Write-Only Disk Caches

Jon A Solworth
University of Illinois at Chicago

Cyril U Orji
University of Illinois and Argonne Natl Lab

Abstract

With recent declines in the cost of semiconductor memory and the increasing need for high performance I/O disk systems, it makes sense to consider the design of large caches In this paper, we consider the effect of caching writes We show that cache sizes in the range of a few percent allow writes to be performed at negligible or no cost and independently of locality considerations

1 Introduction

In this paper we examine Write-Only Disk Caches (WODC), that is caches whose main purpose is to hold blocks to be written to disk Unlike traditional caching which reduces the cost of reads, the primary purpose of WODC is to reduce the cost of writes With sufficiently large caches, the CPU never needs to wait until the disk block is written, instead the block is cached and then written asynchronously with the CPU process The asynchrony enables writes to be logically performed with virtually no latency More importantly, the service time for performing physical writes can be reduced by combining multiple disk accesses into a single access This is in contrast to reads which should be executed as soon as possible to reduce program delays

Unlike more traditional caches, Write-Only Disk Caches require neither spatial nor temporal locality – and in fact, we do not assume any locality properties in most of this paper Instead, WODCs use the excess bandwidth during read operations to perform writes Since, in a traditional UNIX[1] system, only 3% of disk bandwidth is used, there is bandwidth available to support write requests Hence, write requests will be piggy-backed onto read requests, at little or no cost

Because WODC can perform updates in place, and can mix reads and writes effectively, it is efficient even under conditions when log-based files [OD89, FC87] are inappropriate, such as on line transaction processing system (OLTP)

However, deferring disk accesses has reliability implications If the system fails before the disk is written, the file system can be left in an inconsistent state Hence, we assume that the disk cache is protected against power failure by battery backup In a future paper, we will discuss techniques which are reliable even in the presence of faulty memory system

It would seem that WODCs would not have a large payoff for random accesses and reasonable sized disks However, a typical disk has hundreds or thousands of blocks per cylinder Since a random read requires a rotational latency of one-half a track, one-half the blocks on a cylinder are "passed" on a single read location If the cache size is in the order of 1%, this means that on average a few to tens of blocks can be written per read

In this paper we use simulations and analysis to measure and predict the performance of WODCs We show the

1 cost of performing writes based on disk cache size, read/write percentage and disk geometry,

2 size of cache (or percentage) for performing writes at no cost,

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

No of Cylinders	1000
Blocks per Track	80
Average Latency	8 33 ms
Average Seek	13 50 ms

Table 1 **Disk Parameters**

3 effect of purges on write service time, and

4 effect of WODCs on read response time

The rest of the paper is organized as follows Section 2 presents the simulation model In Section 3 the results of our simulation and analysis are given Finally, our conclusions are presented in Section 4

2 The Simulation Model

Our initial model assumes a single recording surface Although almost all disks have multiple recording surfaces, the single-surface model can be used in two multi-surface disk cases The first is when all surfaces can be simultaneously read/written, the second is when the cache is small enough that the probability of sector collision is small

Subsequently, the single-surface model is generalized to a multiple-surface model, corresponding to current disks with large caches

2.1 Disk Parameters

We summarize some of the disk parameters in Table 1 The parameters in this table are typical for a large range of large $5\frac{1}{2}$ inch disk drives and even older drives such as the Fujitsu Eagle Disk access time consists of three primary components, *seek time, rotational latency* and *transfer time*

The seek time is time required for the disk head assembly to move to the appropriate cylinder that holds the requested data In modern systems, disk arm actuators have non-constant speed [STH83] For voice coil actuators, the seek time is given by a non-linear function

$$T(d) = a + b\sqrt{d} \qquad (1)$$

where a is the mechanical setting time, b is a constant associated with the speed of the actuator and the track density on the magnetic media and d is the number of cylinders seeked We use the parameter values 5 0 ms for a and 0 5 ms for b suggested in [BG88], which are typical of a large range of SMD and SCSI disk drives

