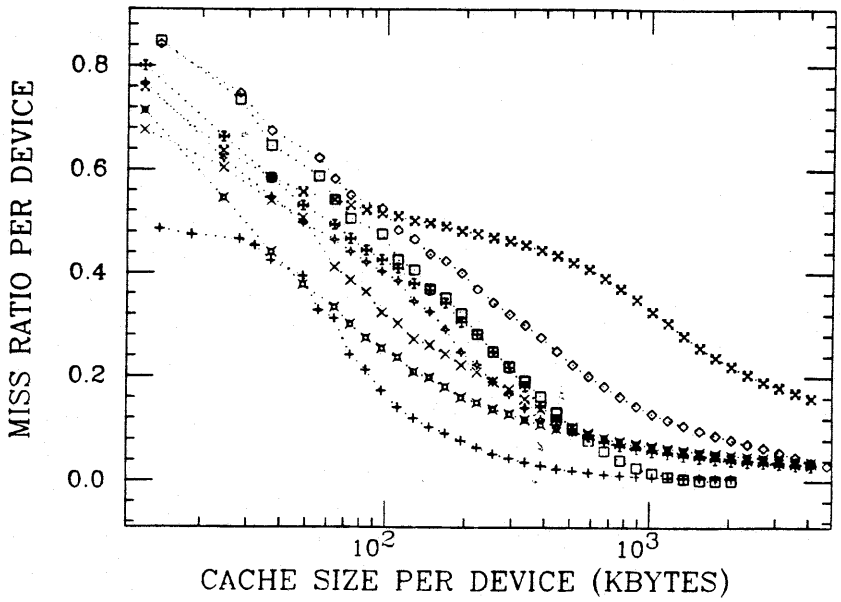


Fig. 18. Miss ratio for various device caches. SLAC: 3.7 million seek address, 1 track blocks, device buffer, 8 most heavily used devices.

and job queue, and the Orvyl (time-sharing) file system. (The next worst results are for user scratch packs.)

On the basis of these observations and those in Section 3.4.1 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) Devices containing user-defined data sets should be cached only if the block sizes are small enough for there to be 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.

3.6 Time of Day Effects

In Tables VII–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: on-line TSO and IMS applications (during the day) have relatively small data working sets. The batch job and batch IMS applications that run at night (e.g., check processing) have

Table VII. Effect of Time Period on Miss Ratio—Crocker Bank: 1 Track Blocks; Global Buffering, 16 Sets

	Trace section				
Cache size (Mbytes)	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 VIII. Effect of Time Period on Miss Ratio—Hughes Aircraft; 1 Track blocks; Global Buffering, 16 Sets

	Trace section		
Cache size (Mbytes)	0–1M	1–2M	2–3M
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 IX. Effect of Time Period on Miss Ratio—SLAC: 1 Track Blocks; Global Buffering, 16 Sets

	Trace section				
Cache size (Mbytes)	0–1M	1–2M	2–3M	3–4M	4–5M
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

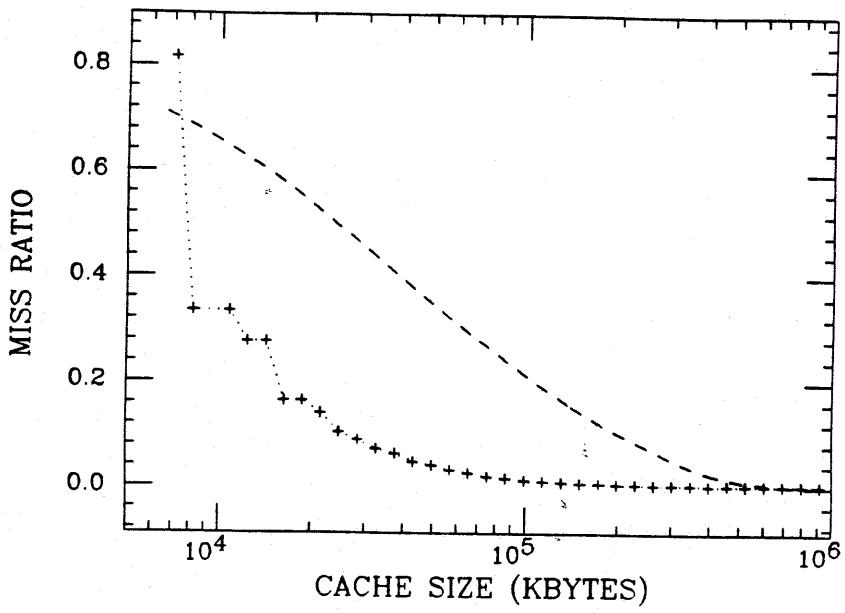


Fig. 19. Comparison of static and dynamic cache. Crocker: 835 thousand seek address, 1 cylinder blocks. + = dynamic; 16 sets; − = static.

poor locality. Conversely, both SLAC and Hughes run similar workloads during the day and night.

