Trace Driven Analysis of Write Caching Policies for Disks

P. Biswas, K.K. Ramakrishnan
and Don Towsley

Computer Science Technical Report 93-30
April, 1993

Trace Driven Analysis of Write Caching Policies for Disks

Prabuddha Biswas
Business and Office Systems
Engineering Performance Group
Digital Equipment Corporation
110 Spitbrook Road, Nashua, NH.
biswas@xanadu.enet.dec.com

K. K. Ramakrishnan
Distributed Systems Architecture and
Performance
Digital Equipment Corporation
550 King Street, Littleton, MA.
rama@erlang.enet.dec.com

Don Towsley
Dept. of Computer Science
University of Massachusetts
Amherst, MA.
towsley@cs.umass.edu

Trace Driven Analysis of Write Caching Policies for Disks*

ABSTRACT

The I/O subsystem in a computer system is becoming the bottleneck as a result of recent dramatic improvements in processor speeds. Disk caches have been effective in closing this gap but the benefit is restricted to the read operations as the write I/Os are usually committed to disk to maintain consistency and to allow for crash recovery. As a result, write I/O traffic is becoming dominant and solutions to alleviate this problem are becoming increasingly important. A simple solution which can easily work with existing file systems is to use non-volatile disk caches together with a write-behind strategy. In this study, we look at the issues around managing such a cache using a detailed trace driven simulation. Traces from three different commercial sites are used in the analysis of various policies for managing the write cache.

We observe that even a simple write-behind policy for the write cache is effective in reducing the total number of writes by over 50%. We further observe that the use of hysteresis in the policy to purge the write cache, with two thresholds, yields substantial improvement over a single threshold scheme. The inclusion of a mechanism to piggyback blocks from the write cache with read miss I/Os further reduces the number of writes to only about 15% of the original total number of write operations. We compare two piggybacking options and also study the impact of varying the write cache size. We briefly looked at the case of a single non-volatile disk cache to estimate the performance impact of statically partitioning the cache for reads and writes.

*To be presented at the 1993 Sigmetrics Conference.

1 INTRODUCTION

The performance of computer systems has been increasing rapidly in the recent past. This has been largely due to dramatic increases in processor speeds and the rapid reduction in the costs of memory. The increase in processor speeds has allowed compute intensive tasks to see the benefit directly. This has not been true for I/O intensive tasks. Although the reduction in the cost of memory has allowed the use of large caches, which has alleviated the problem, the performance of the basic I/O subsystem and particularly the disk subsystem has not increased as quickly as needed. Thus, the I/O subsystem is often the bottleneck. Several research efforts have been addressing this issue. One of them is the traditional use of large caches, both for the file system and in the disk subsystem [7, 14, 15]. In addition, more recently there have been efforts to use disk arrays for improving disk subsystem performance [3, 8, 10, 12].

Most commercial systems use write through disk caches to maintain strict consistency and to allow for crash recovery. Therefore, only the read I/Os get the benefit of the cache and write operations begin to dominate the traffic to the disk. It is becoming increasingly important to address the performance of write operations. The use of Log Based File Systems [11] addresses this problem of improving the performance of writes. Another approach that is also becoming attractive is to use non-volatile caches which allow for write-behind rather than write-through and thus reduce the number write operations to the disk [1, 6, 16]. This is partly because non-volatile RAM (NV-RAM) is becoming cheaper and it is easy to incorporate into existing system designs without major changes to the file system or the rest of the operating system. It must be noted, however, that NV-RAM is still sufficiently expensive so that it is not feasible to use a single large NV-RAM cache for the entire disk cache. We therefore consider the performance issues when there is a separate read cache and a non-volatile write cache. In particular, we are interested in devising efficient management policies for the write cache because it can only consist of a relatively small amount of NV-RAM. We would also like to estimate the size of the write cache that is required to substantially reduce the write I/Os to each disk.

There are several ways to manage a separate write cache. The most obvious policy would be to use a simple write-behind policy, wherein the writes are buffered in the non-volatile cache until it is full of "dirty" (written but not committed to the disk) blocks, at which point in time some number of "dirty" blocks are written out to the disk. We study this and propose enhancements that we consider to be effective. In particular, we look at the usefulness of a hysteresis based write behind policy with variable thresholds to begin and terminate the writing out ("purge") of dirty blocks to the disk. In addition, it is important to examine the interactions between the read and write cache. We also examine several mechanisms which use the read misses from the read cache and the write operations to the disk in a complementary way to enhance performance.

Our study of the performance of these different algorithms for managing the read and separate write cache is based on trace driven simulation. We model the components of the I/O subsystem of interest: the disk controller, the read and write caches which we postulate are part of the controller, and the magnetic media. The traces we use for the simulation are detailed I/O operation traces from several commercial time-sharing VAX/VMS systems [9]. The traces are quite large (between 100-500 K operations per disk) and each of the environments studied had a large number of users. All the I/O operations to the disk subsystem are captured with detailed time stamps and the logical block number of the disk blocks accessed. We also had available the precise disk configurations and architectures to enable us to calculate accurately the position of the disk arm with respect to time. The traces were used to drive a detailed simulation of the disk subsystem and to analyze the caching alternatives. We focus on reducing the number of write operations to the disk without adversely affecting the I/O response time.

Section 2 describes the different policies for managing a write cache that will be studied in this paper. Section 3 provides a brief outline of our methodology and a description of the traces used for the analysis. Subsequently, we compare the different caching policies and provide insights regarding the setting of different parameters of the caches. We also look at the performance impact of varying the size of the non-volatile write cache. The final section summarizes the major conclusions from this study.

2 DESCRIPTION OF CACHING SCHEMES

Disk caches have become a convenient and effective way to close the ever-widening gap between CPU and I/O performance. Smith [15] looks at many of the disk caching options and addresses the important issues of cache location, cache size, cache replacement algorithm, prefetching strategy, block size, etc. Many commercial disk controllers are available that use write-through disk caches for data integrity. Thus, reads to the disk are satisfied directly from the cache. In such systems, as the cache size becomes reasonably large, the cache hit rate for reads can be quite high and therefore the write I/O operations start to dominate the disk I/O traffic. It is clear that if the write operations are also cached in the controller, then I/O performance can be significantly enhanced. Analysis of disks with write behind disk caches can also be found in [15]. One way to achieve that is to use a separate non-volatile buffer to cache the writes. This is used in addition to the traditional disk cache which improves read performance. Therefore, it is becoming increasingly important to analyze the performance of the different caching strategies that are required when there are two separate cache components: (1) a volatile read cache, and (2) a non-volatile write cache.

Our discussion of the different caching strategies assumes a simple I/O sub-system model consisting of a controller, a cache unit and the magnetic disk media. The controller is responsible for receiving the I/O requests in a FIFO queue, checking the cache unit to resolve the request and for scheduling disk operations. The cache unit, as discussed before, has separate caches for reads and writes. The disk is made up of one or more *platters* of magnetic media. There are a certain number of concentric *tracks* on each platter. The tracks at the same position on all platters together comprise a *cylinder*.

2.1 Write Behind

The simplest policy for managing the write cache would be to employ a write behind scheme. As disk blocks are written they are inserted into the non-volatile write cache and when the cache becomes full it is "purged" by committing some of the previously updated blocks to the disk. The controller can optimize the write cache purge operation by identifying the disk track with the most "dirty" blocks in the write cache and writing them all back to the disk in one write operation. The seek distance could also be used to choose the appropriate track to "purge". Once a disk block is committed to disk it is marked "clean" and the entry placed in an available buffer list. A subsequent read to a "clean" block can still be resolved from the write cache. The disadvantage of this scheme is that a write operation that causes the cache to become full is stalled until the cache purge operation is completed and "clean" write cache buffers are available for use.

