Performance Study of RAID-5 Disk Arrays with Data and Parity Cache

Sunil K. Mishra and Prasant Mohapatra
Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa 50011
Email: {smishra, prasant}@iastate.edu

Abstract

Disk array architectures such as RAID-5 have become an acceptable way for designing highly reliable and high-performance storage systems. However, one major drawback of a RAID-5 disk array system is that an update to a data block may involve four disk accesses. Such a high overhead is especially undesirable for workloads with high update rate as in transaction processing. In this paper, we present a new scheme for improving the write performance of disk arrays using controller cache to store data as well as parity information. We have developed a trace-driven model to simulate cached disk arrays for transaction processing environment. We have studied the effect of caching parity information at the controller level along with caching data. The simulation results show a considerable improvement in response time of data and parity cached disk array over disk arrays with only data caching. The improvement in response time for disk array employing parity cache is about 10%-20% for the parameters used in our study.

Key Words: Data Caching, Disk Arrays, Parity Caching, RAID-5, Response Time.

1 Introduction

In the last few years, the performance of processors has been growing steadily, and with multiprocessor organizations, the processing power of computer system has been increasing at a rapid pace. However, the Input/Output (I/O) performance has not kept pace with the gains in the processing power. The large disparity between the I/O subsystem's performance and the processing power is becoming a bottleneck in several computational problems. To achieve high I/O capacity and speed, disk array architectures have been proposed. Disk arrays of small diameter disks often have substantial cost, power, and performance advantages over large drives. These disk arrays have low cost and use simple encoding scheme to provide high data reliability while preserving most of the performance advantages. For these reasons, redundant disk arrays, also known as Redundant Array of Inexpensive Disks (RAID), are strong candidates for nearly all on-line secondary storage systems [1].

Parity encoding is used as a means for data encoding and storing redundant information in RAID level 3 through 5 [1]. Implementation of parity encoding is simple and cost-effective. RAID-5 disk arrays exploit the low cost parity encoding to provide high data reliability. Data is block interleaved over all disks so that large files can be fetched with high bandwidth. The parity information is also rotated to avoid hot spot for random write requests. However, a data update needs four disk accesses for updating data. A small write needs preread of the old value of the user's data, overwriting this with new user's data, preread of the old value of the corresponding parity, then overwriting this parity block with the updated parity. In contrast, mirrored disks simply write the user's data to the two disks, and requires two disk accesses, while non-redundant disks need only one disk access. Such a high overhead in writing a data block in a RAID-5 organization is particularly unacceptable for workloads that require several update operations.

Several schemes have been proposed recently to reduce the write overhead of RAID-5 disk arrays [2]-[12]. One of the ways to reduce this write overhead of RAID-5 is to cache or buffer data blocks at the controller level cache [3, 4]. Among other schemes, parity logging [6], floating parity [7], informed prefetching and caching [8] have shown to provide some degree of improvement in reducing the write with modest overhead.

In this paper, we propose a new scheme to store user's data and parity information in the cache. A part of cache is used to store parity information and is called the *parity cache*. Rest of the controller cache memory is used for caching user's data blocks, and is called *data cache*. The use of parity cache reduces actual disk write latency when the cached data is destaged. With only data caching, a write operation is done in the cache if the block is present or free cache space is available. But, when the cache need to be destaged to accommodate a new block, four disk accesses are still required. Use of parity cache reduces this latency as preread of old parity and writing of

new parity is delayed. This approach improves the performance in an environment that experiences an increase in disk accesses for a transient duration, such as during the peak hours of transaction processing. Read and write caches are not implemented separately in our study, but are implemented as a unified data cache. Extensive study on read and write caching has been done by Menon [10]. Here, we intend to study the effect of parity cache on the response time for a write operation. We have developed a trace-driven simulation model to study the effect of the use of parity cache. The simulation result shows considerable improvement in performance as compared to the data caching (read and write caching) schemes.

The rest of the paper is organized as follows. In Section 2, we review some of the existing performance improvement schemes. Section 3 describes the simulation model and underlying the assumptions. In Section 4, we present the trace driven simulation results followed by conclusions in Section 5.

2 Background

In this section, we discuss the data and parity placement schemes followed by the performance improvement techniques proposed for the RAID-5 architecture.