These time-varying performance figures will reinforce our later discussion (Section 4.4) about the advantages of dynamic on/off for the cache.

3.7 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 that we have discussed so far. The idea is to allocate statically 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 nonlook ahead optimal results, as with the A_0 algorithm for the independent reference model for program behavior.

The method that we use to get a static allocation is optimal in the same way as A_0. 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 19–21, 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

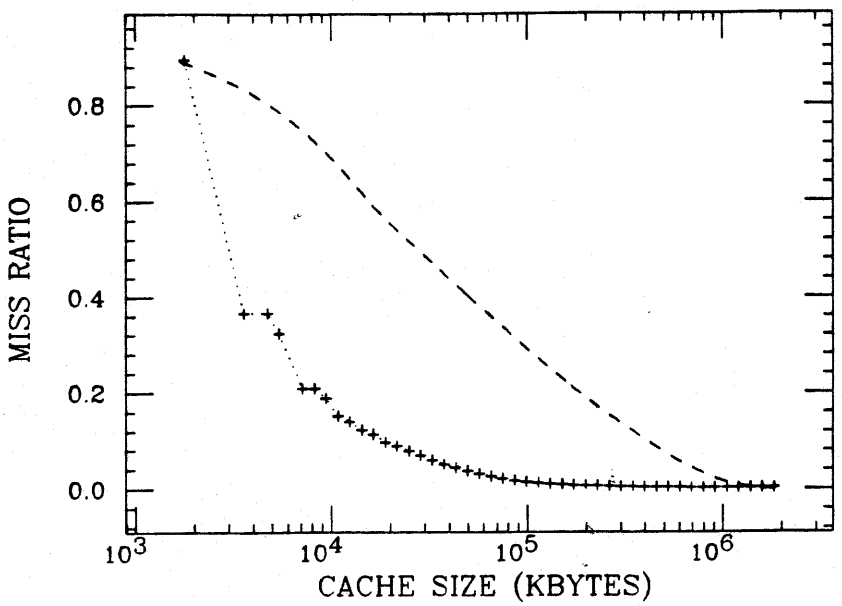


Fig. 20. Comparison of static and dynamic cache. Hughes: 2.7 million seek address, 1 cylinder blocks. + = dynamic, 16 sets; − = static.

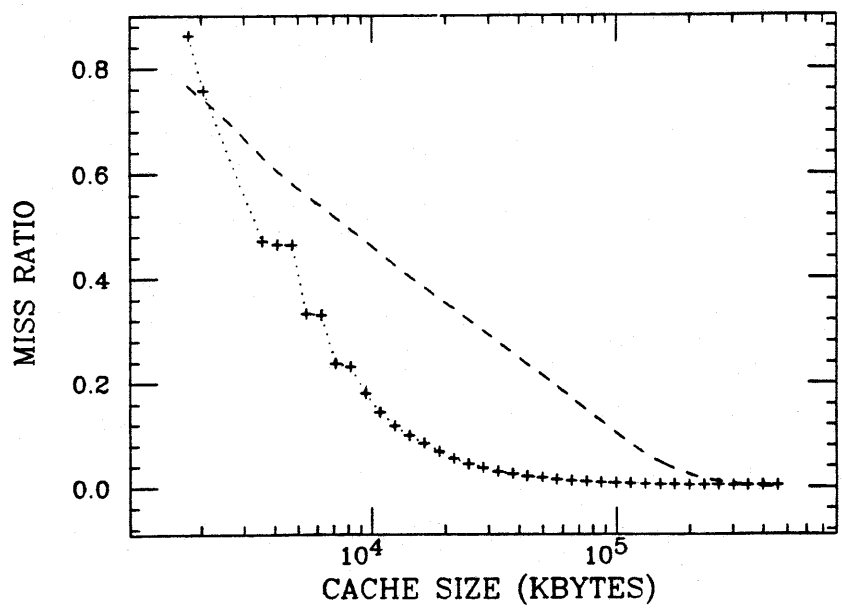


Fig. 21. Comparison of static and dynamic cache. SLAC: 673 thousand seek address, 1 cylinder blocks. + = dynamic, 16 sets; − = static.

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. Note, however, that devices such as electronic drums need not be managed statically—data can be migrated in and out in the same manner as is done for a cache.

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 that are sensitive to the time invariance of the $\lambda_{i,j}$ assumption are, therefore, also not valid.