The rotational latency is the average time taken for the requested data to be positioned under the read/write head The disk in our model makes 3600 revolutions per minute, hence full track rotational latency is the time for one disk rotation or 16 67 ms The average rotational latency is one half of the time for a disk rotation or 8 33 ms

The transfer time is the time for the requested data to be read from or written to the disk surface In our simulation, this is the same as the time for the disk to move over the data area

2.2 Workload Parameters

The workload is characterized by the types of requests, their frequency and arrival pattern, and the size of blocks to access I/O requests are externally triggered and arrive at the scheduler in a FIFO order

The request characteristics are based on the measurements carried out on the UNIX file system [Ous85] and on the studies done in [RB89] In these studies, the I/O request arrival is assumed to be a Poisson process In the simulation we make the same Poisson assumption and specify the mean time between arrivals to vary between 10 ms and 100 ms Two classes of requests are serviced, *read* and *write* requests, and representative mixes of these requests are studied at 0% read, 10% read, 30% read, 50% read, 70% read and 100% read

Most on-line transaction processing (OLTP) systems use about same retrievals (reads) as updates (writes) In UNIX systems, about two-thirds of all I/O requests in computer systems involve reads while the remaining one-third involve writes [NWO88, Ous85]

As the disk caches become larger, reads are increasingly satisfied by the cache, but for reliability, writes must be written through to disk Hence, the trend is that disk traffic becomes increasingly write intensive, and in the extreme all disk traffic becomes write traffic as in main memory database systems

The initial sets of requests used in the experiments are uniformly distributed over the number of tracks in the disk Later, we generalize to non-uniformly distributed requests In particular,

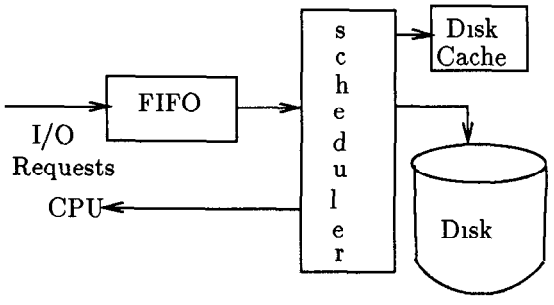

Figure 1 **I/O System Model**

to measure locality effects, we simulate a scenario where 80% of the requests are from 20% of the disk tracks

In each run of the experiment 200,000 requests are serviced and steady state operation is simulated by discarding data for the first 20,000 requests in each run

2.3 System Model

The I/O system is modeled as a single-queue, single-server queuing system with an events scheduler as shown in Figure 1 The system contains a main memory disk cache If possible the request is satisfied from the disk cache If not, the request may be forwarded to the disk The disk controller keeps status information on the disk and carries out the disk scheduling algorithm

2 3.1 Caching Parameters

The caching scheme is characterized by the cache size expressed as a fraction of the disk capacity For example, a 1% cache for a 300 megabyte disk means a 3 megabyte cache Since we assume disk block access is random and independent of block size, block size is independent variable for purposes of this study, but would be an important factor when considering accesses with a sequential component

2 3 2 The Scheduling Algorithms

A write request logically writes the disk but physically the write is deferred If the block is in cache it is updated – the previously deferred write is overwritten and hence eliminated Otherwise a cache entry is created for the block, either by recycling a *clean* block or if necessary, purging A *clean* block in the cache is one that has been written to disk and can be overwritten without a need for an explicit purge operation A *purge* writes out dirty blocks, and hence converts them to clean blocks in the cache

A read request can be satisfied from the cache if the block is cache resident, if not, the block is physically fetched from the disk However, in our simulation such blocks are not cached

If a disk access is required to satisfy a read request, cache-resident dirty blocks in the same cylinder as the read request are piggy-backed onto the read request, adding little or no cost to the read access and reducing the write cost to negligible amounts After the disk head has seeked to the appropriate cylinder, it waits for the desired block to be positioned under the read/write head During this rotational latency, some blocks may pass under the read/write head before the desired block is properly positioned If there are dirty blocks in cache that map to the addresses of these blocks, they can be written (*piggy-backed*) during the rotational delay

