A Study of I/O System Organizations

A. L. Narasimha Reddy
IBM Almaden Research Center
650 Harry Road, K56/802
San Jose, CA 95120
reddy@almaden.ibm.com.

Abstract

With the increasing processing speeds, it has become important to design powerful and efficient I/O systems. In this paper, we look at several design options in designing an I/O system and study their impact on the performance. Specifically, we use trace driven simulations to study a disk system with a nonvolatile cache. Some of the considered design parameters include the cache block size, the fetch size, the cache size and the disk access policy. We show that decoupling the fetch size and the cache block size results in significant performance improvements. A new write-back policy is presented that is shown to offer significant performance benefits. We show that optimal block size in a two-level memory hierarchy is dependent only on the latency, data rate product of the second level as previously conjectured. We also present results showing the effect of a split access operation of a disk read/write head.

1 Introduction

Processor speeds have been improving at a rapid pace over the last few years. It has been realized that if corresponding improvements in the I/O performance are not achieved, the system's performance may not improve at the same rate as of the processor speed improvements. Amdahl's rule of thumb for a balanced computer requires that a system should have 1 Mbyte of main memory capacity and 1 Mbit/sec of I/O bandwidth per 1 MIPS of CPU performance [1]. Recent measurements [2] show that the I/O requirements are nearly an order of magnitude higher, 1 Mbyte/sec of I/O bandwidth per 1 MIPS, for the current machines.

Several proposals have been made recently for employing multiple disk organizations [3, 4, 5, 6, 7]. Most of these studies have been aimed at improving the data throughput of the system. They also study the disk system independent of the other components of the system such as a disk cache or an I/O cache.

Disk cache or I/O cache is normally used to improve the performance of the disk system [8, 9]. We will use the term I/O cache or cache in this paper to denote a cache used within the I/O system, which may be physically located within the main memory (file cache) or within the disk subsystem (disk cache).

Studies so far have looked at the disk system or the I/O cache independently. In this paper, we study the interaction between the I/O cache and the disk system with the aim of understanding how the various policy decisions and system parameters affect the performance of the whole I/O system. There are several parameters and policies that influence the performance of the system. We study the impact of various parameters such as the cache size, block size and fetch size. The policies adopted at the disk system may impact the decisions at the cache. We consider the interaction of various disk policies and the cache design parameters to understand the total effect seen by the system.

The rest of the paper is organized as follows. Section 2 looks at the system model, defines various parameters used in the study and the various disk/cache management policies considered in the study. Section 3 describes our evaluation methodology. Section 4 describes the results of various experiments conducted. Section 5 concludes the paper by outlining the major results presented in the paper.

2 System Model

An I/O system in our model consists of a number of disks together with a cache as shown in Fig.1. Whenever data is written to the I/O system, it is written to the cache. The data is eventually written to the disk at a later time. A write is considered completed when the data is written to the cache. When a read request arrives at the I/O system, it is first checked to see if the requested block is in the cache. If it is, the requested block is returned to the system. If the requested block is not in the cache, the data is read from the disk. Whenever data is read from the disk, it is copied into the cache and returned to the system. A read request is considered complete when the requested block is returned to the system.

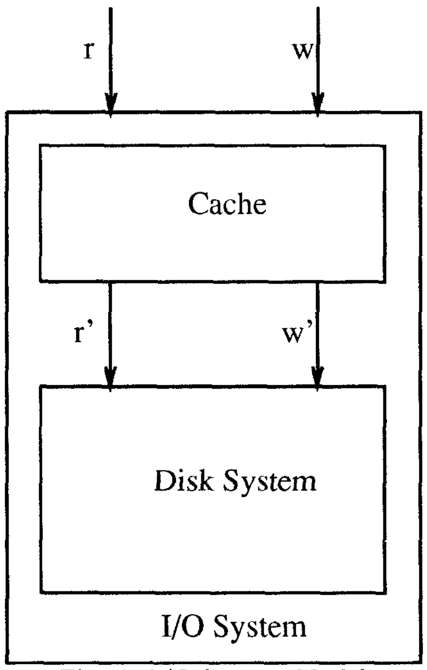

Fig. 1. I/O System Model.

There are four parameters in our model: cache size, cache block size, fetch size, and destage size. Cache size is the amount of cache space in the I/O system. Cache block size is the unit of space allocation in the cache. Fetch size is the amount of data fetched from the disk to the cache on a read miss in the cache. In our model, whenever data is written to disk, all blocks belonging to a *destage unit* are written together. We varied the destage size from fetch size to a track size of 32 kbytes. It was found that destage size of a track always resulted in better performance (lower writes at disk, better read response time and better disk utilization). Hence, we only consider destage sizes of a track in this paper, unless specifically mentioned otherwise. Request size is the number of sectors requested from the I/O system, sector being the smallest unit of data uniquely addressable on the disks. Request sizes are determined from the traces. We considered cache sizes of 1Mbytes to 16Mbytes for our simulations. Cache block sizes from 512 bytes to 32 Kbytes were considered. Fetch size was varied from 512 bytes to 32 Kbytes. The relation between the block size and the fetch size is as follows. The data is always transferred in units determined by the cache block size. For example, if a request asks for 7 sectors starting at sector numbered 3, when block size is 4 and the fetch size is 16, block numbers 1 (sectors 1,2,3,4), 2, 3, and 4, i.e., sectors 1,2,...16, will be brought into the cache. For the same request, if the block size is 2 and the fetch size is 16, block numbers 2 (sectors 3,4), 3, 4,5,6,7,8 and 9, i.e., sectors 3,4,...18, will be brought into the cache resulting in transfers of different sectors.