2.1 Data and Parity Placement

The RAID-5 array consists of N identical disks, where data blocks are interleaved across the N disks. A set of blocks on each disk with the same location constitutes a stripe. Each stripe has a parity block, which is the XOR of all of the other data blocks in the same stripe as shown in Fig. 1. The stripe width, W_s, defined as the number of data blocks in a stripe, is $N-1$. If any one disk fails, the array is still operational, since the failed data can be restored by XORing data from all of the other disks. The overhead for the reliability is that each write operation requires updating of the desired data block as well as the parity block.

Consider a write request that updates less than $N-1$ disks in a stripe. There are two procedures for calculating the new parity block for this stripe. The first procedure reads the old data blocks from the disks being updated and XORs them with the old parity block for that stripe. The result is the new parity block which is written on the parity disk. This is typically referred to as the "read-update procedure" for a write request. The second procedure reads the data blocks from the other disks from the same stripe that are not being updated and XORs them with the new data blocks that are being written. Chen and Towsley [16] have shown that the first procedure performs the best for the transaction type workload that is considered in this paper; hence, we consider only the first procedure in this paper.

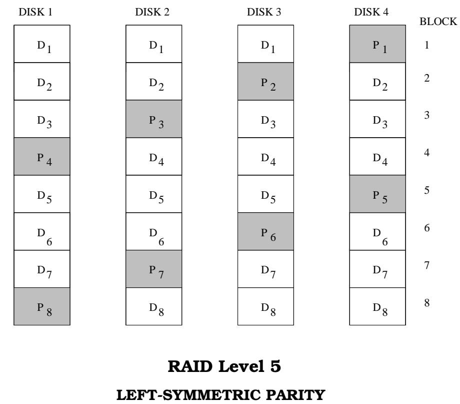

Figure 1: *RAID-5 block interleaving and left-symmetric parity placement. Shaded blocks indicate the parity blocks for that stripe.*

The major performance bottleneck of RAID-5 disk arrays is the high overhead for small writes. Each small write requires four separate disk I/Os, two to read the old data and old parity, and two to write the new data and new parity. These performance penalty of RAID-5 disk arrays relative to non-redundant and mirrored disk arrays are prohibitive in applications such as on-line transaction processing (OLTP) that needs many small writes. In OLTP workload, the request file/record size is usually small and limited to a few blocks only, where each block consists of few contiguous sectors.

2.2 Caching Schemes

Buffering and caching, which are commonly used in I/O system to reduce latency, can also be used in disk arrays. Write caching intercepts a user's write before the write is done on the disk, and the write operation is done on the cache. With write caching several blocks can be written sequentially, thereby reducing several small writes to a full stripe write. To avoid data loss under system failure, non-volatile cache memory is necessary to prevent the loss of buffered data. Data caching further exploits the opportunity to overwrite previous updates. Under high workload, the buffer space gets full quickly needing frequent destage of buffered data. In this case, the response time is the same as a non-cached disk array and is still four times worse than the non-redundant disk array.

Read caching is widely used in the disk drives, by using read ahead policy, to improve the response and

throughput when reading data. In a RAID-5 disk array, read caching can be used with the added advantage that preread of old data, required for generating new parity image, may be avoided [10].

As parity is computed over many logical blocks, parity caching exploits both temporal and spatial locality. By caching parity information, the parity update request can be deferred and sometimes preread of old parity can be avoided. In case of a forked request, without any cache, the parity update request is generated after the last forked request is served. This is done to reduce the workload of the disk containing the parity block. So, parity image for each of the forked requests has to be stored at the controller level. However, if parity cache is present, all the forked requests can be served separately (using any scheduling algorithm) and the parity image can be maintained in the parity cache. This results in reduced buffer space required at the controller level as only one block is required for the entire stripe. As in RAID-5, a disk array having G disks per parity string with N blocks per disk contains N/G blocks of parity information, parity cache size should be about $1/G$ times the size of write cache. Use of parity cache reduces the data cache size, which may increase data miss ratio. However, the use of a small amount of parity cache shows a considerable improvement in the response time as shown in this work.