Two piggy-back schemes were studied, *zero-cost* and *track-based* piggy-backing An example is shown in Figure 2 The block requested is at B Suppose that the disk head seeks to the appropriate track and lands at location A As the disk rotates clockwise, during a complete revolution of the disk starting at A, the disk blocks number 1, 2, 3, 4 and 5 are passed Assume blocks 1 – 5 are dirty in cache In zero-cost piggy-back, the blocks at addresses 1 and 2 will be piggy-backed because they can be written during the rotational latency from A to B In track-based piggy-back, blocks 3, 4 and 5 will also be piggy-backed However, these incur rotation costs after block A is read

Many purge selection schemes such as least recently used (LRU), have been described in the literature However, since we assume no locality, we selected a replacement algorithm that will yield the minimum write cost per block As an approximation, the track with the maximum number of blocks in cache is always purged and all cache blocks that map to this track are written out to disk This amortizes the seek and rotation costs over the number of blocks written Read piggy-backing is one way of performing writes – the other way is purging

When blocks are written out to disk, the blocks are marked clean, but the cache spaces are not freed up By leaving them in the cache, they can satisfy

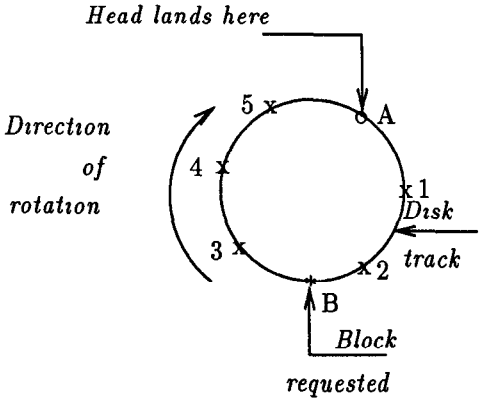

Figure 2 **Sample Block Layout on a Track**

read requests before being overwritten thereby improving the cache hit ratio for reads

Studies for UNIX system [Ous85, OD89, Sat81] indicate that over 40% of all newly created files are deleted within 30 seconds of creation Since blocks tend to stay in the cache for a long period of time, many blocks would be deleted Because we do not consider this effect, WODC would be even more effective than shown here since this effect will reduce the number of writes

3 Results

Two of the primary metrics for evaluating the impact of write-only disk cache in system performance are the service time of write requests and response time of read requests The extent of the effects depends on a number of the simulation parameters

- the piggy-back scheme in effect,
- the mixture of read and write requests,
- the arrival rate of requests,
- the number of blocks per track, and
- the cache size

Piggy-backing provides locality for writes A number of blocks are written together and the cost amortized over the blocks written We shall show conditions under which each of the two piggy-backing schemes is more effective than the other

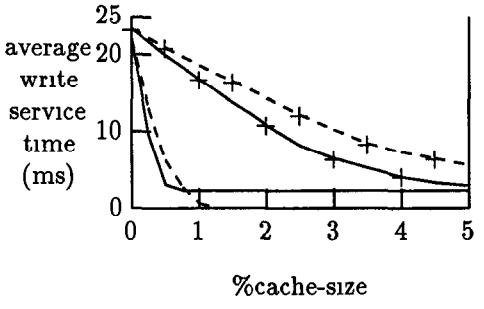

Figure 3 **Write service time vs relative cache size (70% read/30% write, 30 arrivals/sec)**

3.1 Write Service Time

In this simulation, only write requests are cached (although both reads and writes can be satisfied by the cache) In real systems, cached entries would include both read and write requests, but since we assumed no locality, caching reads would have little effect and so was not modeled However, given an appropriate locality model for reads, the formulas derived here can be used to statically or dynamically partition the cache into read and write sections In Section 3 2 we show how the average response time of reads will be affected when read operations are introduced