The following procedure is employed in determining how many sectors of data are to be brought into the cache on a request. If the request size is larger than the fetch size, the number of sectors brought in from the disk equals the request size, otherwise, it equals the fetch size. In other words, fetch size is the minimum number of sectors brought from the disk to the cache on a read miss.

On a miss, cache blocks may have to be written back to the disk. The LRU chain is scanned to see which blocks are at the head of the LRU chain. Since every miss results in fetching max(request size, fetch size) number of blocks, as many blocks have to be evicted from the cache on every miss. If these blocks at the head of the LRU chain were not dirty (i.e., don't have to be written back to the disk), servicing the miss would only involve reading the blocks from the disk to the cache. To facilitate this, the blocks at the head of the LRU chain can be continually written to the disk in the background. If this cleaning activity is carried out when the disk is idle, the cost of cleaning can be hidden from the response time. This is modeled in the following way in our simulations: the number of dirty blocks at the head of LRU chain to be cleaned for servicing the read miss is counted. The read request is issued to the disk system. After the read request is satisfied by a transfer of the requested blocks from the disk to the cache, we simulate the write back of the dirty blocks by issuing a write request to the disk system for an appropriate number of blocks. Even though, this modeling is not exact, it suffices for our evaluation purposes by simulating the write-back activity.

When the block size doesn't equal the fetch size, which blocks should be written back to the disk on a read miss? Should we strictly write the blocks at the head of the LRU chain? Should we write back the blocks that were brought together into the cache (based on the fetch size)? If we follow the first method, a write-back operation may actually involve writing a number of blocks that belong to different tracks and thus incurring a large cost. If we follow the second method, we have not decoupled the cache block size and the fetch size and we might as well have organized the cache on bigger blocks of fetch size. To alleviate these two problems, we adopt a method based on the above two ideas. Blocks at the head of LRU chain are scanned to iden-

Table 1. Disk parameters.

Avg. latency	8.3 ms
Avg. seek	14.2 ms
sectors/track	64
sector size	512 bytes
tracks/cylinder	19
cylinders/disk	746
seek cost function	nonlinear
latency cost function	uniform

tify the blocks that need to be written back. Once these blocks are identified, all the blocks that belong to the same track (destage unit) on the disk that are currently present in the cache are also written back to the disk. The blocks at the head of the LRU chain are considered evicted from the cache and the other blocks that may be written to the disk as a result of such an operation are left in the cache in a clean (i.e., not dirty) state. By adopting such a strategy, it is hoped that dirty blocks belonging to the same track are written to the disk in one single operation while still following an LRU replacement policy. It is possible, using such a write-back policy, a block at the tail of the LRU chain may repeatedly get written back to the disk as a result of evicting other dirty blocks at the head of the LRU chain that belong to the same track. The effectiveness of such a policy needs to be verified through simulations.

Disk service involves several components. A *seek* is performed to seek to the track on which the requested block of data resides. Rotational *latency* is paid to reach the first block of the request on the track. *Read time* or *transfer time* is the time spent in actually reading the data from the disk surface. The disk parameters used for simulations can be found in Table 1.

Disks are assumed to employ a SCAN algorithm for satisfying the disk requests. In this service policy, the disk arm moves outward from the center of the disk servicing requests in its path till it reaches the outermost track. Once it reaches the outermost track, it jumps to the innermost track with an outstanding request and starts the outward journey again. Reads and writes are treated in the same fashion once queued at the disk i.e., reads are not given higher priority over writes at the disk. However, writes to the disk are queued only during the periods when the disk is idle. If a read request is issued after a write request is queued, the read request may have to wait for the write request to complete. The employed SCAN policy also did rotational optimization: if two requests belonging to the same track are waiting to be serviced, they are served in the order that they may be read off from the disk surface, rather than in their arrival order. To further reduce rotational latency, we considered *split access* operations. If a request asks for n number of blocks, in normal operation, on an average half a rotation penalty is paid to get to the first block in the request and then the requested blocks are read from the disk. This operation may result in paying more than one revolution for reading/writing a single track. In split access operation, the disk starts servicing the request as soon as any of the requested blocks comes under the read-write head. For example, if a request asks for reading blocks numbered 1,2,3,4 from a track of eight blocks 1,2,...8, and the read-write head happens to get to block number 3 first, then blocks 3 and 4 are read, blocks 5,6,7,8 are skipped over and then blocks 1 and 2 are read. In such operation, a disk read/write of a single track will not take more than one single revolution.

