

Research Report

Reads and Writes: When I/Os Aren't Quite the Same

A. L. Narasimha Reddy

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).



Research Division
Yorktown Heights, New York • San Jose, California • Zurich, Switzerland

Reads and Writes: When I/Os Aren't Quite the Same

A. L. Narasimha Reddy

IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

ABSTRACT: Disk arrays are gaining increased attention as a feasible I/O organization. In these I/O systems, reads and writes to the system have different performance implications. A write operation actually results in more than one I/O operation. The I/O system's performance and its organization hence depends on the relative costs of read/write operations and their relative frequency in I/O workloads. In this paper, we characterize the read/write behavior of I/O workloads and how caches affect these characteristics. The characterization, based on real traces, is expected to give a clearer understanding of the performance implications for the I/O system. It has been a widely held belief that disk caches increase the percentage of write requests going to the disk. It is shown through analysis and simulations that this may not be the case and in some cases the disk cache can decrease the percentage of write requests going to the disk.

1. Introduction

Processor speeds have been steadily increasing over the last few years and it is expected that this trend continues for some more time. However, there has not been a corresponding growth in the I/O system's performance. These disparities in the processing power and the I/O system's power would eventually make the problem solving speed be limited by how fast the I/O can be done. Recently, there have been a number of proposals to improve the I/O performance [1, 2, 3, 4]. All the proposed systems employ some kind of parallelism for accessing data in parallel from the I/O system. Disk sizes have been decreasing and these parallel systems take advantage of the smaller disks by suitably distributing data across multiple disks. When a large disk is replaced by a number of smaller disks, to achieve the same reliability as a single disk, some kind of fault tolerance needs to be provided. RAID [1] is a proposal to provide this fault-tolerance by storing the parity of data across multiple disks on another disk. If any one of these disks fails, the data on the failed disk can be obtained through proper XOR operation of the data on the functional disks.

The parity information stored on the disks has a number of effects. The most important of these effects is that to write a single block of data to such a disk system requires four I/O operations. Whenever any data are written to the disk system, the corresponding parity information needs to be updated on the disks. To update the parity information, we need to read the old version of the data, XOR this with the old parity information on the disk and the new data, and store the new parity onto the disk. So, one write request results in four I/O operations. Hence, the writes to the system cause significant overload on the system. The impact of this write penalty on the system's performance depends on the relative frequency of occurrence of writes in the disk system workload and this is the main motivation for understanding the read/write characteristics of the I/O workloads.

I/O systems are equipped with a cache to improve the performance. In this paper, we make a distinction between the I/O system and the disk system. The relation between the two is shown in Fig. 1. The cache stores the most recently referenced blocks. Smith presents an evaluation of benefits of caching in [?]. Caches tend to satisfy some of the I/O requests and the blocks that are not found in the cache are fetched from the disks. No previous study has looked at the effects the cache might have on the I/O workload seen by the disks. Specifically, the read, write characterization of the workload is important from the above discussed problem of write penalty in a disk-array environment.

When a cache is used in an I/O system, the problem of reliability surfaces again. If a

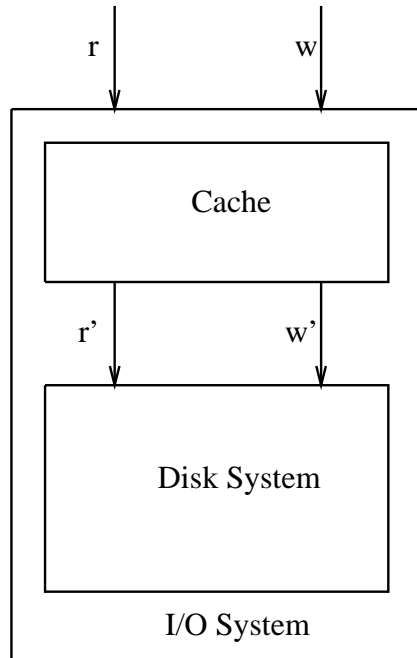


Figure 1: I/O System Model.

write to a disk system is said to be complete when it is written to the cache, the reliability of the I/O system depends on the reliability of the cache. There are two main alternatives for providing this reliability. The first approach is to make the cache nonvolatile such that a power loss would not render the cache unusable. Data in the cache may be inaccessible for several other reasons such as the memory failures etc. To protect against these failures, dual copies of dirty blocks are used. Using a battery backed cache store and dual copies of dirty blocks in the cache, very high reliability of updates to the disk can be guaranteed once the data are written to the cache.

The second approach, adopted in UNIX file systems, is to periodically write the dirty blocks in the cache to the disk. The dirty blocks can be 'flushed' to the disk at some regular intervals, say 30 seconds, to ensure that only a limited amount of work is lost in the event of a cache crash. The period between two flushes, which we will call 'flushing period', can be tuned to suit the reliability and performance needs. This approach requires very little hardware support and can be implemented easily in a file system, though it does not guarantee a reliable update to the disk once a block is written to the cache.

This second approach to ensuring reliability of writes compounds the above discussed

problem of write penalty. The regular flushes of dirty blocks to the disk are expected to increase the write traffic to the disk. Larger caches are expected to make this problem worse. Increased write traffic, along with the write penalty to the disks, prompted Ousterhout and Douglass to propose a log structured file system [5] that optimizes write performance. No empirical evidence for their assumptions is provided. A number of alternatives are also discussed in [5] to reduce the performance penalty of writes in future I/O systems.

In the presence of a cache in an I/O system as shown in Fig. 1, the perceived response time of the I/O system is actually directly related to the read performance of the disk system. The read and write hits in the cache involve no access to the disks and their response time is determined by the cache access time. The write misses can be written to the cache and would involve no immediate disk I/Os if enough space is left open in the cache for them. The read misses cannot be satisfied till the requested blocks are read from the disk system. Hence, the response time observed by the user is dictated by how fast the disk reads can be done. From this perspective, it is important that the disk performance be optimized for reads. This is in conflict with the above observed need for write optimization of the disk system. If both reads and writes can be improved without sacrificing the other's performance, there would be no problem. But when this is not possible, the relative merits of improving the read/write performance have to be investigated.