2.2 Write Behind with Thresholds

The most obvious way to avoid the write stall problem of the previous scheme is to use a proactive write cache purging policy. Each entry in the write cache has a "dirty" flag which is set when a new block is inserted in the cache. When the percentage of the dirty blocks in the write cache exceeds a high-limit threshold (WC-HILIM), the controller notes that condition by raising a "purge-request" flag. Once this flag has been set, the controller finds the first opportunity when the disk becomes idle (i.e. no read operations pending in the controller queue) and "purges" the write cache by committing dirty blocks to the disk. The purge operation is identical to the one used in the previous scheme, in that the track with the most "dirty" blocks in the write cache is chosen for writing to the disk. This scheme has the benefit that the controller attempts to clean the write cache before it becomes full of dirty blocks and tries to do the purges when the disk is quiescent by postponing purge operations if there are read operations pending in the controller queue. Purging is discontinued once the percentage of dirty blocks fall below the threshold. However, the drawback of this scheme is that the purge operation is frequently triggered. This situation can be remedied by introducing a second low limit threshold. The controller continues to schedule purges until the percentage of the dirty blocks in the write cache falls below this low-limit threshold (WC-LOLIM). Purge operations are postponed if there are read operations pending in the controller queue.

The threshold driven proactive purging operations are performed only when the disk is idle. Hence it is possible, during peak load periods, that the controller may not get the opportunity to proactively purge the write cache. In that case, the write cache can become full with dirty blocks. If such a situation arises, the controller schedules an "immediate purge" operation. This operation has the same priority as the read miss requests to the disk and are handled in FIFO order. Immediate purging is discontinued as soon as there are any free buffers in the write cache and the controller returns to quiescent purging if still required.

2.3 Piggybacking

Another mechanism is used to enhance the write behind strategy outlined above. This is to piggyback "dirty" blocks from the write cache with read operations. When a read I/O request is not satisfied by either of the caches, it results in a "Read Miss" and a disk access operation is required. At this point the disk controller checks the write cache for "dirty" blocks belonging to the same disk track as the read request. If such blocks are found, they are "piggybacked" with the read request and written to disk as part of the same disk access operation. The blocks are marked "clean" in the write cache. The disk blocks read are inserted into the read cache which is maintained using the LRU replacement algorithm.

There are two variations of the piggybacking operation - 1) *Free*, and 2) *Full* track piggybacking. In free piggybacking, not all dirty blocks on the target track are eligible for piggybacking. Only the dirty blocks in the write cache in the region between where the head is currently at and the start of the read operation can be written without affecting the read response times. It simply utilizes the rotational latency period gainfully. In the full track piggybacking case, all dirty blocks on the track being visited by the read miss are available for piggybacking. In this case the read service time could potentially be increased as the I/O is not complete until the last dirty block on the track has been piggybacked out. In the example shown in Figure 2.1, only dirty block A is eligible for free piggybacking while blocks A and B are both available for the full piggybacking scheme.

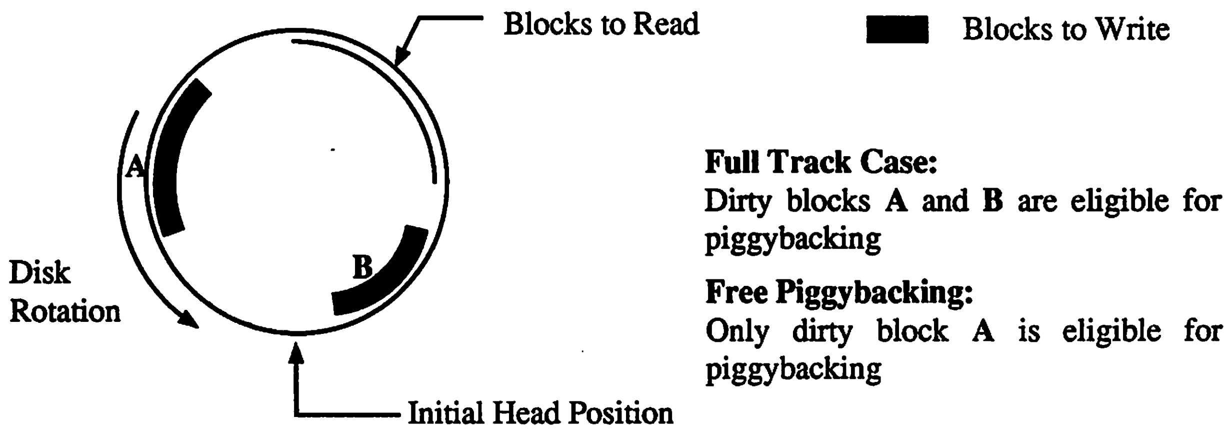

Figure 2.1: Schematic Diagram of a Disk Platter to explain Piggybacking

The way in which the servo information is stored on the disk impacts the purging and piggybacking schemes. Some disks use a *dedicated servo* mechanism where one platter surface is dedicated for servo information. In these disks the heads on each of the platter surfaces are aligned at the same track. In that case the whole cylinder is available for the purge or piggybacking operation. Disks which use the *embedded servo* technique, keep the servo information on every individual platter. These disks achieve greater track density with the side effect that the heads on the different platters are not assumed to be exactly at the same track. When switching from one platter to a new one, the head reads the servo information on the new platter and fine tunes the head position. In these disks there is a significant time delay for switching between platters and all the dirty blocks on one cylinder cannot usually be written out in one rotation. In that case, we pessimistically assume that only one disk track is available for purging and piggybacking. The purging and piggybacking mechanisms function exactly in the same way for the track based policies. We will focus on the track case but also compare it with the cylinder case to provide an upper bound for the reduction in write I/Os that can be achieved.

When a block is written, we need a mechanism to update the read cache. The read cache could use a 1) write allocate, 2) write update, or 3) write purge policy. Under the "write allocate" policy a copy of the block written is also placed in the read cache even if it was not previously in the read cache. The "write update" policy requires that the block be updated in the read cache only if it is already present. On the other hand, under the "write purge" scheme, a block that is written is removed from the read cache if it is present there. The issues surrounding the management of the read cache is not the primary focus of this paper. Hence, we will assume a write purge strategy for most of our results. We will also include a comparison between the write allocate and write purge policies.

3 METHODOLOGY

A trace driven simulation was used to evaluate the different caching options. This is usually the methodology of choice for evaluating cache designs, scheduling algorithms, resource management algorithms, etc. where it is important to capture the correlation in the activity pattern. No simplifying assumption is made about the I/O request arrival stream.

Disk I/O activity was collected at eight commercial customer sites representing interesting and diverse computing environments. The performance impact of the tracing package was small (<1% CPU utilization in all the environments) and the users were unaware of the tracing. Therefore, we believe that the traces capture activity from truly operational, commercial production systems. A detailed description of the tracing mechanism, the traces and the characteristics of file I/O activity can be found in [2, 9]. Based on the environment and file access behavior observed, the traces were broadly categorized into three major interactive classes:

- Office and Scientific Time Sharing
- Time Sharing with some database activity
- Transaction processing

We have chosen one trace from each of these three environments. There were many disks at each of these installations and our analysis was done on many disks from each scenario. For clarity of presentation, we choose one representative disk from each system. In our effort to make the analysis less dependent on the operating system, we did not select the system disks or disks used by some well known VAX/VMS specific applications, even if they were the most active disks in that environment. Private user disks and database disks were chosen for this analysis. Summary data from 6 other disks (two from each environment), including operating system related disks, is presented in Appendix-1 to demonstrate that the results are not affected by the choice of disks for the analysis.

3.1 Trace Descriptions

The traces were collected by instrumenting the I/O processing routines of the operating system. This instrumentation captured every I/O request and recorded detailed information about the system, storage device, file and process identification and disk block request information. Every trace record also contains a timestamp with a 100 nanosecond resolution. The three traces used in this study are classified by their environment and briefly described below:

1. Office Applications and Decision Support (*Off-Day*): The trace was collected at the headquarters of a large Fortune 100 corporation. The installation was a VAXcluster system of 3 processors (~20 SPECmarks) with 51 disk drives. There were about 100 users primarily running office applications and decision support software. Three consecutive days were traced. In this study we used a trace from the *prime time* (7 a.m. to 5 p.m.) period of one day as the access pattern was found to be very similar across the three days [9].
2. Scientific and Office (*Sci-Off*): This trace was collected at a Fortune 100 chemical company site and the workload consisted of a mixture of scientific and office automation applications. The system was a VAXcluster with four processors (~15 SPECmarks) and 19 disk drives. The trace data analyzed was from 7 a.m. to 5 p.m., the usual work day for this site.
3. Airline Reservation (*Air-Rsv*): This is a transaction processing environment for an airline.