4. DESIGN CONSIDERATIONS

It was noted earlier that there are a number of design considerations that we have not evaluated by means of a miss ratio analysis. There are three reasons why we have not done so: First, our data are limited (e.g., no read/write differentiation) and, therefore, some studies are not possible. Second, some items, although 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.

4.1 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 [39], any access time less than 10 ms will result in a physical disk access time being reduced by more than 50 percent. Therefore, the access time can be allowed to be primarily a function of the technology, as discussed below in Section 4.5 MOS RAMs or charge-coupled devices (CCDs) 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 devices access time may not be the limiting factor.

The bandwidth of an outboard 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., 4 kbytes) in less than 1 ms, this implies a data rate of at least 4 Mbytes/s. This compares with a current maximum of 3 Mbytes/s in IBM systems (to/from the 3380 disk) and up to 5 Mbytes/s in Cray I systems. Thus, figures in this range are already available, and it should be straightforward to obtain higher rates 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.) We also note that if the disk cache is in main memory, bandwidth is not a problem.

4.2 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-Mbyte/s 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 per string and j strings per controller. Let the utilization of the disk (excluding seek time and missed rotational position sensing (RPS) delays; that is, including only rotational latency and transmission time) be x. Then the disk transmits $(1/9)x$ time. The string controller is busy with data transmission $(k/9)x$. The storage controller is busy for data transmission $(jk/9)x$. (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 three times as busy as the string controller. (The channel is likely to be even busier than the storage controller, since there are usually fewer channels for access to the disk system than there are storage controllers). Even allowing for seek time (approximately 40 percent of I/Os require seeks, at a mean of 30 ms/seek. (see Table I)), and missed attempts to reconnect to the disk as the desired record comes under the read head (at 16 ms/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 [39]. See [2] and [10] for a discussion of some of these issues.

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; 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; that is 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 4.11. We note that this problem is avoided if the cache is placed in main memory.

4.3 Write Through versus 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, it is 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-fifth to one-third of all I/Os [33], [57], 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. System performance should be better for copy back than for write through, but we have not modeled that issue.

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 [24] that two completions be signaled 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 ensured).

There is a peculiarity to some system architectures (IBM's in particular) which makes it difficult to implement a copy back or write through cache. In IBM's count-key-data architecture [71], a sequence of data commands is followed by the command that indicates whether the transfer is actually a read or write. Therefore, the transfer cannot 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 [71, 49, 50].

A minor complication that 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 [77] 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.

4.4 Dynamic Cache On/Off

It is quite possible for a disk cache to lessen rather than improve performance (see e.g., [11]). Consider, for example, a file that is stored with one large block per disk track and that is read once. In such 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. In [26], for example, a number of situations are suggested in which cache loaded would be inhibited.

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). (Such monitoring is available for commercial disk caches [17], [55].) 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 it 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.

4.5 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 [37], that they would be two–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 $100, 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 may be an advantage to outboard (external) 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 that consistency problems may be worse if data are shared between independent CPUs, and that operating system modifications are certainly required for a main memory cache. Using part of main memory as a disk cache is, of course, another good implementation and is likely to provide greater performance improvements than an outboard cache.

4.6 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 issue is that at any point in time there must 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 take place via

the same cache; this solution may not permit sufficient bandwidth, however. Further, it will not work if the cache is located in the channel or main memory, neither of which need be shared by 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 this 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 [74] 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 such 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. Numerous solutions to these problems exist, and further consideration of this issue is beyond the scope of this paper.

4.7 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 [77].

There are, therefore, four 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) An external copy back cache must be nonvolatile, which implies either magnetic bubbles or battery backup. 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. (4) There should be provision for forcing writes to the disk surface; this feature would be used by reliability and recovery software.

4.8 Software versus Hardware Management and Locus of Control

An external 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; (4) the *data transfer logic*. The question about the control mechanism and the transfer logic is whether they should be implemented in software or hardware, and where.

Either the locus of control for the cache can be 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 why the latter should be used only when the cache is in main memory or is directly associated with the CPU: (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 are likely to be changed and updated, which is done much more easily in software or microcode. Conversely, the data transfer logic, which must transfer data at high (≥3 Mbytes/s) rates, will have to be hardwired; software is not fast enough. Some of these points are made in [22].

4.9 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 [56] 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 those that 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[1] [63], for example, keeps a main memory I/O buffer. IMS [20] maintains a buffer of recently accessed blocks of the database. It is preferable that mutiple 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 memory 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. Thererefore, 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 trade-offs 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

[1] UNIX is a trademark of AT&T Bell Laboratories.

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 handling I/O interrupts and executing the dispatcher at least twice. If the time to complete the I/O or page fault 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 that is using a lot of cache blocks.