In the light of above arguments, to design an efficient I/O system, we need to look at the read/write characteristics of the I/O workloads. Also, it is necessary to understand the effects of cache on the I/O workload in the two scenarios discussed above. In this paper, we look at the read/write characterization of the I/O workloads in the presence of caches and the impact the observed behavior might have on the performance of the I/O system. Specifically, we would like to know the read/write ratios of the I/O workload, how this is affected by the cache size and the flushing period (as explained above). What is the impact of making the cache non-volatile? How do improvements in processor speed affect the I/O system organization?

We answer some of these questions through trace driven simulation and an analytical model. We present an analytical model to understand the effects of cache on read/write characteristics of I/O workloads and the interaction of various elements in the system. The results obtained by the analytical model are compared to the results obtained through trace driven simulations of I/O workloads. Section 3 presents the analytical model and Section 2 discusses the simulation model. Results from the simulations are presented in Section 4 and compared to what is expected from the analytical model. Section 5 concludes the paper.

Trace	Reads		Writes		Trace length seconds	# of disks in system
	# of req.	kBytes	# of req.	kBytes		
1	37037	114462	30079	67752.5	3751.45	4
2	46215	120931.5	23532	21474	6053.96	7
3	49545	140215.5	54735	38211	2003.89	16
4	54171	94569.5	48558	82833	5422.18	8

Table 1: Characteristics of the traces.

2. System Model and the Traces

We use a number of traces obtained on the IBM AS/400 systems. Four traces are used in our evaluation. The traces are collected on different systems. The traces are obtained using a hardware tracing facility that listens to the I/O activity on the bus and dumps them to a memory module. 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. The second trace, Trace 2, contains mostly batch requests obtained from a System 38 with the same architecture as a AS/400 system. The third trace, Trace 3, contains only interactive requests. The fourth trace, Trace 4, is obtained from a server application, where the machine is used as a server for a number of terminals. Table 2 lists the characteristics of these traces. The number of disks used in the different systems is noted in Table 2. Different types of disks were used in these systems. However, the types of disks used in the system do not affect the results reported in our study here.

The disk cache is modeled in the following way. The cache is assumed to be organized on a track basis. The data are always read from the disk in units of one track. If data are requested from the disk, the entire track to which this request belongs to is read into the cache. The cache was organized on a track basis for two reasons: (1) the future disk systems are expected to use track as a basis of transfer for performance reasons [6, 2]. (2) Earlier reports on disk caches show that when the cache block is a track, good performance is obtained for most cache sizes [?].

When an I/O request is issued, a check is made to see if this block resides in the cache. If it does, the block is marked as the most recently used block in the cache. If a miss occurs, a cache block is deallocated/evicted from the cache and the new block replaces it. The blocks

in the cache are deallocated on the basis of LRU policy. When any data element is referenced in the block, the entire block is marked as the most recently referenced block. The traffic patterns can be quite different based on what policy is used on a write miss. If an allocate on write miss policy is used, then by this policy alone we can guarantee that the read traffic at the disks will be higher than write traffic (since every miss, whether read or write would involve a disk read, but only dirty evictions would cause a disk write). To avoid this problem with large cache blocks, we employed the following method: on a read miss, the entire cache block is assumed to be read into the cache; but on a write miss, a cache block is allocated. Subsequent writes to sectors in this block are considered a hit. But on a read attempt to this block a read miss is assumed and the entire block is brought in from the disk and written to cache without overwriting the already existing dirty sectors in this block. For each block, a vector of dirty bits are used to indicate which of the sectors in the block are dirty. The block is considered dirty if any of the sectors in the block is dirty. When a block is deallocated, all the dirty sectors in the block are written back to the disk.

The cache is assumed to be organized in a fully associative fashion. We considered two possible hardware configurations for the cache: (1) volatile and (2) non-volatile. In a volatile cache, the dirty blocks in the cache are written to the disk at regular intervals so that the reliability of updates can be guaranteed to some degree. The time period between two flushes, flushing period, is varied in our simulations among 30, 60, 300, 600 and 3600 seconds. When a non-volatile cache is used, disk writes are initiated only when a cache miss results in evicting a dirty block from the cache. The cache size is varied from 1-32 Mbytes. The traces were not long enough to get accurate results for larger cache sizes.

The cache misses/hits for read/write operations are recorded separately. When a volatile cache is used, the writes to the disk are marked separately depending on whether they are a result of an eviction from the cache or due to a flush. The number of blocks written in each flush are noted. Total number of reads and writes to the disk system are recorded. Various other statistics such as average request size etc. are marked, but not reported here.

3. A Model for Read/Write I/O Traffic

In this section, we will show how the cache affects the read/write traffic through an analytical model. Assume the read, write request rate to the I/O system (including the cache) is r reads/sec and w writes/sec. In this section, distinction will be made between

Cache Size	Trace 1	Trace 2	Trace 3	Trace 4	Average
1	40.52	29.40	35.98	90.49	49.10
2	41.18	29.25	35.18	88.88	48.63
4	41.42	30.40	34.69	85.38	47.92
8	40.92	32.02	34.54	83.62	47.78
16	40.76	35.87	34.88	80.19	47.93
32	40.50	36.15	35.63	65.69	44.49
Average	40.88	32.18	35.15	82.38	47.65

Table 2: Fraction of dirty evictions from the cache.

the total I/O system (that includes the cache and the disks) and the disk system alone. The relationship between these two systems is shown in Fig. 1. Let m_r and m_w represent the read/write miss ratios in the cache. We would like to establish a relation between the read/write traffic seen by the entire I/O system and the read/write traffic seen by the disk system.