In normal operation of serving the request from the first block, latency plus read time , on an average, equal

$$l_{max}/2 + n * R \qquad (1)$$

for reading n blocks from a single track, where l_{max} is the time for one revolution and R is the data rate in secs/block. The seek time is not affected in employing the split access operation and hence we only look at the difference in the read time plus latency. In split access operation, the latency plus read time is given by the following:

$$f * l_{max} + (1-f) * ((1-f)\frac{l_{max}}{2} + n * R), \qquad (2)$$

where f is the fraction of blocks on a track that are being read = n/N, N is the total number of blocks on a single track. The first term in the above expression counts the probability of finding the read/write head in the middle of the requested sequence (and hence paying a full revolution to serve the request) and the second term counts the probability of finding the read/write head positioned in the rest of the track. The above simplifies to

$$l_{max}/2 + l_{max} * f(1 - f/2). \qquad (3)$$

This function grows less rapidly than the earlier function. For example, when f = 0.25, the above cost = $0.719 l_{max}$, when f = 0.5, it is $0.875 l_{max}$, and when f = 1, it is l_{max}. Hence, the split access operation of serving disk requests flattens the latency+read time cost function. This indicates that, split access operation may favor larger fetch sizes than normal mode operation. The effective average latency of a disk operation in split access operation $= l_{max}/2 + l_{max} * f(1 - f/2) - n * R$ $= \frac{l_{max}}{2}(1 - f^2)$, which again indicates that the transfers are more efficient as f approaches 1. The benefits of larger accesses in split access operation have to be weighed against the problem of cache pollution. This balance needs to be studied through simulations.

Table 2. Characteristics of the traces.

Trace	Reads		Writes		Trace length	System Capacity	Read ratio
	# of req.	kbytes	# of req.	kbytes	seconds	Mbytes (# of disks)	
1	37037	114462	30079	67752.5	3751.45	1672 (4)	0.55
2	46215	120931.5	23532	21474	6053.96	2926 (7)	0.66
3	49545	140215.5	54735	38211	2003.89	6688 (16)	0.48
4	54171	94569.5	48558	82833	5422.18	3344 (8)	0.53

3 Evaluation Methodology

Physical I/O traces were obtained from IBM AS/400 systems running real applications. Four traces were employed in this study. The characteristics of these traces can be found in Table 2. All the traces used in this study are from commercial environment with predominantly transaction oriented workloads. The first trace, Trace 1, has both interactive and batch requests. Trace 2 contains mostly batch requests. Trace 3 contains only interactive requests. Trace 4 is obtained from a server application, where the machine is used as a server for a number of terminals. Each trace entry contained the following information: the time the request was initiated, the disk address, the starting sector address on the disk, the number of sectors requested and whether the request is a read or a write operation. The AS/400 architecture is segment-based and distributes blocks of a file across the disks in the system. The file is initially allocated on a single disk, but further allocations of blocks to the same file can get distributed across all the disks. In this study, we do not consider the effects of file/block allocation on disks since the trace information we have is only from physical I/Os.

When a nonvolatile cache is used, the writes can be declared completed once the data is written to the cache. If sufficient clean (or empty) space is left in the cache for writes to take place without waiting for some dirty pages to be evicted to the cache, the write performance seen by the user is limited by how fast the data can be written to the cache. The dirty blocks in the cache can be written to the disk by a background process. This process may keep a certain number of clean blocks in the cache to ensure that the writes can be completed by writing to the cache. However, when a read request is issued to the I/O system, the read request is completed only when the requested block is returned to the user. Hence, the perceived performance of the I/O system is determined by how fast the read requests can be serviced (assuming that the load on the system is not so high as to be unable to keep writing the dirty blocks of data from the cache to the disk). Hence, in this paper, read response time is used as a measure of performance of the system. This is in contrast to the approach taken in [10, 11] where write bandwidth of the system is the primary concern.

Since disk accesses take considerably longer time compared to accesses from the cache, we consider the performance to be mainly dictated by the misses in the cache. Since write misses can be satisfied by writing the data blocks to the cache, read misses are considered to be the main determinants of performance. In the model we employ here, the read/write hits and write misses to cache are assumed to be satisfied in zero time. The response time for servicing a read miss is considered as the time for satisfying that request.

At each cache size, different block sizes and fetch sizes were considered for the I/O system organization. For each organization, the effective access time was computed. Effective access time of each organization is the sum of response times of all read misses divided by the total number of requests in the trace. In our model, read response time is the measure of performance. If we used only the read miss ratio as the measure, we would not account for the different costs associated with fetching different number of blocks from the disk. For each trace, the effective access time of each organization is computed. The access times of all the organizations are scaled with respect to the acess time of an organization with a block size of 4 kbytes and a fetch size of 4 kbytes to obtain their *relative access times*. Hence, by definition, for all traces, the relative access time of a system with a block size of 4 kbytes and a fetch size of 4 kbytes is 1.0. The relative access times of each organization are computed for all the four traces. Then, a geometric mean of these relative access times is computed for each organization. We call this the *normalized access time*. Normalized access time is used as the final measure of performance in our study. System organizations are compared on the basis of normalized access time. Lower the normalized access time, better the performance. This approach of evaluation is similar to the approach adopted by the SPEC benchmark group [12].

4 Results

In this section, we present the results obtained from the simulations.