There were about 500 agents making airline and hotel reservations on a VAXcluster system of 4 processors (~30 SPECmarks) and 28 disk drives. The system was traced from 9:30 a.m. to 6:30 p.m., the prime time for this environment.

Some of the important static and dynamic characteristics of the of the three disks chosen for this study are summarized in Table 3.1. Note that the disk space is the formatted capacity in megabytes (MBytes). The utilization of the disk (% disk busy in Table 3.1) was calculated approximately based on the number of I/Os and the rated capability of the disk subsystem.

Disk	Disk Space (MBytes)	# Platters	Tracks / Platter	Blocks / Track	Rotational Speed (rpm)
Off-Day	456	14	1251	33	3600
Air-Rsv	1200	13	2652	69	3600
Sci-Off	622	15	1426	57	3600

Disk	# Read I/Os	Avg. Read Size (Kbytes)	# Write I/Os	Avg. Write Size (Kbytes)	Approx. % Time Disk Busy
Off-Day	58468	1.64	91726	2.40	16%
Air-Rsv	326579	5.27	79635	3.40	36%
Sci-Off	432218	1.30	70535	1.70	43%

Table 3.1: Characteristics of disks chosen for analysis

3.2 Simulation Methodology

The simulation model used for this analysis is driven by a time ordered trace of I/O requests. The time resolution of the trace is 100 nanoseconds which is adequate to accurately simulate the disk arm traversal on the disk platters. A device status and a device statistic block is maintained for the disk being simulated. The status block keeps track of the time when the last I/O completed and the head position at that point. As an I/O request is generated from the trace, the disk status block of the corresponding disk is checked to see if the disk is busy. If it is idle, then the time between the completion of the last I/O request and the current time time is taken to calculate the current head position. If the disk is busy at the I/O arrival instant, then the starting head position at the time this request is serviced is the point where the last I/O is considered complete. The I/O request provides the logical block number of the start of the request and the transfer count. That information and knowledge of the disk architecture is used to calculate the exact cylinder, platter and track on disk being visited. I/O requests crossing track and/or cylinder boundaries are also accurately handled. A table of the exact seek times for a large number of selected seek distances is kept for all the disk types handled by the simulation. Linear interpolation is used to calculate the seek time for requests with seek distances not available in the table. An example of the seek time distribution is presented in Figure 3.1. Note that a large number of measured points were used in the non-linear region. A count of the number of I/Os that had to queue for service and a cumulative value of the total waiting time is maintained in the statistic block along with a variety of other cache related information.

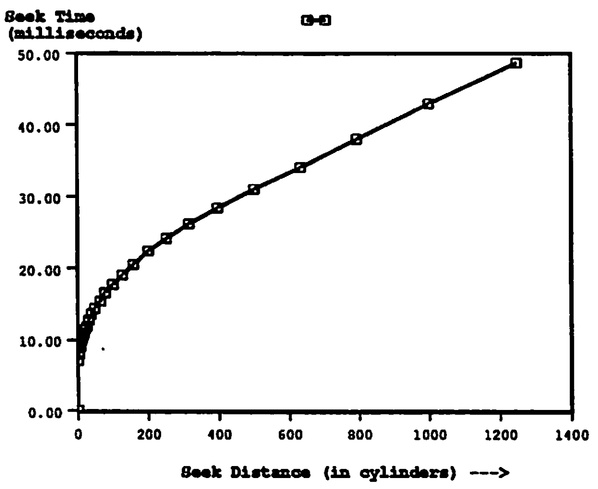

Figure 3.1: Example of Seek Time Distribution

The operation of the caches, as explained in Section 2, was simulated in detail. It is important to point out that the simulation was at the level of 512-byte disk blocks. The read cache is maintained as an LRU-list and uses a hash table for efficient access. The write cache is maintained as a tree structure facilitating access to blocks belonging to the same disk track and cylinder.

3.3 Metrics

The focus of the study is to compare different write caching policies and therefore the most important metric is the number of write operations generated in each scheme. To provide a feel for the factor by which the write I/Os were reduced, we present the number of write operations as a percentage of the original number of write operations without using any write caching policies.

$$\textit{Write with Caching } (W_{wc}) = \frac{\textit{Number of Write Ops with Caching}}{\textit{Total Number of Writes}} * 100\%$$

Similarly, R_{WC} will be used to denote the percentage of the read operations that need to visit the disk. It is important to look at R_{WC} even when the read cache is kept constant because we would like to account for the read operations that are satisfied by the write cache. Not only is it important to reduce the total number of I/O operations but it is also crucial to ensure that the number of I/O operations that have to queue for service should not increase. We express this metric as the percentage of all I/O operations that has to queue for service.

$$\textit{IO Stall} = \frac{\textit{Number of IO Ops Stalled}}{\textit{Total Number of IO Ops}} * 100\%$$

Not only is it important to know the percentage of I/O operations that queue for service but to also estimate the average queueing time. The write cache purging operations are usually done when the disk is idle, so we shall focus on the average read queueing time.

$$\textit{Average Read Queueing Time} = \frac{\textit{Total Read Ops Queueing Time}}{\textit{Total Number of Read Ops to Disk}}$$

Another interesting metric for the schemes that use piggybacking is the percentage of the read miss I/Os that are successful in piggybacking dirty blocks out of the write cache. We will present the piggyback success parameter as the following:

$$Piggyback\ Success = \frac{Number\ of\ Read\ Miss\ IOs\ with\ Piggybacked\ Writes}{Total\ Number\ of\ Read\ Miss\ IOs} * 100\%$$

We shall always refer to the cache sizes in terms of megabytes. The thresholds used in the purging policies are expressed as percentages of the total write cache size.

4 CACHE ANALYSIS RESULTS

4.1 Purging with Thresholds

The write cache contains the disk blocks that were modified which must eventually be committed to disk to make room for new blocks being written. One policy is to wait until the write cache becomes completely full and then perform a write cache purge operation, as explained in Section 2. We believe that such a policy would result in unacceptably poor performance because the write I/O that causes the the cache to become full will stall until the purge operation makes room in the cache. Therefore, we study a track based cache purge policy triggered by a high limit threshold (WC-HILIM) before the cache becomes full. The read and write cache sizes were 2 and 1 Mbytes respectively. In Table 4.1 we observe that the total number of write I/Os is reduced to about 40-50% of the original number of write operations for all three disks chosen for this analysis.

Threshold WC_HILIM	Off-Day		Air-Rsv		Sci-Off	
	W_{wc}	I/O Stalls	W_{wc}	I/O Stalls	W_{wc}	I/O Stalls
80%	41.15%	1.82%	43.29%	13.44%	53.14%	3.63%
90%	39.25%	1.78%	41.52%	13.30%	50.82%	3.68%
95%	39.25%	1.77%	41.19%	13.36%	49.48%	3.65%
99%	43.12%	2.96%	41.19%	14.23%	49.44%	3.81%

Table 4.1: Number of Writes with Caching and I/O Stalls for varying WC-HILIM Threshold values

It is also observed from Table 4.1 that the correct choice of WC-HILIM is in the 90-95% range. Although in some cases, the number of write operations may sometimes be lower at a threshold of 99%, the percentage of I/O operations that have to queue for service also increases at that point. As a result, it is necessary to examine improvements to such a simple purge operation. In particular, we examine the use of hysteresis in the write cache purging process. A low limit threshold (WC-LOLIM) is added and purge operations continue to be performed until the percentage of dirty blocks in the write cache drops below this threshold.

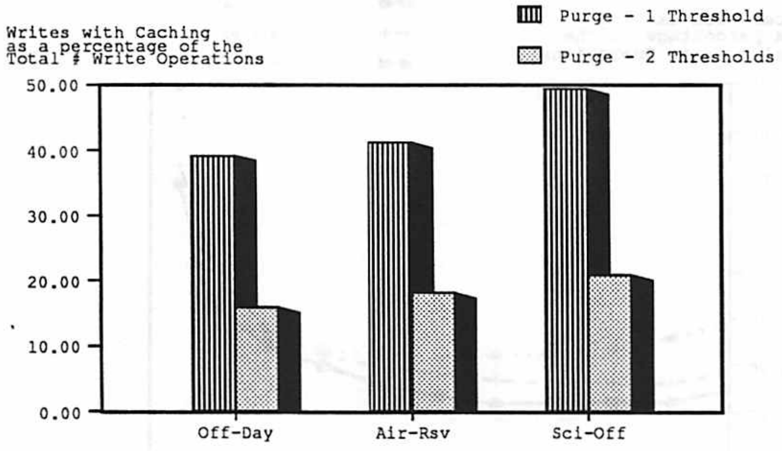

Figure 4.1: Comparison of W_{wc} for Purging with 1-Threshold versus 2-Thresholds

Figure 4.1 shows the total number of write purge operations performed (as a percentage of the total number of writes) for the best settings of one and two thresholds. It is clear that in all cases, the total number of disk write operations is significantly reduced by using two thresholds as opposed to one. The two threshold write behind purging strategy reduces the the number of writes to disk to only about 15-20% of the total number of write I/Os in the trace. This is more than 50% improvement over the single threshold policy.