First, consider a nonvolatile cache. Then the number of read requests going to the disk system is given by the number of read requests missing in the cache. Hence, the read request rate seen by the disk system is $m_r * r$ /sec. Each miss in the cache, whether a read or write, results in a block being evicted from the cache. If the evicted block is dirty, then it would result in a disk write operation. Hence, the rate of write requests seen by the disk system is given by $q * (m_r * r + m_w * w)$, where q is the probability that an evicted block is dirty.

The observed values of q for various cache sizes and different traces are shown in Table 2. The average across different traces for each cache size is also shown in the table. It is observed that the average value of q is around 50% for most cache sizes and we will use this value in our discussions further on. The value of q depends on the interaction between the reads and writes on individual blocks in the cache and is hard to determine analytically. It is also to be noted that the value of q will be lower with a smaller cache block size than that considered here (a track of approx. 30 kbytes).

In a volatile cache, every f seconds, the flushing period, all the dirty blocks are written to the disk. Besides these block flushes, there may be some cache misses that may result in writes to the disk between two flushes. Hence, the write traffic is given by $q' * w + q''(m_r * r +$

q	Initial Read ratio R			
	0.5	0.67	0.75	0.8
0.1	0.83	0.87	0.88	0.89
0.3	0.63	0.69	0.71	0.73
0.5	0.50	0.57	0.60	0.62
0.7	0.42	0.49	0.56	0.57
0.9	0.36	0.43	0.45	0.47

(a) $m_r = m_w$ ($= 0.1, 0.2, 0.3, \dots$)

q	Initial Read ratio R			
	0.5	0.67	0.75	0.8
0.1	0.87	0.89	0.90	0.90
0.3	0.69	0.73	0.74	0.75
0.5	0.57	0.62	0.63	0.64
0.7	0.49	0.53	0.55	0.56
0.9	0.42	0.47	0.49	0.50

(b) $m_r = 2 * m_w$ ($= 0.1, 0.2, 0.3, \dots$)

Table 3: Read ratios at the disk system with nonvolatile cache.

$m_w * w$), where q' gives the fraction of the writes that result in dirty blocks by the time for the next flush and q'' represents the fraction of misses in the cache that result in disk writes between two flushes. For small cache sizes, the second term dominates the write workload and for larger caches most of the writes are a result of flushing the cache.

For the above two cases, the fraction of read operations in the disk traffic are calculated below. For nonvolatile caches, the fraction of read operations is given by $m_r * r / (m_r * r + q * (m_r * r + m_w * w))$. If the initial read ratio of I/O requests is R, then the resulting read ratio to the disk is given by $R' = m_r * R / (m_r * R + q * (m_r * R + m_w * (1 - R)))$. Table 3 shows the expected read ratio of requests at the disk system as a function of the miss ratios and the read ratio of requests seen at the I/O system. From the table, it can be seen that in quite a few cases, the read ratio at the disk system actually goes up as a result of the cache. This is contrary to the popular belief that caches would increase the write traffic ratio at the disk system. For the average value of $q = 0.5$, as obtained from Table 3. , the read ratio

q'	Initial Read ratio R			
	0.5	0.67	0.75	0.8
0.1	0.50	0.67	0.75	0.80
0.3	0.25	0.40	0.50	0.57
0.5	0.17	0.29	0.38	0.44
0.7	0.13	0.22	0.30	0.36
0.9	0.10	0.18	0.25	0.31

(a) $m_r = 0.1$

q'	Initial Read ratio R			
	0.5	0.67	0.75	0.8
0.1	0.67	0.80	0.86	0.89
0.3	0.40	0.57	0.67	0.73
0.5	0.29	0.44	0.55	0.62
0.7	0.22	0.36	0.46	0.53
0.9	0.18	0.31	0.40	0.47

(b) $m_r = 0.2$

Table 4: Read ratios at the disk system with volatile cache.

at the disk system is in the range of 0.5-0.65 for all the cases considered. Hence, the model predicts that when $q = 0.5$, the read traffic at the disk system is larger than the write traffic if the read traffic was larger than the write traffic at the cache.

Carrying out a similar analysis for the volatile cache, we obtain the read ratio observed at the disks is given by $R'' = m_r * r / (m_r * r + q' * w + q''(m_r * r + m_w * w))$. Assuming that the value of q'' is zero, we tabulate the expected read ratio at the disk system as a function of q' . Table 4 shows the expected read ratio of the I/O traffic seen at the disk system when a volatile cache is used. When $m_r * r \ll q' * w$, for a given flushing period, R'' is then proportional to $m_r * r$. This implies that the read ratio of the disk traffic can be arbitrarily reduced by appropriately increasing the cache size. The table shows that a volatile cache significantly reduces the read ratio of the I/O traffic seen by the disk system.