2.2.1 Caching Location and Size

Among important parameters for cached disk arrays are the size of cache, cache block size, and its location. All modern disk drives use cache to improve throughput. In RAID architecture, the controllers are connected to a number of disks. So caching in the controller would help reduce the response time and improve the throughput. Caching at controller level has many advantages - cache coherency is achieved since the controller sees the I/O streams from all hosts and since data from multiple hosts resides in the controller cache. The cache pollution at the controller level is minimized as only the more frequently used portions of the prefetched data from each drive reside in the controller cache. Again, the cache can be efficiently divided among different disk drives based on their utilization and workload pattern.

In general, the performance of a disk array improves with the increase in cache size. However, the increase in performance is not linear. Finding a proper size of cache is important because of its higher cost as compared to the magnetic storage. To achieve an optimum cost-to-performance ratio, cache size should be 0.1 to 0.3 percent of the total disk storage [3]. A detailed study of cache size and block size, fetch block size has been done by Reddy in [2].

2.2.2 Caching and Replacement Policies

Cache write policies and cache replacement policies are two significant factors that affect the performance of a cache-based organization. Two different widely used write policies are write-back and write-through. Other variants of these policies, write-behind and write-free policy, are discussed in [5]. Use of write-back reduces the number of disk writes exploiting the temporal locality of data access pattern and is widely used in all types of cache.

When a cache is full, to accommodate new data, the old data in the cache need to be *destaged* (the term *flushed* is also widely used). Thus selection of a suitable destaging scheme is crucial when the type workload has a wide fluctuation in its access pattern. The widely employed schemes are random replacement (RR), least recently used (LRU), most recently used (MRU), most frequently used (MFU) and least frequently used (LFU). The use of different replacement algorithms is studied in [3, 5, 11]. The most widely used scheme, LRU replacement scheme, has been used in the study of cached RAID controller by Karedla et. al [3] and by Menon [10]. As the LRU scheme has been shown to demonstrate better performance than the other schemes, it has been used in our simulation study.

2.3 Parity Logging

Stodolsky et. al [6] have proposed *parity logging* to reduce the update penalty of small writes in RAID-5 disk arrays. This scheme delays the preread of old parity and write of new parity to the disks. Instead, an *parity update image*, which is the difference between the old and new parity, is temporarily stored in a log manner on a disk. Delaying the actual parity update allows the parity to be grouped together in large contiguous blocks that can be updated more efficiently. When the log disk fills up, parity update image is read to memory in a large sequential read along with the old parity information from different disks, and the new parity is generated. Then the new parity is written in a large sequential write to disk. Parity logging reduces the small write overhead from four accesses to a little more than two accesses, which is nearly same as in mirrored disk arrays. The overhead in this case is an extra log disk and the additional memory required while generating new parity. Even though Parity Logging provides sufficient savings in number of disk I/Os for a disk write, the update of data still needs two disk accesses. This access time may result in a long wait period for many requests, during the high I/O arrival rate for a transient time.

2.4 Floating Parity

Floating parity scheme reduces small write update penalty to a little more than single disk access time on average [7]. It is a greedy approach of writing parity to the disks. Floating parity scheme clusters parity stripes into cylinders, each containing a track of free blocks. Whenever a parity block needs to be updated, the new parity block can be written on the rotationally nearest, unallocated block following the old parity block. To efficiently implement floating parity, however, directories for the locations of unallocated blocks and parity blocks must be stored. Also, it needs more free space in each disks, and thus has high overhead cost.

2.5 Informed Prefetching and Caching

The improvement in response time as obtained by caching is solely dependent on the cache hit ratio. The use of a large cache does not result in proportionate increase in the hit ratio as the file size increases. A new technique, informed prefetching and caching, has been investigated by Patterson et. al in [8]. In any given application, the access pattern is predictable; thus, it is possible to inform the file system about the access pattern based on the predictability of an application. Each application gives a hint as to their future demands on the file system. Although this scheme enhances the performance, it is not always possible and convenient to predict the future demand on a file system.

3 Caching Parity

Parity cache can be used in conjunction with the data cache to reduce the number of disk accesses required for a write operation. In general, if there is no data cache at the controller level, use of parity cache can reduce the number of disk accesses by 1 to 2 for each write operation. However, the average response time is still governed by the data update time which needs two disk I/Os.