Next, in Figure 4.2 and Table 4.2 we look at the sensitivity of the number of write cache purge operations to the two thresholds. In Figure 4.2, the WC-HILIM is kept constant at 95% of the write cache size (i.e., the purge operation is triggered when the write cache contains more than 95% dirty pages) and the WC-LOLIM is varied. It is clear that a choice of WC-LOLIM close to WC-HILIM is bad because purging is triggered frequently and only a small number of dirty pages are written out each time. The effectiveness of the cache purging scheme is not affected significantly once the WC-LOLIM value is below 70%. A very low setting (close to 0%) is also not a good choice because it appears to increase the number of write cache purge operations. Dirty pages that may be overwritten are needlessly purged out early (as seen in the Sci-Off environment). The performance of the scheme appears to be quite stable for a wide range (20-60%) of WC-LOLIM values -- that is a desirable feature.

Next we fixed WC-LOLIM at 40% and varied WC-HILIM to understand the impact of this parameter on the performance of the write cache. We observe (Table 4.2) that both W_{wc} and I/O Stalls are very high when WC-HILIM is set at 100%. Clearly, WC-HILIM should be less than 100% so that purging is done proactively. However, if the value is too low, then purging is triggered early and the write cache will be under utilized. This is reflected in an increase in the number of purges to disk. It appears that a value for WC-HILIM in the 95-99% range is an appropriate choice. The difference in W_{wc} and I/O Stalls for WC-HILIM values of 95% and 99% is small and the 95% threshold provides more headroom for the write cache to grow before triggering an immediate purge operation. It was observed that "immediate" purge operations were either negligible or completely eliminated when WC-HILIM was set at 95%.

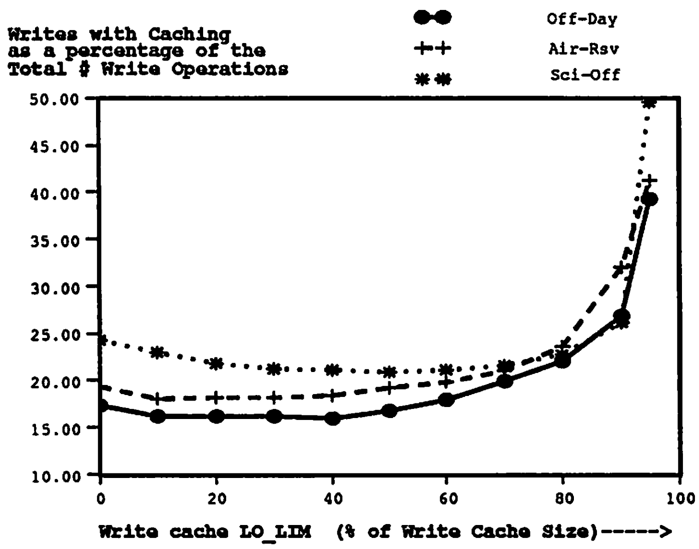

Figure 4.2: Effect of WC-LOLIM on W_{wc}. WC-HILIM = 95%

WC-HILIM	Off-day		Air-Rsv		Sci-Off	
	W_{wc}	IO Stalls	W_{wc}	IO Stalls	W_{wc}	IO Stalls
80%	19.43%	1.09%	20.39%	10.88%	22.54%	2.94%
90%	17.18%	0.89%	19.24%	10.88%	21.47%	3.08%
95%	15.96%	1.00%	18.44%	10.66%	21.09%	2.97%
99%	15.72%	0.95%	18.30%	10.65%	20.55%	2.77%
100%	59.48%	38.37%	42.47%	22.29%	68.06%	13.46%

Table 4.2: Effect of WC-HILIM on W_{wc} and I/O Stalls. WC-LOLIM = 40%

4.2 Cache Size Variation

Before we try to understand the impact of piggybacking in improving the write cache management algorithms, we look at the effect of varying the write cache size while the read cache size is held constant at 2 MBytes. We used the purging scheme with the WC-HILIM and WC-LOLIM thresholds set at 95% and 40% respectively.

Figures 4.3-4.5 show the percentage of total read I/Os that go to the disk (R_{wc}) and the number of cache purge operations as a percentage of the total number of write I/Os(W_{wc}) for the three disks. W_{wc} decreases rapidly as the write cache is increased to about 1 MBytes. Thereafter, we begin to see diminishing returns for increasing the size of the write cache. For example, in the Sci-Off disk, W_{wc} reduces from 40.6% to 21.1% as the write cache size is increased from 0.125 MByte to 1 MByte. But going from 1 MByte to 2 MByte, W_{wc} reduces only by 5.9% (21.1% to 15.2%). For the other disks the flattening of the W_{wc} curve in the 1-2 MByte region is even more pronounced. We observe that R_{wc} does not decrease noticeably for the Sci-Off and Air-Rsv disks as the write cache is increased while it shows some improvement for the Off-Day disk.

This implies that the Air-Rsv and Sci-Off disk had very little "read-after-write" activity. From this data, it appears that a 1-2 MByte non-volatile write cache is adequate for all the disks presented.