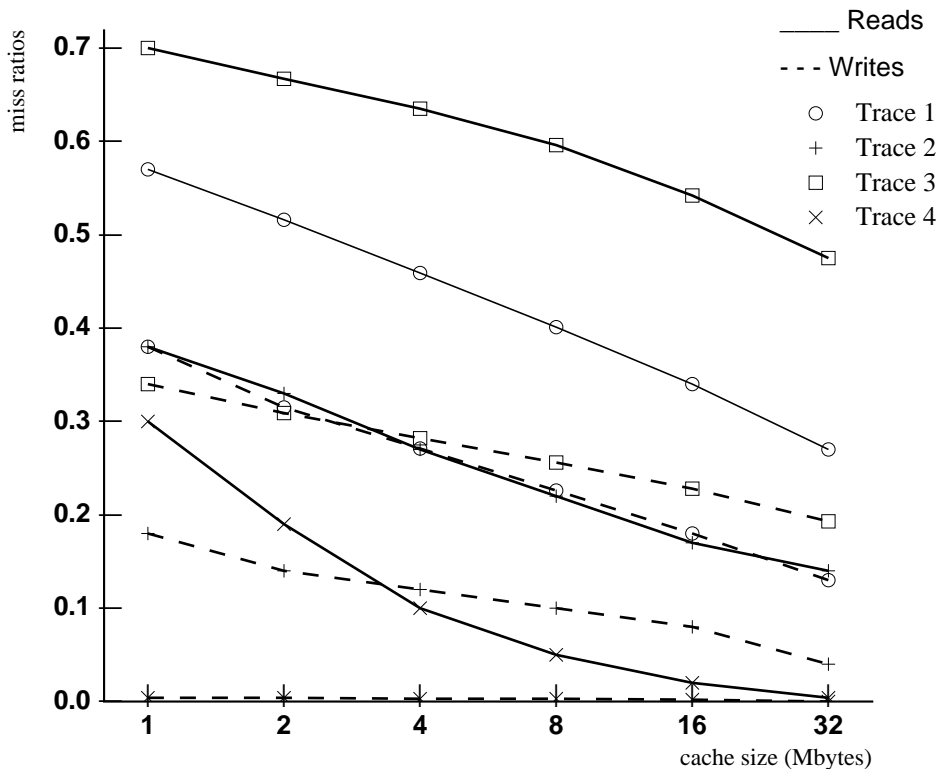


Figure 2: Effect of cache size on miss ratios

4. Results

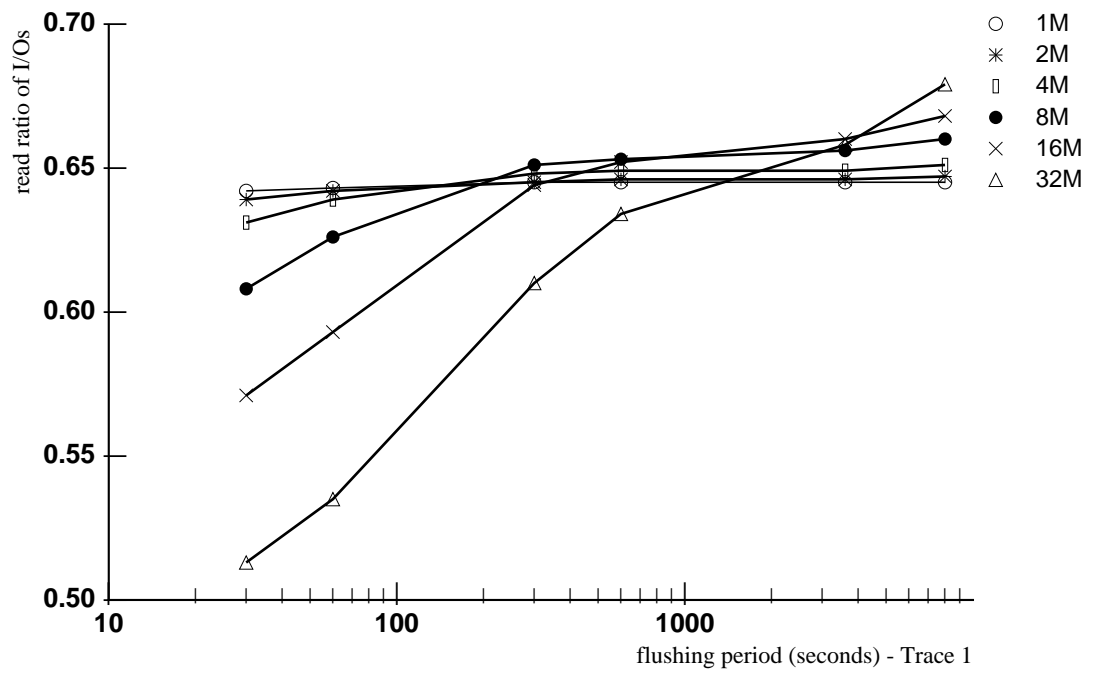
In this section, we report various results obtained through the simulation model discussed in Section 2 and compare them with the predicted results through the analytical model of Section 3.

Fig. 2 shows the miss ratios observed for all the four traces when the cache size is varied from 1 Mbytes to 32 Mbytes. Miss ratios decrease rapidly upto a point and from then onwards decrease at a much slower pace. It was observed that each doubling of cache size brought approximately 15%-20% drop in miss ratios. This effect is visible in Fig. 2 by the nearly linear miss ratio curves on a log scale of cache sizes. The miss ratios range from 0.7 to 0.01 for the considered traces at various cache sizes. This wide range of miss ratios is expected to give us an understanding of the trends in read/write characteristics of the workloads over a broad range of operation. It is observed that the writes have lower miss ratios than the reads for all the traces considered in this study.

In Fig. 3, the read ratios of disk traffic are plotted as a function of the flushing period and the cache size. The results for the nonvolatile cache are also plotted at the end of each curve. First, it is observed that volatile caches have small read ratios. It is observed that increasing flushing period does not have a significant effect on the read ratio of the disk I/O traffic at smaller cache sizes. But as the cache size grows, the flushing period makes more impact on the observed trends. It is observed that the read ratio is almost constant across all the cache sizes for the nonvolatile cache. As seen from Fig. 3, the analysis of Section 3 predicts the read ratios of the I/O traffic fairly accurately. It is observed that the read ratio of the disk traffic is fairly constant across a wide range of nonvolatile cache sizes. It is observed that as the volatile cache size increases, the read ratio drops significantly. This is due to two factors: constant write traffic due to flushes beyond a certain cache size and the decreasing read traffic due to improved miss ratios. The read ratio can be made arbitrarily small in this case by suitably increasing the cache size. For trace 4, the reduction in the read ratios are more dramatic than the other traces because of much lower read miss ratios observed for this trace.

It is observed that the flushing period has very little effect on the read ratios for small cache sizes. This is due to the fact that the main write traffic for these cache sizes is dominated by the dirty evictions from cache due to misses (the second term in the analysis of Section 3). Hence, the flushing period has very little effect on the read ratio of the I/O workloads at small cache sizes. But, as the cache size increases, the write flushes become dominant and hence the flushing period has more impact on the read ratio of I/Os. Larger flushing periods result in a reduction of write traffic and hence an improved read ratio at the larger cache sizes.