4.1 Fetch size and Block size

Fig. 2. shows the performance of different organizations for a cache size of 1 Mbytes. The different organizations are compared on the basis of normalized access time, as explained earlier. It is observed that an organization with a fetch size of 4 kbytes and a block size of 512 bytes achieves the best performance. However, there are several other organizations with almost equal performance. It is also noted that at the optimal fetch size of 4 kbytes, the differences in performance due to varying the block size are about 10%. Among these organizations, a choice could be made based on some other criterion. For example, in this study, we did not include the cost of managing larger LRU chains with smaller block sizes. If these costs are a concern, among the organizations with almost equal performance in Fig.2., an organization with a slightly larger block size may be preferred. The relatively flat region around the optimal block size indicates that choosing a slightly different fetch size would not hurt performance drastically. It is noted that decoupling block size from fetch size produced considerable performance improvements (roughly 25% improvement from a block size = fetch size = 16k to 16k fetch size and 2 kbyte block size). Even when the cost of handling larger LRU chains is included, this benefit is likely to be significant since the cost of a disk access is much more expensive.

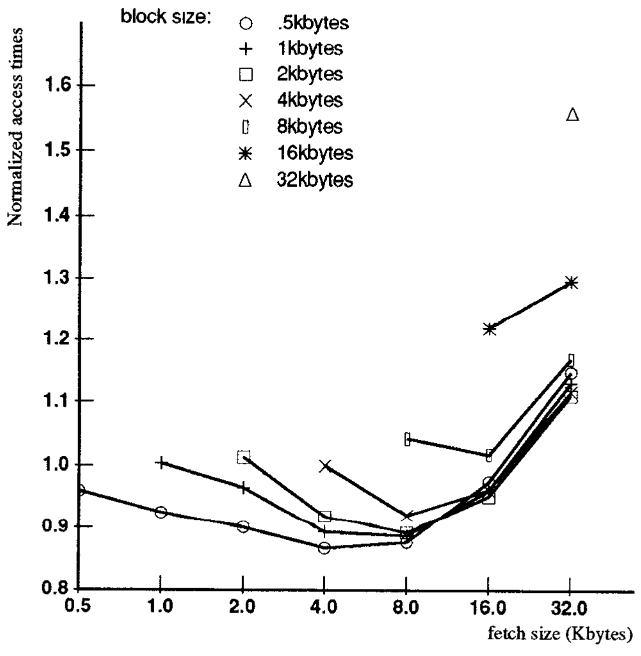

Fig. 2. Performance of various organizations at cache size = 1M.

4.2 Cache size

Fig. 3. shows the effect of cache size on the performance of various organizations when block size equals fetch size. To show the impact of cache size on the average access times, actual values are plotted in Fig. 3. rather than the normalized access times.

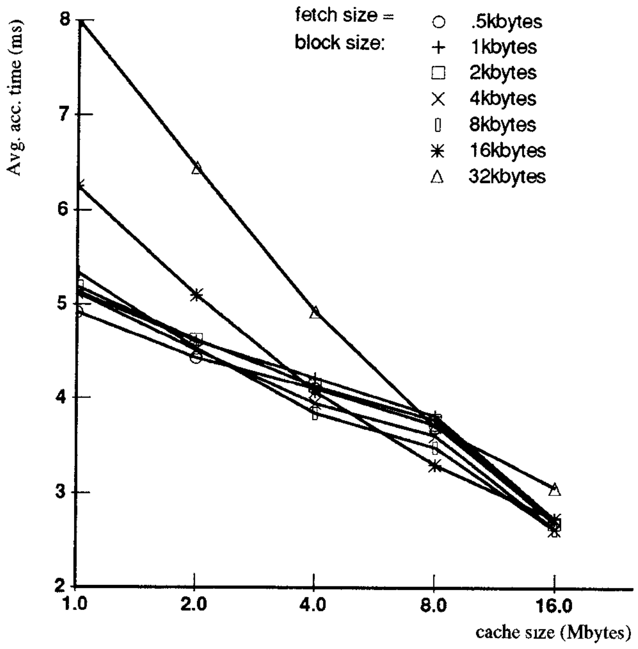

Fig. 3. Performance of various organizations at different cache sizes.

It is observed that the difference in performance between different organizations is more significant at smaller cache sizes than at larger cache sizes. It is also noted that organizations with fetch sizes larger than 8 kbytes have nearly equal performance at higher cache sizes even though they differ considerably at smaller cache sizes. Larger fetch size tries to exploit spatial locality. However, as a result of fetching more blocks into the cache, the cache may get polluted. At smaller cache sizes, this pollution is a bigger problem. But at larger cache sizes, this is less of a problem and hence the observed trend. This tends to indicate that if sufficiently large caches are employed, the implications of choice of other parameters are less significant. This fact is also observable in Fig. 4, which shows the performance of different organizations at a cache size of 16 Mbytes. It is observed that the normalized access times now range from 0.9 to 1.2 compared to a range of variation of 0.9 to 1.6 at a cache size of 1 Mbytes. It is also noted that the optimal fetch size has increased with an increase in cache size.