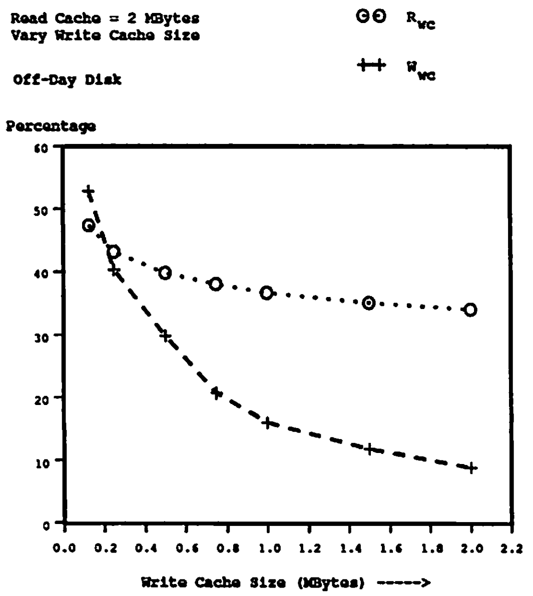

Figure 4.3: R_{wc} and W_{wc} for Off-Day Disk for varying Write Cache Sizes

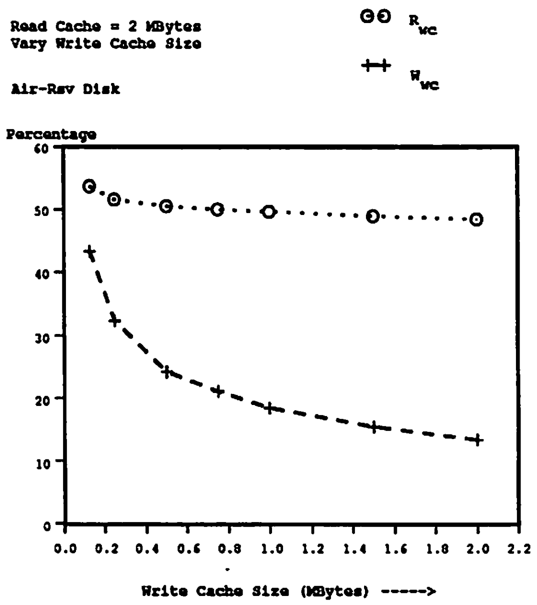

Figure 4.4: R_{wc} and W_{wc} for Air-Rsv Disk for varying Write Cache Sizes

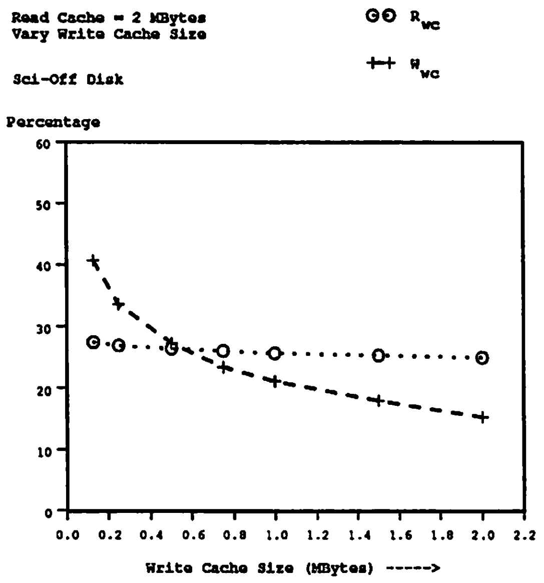

Figure 4.5: R_{wc} and W_{wc} for Sci-Off Disk for varying Write Cache Sizes

4.3 Piggybacking

Next we investigate the impact of piggybacking "dirty" blocks from the write cache on read I/O requests. Figure 4.6 shows the benefit of track based piggybacking over the 2-Threshold purging scheme (WC_HILIM=95%, WC-LOLIM=40%). The diagram also compares W_{wc} for the "full track" and "free" piggybacking schemes (explained in Section 2). The Read and Write cache sizes are once again set at 2 MBytes and 1 MByte respectively.

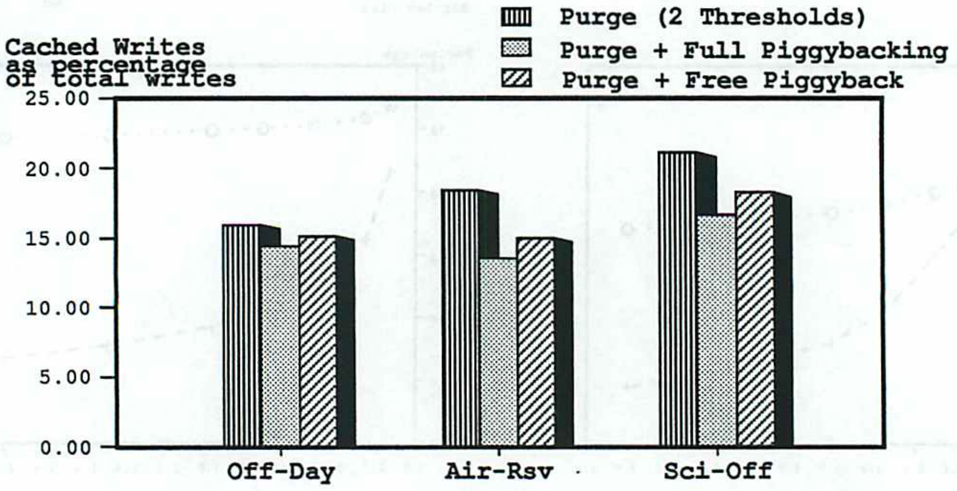

Figure 4.6: Comparison of W_{wc} for the *Track* based Piggybacking Schemes versus the 2-Threshold Purge Scheme

From Figure 4.6 we see that "full track" and "free" piggybacking policies are successful in reducing W_{wc} by about 10-25% and 5-20% respectively over the 2-Threshold purging scheme. The improvement in write I/Os achieved with track based piggybacking, as we had mentioned earlier, is a pessimistic estimate. The piggybacking of dirty blocks on the whole disk cylinder as the read miss operation, on the other hand, provides the upper bound on the reduction of write I/Os that can be achieved by the piggybacking policy. Figure 4.7 shows that dramatic reductions in write I/Os can be attained using the"full cylinder" piggybacking policy. W_{wc} is reduced by about 45-99% compared to the 2-threshold purge scheme. For the Air-Rsv database disk the full cylinder piggybacking policy is so effective that explicit write operations are almost completely eliminated! The Sci-Off disk also shows an order of magnitude improvement. Piggybacking was quite effective for the Off-Day disk as well, achieving about 45% reduction in write I/Os over the best purge-only scheme. Even "free" piggybacking is quite effective - reducing W_{wc} to only 0.6-5.6% of the total number of write I/Os in the trace, for all the three disks.

In the cylinder case, the total amount of blocks that can be purged or piggybacked in one operation can be expected to be higher. That should lead to fewer write operations for the cylinder case than the track case. In Table 4.3 we present a comparison between the "full track" and "full cylinder" piggybacking and purging policies. We observe (from the *piggyback success* value) that about 2-3 times more of the read miss operations are able to piggyback dirty blocks in the cylinder case than in the track case. Better piggybacking success coupled with more efficient purges (observe average KBytes per purge in Table 4.3) helps to reduce the number of write operations in the cylinder case (W_{wc}) to 0.1-4.4% of the original number of writes in the trace. The track based policy on the other hand, is successful in reducing W_{wc} to 14-17%. One might expect the percentage of I/Os that stall to increase in the cylinder case because the I/O service times might be greater than in the track case but it appears from Table 4.3 that the percentage of all I/Os that stall is always less for the cylinder than the track based policy. This is due to the fact that the reduction in write I/Os with the "full cylinder" piggybacking more than compensates for the slightly higher probability of stalls due to the larger I/O service times.

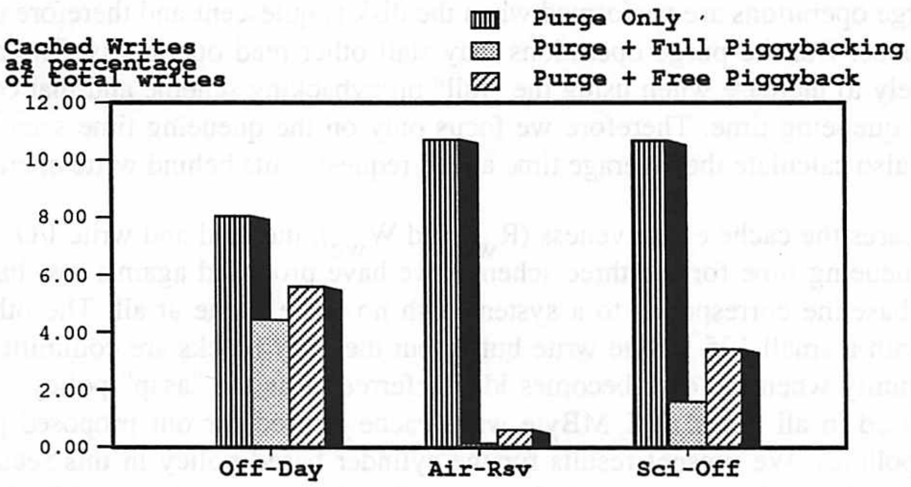

Figure 4.7: Comparison of Writes with Caching for the *Cylinder* based Piggybacking Schemes versus the 2-Threshold Purge Scheme

Disk	Policies	W_{wc}	%I/O Stalls	Piggyback success	Avg. KBytes per Purge
Off-Day	Track	14.34%	1.00%	18.52%	4.38
	Cylinder	4.41%	0.92%	34.08%	10.14
Air-Rsv	Track	13.62%	11.27%	4.93%	6.58
	Cylinder	0.10%	11.27%	10.52%	14.58
Sci-Off	Track	16.67%	3.02%	5.39%	2.58
	Cylinder	1.59%	2.79%	14.03%	7.74

Table 4.3: Comparison of R_{wc}, I/O Stalls, Piggyback success and Kbytes per Purge where a disk track or cylinder is available for write cache management

4.4 Service and Queueing Time Analysis

Until this point in the analysis we have focused on reducing the total number of write operations to disk. We mentioned that the I/O service times may be higher for some of our policies and showed that it did not have a negative effect on the percentage of I/O operations that had to queue for service. In this section we look at the impact of our proposed cache management policies on the the disk I/O service and queueing times. When a non-volatile write cache is used, the write cache purge operations are performed when the disk is quiescent and therefore these operations do not queue. But the purge operations may stall other read operations. The read service time is also likely to increase when using the "full" piggybacking scheme and that could lead to additional read queueing time. Therefore we focus only on the queueing time seen by the read operations and also calculate the average time a read request waits behind write operations.