Fig. 4 shows the relative contribution of flushes to the write traffic across different traces. It is observed that the percentage of writes due to flushes increases with the cache size and very quickly reaches 100%. This is expected since the number of misses in the cache decreases as the cache size increases and hence the relative contribution of flushes to writes increases with the cache size. At small cache sizes, the write traffic is mainly due to misses in the cache. A block evicted from cache results in a disk write only if a block written since the last flush needs to be evicted from the cache. But, if the block is written since the last flush, then it is not likely to be LRU block. Hence, a large number of misses (relative to the number of blocks in the cache) are required for a dirty block to be evicted from the cache between two flushes. As the cache size grows, the number of misses go down and also a relatively larger number of misses are required for a dirty block to be evicted from the cache. These two factors contribute to the fact that the flushes constitute 100% of the write traffic



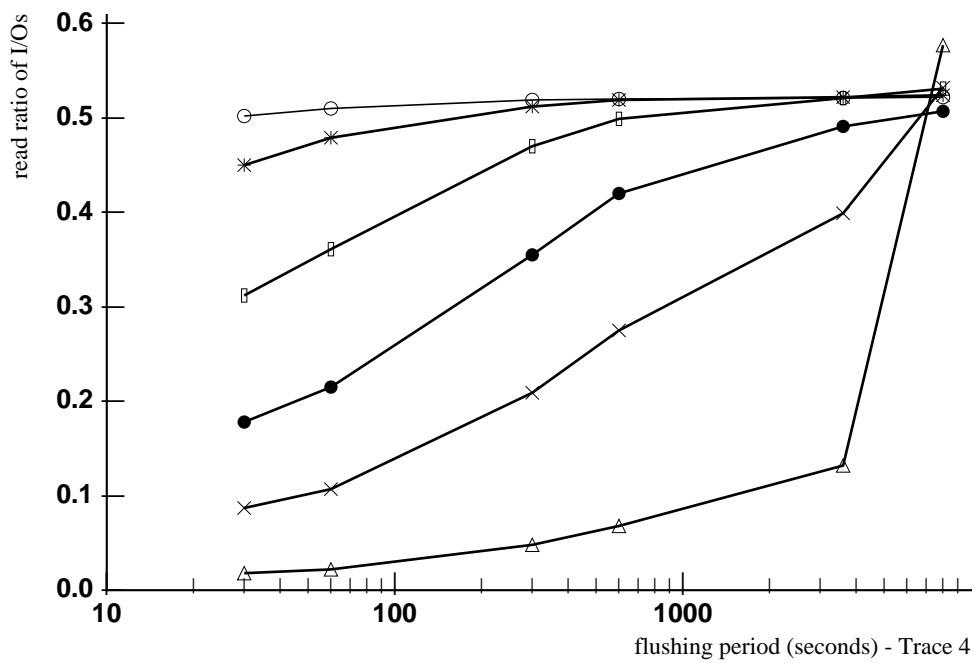
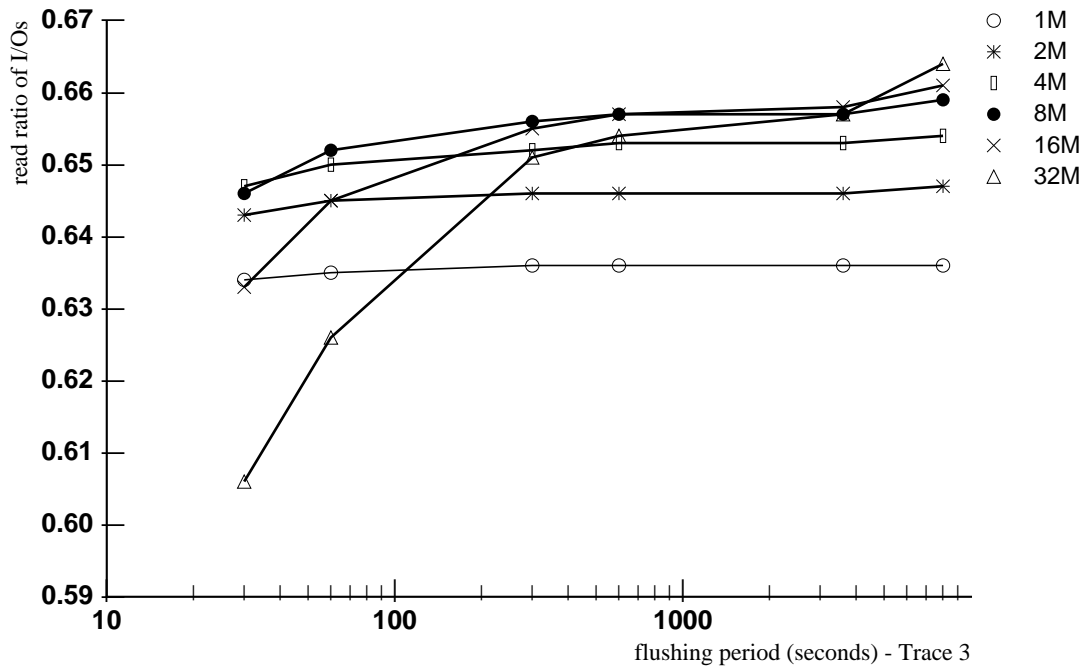
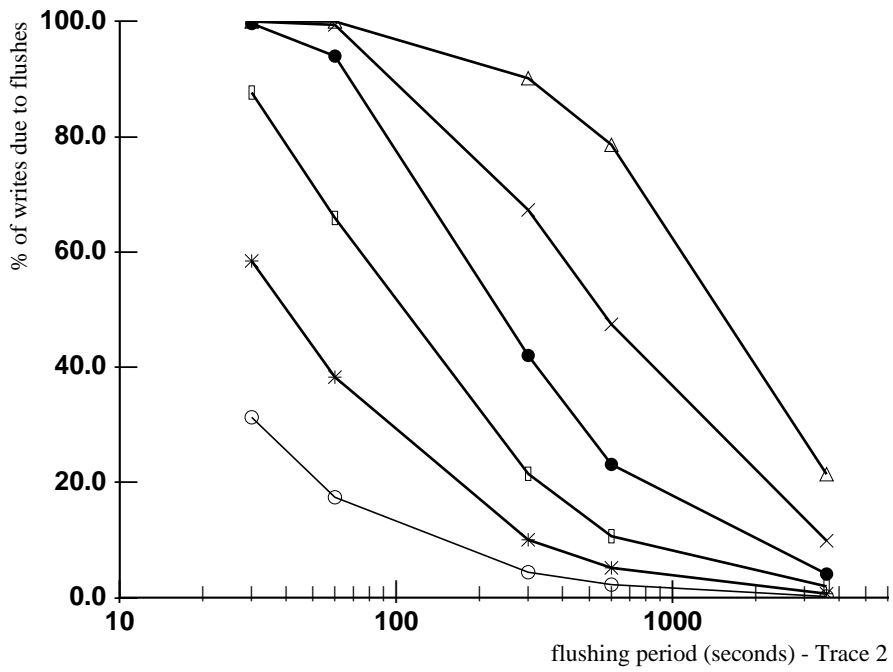
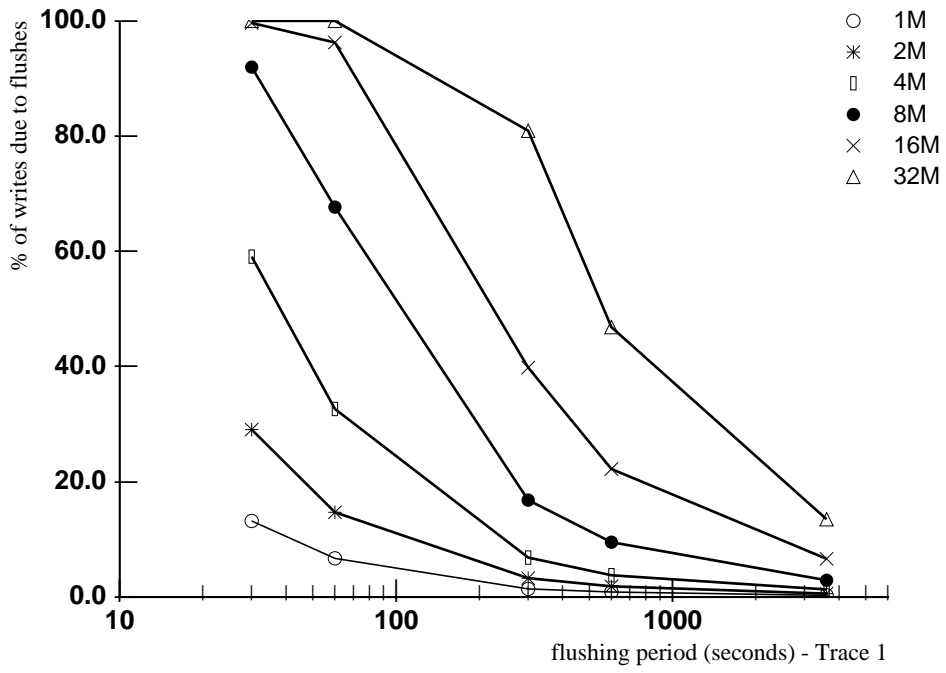