4.10 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 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 3.2), 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 [77]. In that system, when running real commercial workloads, performance improvements using all possible file modes (described further in Section 7.1) 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 a file. In an IBM OS-based system, for example, job control language (JCL) parameters can be added.

4.11 Performance Impact

As noted earlier, we have not attempted to calculate directly the performance impact of disk cache in this paper. In the literature there are some papers that do look at the effect of disk cache on the mean I/O access time. In [11] it is

observed that if the read/write ratio is high enough and the hit ratio is high enough, then the use of disk cache can significantly cut mean I/O time; conversely, if the hit 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. Additional analysis is available in [59]; it also shows performance improvements from disk cache.

In [77] and [57], it is reported that the use of disk cache (respectively, that of NEC and IBM) yields overall gains in system performance.

5. 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 [39], [2], 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* [40]). 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. Previously, if the channel utilization became too high (above 30 or 40 percent) [9], RPS misses became quite frequent and costly. With a spindle buffer, queuing 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 slowed down if the channel or storage controller is otherwise too busy to operate at the full data rate.

6. 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., [8]) that block sizes are frequently small and that larger block sizes improve performance [8, 73]. 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 I/O 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 portion 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 database system keeps its own buffer of recently used blocks. Similarly, UNIX maintains its own main memory I/O buffer. These buffers are effectively disk caches located in main memory. 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 would 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.

Disk cache can be implemented in main memory by using the already existing virtual memory management system. The idea is to map the disk address space into the program address space and then rely on the paging mechanism, as is done in the Multics system [62]. Provided that either prefetching or large block sizes are used for those data sets benefiting from them, this approach should achieve performance gains similar to the more single-purpose disk cache.

The idea of disk cache is that it is a self-managed cache that retains recently used blocks of data. An alternative is to create an explicitly managed electronic storage device, such as the electronic drugs made by Intel (FAST-3805 [42] and 3825) and Storage Technology (4305). There already exists software to manage drums and maintain on them the most frequently used information. In some cases that software may work fairly well. We note that these electronic drums are managed dynamically, not statically, and thus should be able to offer performance comparable to the disk cache, rather than similar to the static device design.

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 the disk cache does; in fact, they are a form of disk cache. Reblocking explicitly accomplishes the prefetching/large block size function of disk cache; it does not capture locality by time. Electronic drums can

function much as a cache. 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 an explicit disk cache becomes unnecessary, since all of its functions have been provided in other ways. *The appealing aspect of disk cache is that it is a simple and elegant solution that avoids the need for many small and costly system changes and improvements.*

7. COMMERCIAL DISK CACHE PRODUCTS

In the last few years, a number of disk cache 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.

7.1 The Nippon Electric (NEC) Integrated Disk Cache

NEC has designed and tested a disk cache for their ACOS 1000 system [77]. That machine is structured around a *system control unit* (SCU) through which all I/O 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. (A design similar to this is suggested in [5].) Further, the operating system is aware of the disk cache, and the type of caching provided is determined by the operating system on the basis of file type. Performance is also improved by providing a 5.3-MByte/s 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 percent are observed in practice, and the elapsed job times decrease by 10–35 percent. Response time decreases for interactive use by

15–25 percent. It has also been observed [private communication with T. Tokunaga, Sept. 1980] that the use of basic mode alone yields only about half of the potential performance gains.

7.2 IBM 3880 Model 11 and Model 13

The IBM 3880 Model 11 storage controller [51, 52] is a storage controller expressly designed for paging and swapping. It has an internal disk cache of 8 Mbytes and can transfer data to the channel at up to 3 Mbytes/s. (More recently, the capacities of the Model 21 and the Model 23 have been expanded to 32 Mbytes [30].) 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 eight model 3350 disks, although two disks are recommended per cache. Access time on a page read hit is 2.4 ms. It uses copy back rather than write through. It appears that there is only one data path in the storage controller (to which the cache is connected); thus one I/O operation at most can be transferring data at a time.

There appear to be three problems with the 3880 model 11: First, our earlier results suggest that paging data sets have low locality and, therefore, that the hit ratio will be low. It may be possible, however, 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. Finally, there seem to be few if any advantages to caching pages in the same type of MOS RAM storage as main memory; the cost is just as high and the performance worse. The introduction of the IBM 308x architecture with MVS/XA and 31-bit addressing suggests that the one advantage of the 3880/11, a mechanism for using more than 16 or 32 Mbytes of RAM memory, has disappeared. 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 model 13 storage controller [53, 54, 19, 57], uses a cache design different from the model 11 and is explicitly designed for nonpaging I/O. It can have either 4 or 8 Mbytes and can attach to two storage directors. It supports only IBM 3380 disks. The I/O transfer rate is 3 Mbytes/s. 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 and purge behind are implemented for sequential data sets. Tracks are loaded from the point of reference forward; the preceding portion of the track is not fetched. Write through is used with write update, but not write allocate; write hits write first to the cache, and then subsequently to the disk. 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 percent at 4 Mbytes, and 65 to 90 percent at 8 Mbytes. Other benchmark data, from [57], show read hit ratios from 3 installations of 92, 88, and 84 percent.