Table 4.4 compares the cache effectiveness (R_{wc} and W_{wc}), the read and write I/O service time and read I/O queueing time for the three schemes we have proposed against two baseline policies. The first baseline corresponds to a system with no write cache at all. The other baseline policy is one with a small 125 KByte write buffer but the dirty blocks are committed to disk at the first opportunity when the disk becomes idle (referred to as the "asap" policy). A 2 MByte read cache is used in all cases. A 1 MByte write cache is used for our proposed purging and piggybacking policies. We present results for the cylinder based policy in this section because the impact on service and queueing time is expected to be more pronounced in that scenario. Data for the track based case is available in Appendix-2.

We observe from Table 4.4 that the write service time for the proposed policies is higher compared to the baseline schemes. One would expect that because effort is made to purge out the track/cylinder with the most dirty blocks. But the increased write service time does not have any negative impact on the average queueing time seen by the read I/O requests that visit the disk. In fact, it is almost always lower for our proposed schemes compared to the baseline cases because the total number of I/O operations are significantly reduced by the write cache. Even the R_{wc} statistic improves with the introduction of a write cache because of read hits in the write cache due to "read-after-write" I/O activity. Next, we notice that the read service time does not vary much across all the policies. The read service time for the "full" piggybacking scheme, as we expect, is slightly higher than the "free" piggybacking scheme. The increase in read service time for "full cylinder" piggybacking appears to significantly increase the read queueing time only for the Air-Rsv disk. The last column of Table 4.4 shows that the average amount of time a read operation spend waiting behind write operations is greatly reduced by our policies compared to the baseline policies. So we conclude that the proposed write cache purging and "free" piggybacking policies have no detrimental effect on the read queueing time.

Disk	Policies	R_{wc} %Reads with Caching	W_{wc} %Writes with Caching	Read Service Time (ms)	Write Service Time (ms)	Avg. Read Q-Time (ms)	Avg. Read Q-Time behind Write (ms)
Off-Day	Write Cache = 0	66.70%	100.00%	26.23	25.74	4.71	2.57
	Write Cache =125KB, asap	55.58%	87.51%	20.29	23.44	3.35	1.78
	Purge with 2 Thresholds	36.70%	8.05%	25.65	34.26	0.95	0.18
	Full Track Piggybacking	38.34%	4.41%	26.68	35.09	0.93	0.13
	Free Piggybacking	37.76%	5.58%	25.91	34.86	0.83	0.11
Air-Rsv	Write Cache = 0	59.71%	100.00%	27.14	21.73	134.43	13.73
	Write Cache =125KB, asap	57.63%	89.62%	26.71	22.84	45.42	1.14
	Purge with 2 Thresholds	49.40%	10.71%	25.25	29.03	32.70	0.17
	Full Track Piggybacking	50.02%	0.10%	27.37	26.60	69.29	0.00
	Free Piggybacking	49.97%	0.61%	25.23	26.74	38.89	0.00
Sci-Off	Write Cache = 0	33.04%	100.00%	21.27	21.99	3.64	1.11
	Write Cache =125KB, asap	29.20%	88.80%	20.29	23.44	2.98	0.84
	Purge with 2 Thresholds	25.66%	10.65%	18.69	25.10	1.90	0.12
	Full Track Piggybacking	25.96%	1.59%	19.02	27.86	1.82	0.02
	Free Piggybacking	25.90%	3.42%	18.72	26.18	1.74	0.05

Table 4.4: Comparison of service and queueing time statistics for *Cylinder* based case; Read Cache = 2 Mbytes; Write Cache = 1 Mbytes (unless otherwise noted)

4.5 Write-allocate vs. Write-purge

Since we have focused on the caching of writes, we felt it important to analyze the different policies in which writes affect the read cache. Table 4.5 compared two such policies - the write allocate and write purge in the context of the full track piggybacking scheme. Recall that in the write allocate scheme a copy of a block written is also kept in the read cache while in the write purge scheme a block that is updated is removed from the read cache. If the probability of reading a disk block shortly after it has been updated is small, then in the write allocate scheme, these blocks will waste read cache space. The penalty will also not be very high if the system is operating at a point where the marginal benefit of increasing the read cache size is small.

For this comparison, the read and write cache sizes are 2 Mbytes and 1 Mbyte respectively. We observe that the difference between the two schemes appears to be small; however the write purge scheme is recommended for all environments except for storage devices with a large amount of read-after-write activity.

Disk	Policies	R_{wc} %Reads w/ Caching	W_{wc} %Writes w/ Caching	I/O Stalls	Piggyback success
Off-Day	Write Allocate	38.32%	14.34%	1.00%	15.46%
	Write Purge	37.51%	14.38%	1.09%	18.52%
Air-Rsv	Write Allocate	51.30%	13.55%	12.23%	4.61%
	Write Purge	49.89%	13.62%	11.27%	4.93%
Sci-Off	Write Allocate	26.89%	16.59%	3.33%	5.07%
	Write Purge	25.80%	16.67%	3.02%	5.39%

Table 4.5: Comparison of Write Allocate versus Write Purge read cache management policies

4.6 Multi-Disk Environment

In this set of experiments we examine how our write caching scheme works when the cache is located in a disk controller serving multiple disks. We study two variations. The first has common read and write caches for all the disks. In the second case the caches are strictly (equally) partitioned between all the disks. We keep the read to write cache size ratio fixed at 4:1. WC-HILIM and WC-LOLIM is kept constant at 95% and 40% respectively. The full cylinder purging and piggybacking policy is used. The results for varying the cache size is presented in Table 4.6

and Figure 4.8. In this example, four of the most heavily used disks from the Off-Day environment were chosen. These included the system disk, an application code disk, and two private user data disks. In Figure 4.8 we observe that the total number of I/O operations is always lower by about 16-19% in the common cache case than the strictly partitioned cache case. Table 4.6 shows that a common cache shared by the four disks is better than the strictly partitioned cache for all the metrics shown (in fact, for all metrics analyzed).

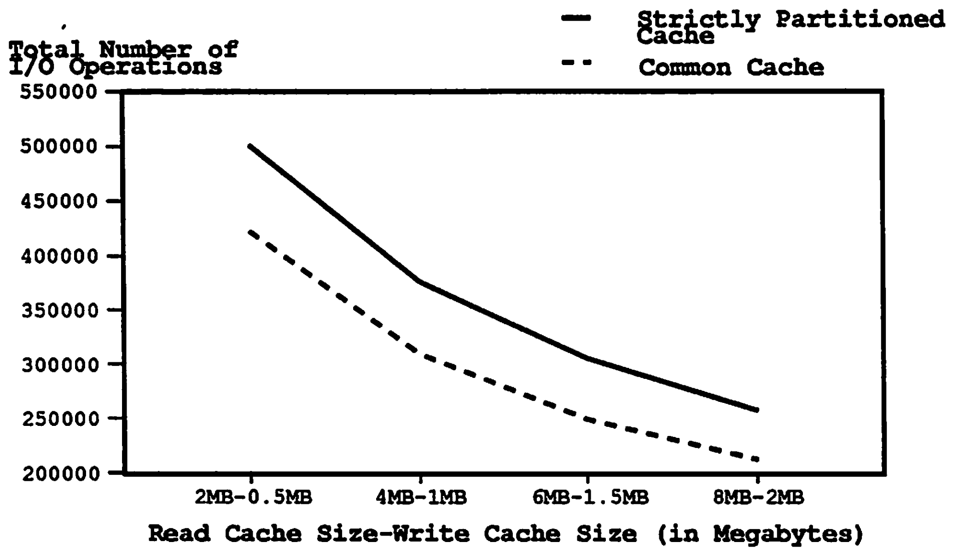

Table 4.6: Comparison of Writes with Caching, I/O Stalls and Piggyback success for a Common Cache versus a Strictly Partitioned Cache in a multi-disk environment