Figure 3: Read ratios for various configurations

very quickly as the cache size grows.

When the majority of the write traffic is contributed by flushes to the disk, the cache does little to reduce the write traffic. The cache reduces the write traffic in the following two ways: (1) multiple writes to a single cache block result in only one disk write when this block is evicted from the cache and (2) when the cache block is larger than the average write operation, multiple write operations may fall in one cache block and will result in one disk write rather than multiple smaller writes. For a volatile cache, these two effects result in a benefit only if these effects take place within a flushing period. For a nonvolatile cache, the blocks have longer time to accrue the benefits from these two effects and as a result the cache becomes more effective in reducing the writes appearing at the disk. The number of blocks written to the cache between two flushes is only dependent on the characteristics of the application and hence a larger cache size does not reduce the write traffic to the disk. This effect is visible in Fig.4, where the total write traffic to the disk is plotted as a function of cache size when the flushing period is 30 seconds. For all the traces, the write traffic reaches a non-zero minimum value and stays constant even with larger cache sizes. Increasing cache size reduces the read miss ratio and hence the number of disk reads. This is the reason for seeing decreased read ratio in the disk traffic with increased cache sizes. This observation confirms the assumptions made in [5] for proposing the log structured file system. The traffic to the disk cannot be reduced to zero as a result of the constant write traffic due to flushes. Hence, in the systems where flushing is employed, the performance benefits of increasing the cache size are seen only in a reduction of read misses.

What effect does improved processor speed have on the traffic seen at the disks? Does the constant write traffic due to flushes result in 10 times higher I/O rate at the disk in a system with a processor that is 10 times faster? How does read ratio get affected due to improvements in processor speed? In a system with a nonvolatile cache, the only characteristic that changes would be the I/O rate (/sec) of the system. To study these effects in a system with a volatile cache, we could reduce the trace times by a factor of 10 to simulate a system issuing requests 10 times faster. A flushing period of 30 seconds to a processor with a speed of 10x effectively looks like a flushing period of 300 seconds for a processor with speed x. Hence, to study the effect of a faster processor on the I/O workloads, we need only look at the results at a suitably larger flushing period. Continuing our example, the characteristics of a system that is 10 times faster than that used in the traces with a flushing period of 30 seconds would be identical to the results obtained from these traces with a flushing period of 300 seconds (keeping in mind the factor of 10 in time when calculating the I/O rates). Due to locality, we don't see 10 times as many blocks being flushed to the disk in each flushing period. The