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 [11]. Actual performance figures, from [57], show decreases in average TSO response time of 12, 25, and 13 percent for three sites; batch throughput changes were −3, + 4, and + 6 percent.

7.3 Other Disk Cache Systems

Some time ago, Memorex [61] designed and built a disk cache (the model 3770) into one of their storage controllers. It contained 1–18 Mbytes of storage which buffered recently used tracks using LRU replacement. The 3770 was never commercially successful, owing, 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 [18, 76, 33]. It consists of from 1.5 to 12 Mbytes of RAM storage associated with two storage directors. (More recently, the cache size has been expanded to 18 Mbytes [31].) A "typical" configuration would be 6 Mbytes of storage and 16 spindles. The design uses write through, with write purge, and full track blocks. Some actual performance figures are presented in [33], where data from two sites are given. Read hit ratios were 76 and 84 percent with 6-Mbyte caches, and write hit ratios were 22 and 77 percent, respectively. For site A, the cache was varied in size, over consecutive 24-hour periods, and read and write hit ratios were, respectively, (.74, .24), (.74, .22), (.76, .22), and (.78, .24) at 1.5, 3.0, 6.0 and 12.0 Mbytes. The paging volume was found to have a low hit ratio.

Amperif manufactures a disk cache that can be used on Sperry Univac computers. Significant performance improvements [15] are reported from its use. Sperry Univac also manufacturers its own disk cache system [75]. Amperif also has a model of its cache [14] that is specifically designed to run with the Airline Control Program on IBM computers.

Computer Automation [38] makes a disk cache to run with its SyFA line of minicomputers. It uses from 0.5 to 2 Mbytes of RAM, achieves an average access time for a hit of 4 ms, and a hit ratio average of 85 percent is claimed. Replacement is by a complex algorithm reflecting both the amount of reference and time since last reference. Write through is used.

Hewlett Packard [29] provides disk caching in versions of its MPE operating system for its series 3000 computers. The Masscomp workstation [34] incorporates disk caching in its main memory.

Other disk caches are announced and/or manufactured by Minicomputer Technology [27] for Digital Equipment Corporation's PDP-11 and VAX, by Point4 Data [12] for its Mark 5 or Mark 8 minicomputers, by Qualex Technology [13] for the HP 3000 computers, by Integrated Business Computers [43], by IBM for the Series I [28], and by Amdahl [16] in its 6880 control unit. One vendor has a software package that simulates a disk cache in main memory. [41]

8. 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, either in main memory or as a separate device, 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 (in a large IBM-type system) of less than 8 Mbytes can capture 60–95 percent of I/Os, with access times for read operations of 2 to 4 ms for an outboard cache, and with shorter access times for a cache in main memory.

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, because of the lack of multiple data paths to/through the cache; those products may not be very useful.

The experiments reported in this paper are only a fraction of the range of data we have gathered. Nevertheless, 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.

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. Our data are not sufficient to study these questions, and more complete data are needed.

ACKNOWLEDGMENTS

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.

REFERENCES

1. AMDAHL, G. M. Storage and IO parameters and systems potential. In *Proceedings of the IEEE Computer Group Conference* (Washington, D.C., June 16–18). IEEE. New York, 1970, pp. 371–372.
2. ARTIS, H. P. Calculating head of string utilization in a shared DASD environment. In *Proceedings of the CMG International Conference* (Dec. 6–8, Washington, D.C.). Computer Measurement Group, 1983, pp. 62–65.
3. BASTIAN, A. L., HYDE, J. S., AND LANGSTROTH, W. E. Characteristics of DASD use. In *Proceedings of the CMG 12th International Conference* (Dec. 1–4, New Orleans, Dec.). Computer Measurement Group, 1981, pp. 107–109.
4. BASTIAN, A. L. Cached DASD performance prediction and validation. In *Proceedings of the CMG 13th International Conference* (Dec. San Diego, Calif.). 1982, pp. 174–177.