Cache Size (Mega Bytes)	Policies	Writes with Caching	%I/O Stalls	Piggyback success
Read cache = 2 MB Write cache = 0.5 MB	Common Cache	2.13%	2.91%	6.40%
	Strictly Partitioned	3.20%	4.42%	5.15%
Read cache = 4 MB Write Cache = 1 MB	Common Cache	1.17%	1.55%	8.51%
	Strictly Partitioned	2.06%	2.55%	6.40%
Read Cache = 6 MB Write Cache = 1.5 MB	Common Cache	0.74%	1.00%	9.77%
	Strictly Partitioned	1.52%	1.72%	7.40%
Read Cache = 8 MB Write Cache = 2 MB	Common Cache	0.53%	0.74%	10.87%
	Strictly Partitioned	1.07%	1.25%	8.13%

Figure 4.8: Total number of I/O operations for 4 disks sharing a Common Cache versus a Strictly Partitioned Cache

4.7 Single non-volatile cache

So far, all of the caching schemes have contained two caches - a volatile read cache and a non-volatile write cache. It was shown that a small, non-volatile write cache together with a read cache improves performance significantly. However, in the separate cache environment the size of the two caches cannot be dynamically varied and therefore they have to be conservatively sized to work well in different environments. In the separate cache design, there are situations when disk blocks need to be copied between the two caches and that could limit system performance. Technology trends show that the standby power required by DRAMS is decreasing rapidly [4] and therefore it is important to quantify the performance of a simpler design that uses a single large non-volatile cache for both reads and writes. The single NV-cache scheme is also used as a baseline against which other schemes are compared.

In the single, non-volatile cache environment the system could use the traditional write-back scheme where a "dirty" block is written out to disk only when it is replaced by another block being brought into the cache. In that case, a cache insert operation that causes a "dirty" block to be replaced will stall for the duration of an I/O operation. Such a stall may be unacceptable. We therefore devise a scheme that attempts to avoid a stall when a new item has to be inserted without requiring any additional hardware support for the disk cache.

All the cache blocks are maintained in a least recently used (LRU) chain with hashed access for efficient searching. When the cache is full and the LRU element in the cache is a "dirty" block, the insertion of a new block results in the LRU "dirty" block being simply marked as a "target" for purging. The LRU "clean" block is replaced to make room for the incoming block. Thereafter, at the first opportunity when the disk is detected to be quiescent, all dirty blocks on the cylinder containing the "target" block are written out to disk and are marked "clean" in the cache. If it so happens that the cache is completely depleted of "clean" blocks then an "immediate purge" to the cylinder containing the least recently used dirty block is done. To avoid immediate purges, piggybacking of dirty blocks on read misses from the cache and the threshold-based quiescent purging (as described for the separate cache case) mechanisms are also used.

A single set of values for the high and low limit thresholds for quiescent purging may not be appropriate across different environments. The ideal scenario would be one where the threshold values are dynamically determined from the characteristics of the workload and are adjusted periodically. Such a policy should be able to adapt to changes in the workload pattern and perform well uniformly. We experimented with several different heuristics and found that the high limit threshold value determined by the ratio of the number of disk block written to the total number of disk blocks accessed for read and write to be a good choice. The system keeps track of the number of blocks read and written for a certain interval and uses that to calculates the high limit threshold value for the next period. The low limit threshold is calculated as a fixed fraction (0.4 in our simulations) of the high limit threshold.

$$\textit{High Limit Threshold} = \frac{\textit{Number of Blocks Written}}{\textit{Number of Blocks Read} + \textit{Number of Blocks Written}} * 100\%$$

Note that this formula is applicable only in the single NV-cache case. Table 4.7 compares the performance of the dynamic threshold policy against the best results obtained with fixed threshold values for a single, 2.5 MByte cache. In the dynamic scheme the threshold values were adjusted every 10,000 I/O operations. The results of the dynamic policy did not vary significantly when the threshold adjustment interval was varied in the 5,000-25,000 I/O range. It is observed that the dynamic policy is almost always as good as the best achieved with fixed thresholds. This is a satisfactory result in favor of the dynamic threshold scheme because it will usually not be possible, in practice, to predetermine the best threshold values for different workload environments.

Table 4.7 also compared this single non-volatile cache case against the separate read and write cache scheme. The data presented for the separate cache configuration represents the results obtained when the same 2.5 MBytes of memory is statically partitioned in the best possible way between the two caches for each workload. We observe that the performance of the separate cache is close to the single cache in two out of the three cases. For the Off-Day disk, the total number of I/O operations with a single cache is about 10% lower than the separate cache case. Results in Sections 4.1-4.3 showed that a small amount of non-volatile write cache is effective in reducing the total number of I/O operations without increasing the read queueing time. Therefore, having a small and separate non-volatile write cache could result in substantial cost savings without a large performance penalty.

Disk	Policies	W_{wc} %Writes with Caching	R_{wc} %Reads with Caching	Total # I/Os to disk
Off-Day	Single Cache Fixed Thresholds	1.87%	39.07%	24556
	Single Cache Dynamic Thresholds	1.93%	38.95%	24539
	Separate Read and Write Caches	2.39%	42.90%	27272
Air-Rsv	Single Cache Fixed Thresholds	3.51%	48.92%	162555
	Single Cache Dynamic Thresholds	4.43%	48.93%	163330
	Separate Read and Write Caches	2.19%	50.63%	167084
Sci-Off	Single Cache Fixed Thresholds	4.49%	26.25%	116627
	Single Cache Dynamic Thresholds	8.81%	25.77%	117583
	Separate Read and Write Caches	6.78%	26.42%	118994

Table 4.7: Comparison of Fixed vs. Dynamic Threshold for the Single Cache and Comparison of Single Cache vs. Separate Read and Write Cache

5 CONCLUSIONS

In this paper we analyzed the performance of several mechanisms for caching writes to disks. With the current trend of having much faster processors, but comparatively slower disk subsystems, such mechanisms are increasingly important for removing the I/O bottleneck. We showed that having a relatively small amount of NV-RAM for a write cache is very effective in reducing the number of I/O writes to the disk and still allows for a write behind strategy for the write cache to ensure consistency and crash recovery. Our analysis was based on a trace driven simulation of the disk subsystem. The traces used were taken from three different commercial production sites of VAX/VMS time-sharing clusters and each trace comprised several hundred thousand I/Os. We presented results for one disk from each of these environments to examine the effectiveness of the write cache in conjunction with a separate read cache. Data for six other disks were summarized in the appendix.

We found that a simple purging policy (write-behind) triggered by a high limit threshold in the 90-95% range for the write cache was effective in reducing the write I/Os to the disk to about 40-50% of the number of write operations without such a write cache. We then showed that a hysteresis policy for purging the write cache reduces the number of I/Os even further, to about 15-20% of the total writes without a write cache. A properly chosen hysteresis policy thus reduces the number of writes to less than half the number of I/Os with a single threshold policy. The high limit of dirty blocks in the write cache for the hysteresis policy is in the range of 95-99% of the total number of blocks in the write cache and the low limit at which point the purging of the write cache ceases is in the range of 20-60%. It was heartening to note that the performance variation is relatively flat for quite a wide range of threshold values, which helps in designing a policy to be suitable for a variety of environments.

Next we looked at the effect of varying the write cache size on the number of I/Os that access the disk. We observed that the number of write I/Os decrease as the write cache size is increased but the point of diminishing returns is reached for write cache sizes beyond 1 Mbyte. We also noticed that the read miss I/Os do not decrease noticeably implying that there isn't a lot of "read-after-write" activity. We conclude that a non-volatile write cache of about 1 Mbyte is adequate for all the disks presented.

We then incorporated a mechanism to "piggyback" writes of dirty blocks from the write cache along with read misses from the read cache. In this manner, the read and write cache complement each other. We first examined the piggybacking of dirty blocks in the write cache which belong in the same track as the blocks of the read miss. In comparison to the purging with hysteresis scheme, the addition of the "full track" piggybacking further reduces the number of writes to disk by about 10-25%. The "full cylinder" piggybacking provides the upper bound on the reduction in write I/Os that can be achieved using piggybacking. Cylinder based "full" piggybacking produced dramatic improvements. In fact, it almost completely eliminated the the write I/Os for the Air-Rsv disk.