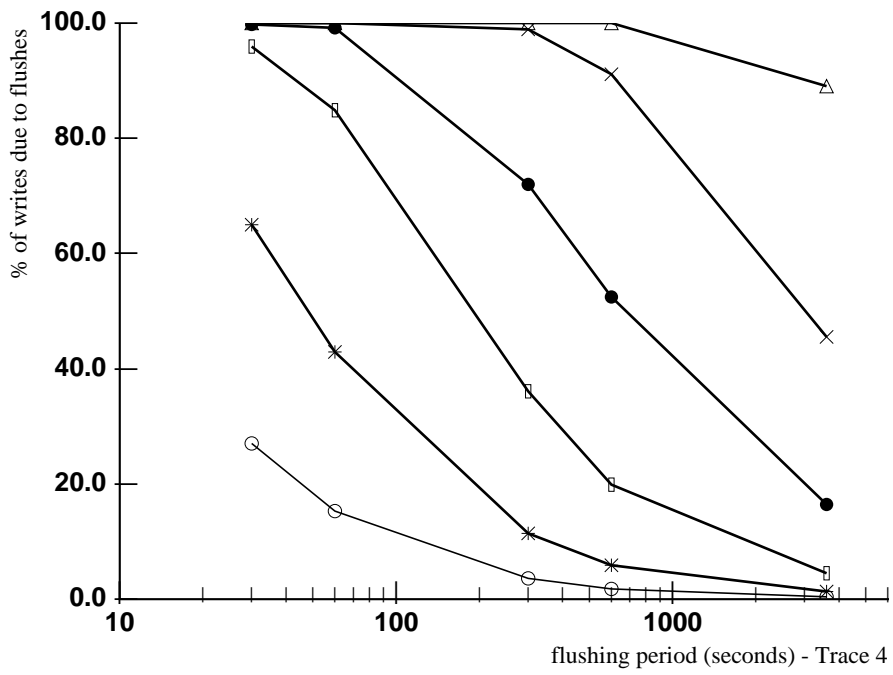
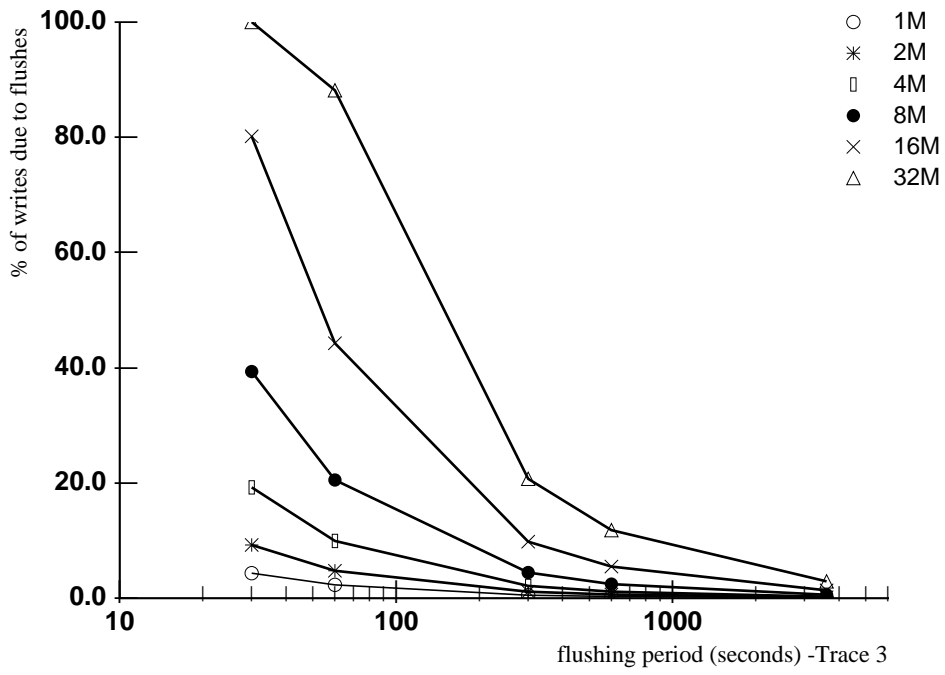


Figure 4: Contribution of flushes to write traffic.

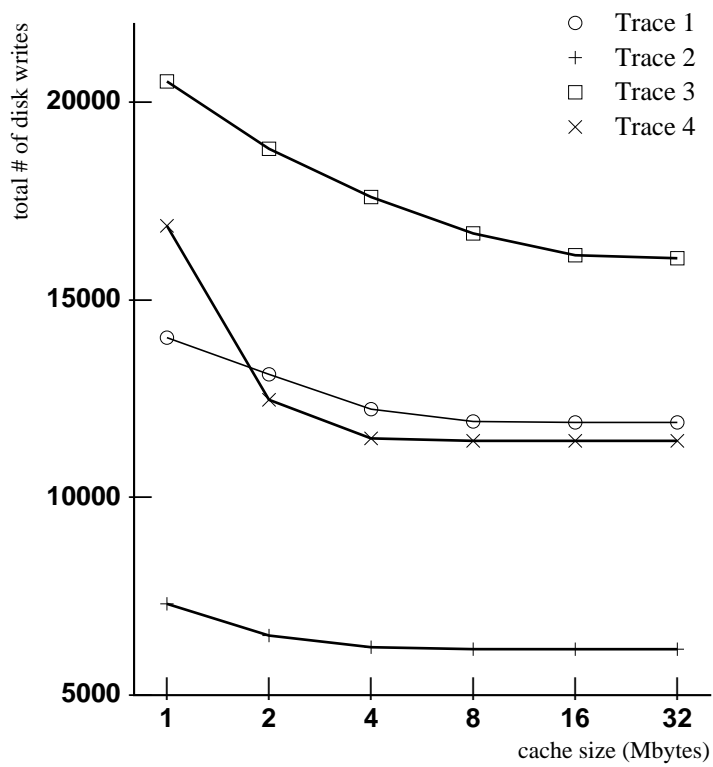


Figure 5: Effect of volatile cache on write traffic.