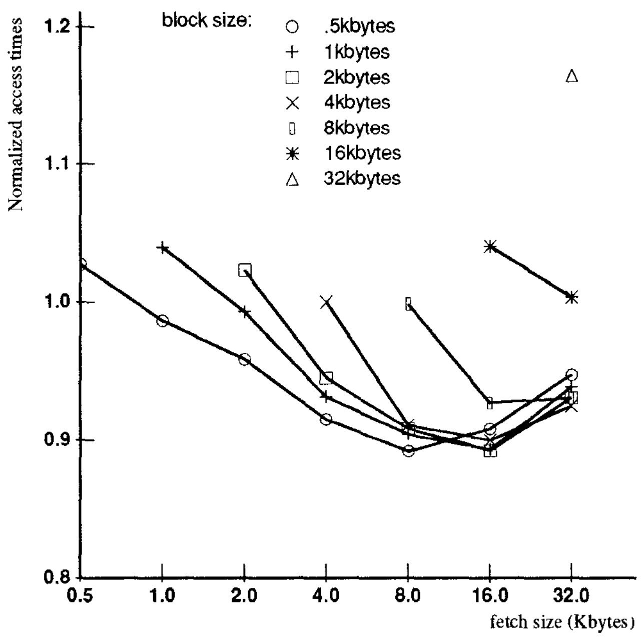

Fig. 4. Performance of various organizations at different cache sizes.

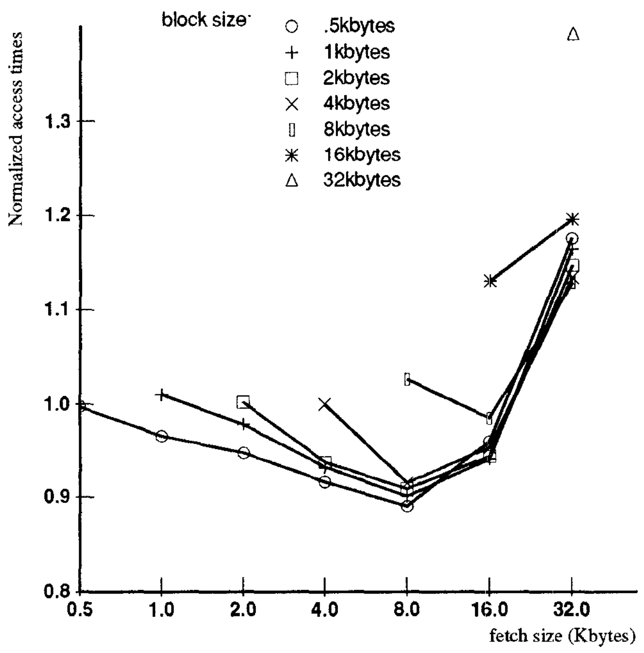

Fig. 5. Normal-mode performance at cache size = 16M.

4.3 Normal-mode Disk Operation

Fig. 5. shows the performance of various organizations at a cache size of 16 Mbytes, employing a normal-mode disk operation. The best fetch size for normal-mode operation is observed to be 8 Kbytes. In normal-mode operation, the latency cost of each disk access is higher than that in a split access operation. To offset this higher cost of latency, we fetch more data in each disk access in normal-mode operation. We observe that the performance in normal mode degrades quickly at larger fetch sizes compared to a more gradual degradation of performance in split access operation. This is as expected from the analysis earlier owing to a relatively flat read time plus latency cost of split access operations. At the optimal fetch size, the split access operation resulted in 5-10% (depending on cache size) performance improvement over normal mode operation of disks. In a more highly loaded environment, these performance gains are likely to be higher. To assess these possible improvements, we conducted two other experiments where the time between two requests was reduced by a factor of 2 and 4. Split access operation in these two cases resulted in 20% and 45% improvement in performance at the best fetch sizes.

4.4 Disk Reads and Writes

Fig.6. shows the fraction of total disk operations that are read operations as a function of block size and fetch size for Trace 1 at a cache size of 16 Mbytes. For this experiment, the destage size was made equal to the fetch size. As the fetch size is increased, reads constitute larger fraction of the total disk operations. At larger fetch sizes, smaller block sizes result in larger read ratios. With larger block sizes, there is a higher probability of a block being dirty and hence a miss results more often in an eviction of a dirty block and hence the resulting higher write ratios. The reason that the write ratio decreases with increasing fetch size is that the writes have more spatial locality than reads, at least for the traces under study. This can also be observed by the miss ratio curves shown in Fig. 7, where it is seen that writes have smaller miss ratios than reads. The read/write ratios in the I/O workload have significant impact on the performance in systems where the reads and writes have different costs such as disk arrays with parity protection. The larger the readratio, the better the performance of disk arrays since the writes are more expensive [?].

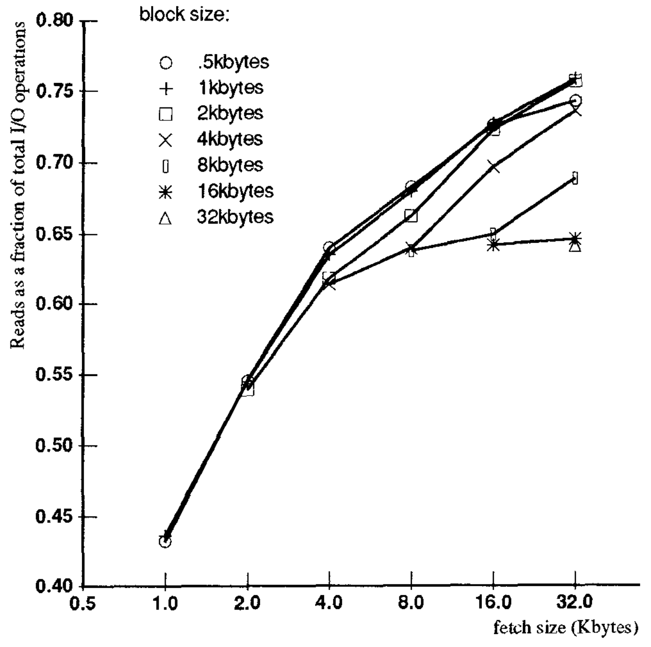

Fig. 6. Read ratio as a function of block size and fetch size -Trace 1.