In figure 3 the average write service time is plotted as a function of the cache size for two different block sizes The solid line curves represent values for track-based piggy-backing while the dashed line curves represent values for zero-cost piggy-backing The curves with "+" represent values obtained with 10 blocks per track The other curves represent values for 80 blocks per track

Two trends are obvious from the graphs

- Track-based piggy-backing is more effective at small cache sizes As the cache size is increased, the extra half rotation of track-based piggy-backing becomes an overhead
- The larger the number of blocks per track, the greater the reduction in average write service time

Although purges may be performed asynchronously during otherwise idle time for the disk, write service time is used to compute the amount of disk bandwidth necessary for writes Read response time, on the other hand, does include time for the purges to complete If purges can be com-

pleted in the background, then read response time is not affected at all by zero cost piggybacking, and only a small amount by full track piggybacking

We shall show that under certain mixes of system and disk parameters, with zero-cost piggy-backing, write operations can be performed at no cost and without the need to ever perform a purge operation Assuming a uniform distribution of requests, we derive simple analytic models in terms of these parameters to show how the zero-cost writes can be achieved

We define the following

- t rotation time of disk,
- $\bar{t}$ average rotational latency of disk,
- b number of blocks per cylinder,
- c ratio of cache size to disk size,
- r read ratio,
- w write ratio,

where $(0 \leq r, w, c \leq 1)$

At steady state, when all writes are piggybacked, the number of *writes* per *read* can be given as

$$\frac{w}{r} = \frac{\bar{t}}{t} \times b \times c \qquad (2)$$

where $\bar{t}/t$ is the average proportion of blocks in a track that pass under the read/write head during the rotational latency each time there is a disk access This is the proportion of blocks in a track that can be written at no cost during every read request that requires a disk access The product bc is the expected number of blocks per track that is cache resident at any time The terms on the right hand side of equation (2) represent the number of cache blocks written per read request, those on the left the expected number of blocks passed over performing a read

If we let $\bar{t}/t = k$, we can rewrite equation (2) as

$$\frac{w}{r} = k \times b \times c \qquad (3)$$

where $k = 1$ for track-based piggy-backing and $k =$ 0 5 for zero-cost piggy-backing Rewriting w as *1 - r* and rearranging terms, equations can be derived from equation (3) either in terms of the read ratio or cache size,

$$r = \frac{1}{1 + kbc} \qquad (4)$$

fixed cache size (%c)	*%read (r)*	
	analytic value	*simulation value*
1	71 43	75
2	55 57	60
5	33 33	35
10	20 00	22

Table 2 **Some model and simulation values for zero-cost piggy-backing (t = 16.67 ms, $\bar{t}$ = 8 33 ms, b = 80)**

fixed read ratio (%r)	*cache size (%c)*	
	analytic value	*simulation value*
10	22 50	25 00
30	5 83	7 00
50	2 50	3 00
70	1 07	1 50
90	0 00	0 50

Table 3 **Some model and simulation values for zero-cost piggy-backing (t = 16 67 ms, $\bar{t}$ = 8 33 ms, b = 80)**

and

$$c = \frac{1 - r}{kbr} \qquad (5)$$

In equation (4) it is assumed that the system characteristics in terms of the disk parameters (t, $\bar{t}$, and b) and the relative cache size (c) are known It is then easy using the equation to design a proper work load in terms of the read/write ratio (r, w) that will guarantee that all write blocks are written at no cost (zero-cost piggy-backed)

From equation (5) we can determine the relative cache size (c) that will guarantee a zero-cost piggybacking operation assuming we know our work load characteristics in terms of the read/write ratio (r, w) and the disk parameters (t, $\bar{t}$, and b)