5. BATALDEN, G. D., CRABTREE, M. R. AND GOURNEAU D. A. DASD cache for file subsystem. *IBM Tech. Disc. Bull. 27*, 6 (Nov. 1984), 3433–3435.
6. BELADY, L. A. A study of replacement algorithms for a virtual storage computer. *IBM Sys. J. 5*, 2 (1966), 78–101.
7. BENNETT, B. T. AND MAY, C. Improving performance of buffered DASD to which some references are sequential. *IBM Tech. Disc. Bull. 24*, 3 (Aug. 1981), 1559–1562.
8. BERBECK, S., SHIBAMIYA A., TOGASAKI, S., AND YOSHIDA, H. Use of direct access storage devices by MVS customers—Guide survey results. In *Proceedings of the Guide 47 Conference* (Chicago, Nov. 10). 1978, pp. 1121–1138.
9. BERETVAS, T. Performance tuning in OS/VS2 MVS. *IBM Sys. J. 17*, 3 (1978), 290–313.
10. BRANDWAJN, A. Models of DASD subsystems: Basic model of reconnection. *Perform. Eval. 1* (1981), 263–281.
11. BUZEN J. P. BEST/1 analysis of the IBM 3880-13 cached storage controller. In *Proceedings of the CMG 13th International Conference* (Dec. San Diego, Calif.). 1982, pp. 156–172.
12. COMPUTERWORLD. Cache memory reduces data transfer time. *Computer* (Feb., 1982), 103
13. COMPUTERWORLD. HP 3000 access time slashed 400%. *Computerworld* (Feb. 8, 1982), 109.
14. COMPUTERWORLD. Cache disk system out for IBM ACP. *Computerworld* (June 7, 1982), 101.
15. COMPUTERWORLD. Cache disk system speeds access for power firm. *Computerworld* (Dec. 6, 1982), 34.
16. COMPUTERWORLD. Amdahl offers cache for new, older, drives. *Computerworld* (Mar. 12, 1984), 4.
17. COMPUTERWORLD. Storage Technology Corp., Sybercache statistical product. *Computerworld* (July 9, 1984), 80.
18. COTE, H. J. AND DUHL, B. New horizons for cached disk and buffered tape. In *Proceedings of the CMG 13th International Conference* (Dec. San Diego, Calif.). 1982, pp. 333–337.
19. DAHMAN, K. AND GROSSMAN, G. Effective use of cached DASD in a data base/data communications environment. In *Proceedings of the 1983 CMG International Conference* (Washington, D.C., Dec.). 1983, pp. 425–431.
20. DATE, C. J. *An Introduction to Database Systems.* 2nd ed., Addison-Wesley, Reading, Mass., 1977.
21. DENNING, P. J. On modelling program behavior. In *Proceedings of the Spring Joint Computer Conference*, vol. AFIPS Press, Reston, Va., 1972, pp. 937–944.
22. DIXON, J. D., MARAZAS G. A., AND MCNEILL, A. B. Mini-ops—A microcoded data transfer scheduling and execution systems for the optimized control of an I/O controller cache memory. *IBM Tech. Disc. Bull. 27*, 2 (July, 1984), 1226–1227.
23. DODSON, G. W. Cached DASD evaluations for paging and non-paging data. In *Proceedings of the CMG 13th International Conference* (Dec. San Diego, Calif.). 1982, p. 338.
24. DUKE, A. H., HARTUNG, M. H., HUNTLEY, J. D., AND MARSCHNER, F. J. Buffered writing in a peripheral storage hierarchy. *IBM Tech. Disc. Bull. 25*, 4 (Sept. 1982), 2075–2076.
25. DUKE, A. H. AND HARTUNG, M. H. Controlling multitrack references in a cached storage system. *IBM Tech. Disc. Bull. 25*, 7B (Dec. 1982), 3756–3757.
26. DUKE, A. H., HARTUNG, M. H., HUNTLEY, J. D., AND NOLAN K. P. Inhibiting cache loading. *IBM Tech. Disc. Bull. 25*, 12 (May 1983), 6351–6353.
27. ELECTRONICS 1. Cache memories catch on for disks. Electronics (Apr. 21, 1981), 62–63.
28. ELECTRONIC NEWS. IBM expands CPU for system 38; adds series 1 processor, disk drive. *Electron. News* (Apr. 11, 1983).
29. ELECTRONIC NEWS. New HP 16-bit CPUs Based on OS Upgrades. *Electron. News* (May 30, 1983), 18.
30. ELECTRONIC NEWS. IBM Increases Memory, Cuts Tag on 3880 Controller. *Electron. News* (Sept. 24, 1984), 27, 39.
31. ELECTRONIC NEWS. Storage Tek Offers Controller Upgrade. *Electron. News* (Aug. 13, 1984), 33.
32. FAJMAN, R. AND BORGELT, J. Wylbur: An interactive text editing and remote job entry system. *Commun. ACM 16*, 5 (May, 1973), 314–322.
33. FRIEDMAN, M. "DASD access patterns" In *Proceedings of the 1983 CMG International Conference*, (Dec. 1–4 Washington, D. C.). 1983, pp. 51–61.