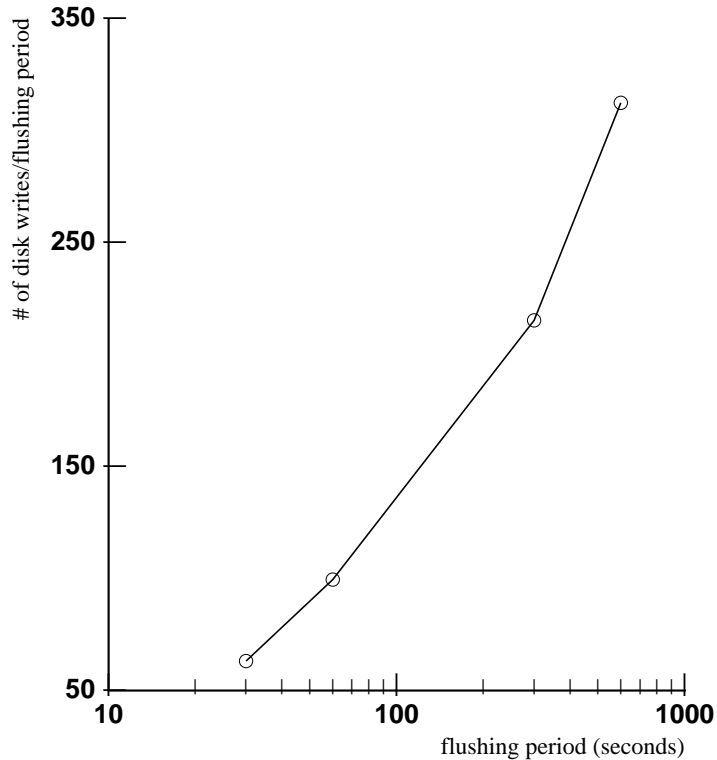


Figure 6: Average # of blocks flushed per flushing period.

hot blocks in the cache get flushed repeatedly to the disk and among successive flushes, the set of dirty blocks differ very little. This effect is shown in Fig. 4 for trace 4 with a 32 Mbyte cache. This implies that the write traffic rate (/sec) to disk grows slower than the speed of the processor. However, the read traffic rate is proportional to the read misses in the cache and hence grows at the same rate as the processor speed. As a result, with increased processor speed, with the same I/O system configuration, we would see higher read ratios. This can be observed in Fig. 3, where the read ratio for any size cache is higher with larger flushing periods (or faster processor speeds). At smaller cache sizes, the write traffic to disk is dominated by dirty evictions from the cache which grows with the processor speed, and hence the read ratio remains constant as the processor speed is increased.

5. General Discussion

From the analytical model and the results obtained through trace driven simulations, two points emerge clearly: the read traffic dominates the I/O workload at the disk with a nonvolatile cache and the read ratio of the I/O traffic at the disk can be made arbitrarily small by choosing a suitable volatile cache size and a flushing period. Our results seem to indicate that the assumptions made in [5] for proposing a log structured file system for UNIX environment are valid. It is clear in such a system, the file system needs be write optimized.

This seems to be the case even if a disk system with relatively higher write cost, such as a disk array, is not used. Methods to use the cache more effectively in such an environment need to be investigated since the cache helps in reducing only the read traffic to the disk (beyond a certain cache size).

But, how should the I/O system be designed when a nonvolatile cache is used with a disk system that has a higher write cost? The answer to this question is not very clear and cannot be answered from this study alone. Even though the read traffic is dominant in such a system, if the relative cost of a write operation is much higher than a read operation, the disk system may have to be write optimized. As argued earlier, user response time is dictated by the read response time of the disk system and hence there is a conflicting demand to optimize the I/O system for read performance. Also it is not clear how a write-optimized system affects the read performance in general. If the read cost in such a system is far worse than that in a regular I/O system, the overall system performance may not be improved. The various factors involved in such a tradeoff remain to be studied. This issue can only be resolved by simulating the disk system along with the cache simulations studied in this paper. The answer to this question may depend on the performance metric used: maximum throughput, maximum throughput for a given limit on response time or the response time of the system itself. This will be a topic for our future study.

6. Conclusions

In this paper, we studied the read/write characteristics of the I/O workloads and how these characteristics are affected by the presence of a cache in the I/O system. We presented an analytical model to derive how the cache affects the read ratio of the I/O workload. It was shown that it is possible to increase the read ratio of the workload as a result of using a cache contrary to popular beliefs that a cache reduces the read ratio of I/O traffic. Extensive trace driven simulation results were presented to show that the read ratio remains high when a nonvolatile cache is used. It is also shown that with a periodic flushing of the cache, the disk traffic is dominated by writes. It was shown that increased flushing period or increased processor speed increases the read ratio of the traffic.

7. Acknowledgements

The author wishes to gratefully acknowledge the help of Aare onton of IBM Almaden Research Center in obtaining the traces used in this study. Discussions with Jai Menon, Rich Freitas, Dick Mattson and John Palmer of IBM Almaden Research Center have helped improve the quality of the paper.

References.

- [1] D. A. Patterson, G. Gibson, and R. H. Katz. A case for redundant arrays of inexpensive disks (RAID). *ACM SIGMOD Conference*, June 1988.
- [2] A. L. Narasimha Reddy. Parallel input/output architectures for multiprocessors. *Ph. D Thesis, CRHC Tech. Rep.:90-5*, 1990. Coordinated Science Lab., University of Illinois, Urbana-Champaign.
- [3] K. Salem and H. Garcia-Molina. Disk striping. *Int. Conf. on Data Engineering*, pages 336–342, 1986.
- [4] M. Y. Kim. Synchronized disk interleaving. *IEEE Trans. Comput.*, C-35, no. 11:978–988, Nov. 1986.
- [5] J. Ousterhout and F. Douglass. Beating the I/O bottleneck: A case for log-structured file systems. *Tech. Rep., Dept. of EECS, Univ. of California, Berkeley*, Aug. 1988.
- [6] P. M. Chen and D. Patterson. Maximizing performance in a striped disk array. *Proc. 17th Ann. Int. Symp. on Computer Architecture*, June 1990.