In table 2 we tabulate some values to show the analytic and simulation values of read ratio that guarantee zero-cost piggy-backing for a given cache size As an example, in the first row of the table, with a 1% cache, the analytic model forecasts a 71 43% read ratio to achieve zero-cost piggybacking while we obtain 75% by simulation The values in table 3 can be analogously described In the first row of the table, we obtain analytically that a 22 5% cache is required for zero-cost piggybacking for a 10% read ratio, while we obtain a 25%

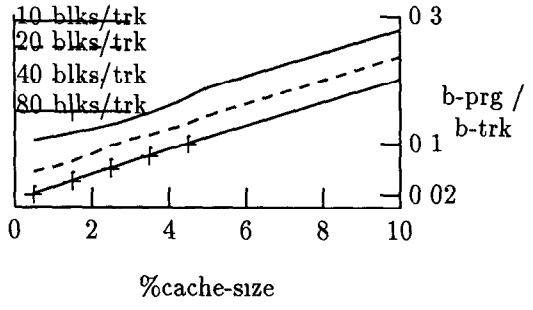

Figure 4 **%cache size vs blks per prg/blks per trk (zero-cost piggy-back, 0% read/100% write)**

cache value by simulation The other values in the tables compare very strongly with their simulation counterparts (The slight fluctuations are due to the fact that we are using expected values)

The average number of blocks purged per purge operation with respect to the cache size is shown in figure 4 The average number of blocks per purge shown on the y axis is normalized with respect to the number of blocks per track The relative cache size is given on the x axis

Two points should be noted from figure 4 The first is the linear nature of the curves, and the second is the fact that all the curves have approximately the same slope We can therefore model the curves using a straight line, at least for the range of cache sizes between 1 and 10 percent If we define

- b number of blocks per track,
- $\bar{b}$ average number of blocks per purge,
- c relative cache size,

then the general model for the curves becomes

$$\frac{\bar{b}}{b} = mc + h \tag{6}$$

Substituting for $\bar{b}$, b and c, we derive two equations to solve for the slope m and intercept h

Observe from the curves that as the relative cache size tends to zero, the average number of blocks per purge ($\bar{b}$) tends to one In other words, with no cache we need to perform a purge operation for every write Therefore, since $h = \bar{b}/b$, as $c \rightarrow 0$

$$\bar{b} = 2cb + 1 \tag{7}$$

	equation data	*simulation data*
relative cache-size (%)	*average blks-prg*	*average blks-prg*
1	2 60	2 48
2	4 20	4 13
3	5 80	5 67
4	7 40	7 25
5	9 00	8 79

Table 4 **Some model and simulation values (b = 80)**

Equation (7) was derived assuming an all write environment However, as reads are introduced, the estimate becomes conservative since more blocks are written per purge operation further reducing the cost per write

If $2cb << 1$, $\bar{b} \approx 1$, but if $2cb >> 1$, then $\bar{b} \approx 2cb$ In table 4 we show how the values of the average number of blocks per purge (*av-blks-prg*) obtained from equation 7 compare with the corresponding simulation values for various cache sizes

Consider the case when all blocks in the cache are dirty and there is need for an explicit purge operation In other words, this is the case when all blocks are not piggy-backed From equation (4) this is when

$$r < \frac{1}{1 + kbc}, \tag{8}$$

then the total number of writes w is given by

$$w = rkbc + w_{prg} \tag{9}$$

where $rkbc$ is the number of piggy-backed writes and w_{prg} the number of purge writes Rewriting w as $1 - r$, and rearranging terms, we obtain the proportion of purge writes to the total writes, p

$$p = \frac{w_{prg}}{w} = \frac{1 - r - rkbc}{1 - r} \tag{10}$$

We derive the average cost for performing write operations in write-only disk cache systems The average cost of a write operation is the sum of the weighted average cost of a piggy-backed write and the weighted average cost of a purge write The weights are based on the relative proportion of writes that are purge writes and those that are piggy-backed writes

The cost of a purge operation is the sum of the average seek time and the transfer time We