34. GALE, L. Work station performs at the superminicomputer level. *Electronics* (Sept. 8, 1983), 119–123.
35. HARKER, J. M., BREDE, D. W., PATTISON, R. E., SANTANA, G. R., AND TAFT, L. G. A quarter century of disk file innovation. *IBM J. Res. Devel. 25*, 5 (Sept. 1981), 677–689.
36. HOAGLAND, A. S. Storage technology: Capabilities and limitations. *Computer 12*, 5 (May 1979) 12–18.
37. HODGES, D. A. A review and projection of semiconductor components for digital storage. *Proc. IEEE 63*, 8 (Aug. 1975), 1136–1147.
38. HUGELSHOFER W. AND SHULTZ, B. Cache buffer for disk accelerates minicomputer performance. *Electronics* (Feb. 10, 1982), 155–159.
39. HUNTER, D. Modeling real DASD configurations. IBM Res. Rep. RC 8606, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1980.
40. HUNTER, D. W. DASD arm buffers. *IBM Tech. Disc. Bull. 24*, 4 (Sept. 1981), p. 2035.
41. INFOWORLD. Cache/Q Disk buffering enhancement for CP/M. *Infoworld 5*, 7 (Jan. 14, 1983), 50–54.
42. INTEL. FAST-3805 Functional Description. PN 19-1619-006 (Aug. 1979), Intel Commercial Systems Division, Phoenix, Ariz.
43. IBC/INTEGRATED BUSINESS COMPUTERS. CADET/10 cache disk memory reference manual. Integrated Business Computers, Chatsworth, Calif., 1982.
44. IBM. Reference manual for the IBM 2835 storage control and the IBM 2305 fixed head storage module. GA26-1589, IBM Corporation, 1972, San Jose, Calif.
45. IBM. Reference manual for IBM 3830 storage control and IBM 3330 disk storage. GA26-1592, IBM Corporation, Armonk, N.Y.
46. IBM. OS/VS2 system programming library: Service aids. GC28-0674-1, IBM Corporation, Gaithersburg, Md., 1976.
47. IBM. Reference manual for IBM 3350 direct access storage. GA26-1638-2, IBM Corporation, San Jose, Calif., 1977.
48. IBM. OS/VS MVS systems programming library: System management facilities (SMF). GC28-0706-1, IBM Corporation, Poughkeepsie, N. Y., 1977.
49. IBM. IBM 3310 direct access storage reference manual. GA26-1660, IBM Corporation, San Jose, Calif., 1979.
50. IBM. IBM 3370 direct access storage description. Pub GAZ6-1657-2 IBM Corporation, General Products Division, San Jose, Calif., 1979.
51. IBM. Introduction to IBM 3880 storage control, model 11. GA32-0060, IBM Corporation. Tucson, Ariz., 1981.
52. IBM. IBM 3880 storage control model 11 description. GA32-0061, IBM Corporation, Tucson, Ariz., 1982.
53. IBM. Introduction to IBM 3880 storage control model 13. GA32-0062, IBM Corporation, Tucson, Ariz., 1983.
54. IBM. IBM 3880 storage control model 13 description. GA32-0067, IBM Corporation, Tucson, Ariz., 1983.
55. IBM. Cache RMF reporter. G320-0362, IBM Corporation, Irving, Tex., 1984.
56. KNUTH, D. E. *The Art of Computer Programming.* Vol. 3, *Sorting and Searching* Addison-Welsey, Reading, Mass., 1973.
57. LOWMAN, R. IBM 3880 model 13 storage subsystem. Rep., IBM Corporation, General Products Division, Tucson, Ariz., 1983.
58. LYNCH, W. C. Do disk arms move? *Perform. Eval. Rev. 1* (Dec. 1972), 3–16.
59. MANKEKAR, P. S. AND MILLIGAN, C. A. Performance prediction and validation of interacting multiple subsystems in skew-loaded cached DASD *Proceedings 1983 CMG International Conference* (Washington, D. C., Dec.). 1983, pp. 383–387.
60. MATTSON, R. L., GECSEI, J., SLUTZ D. R., AND TRAIGER, I. L. Evaluation techniques for storage hierarchies. *IBM Syst. J. 9*, 2 (1970), 78–117.
61. MEMOREX. 3770 Disc cache product description manual. Memorex Corporation, Santa Clara, Calif., 1978.
62. ORGANICK, E. *The Multics System: An Examination of its Structure*, MIT Press, Cambridge, Mass., 1972.