For a non-cached RAID-5 disk, a data block read time is $(S_d+R)+(2R/D)$ where S_d is the seek time for disk containing data, R is the average rotational delay, D is the block size in sectors [6]. The term $(S_d + R)$ corresponds to the time taken until the head is over the requested data sector. Average rotational delay can be assumed to be half the revolution time. A data write operation is done immediately following a data read operation. Thus a data write operation does not have any seek time. The wait period is the rotational delay until the data sector is under the read/write head which is $(2R - 2R/D)$. The block read or write time corresponding to actual block transfer time is $2R/D$. So, a write operation will take $(S_d + R) + (2R/D) + (2R - 2R/D) + 2R/D$ disk seconds, which is equal to $S_d + (3 + 2/D)R$ seconds.

A parity update request is generated only after the old data is read and parity image is generated. Parity image is referred to as the XOR of *olddata* and *newdata*. Assuming that the disk containing parity is ready to serve the parity update request, and the time taken to XOR the data and parity is negligible, parity update time will be $(S_d + R) + (2R/D) + (S_p + R) + (2R/D) + (2R - 2R/D) + 2R/D$ disk seconds, which equals to $S_d + S_p + (4 + 4/D)R$ seconds.

With only parity caching, the above expression reduces to $S_d + (3R + 2R/D)$ seconds, if there is a hit in the parity cache or free blocks are available.

In RAID-5 architecture, the data is block-interleaved. So any request for larger than a block size is forked and sent to different array controllers. For a write request, if the controllers are not using any aggressive scheduling policy to synchronize all forked write operations together, then each forked write request will generate a parity update request at a different time instant for the same parity block. This increases the workload seen by the disk containing parity, and the response time increases. This problem can be obviated by the use of a suitable scheduling policy in which all the forked requests are served at the same time. A considerable computation and communication is required at the controller level to implement an effective scheduling scheme. A simple solution to take care of this scenario is to use parity cache at the controller level. A part of the controller cache is set aside for caching parity. Although, it results in a reduction of data cache size and hence an increase in miss ratio, there is a significant reduction in the response time of a write request consisting of more than one block. Use of parity cache is suggested for workload consisting of write requests to more than one block. Furthermore, the use of parity cache can sometimes eliminate the read of the old parity.

4 Simulation Model

We have developed a detailed disk array simulator to study the response time of cached RAID-5 disk array under OLTP workload.

4.1 Model Assumptions

The trace driven simulator used to evaluate the performance of the cached RAID-5 disk array is based on the following assumptions.

- The disk array modeled is RAID-5 using left symmetric parity distribution [14], as it has shown to have the best performance.

- The I/O arrival process to the disk array is assumed Poisson. Furthermore, it is assumed that requests are of OLTP type, where small amounts of data are frequently accessed. The number of data stripe units in an I/O request is less than or equal to the number of data stripe units in a stripe, $W_S - 1$. Such distribution can be modeled using quasi-geometric distribution [16]. Let

$$P_n = \begin{cases} \sigma, & n = 1, \\ (1-\sigma)\frac{\rho-(1-\rho)^{n-1}}{(1-\rho)-(1-\rho)^{W_s-1}}, & 2 \leq n \leq W_s - 1. \end{cases}$$

  where P is the probability that number of data stripe units requested is n, σ is the probability of single block access, and ρ is the parameter of geometric distribution.

- The OLTP workload is predominantly read request, and the ratio of reads to writes ratio for such systems is typically 2 or 3 to 1.

- I/O request are uniformly distributed over all disks. With equal probability any disk in the array can be the starting disk of a multiple stripe unit request.

- Each controller can serve one disk request at any time, and a disk can service only one request at a time. Other requests wait and are served in first-come first-served (FCFS) order.

- The controller cache is fully associative and uses LRU replacement scheme.

- Once a disk request comes to the bottom of that queue, it is served immediately, and if it is a write operation, parity update is given highest priority at the disk containing parity block. The write synchronization model uses *after read-out (AR) policy* described in [16].

- The parity update cannot preempt any other request being served. It must wait until that request being served is finished. The disk maintains a separate high priority queue for parity updates.