% read	% cache			
	1	2	5	10
0	18 76	9 38	3 75	1 87
10	17 93	8 55	2 92	1 04
30	15 55	6 17	0 54	0 00
50	11 29	1 88	0 00	0 00
70	1 25	0 00	0 00	0 00
90	0 00	0 00	0 00	0 00

Table 5 **Average write cost (zero-cost piggyback, 80 blks/trk)**

assume that writing starts immediately the disk seeks to the correct track (all the track blocks are purged), hence no rotational latency is incurred This is given as $(s + t)$ ms, where s is the average seek time of the disk and t is the time for a complete rotation If we assume from equation (7) that the average number of blocks per purge operation is $2cb$, then the average cost of purging a block is

$$\frac{(s+t)}{2cb} \tag{11}$$

Track-based piggy-backing adds to the cost for doing a read half disk rotation From equation (3) the average number of piggy-backed blocks per read is given as $krbc$, hence the average cost of a piggy-backed write in track-based piggy-backing can be given as

$$\frac{\bar{t}}{krbc} = \frac{\bar{t}}{rbc} \tag{12}$$

since $k = 1$ for track-based piggy-backing and $\bar{t}$ is the time for one-half disk rotation For zero-cost piggy-backing, the average cost of a piggy-backed write is zero

As before, let p represent the proportion of total writes that are purges and (1 - p) be piggy-backed writes (refer equation 10 for the value of p), then the average cost of a write in the track-based piggy-backing scheme is given by

$$p\left(\frac{s+t}{2cb}\right) + (1-p)\frac{\bar{t}}{rbc} \tag{13}$$

For zero-cost piggy-backing, the average cost of a write is given by

$$p\left(\frac{s+t}{2cb}\right) + 0 \tag{14}$$

In table 5 we show the average write cost for some read/write mix and cache size for write-only disk cache derived from equations 13 and 14 We have shown 0% reads to show the effect of pure purging Since the number of blocks per purge is linear with the cache size, the cost for even 0% reads decreases as the cache size approaches 10% As the number of reads increases, the effect of piggy backing plays a larger role For example, with a 1% cache and 70% reads, almost 94% of the writes are piggy backed These numbers agree very strongly with simulation data

This means that

- at low read rates, cache size dramatically increases the effects of purges, but has little effect on piggy-backing, and
- at high read ratios, cache size dramatically effects piggy-backing and purges are unnecessary

3.2 Read Response Time

As the percentage of read requests increases the read response time increases This is because less writes are cached resulting in less requests being satisfied from the cache In the limiting case of 100% reads, every request causes a disk access (recall that only write blocks are cached)

Conversely, an increase in the number of blocks per track, causes the response time of reads to decrease The increase in the number of blocks per track causes an increase in the cache hit ratio for reads and hence less read requests require a disk access

The arrival rate of requests and the cache size have related impacts on the average response time of read requests We found that holding certain parameters constant, there exists an arrival rate of requests below which using a cache of a certain size adversely affects system performance

From figure 5 we observe that if the arrival rate of requests is below about 28 arrivals per second, introducing a 1% cache produces a read response time that is worse than when no cache is used It is also observed that this behavior is independent of the piggy-backing scheme in effect

The piggy-back scheme impacts on the average response time of read requests the same way it does on the average service time of write requests (refer figure 3) With a small cache, track-based piggy-backing is more attractive than zero-cost piggy-backing, but as the cache becomes larger, zero-

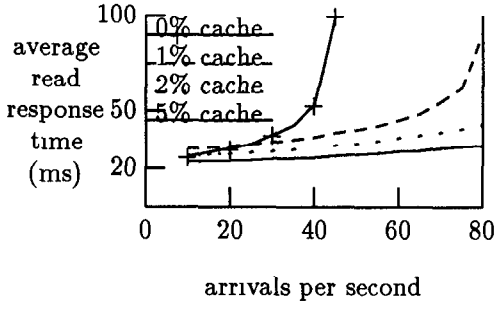

Figure 5 **Average read response time vs request arrivals per second (zero-cost piggyback, 30% read, 80 blks/trk)**