63. RITCHIE D. AND THOMPSON, K. The UNIX time sharing system. *Commun. CACM 17,* 7 (July 1974), 365–375.
64. SHERMAN, S., BASKETT F. AND BROWNE, J. C. Trace driven modeling and analysis of CPU scheduling in a multiprogramming system. *Commun. CACM 15,* 12 (Dec. 1972), 1063–1069.
65. SMITH A. J. A locality model for disk reference patterns. In *Proceedings of the IEEE Computer Society Conference* (Feb., San Francisco, Calif.) 1975 IEEE, New York, pp. 109–112.
66. SMITH, A. J. Analysis of a locality model for disk reference patterns. In *Proceedings of the 2nd Conference on Information Sciences and Systems* (Baltimore, MD., Apr.). 1976, pp. 593–601.
67. SMITH, A. J. Bibliography on paging and related topics. *Oper. Syst. Rev. 12,* 4 (Oct. 1978), 39–56.
68. SMITH, A. J. On the effectiveness of buffered and multiple arm disks. In *Proceedings of the 5th Computer Architecture Symposium* (Palo Alto, Calif., Apr.) 242–248.
69. SMITH, A. J. Sequentiality and prefetching in data base systems. *ACM Trans. Database Syst. 3,* 3 (Sept. 1978), 223–247.
70. SMITH, A. J. Sequential program prefetching in memory hierarchies. *IEEE Computer 11,* 12 (Dec. 1978), 7–21.
71. Smith, A. J. Input/Output optimization and disk architecture: A survey. *Perform. Eval. 1,* 2 (1981) 104–117.
72. SMITH, A. J. Bibliography on file system and Input/Output optimization and related topics. *Oper. Syst. Rev. 15,* 4 (Oct 1981) 39–54.
73. SMITH, A. J. Optimization of I/O systems by cache disk and file migration, A summary. *Perform. Eval. 1,* 3 (1981), 249–262.
74. SMITH, A. J. Cache Memories. *ACM Comput. Surv. 14,* 3 (Sept., 1982), 473–530.
75. Cache disk system. Sperry Univac Product Announcement, for 5057 Cache Disk Processor and 7053 Storage Unit, Sperry Univac, 1981.
76. Storage Technology Corporation. Sybercache 8890 intelligent disk controller. *Storage Technology Corporation,* Louisville, Colo. 1982.
77. TOKUNAGA, T., HIRAI, Y., AND YAMAMOTO S. Integrated disk cache system with file adaptive control., In *Proceedings of the IEEE Computer Society Conference,* (Washington, D. C., Sept.) IEEE, New York, 1980, pp. 412–416.
78. WELCH, T. Effects of sequential data access on memory hierarchy design. In *Proceedings of the IEEE Computer Society Conference,* (San Francisco, February). IEEE, New York, 1979 pp. 65–68.
79. WELCH, T. A. Analysis of memory hierarchies for sequential data access. *Computer 12,* 5 (May 1979), 19–26.

Received May 1983; revised August 1984; accepted December 1984.

REMARK ON "DISK CACHE—MISS RATIO ANALYSIS AND DESIGN CONSIDERATIONS"

Disk Cache—Miss Ratio Analysis and Design Considerations [Alan Jay Smith, *ACM Trans. Comput. Syst. 3*, 3 (Aug. 1985), 161–203]

My paper, "Disk Cache—Miss Ratio Analysis and Design Considerations," which appeared in the August 1985 issue of TOCS, provoked some discussion of early disk cache development. Two early papers not cited in my paper are particularly noteworthy.

The first report of a disk cache appeared in [2]. This paper explains the concept of a disk cache and evaluates its utility using both modeling and trace-driven simulation from an IBM OS/360 system with IBM 2311 disks. Cache effectiveness was further demonstrated with an experimental implementation on a 360/50 and a 360/65 using Large Core Storage (LCS).

Another early disk cache implementation is reported in [1]. In that system the Triangle University's Computation Center used LCS for several functions, in particular an automatically managed disk-cache to which substantial performance improvement was attributed.

ALAN JAY SMITH
University of California
Berkeley, CA 94720

REFERENCES

1. FREEMAN, D., AND RAGLAND, J. The response-efficiency trade-off in a multiple-university system. *Datamation* (Mar. 1970), 112–116.
2. STEVENSON, D. A., AND VERMILLION, W. H. Core storage as a slave memory for disk storage devices. In *Proceedings of IFIP '68*. pp. F86–F91.

Received August 1986; revised September 1986