Piggybacking writes of dirty blocks for an entire cylinder potentially increases the amount of read stalls in the environment, which we consider to be undesirable. To alleviate this problem, if any, we investigated a mechanism which we call "free" piggybacking. This policy allows the piggybacking of dirty block writes only to the range of the blocks that are between the current head position and the blocks that have to be read due to the read miss. This policy does not increase the read service time and thus there is no potential increase in the amount of read I/O stalls due to piggybacking. We found that "free" piggybacking helps in reducing the amount of read stalls, especially for heavily loaded disks, but does generate more write I/Os in all the cases compared to "full cylinder" piggybacking.

The percentage of I/O operations that queue for service does not provide the complete picture. So we then looked at the impact of our proposed policies on the I/O service and queueing times. The write cache purge operations are performed when the disk is idle so these operations do not queue but they may stall other read I/Os. Further, the write service time increases when our purging policy is used. But this does not adversely affect the read queueing time because the total number of I/O operations are significantly reduced.

We also looked at the interaction between the read and write cache. We compared two ways of dealing with the write operations in the read cache - the write allocate and the write purge policies. We found that the write purge policy is marginally better than the write allocate scheme except in environments with a substantial amount of read-after-write operations on disk blocks.

We further considered the case of using a single non-volatile disk cache. The single cache has the advantage that the portion of the cache allocated for reads or writes is not static and it was therefore expected to provide a baseline against which the separate caching scheme could be compared. We observed that the difference in performance between the single and separate cache configurations is small in most cases. We therefore believe that a separate and small non-volatile write cache would be the appropriate solution in most environments. Such an approach results in substantial cost savings without sacrificing performance.

We plan to continue this work in several ways. We want to improve the disk model to include sophisticated disk arm movement optimizations [5, 13]. We need to further explore the issues of performance and fairness of multi-disk configurations. We are also working on understanding the issues around extending some of the proposed mechanisms to the distributed systems environment.

6 REFERENCES

1. Baker, M., et al, "*No-Volatile Memory for Fast, Reliable File Systems,*" To appear in the Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V), October, 1992.
2. Biswas, P., Ramakrishnan, K.K., "*File Access Characterization of VAX/VMS Environments,*" Proceedings of the 10th International Conference on Distributed Computing Systems, May 1990.
3. Chen, S., Towsley, D., "*A Queueing Analysis of RAID Architectures,*" Tech. Report 91-71, Department of Computer Science, University of Massachusetts, Amherst, MA, September 1991.
4. Copeland, G., et al, "*The Case For Safe RAM,*" Proceedings of the 15th International Conference on Very Large Data Bases, August 1989.
5. King, R.P., "*Disk Arm Movement in Anticipation of Future Requests,*" ACM Transactions on Computer Systems 8(3), August 1990.
6. Menon, J., Hartung, M., "*The IBM 3990 Disk Cache,*" Proceedings of the IEEE Computer Society International COMPCON Conference, 1988.
7. Nelson, M.N., et al, "*Caching in the Sprite Network File System,*" ACM Transactions on Computer Systems, February, 1988.
8. Patterson, D.A., Gibson, G., Katz, R.H., "*A Case for Redundant Array of Inexpensive Disks (RAID),*" Proceedings of the 1988 ACM SIGMOD International Conference on the Management of Data, June 1988.
9. Ramakrishnan, K.K., et al, "*Analysis of File I/O Traces in Commercial Computing Environments,*" Proceedings of the 1992 ACM SIGMETRICS and PERFORMANCE '92 International Conference on Measurement and Modeling of Computer Systems, June 1992.
10. Reddy, A.L., Banerjee, P., "*An Evaluation of Multiple-Disk I/O Systems,*" IEEE Transactions on Computers, Vol. 38, No. 12, 1989.
11. Rosemblum, M., Ousterhout, J.K., "*The Design and Implementation of a Log-Structured File System*", Proceedings of the 13th Symposium on Operating Systems Principles (SOSP), October, 1991.
12. Salem, K., Garcia-Molina, H., "*Disk Striping,*" Proceedings of the 2nd International Conference on Data Engineering, 1986.
13. Seltzer, M., et al, "*Disk Scheduling Revisited*", Proceedings of the Winter 1990 USENIX Conference, January, 1990.
14. Smith, A.J., "*On the Effectiveness of Buffered and Multiple Arm Disks,*" Proceeding of the 5th Computer Architecture Symposium, April, 1987.
15. Smith, A.J., "*Disk Cache - Miss Ratio Analysis and Design Considerations,*"ACM Transactions on Computer Systems 3, August 1985.
16. Solworth, J.A., Orji, C.U., "*Write-Only Disk Caches,*" Proceedings of the 1990 ACM SIGMOD International Conference on the Management of Data, May 1990.

APPENDIX -1

Disks	Total # Reads	Total # Writes	R_{wc} Reads with Caching	W_{wc} Writes with Caching	% IO Stall	% Piggybck Success	Avg. KBytes per Piggybck
Off-Day System Disk	330916	12868	12.38%	3.51%	1.13%	0.74%	2.20
Off-Day Application	436041	45719	29.30%	1.58%	4.85%	2.80%	1.55
Air-Rsv System Disk	143272	37065	26.66%	9.62%	4.13%	8.73%	2.43
Air-Rsv Database	251095	67546	31.39%	3.44%	4.21%	12.54%	5.01
Sci-Off System Disk	812034	131906	12.09%	2.77%	1.80%	12.15%	1.77
Sci-Off Database	576272	61792	76.46%	55.55%	23.71%	3.22%	2.28

Table A1: Comparison of major statistics for 6 disks. *Track* based case, Read Cache = 2 MBytes; Write Cache = 1 Mbytes; WC-HILIM=95%, WC-LOLIM=40%

Disks	Total # Reads	Total # Writes	R_{wc} Reads with Caching	W_{wc} Writes with Caching	% IO Stall	% Piggybck Success	Avg. KBytes per Piggybck
Off-Day System Disk	330916	12868	12.62%	1.56%	1.14%	2.07%	5.48
Off-Day Application	436041	45719	29.87%	0.89%	5.01%	3.88%	2.37
Air-Rsv System Disk	143272	37065	27.29%	4.60%	4.05%	15.19%	5.92
Air-Rsv Database	251095	67546	33.53%	0.20%	4.45%	18.61%	5.81
Sci-Off System Disk	812034	131906	12.58%	1.06%	1.83%	16.99%	2.56
Sci-Off Database	576272	61792	76.48%	16.41%	21.97%	8.62%	1.81

Table A2: Comparison of major statistics for 6 disks. *Cylinder* case, Read Cache = 2 MBytes; Write Cache = 1 Mbytes; WC-HILIM=95%, WC-LOLIM=40%

APPENDIX 2

Disk	Policies	R_{wc} Reads with Caching	W_{wc} Writes with Caching	Read Service Time (ms)	Write Service Time (ms)	Avg. Read Q-Time (ms)	Avg. Read Q-Time behind Write(ms)
Off-Day	Write Cache = 0	66.70%	100.00%	26.23	25.74	4.71	2.57
	Write Cache =125KB, asap	55.59%	94.72%	26.53	25.22	3.44	1.87
	Purge with 2 Thresholds	36.73%	15.96%	25.80	28.42	1.15	0.23
	Full Track Piggybacking	37.51%	14.34%	26.72	28.26	1.01	0.21
	Free Piggybacking	37.31%	15.15%	25.83	28.45	1.02	0.25
Air-Rsv	Write Cache = 0	59.71%	100.00%	27.14	21.73	134.43	13.73
	Write Cache =125KB, asap	57.65%	95.65%	26.75	22.34	45.44	1.21
	Purge with 2 Thresholds	49.61%	18.44%	25.29	27.10	35.20	0.22
	Full Track Piggybacking	49.89%	13.62%	27.51	26.27	67.45	0.17
	Free Piggybacking	49.81%	14.97%	25.31	26.98	35.99	0.21
Sci-Off	Write Cache = 0	33.04%	100.00%	21.27	21.99	3.64	1.11
	Write Cache =125KB, asap	29.21%	91.58%	20.28	23.13	2.96	0.84
	Purge with 2 Thresholds	25.69%	21.09%	18.72	21.13	1.95	0.18
	Full Track Piggybacking	25.80%	16.67%	19.14	20.92	2.00	0.13
	Free Piggybacking	25.77%	18.24%	18.74	21.11	1.92	0.15

Table A3: Comparison of service and queueing time statistics for *Track* based case, Read Cache = 2 Mbytes; Write Cache = 1 Mbytes when not mentioned.