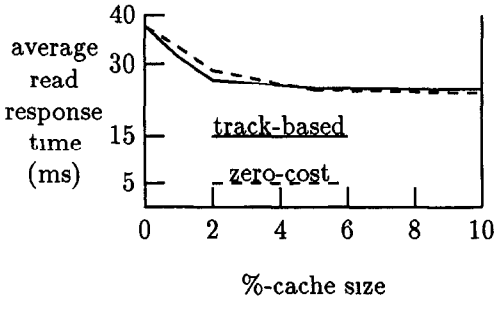

Figure 6 **Average read response time vs relative cache size (30% read, 30 arr/sec, 80 blks/trk)**

cache size (%)	*track-based*		
	r-res (ms)	*w-svc (ms)*	*b-prg*
0	37 46	21 94	0
1	31 32	6 44	36991
2	26 68	2 73	18520
5	24 88	1 38	0
10	24 73	1 34	0

Table 6 **Effect of Piggy-back Scheme (track-based scheme)**

cache size (%)	*zero-cost*		
	r-res (ms)	*w-svc (ms)*	*b-prg*
0	37 46	21 94	0
1	33 26	7 84	46258
2	28 73	3 91	36769
5	24 58	0 38	9161
10	23 94	0 00	0

Table 7 **Effect of Piggy-back Scheme (zero-cost scheme)**

cost piggy-backing outperforms track-based piggy-backing (refer figure 6) In table 6 the data of figure 6 is repeated in tabular form for the track-based scheme, and in table 7 the same data is repeated for the zero-cost scheme In both tables, we also show data for the average service time of write requests (*w-svct*) and number of blocks purged (*b-prg*) during a particular run of the experiment for a given cache size Average read response times are given under the heading *r-res*

When the cache is small, we observe that in the zero-cost scheme more time is spent freeing up cache space in purge operations than in the track-based scheme For example, in the zero-cost scheme about 25% more blocks are purged than in the track-based scheme when 1% cache is used Although, pending read requests are delayed by the rotational delay incurred in track-based piggy-back scheme, our experiments show that when the cache is small, the cost associated with this delay is still smaller than the seek and rotation costs that are incurred by the extra purge operations in zero-cost piggy-backing This behavior is clearly illustrated in tables 6 and 7 by the entries associated with a 1% cache

The relative performance of the two piggy-back schemes is clearly a function of the cache size With a small cache the track-based scheme outperforms the zero-cost scheme For example, with a small cache of about 1%, the average read response time for the track-based scheme is about 6 5% less than the corresponding value in the zero-cost scheme, showing the track-based scheme as the preferred method with a small cache However, with 10% cache, the track-based value is now about 3 3% more than the corresponding zero-cost value The same trend is also observed with the write service time

Zero-cost scheme is clearly a better scheme when

a large cache is used since all blocks are written on the zero cost part of disk rotation and there is hardly ever a need for a purge operation

3.3 Non-uniform distribution of requests

The experiments were repeated with a skewed distribution of requests The requests were generated such that 80% of the requests were serviced from 20% of the disk tracks The results from this set of experiments did not vary significantly from the uniformly distributed requests

This was surprising, but easily explained Initially, each purge contained a larger set of blocks as blocks accumulated faster on the most heavily used tracks However, as the simulation continued, less used tracks would slow accumulate the same number of blocks as heavily used sectors, since the reduced writes were exactly compensated for by the reduced reads The result is that the blocks on the lightly used cylinders were older, but no less numerous, than those on the more heavily used sectors

3.4 Multi-surface Disks

Our basic disk model is a single track, single surface disk system In this section multi-surface disk results are given, where only one read/write head is active at a time

Because of this constraint, interference exists when multiple blocks in the same sector of a cylinder are to be piggy-backed or purged In figure 7, piggy-backing effectiveness is shown with a fixed number of blocks/cylinder but a variable number of surfaces