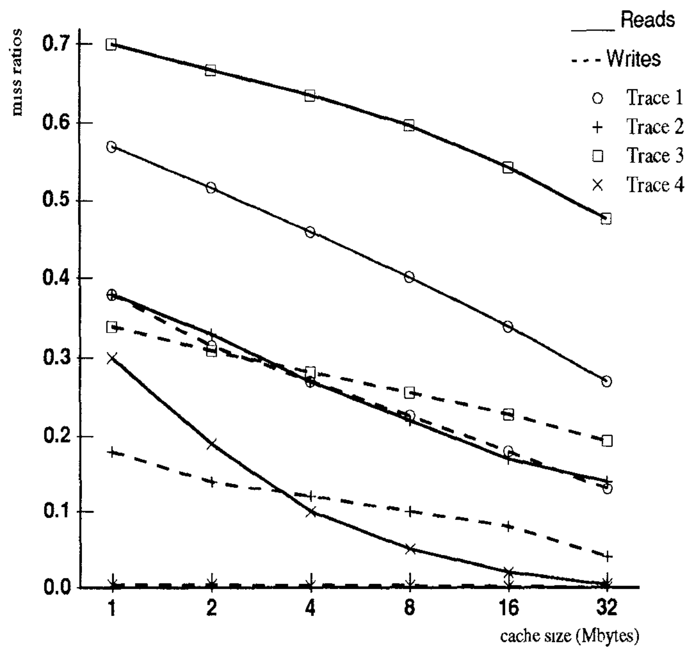

Fig. 7. Miss ratio as a function of block size.

From Fig. 6, it is seen that by choosing the block size and fetch size independent of each other, we can obtain desired workload characteristics. Besides the observed performance benefits in non-arrayed systems as reported in this paper, the ability to change the workload characteristics seen by the disk system is an important benefit of decoupling block size and fetch size. It is also observed from Fig. 6 and Table 2 that for some fetch sizes, the workload seen by the disk system after the cache has fewer fraction of write requests than that of the workload before the cache. All this discussion points to the fact that the write penalty of the array systems can be managed by using nonvolatile caches and appropriate cache management techniques.

4.5 Optimal Block Size

Previously, it was conjectured that the optimal block size for minimizing the access time in a two-level cache depended only on the product of the latency and the data rate of the second level cache and not on the individual parameters separately [14]. Similarly, optimal striping unit in a disk array system is shown to be directly related to the product of data rate and the positioning time of a disk [15]. We show that this is indeed the case in any two-level memory system under some assumptions. In fact, from the dimensions of the parameters alone, it could be argued that this has to be the case. Here, we present another simple argument to show why this is true. For the analysis here, we assume that the block size and the fetch size are equal. The cost of serving a miss is given by $c_b = c_0 + b * c_1$, where c_0 is the latency of the data access from the second level memory and c_1 is the time to access a unit of data. The optimal block size is one that minimizes the total cost of serving the misses, given by,

$$T_b = m_b * c_b, \tag{4}$$

where m_b is the total number of misses at block size b or alternately the miss ratio at that block size if we consider T_b to be normalized cost of serving misses. When the block size is increased, the miss ratio decreases till a point beyond which the miss ratio starts to increase again. There is no reason to choose the cache block size larger than this minimum point since both the miss ratio and the cost of serving a miss increase beyond this point. Assume that the cache block size at which this minimum in miss rate occurs to be b_{min}. The miss ratio in the range of (1, b_{min}) can be approximated by:

$$ln(m_b) = ln(m_{min}) + k * ln(b_{min}/b). \tag{5}$$

The above states that log of miss ratio is a linear function of log of the block size. This behavior is observed in several studies, for example [14, 16]. If we substitute the values of m_b and c_b from above and differentiate with respect to b to obtain the optimal block size, we find that the optimal block size is given by

$$b_{opt} = \frac{kc_0}{(1-k)c_1}. \tag{6}$$

This shows that the previous conjecture about optimal block size is indeed true. Typically, the miss ratio drops 20%-30% for every doubling of cache size in the range of (1, b_{min}) [14]. If this is the case, we can choose k to be approximately 0.33 in the above calculations to obtain $b_{opt} = c_0 / 2c_1$. Using the above equations, it can also be shown that the differences in performance around the optimal block size would be small.

In I/O systems, cache and the disks function as a two-level memory hierarchy. If the assumptions in the above analysis are valid in such a hierarchy, we could apply the analytical results to such a system as well. Even though disk access can fit the above cost model of accessing a block of size b, the latency parameter (or positioning time = rotational latency + seek time for disks) is not a constant, but is a random variable. We could approximate this by choosing average positioning time for this parameter. Another factor that is ignored in the above analysis is the queuing delays that may be incurred in the disk systems. If queuing delays are large, this model can not be applied to the disk systems. The constant factor relating the optimal block size to the latency and data rate parameters may depend on the workload characteristics. How the workload characteristics may impact these constants remains to be established.