For small caches, the number of piggy-backed blocks is independent of the number of read/write heads This is understandable since the small cache can only contain a very few number of blocks, and hence there is little interference The size of the cache becomes the limiting factor in this situation and not the number of read/write heads available for data transfer

As the cache becomes larger (5% and 10%), some improvements in the number of piggy-backed blocks are observed with fewer surfaces For a 10% cache with all read write heads operational, about 8 15 blocks are piggy-backed per read request as against 4 28 blocks piggy-backed when only a read/write head is operational in a 20 surface disk

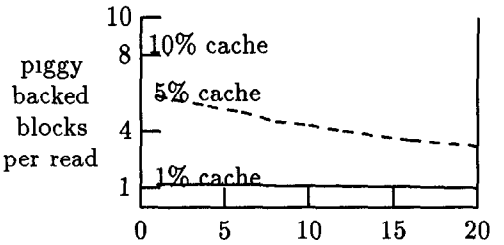

Figure 7 **Average no of piggy-backed blocks per read vs number of disk surfaces (10% read, 30 arr/sec, zero-cost piggy-back, 240 blks/cyl, uniform distribution)**

Hence, for multi-surface disks with small caches, all of the results in the previous section hold, except that transfer time (for the read) is larger Hence a 10 block/track disk with 20 recording surfaces can be modeled as a 200 block/track disk with one recording surface

This means that for small size cache and a fixed number of blocks per track

- number of piggy-backed requests grows linearly with the number of surfaces, or
- size of cache decreases in proportion to the number of surfaces

4 Conclusion

We have described zero-cost and track-based piggybacking, and have shown simulation results and analytical formulas of the cost of performing these algorithms These techniques are shown to be effective methods of increasing disk bandwidth

The general effects of disk caching and piggybacking can be summarized as follows

1 Write operations may be performed at no cost

2 With zero-cost piggy-backing, writes may be *invisible* and can be performed *lazily*

3 Purge operations never need to be performed if read rates are sufficiently high

4 Purge operations which are performed substantially reduce write costs in proportion to cache size

5 Disk traffic can be reduced to that required for reads With caching, disk utilization goes down as operations satisfied by the cache are never forwarded to disk

6 Effectiveness is independent of locality effects

The techniques described here are also independent, and hence can be combined with read caches

Acknowledgments

Partial support for this work was provided by ONR grant #88-K-0423 We would also like to thank one anonymous referee for his many insightful comments

References

[BG88] D Bitton and J Gray Disk shadowing In *Proceedings International Conference on Very Large Data Bases*, Long Beach, California, Sep 1988

[FC87] R S Finlayson and D R Cheriton Log Files An Extended File Service Exploiting Write-Once Storage In *Proceedings of the 11th ACM Symposium on Operating Systems Principles, Vol 21 #5*, pages 139–148, Nov 1987

[NWO88] M Nelson, B Welch, and J Ousterhout Caching in the Sprite Network File System *ACM Transactions on Computer Systems*, 6 No 1 134–154, Feb 1988

[OD89] J Ousterhout and F Douglis Beating the I/O bottleneck A Case for Log-Structured File Systems *Operating Systems Review*, pages 11–28, Jan 1989

[Ous85] J Ousterhout A Trace-Driven Analysis of the UNIX 4 2 BSD File System In *Proceedings Tenth Symposium on Operating Systems Principles*, pages 15–24, Dec 1985

[RB89] A L Narasimha Reddy and P Banerjee Performance Evaluation of Multiple-Disk I/O Systems In *Proceedings of the 1989 International Conference on Parallel Processing*, pages 315–318, Aug 1989

[Sat81] M Satyanarayanan A Study of File Sizes and Functional Lifetimes In *Proceedings Eight Symposium on Operating Systems Principles*, pages 96–108, Dec 1981

[STH83] R A Scranton, D A Thompson, and D W Hunter The Access Time Myth IBM Technical Report RC 10197 (#45223), Sep 1983