Fig. 8. compares the performance of various organizations at a cache size of 16 Mbytes, using normal mode disk operation, at different data rates. For this set of experiments, the block size = fetch size = destage size. It is observed that the optimal block size doubled when the data rate of the disks is improved by a factor of two. This conforms to the above analysis. It is also noted that the variations in performance around the optimal configuration point are not very significant.

4.6 Write-back policy

Fig. 9. shows the effect of the write-back policy on the number of write operations for Trace 1 at the fetch size of 32 kbytes. The figure shows the number of write operations of the proposed write-back policy as a fraction of the number of write operations in a strict LRU policy. In a strict LRU policy, only the blocks at the head of the LRU chain are written to the disk. If n number of blocks are to be written to disk from the head of the LRU chain, these n number of blocks are grouped into as few write operations as possible based on their track location. It is seen that the proposed write-back policy considerably reduces the number of write operations seen at the disk. The write operations are reduced by more than half at a block size of 512 bytes. The reduction of the number of write operations is significant because of the earlier discussed write penalty in disk arrays. The policy is seen to be more effective at smaller block sizes. At smaller block sizes, it is more likely that blocks belonging to the same track are not bunched together in the LRU chain and hence a higher likelihood of the proposed write-back policy being effective at reducing the number of write operations.

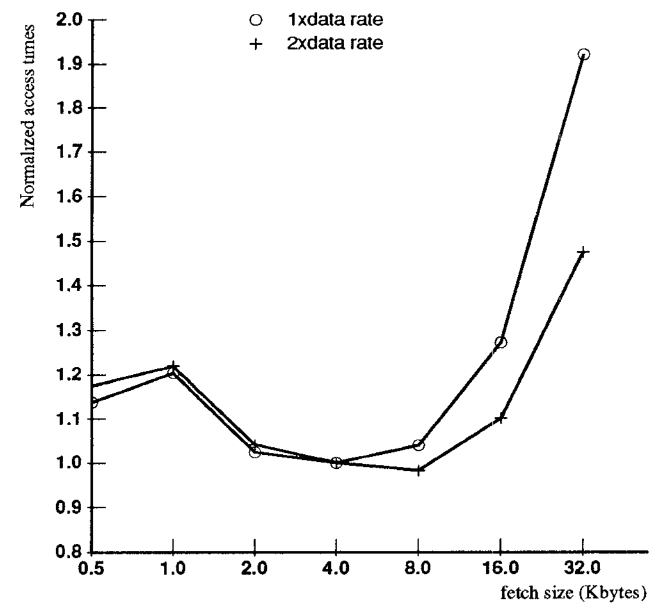

Fig. 8. Performance of various organizations at different data rates.

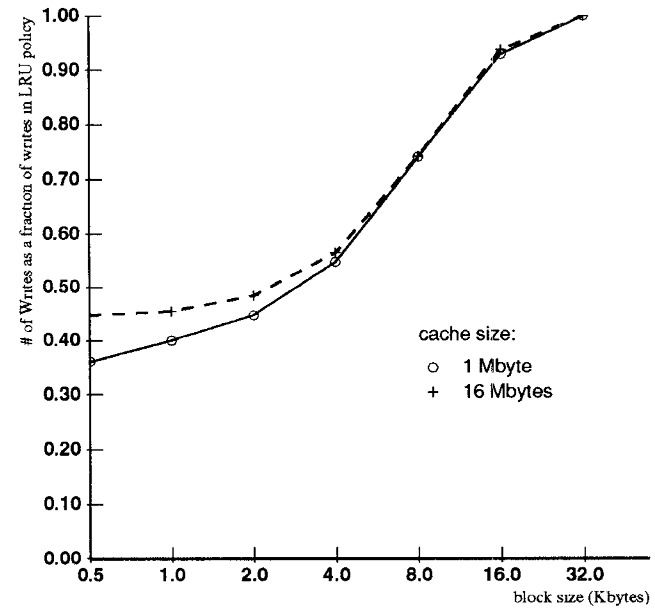

Fig. 9. Performance of write-back policy.

4.7 Impact of two levels of caching

In most systems, part of the main memory is used as an I/O cache (file cache). Besides this cache, the I/O system may have a cache on its own (disk cache). If I/O cache is used in two places, in the memory and the I/O system, the second level cache in the I/O system may be of little benefit depending on the size of the cache employed in the main memory. This is illustrated in Fig. 10. The figure shows the effective access times observed at the second level cache in the I/O system when the system has a first level I/O cache of 4 Mbytes, in both caches the block size and the fetch size being 4 kbytes. It is observed that up to a size of 4 Mbytes, there is very little improvement in access times. It is also observed that even at higher cache sizes, the second level in a two-level cache has effective access times that are higher than a single level cache of the same size. The reason for this is that the first level cache takes away much of locality and hence the second level cache observes higher miss ratios and hence higher effective access times than a single level cache of same size. This indicates that benefits of cache are more significant if the cache space is located in one level. This also raises the question that if there should be a cache close to the disk system since a main memory I/O cache reduces its benefit significantly.

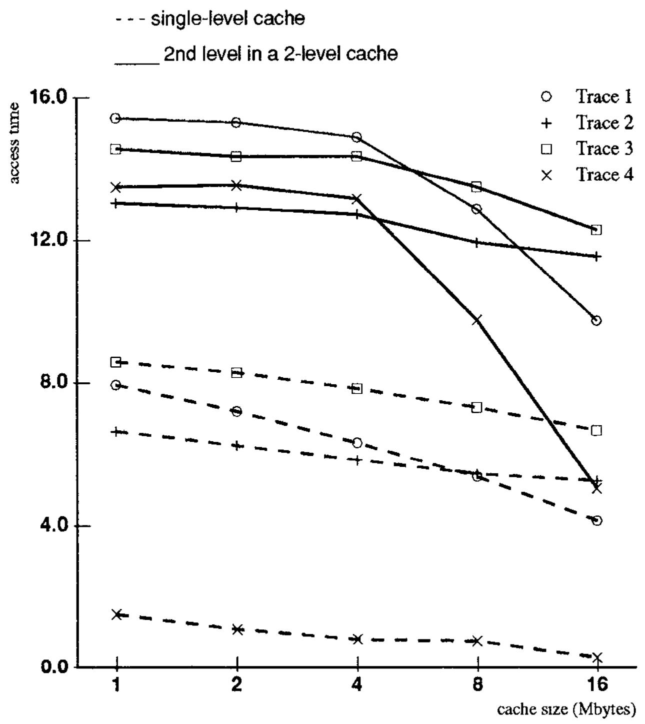

Fig. 10. Access times of a two-level cache.

Keeping the cache closer to the processor (in main memory) reduces page fault ratio and other processing overheads as well. Keeping it closer to the disks enables sharing the cache between different processors connected to the same I/O system. This question is also addressed in [17]. The optimal configuration for I/O caching (between main memory and the disk system) remains an open question and further study is required.

5 Conclusions and Future work

In this paper, we have presented a study of I/O system organizations. Several design choices and policies were evaluated through trace-driven simulations. It was shown that decoupling the cache block size and the fetch size yielded significant performance benefits. It was also shown that by suitably choosing these parameters, the read/write characteristics of the workload seen by the disk system can be considerably altered. This is particularly useful in I/O systems such as disk arrays where the costs of reads and writes are different. A new write-back policy presented in this paper is shown to reduce the number of disk write operations considerably. It was also shown that a previous conjecture about block size being determined by the product of latency and data rate is indeed true. It was shown that split access operation of disk could result in considerable performance benefits. Some results were presented to show the effects of two levels of caching in the I/O system.

The impact of various parameters and policies in a cached disk array system remains to be studied. The effect of two levels of caching is only briefly explored here. Given a performance goal, what is the best way to organize a cached I/O system at a fixed cost? We intend to pursue these questions in our future work.

6 Acknowledgements

Discussions with John Fu, of University of Illinois, Jai Menon, Dick Mattson, Robert Morris, Rich Freitas, John Palmer and Jim Brady, of IBM corporation, have been helpful.

References

[1] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *Proc. AFIPS Spring Joint Comput. Conf.*, 30:483–485, April 1967.

[2] J. Akella and D. P. Siewiorek. Modeling and measurement of the impact of input/output on system performance. *Proc. of 18th Ann. Symp. on Computer Arch.*, May 1991.

[3] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.

[4] A. L. Narasimha Reddy and P. Banerjee. An evaluation of multiple-disk I/O systems. *IEEE Trans. Comput.*, C-38, no. 12:1680–1690, Dec. 1989.

[5] K. Salem and H. Garcia-Molina. Disk striping. *Int. Conf. on Data Engineering*, pages 336–342, 1986.

[6] M. Livny, S. Khoshafian, and H. Boral. Multi-disk management algorithms. *Proc. ACM SIGMETRICS Conf.*, pages 69–77, May 1987.

[7] M. Y. Kim. Synchronized disk interleaving. *IEEE Trans. Comput.*, C-35, no. 11:978–988, Nov. 1986.

[8] A. J. Smith. Disk cache-miss ratio analysis and design considerations. *ACM Trans. on Comput. Systems*, 3, no. 3:161–203, Aug. 1985.

[9] J. K. Ousterhout, H. DaCosta, D. Harrison, J. Kunze, M. Kupfer, and J. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. *Proc. 10th Symp. on Operating System Principles*, pages 15–24, Dec. 1985.

[10] J. Ousterhout and F. Douglis. Beating the I/O bottleneck: A case for log-structured file systems. *Tech. Rep., Dept. of EECS, Univ. of California, Berkeley*, Aug. 1988.

[11] M. Rosenblum and J. K. Ousterhout. The LFS storage manager. *Proc. of ACM Symp. on Oper. Syst. Principles*, 1991.

[12] The SPEC benchmark suite. SPEC, Fremont, CA, Dec. 1990.

[13] A. L. Narasimha Reddy. Reads and writes: when I/Os aren't quite the same. *IBM Tech. Report: RJ 8033*, March 1991.

[14] A. J. Smith. Line (block) size choice for cpu cache memories. *IEEE Trans. on Computers*, Sept. 1987.

[15] P. M. Chen and D. Patterson. Maximizing performance in a striped disk array. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, June 1990.

[16] S. Przybylski. The performance impact of block sizes and fetch strategies. *Proc. of 17th Ann. Symp. on Computer Archicture*, May 1990.

[17] K. Li and K. Petersen. Evaluation of memory system extensions. *Proc. of 18th Ann. Symp. on Computer Arch.*, May 1991.