4.2 Disk Array Simulator

The simulator has four main components: *synthetic trace generator, controller cache and queue handler*, *disk queue handler*, and a *model of disk behavior.* Fig. 2 shows a schematic diagram of the cached controller used in our simulator.

Synthetic Trace Generator: Because of the unavailability of the actual traces for the OLTP workload, we use synthetic traces to evaluate our proposed scheme. Synthetic traces have the flexibility that those can be altered to emphasize on certain aspects, such as the trace arrival rate.

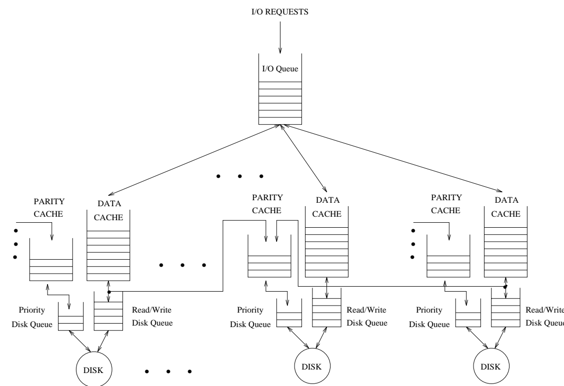

Figure 2: *Cached Disk Array Model*

The synthetic trace generator has been tailored to generate traces as seen in OLTP environments. In OLTP, the fetch size of typical requests are smaller in size, typically limited to few blocks. Since RAID-5 uses block interleaving, a request to more than a block is forked to different disks in the logical parity group. The simulator uses quasi-geometric distribution, as described earlier, to determine the fork-size of a request. In OLTP, the hit ratio in the controller cache is fairly high. Ramakrishnan et. al [13] have shown from actual trace analysis that number of read and write to a block in transaction processing is fairly high, and the same block is accessed often within a period of time. We have modeled this behavior to generate synthetic traces. Trace generator randomly (uniform disk access pattern) distributes the workload among all disks in the array, and the requests are put in the controller queue. The input parameters for trace generator that can be varied are number of I/Os per second, read-to-write ratio, read-hit ratio, and write-hit ratio. To model the effect of cache size, the read-hit ratio and write-hit ratio can be varied in proportion with the cache size.

Controller Queue and Cache Handler: This unit serves the queued requests, maintains the cache coherency and acts as an interface between user requests and disks. FIFO scheduling policy is used to serve the queued requests and write-back policy, and LRU replacement scheme is used for write and parity caches. Cache handler also marks blocks dirty or clean depending on the status of the cache block, and maintains the cache coherency. For parity cache, along with the flag for dirty/clean, another flag is used to represent whether the block is an old parity block or the

parity image. This flag is used to determine if preread of old parity is required. When the cache becomes full, the least recently used parity block is destaged, and this request is queued in the disk I/O parity queue.

Disk Queue Handler: For a disk read, the queue handler accesses the requested disk, and reads the block. For a write request, it finds out where the parity block is located. After the old data is read, if required, two requests are generated - one data write request to the same block, and another parity update request which is sent to another controller. We have assumed that all these controllers are connected through a bus for parity image transfer, and there is no contention. The parity image is updated by the cache handler if the block is already present or the parity cache is not full. Otherwise, it destages the bottom of the parity cache (the LRU block). This destage request goes to the disk parity queue. As described earlier, parity update requests are given higher priority, and a separate parity queue is maintained. Since disk preemption is not allowed, a parity update request is serviced immediately if the disk is free, else after the disk serves the current request.

Disk Behavior Model: We have used the disk behavior model used in RAIDSIM, a disk array simulator derived from the Sprite operating system disk array driver [14].

The seek time is a function of track separation and modeled as a polynomial equation [15],

$$seekTime(x) = \begin{cases} 0, & \text{if x = 0,} \\ a\sqrt{x-1} + b(x-1) + c, & \text{if x > 0} \end{cases}$$

where x is the seek distance in cylinders and a, b and c are chosen to satisfy the single-cylinder-seek-time, average-seek-time and max-stroke-seek-time constraints. The parameters a, b and c are approximately given by the following formulas [15]:

$$\begin{aligned} a &= \frac{(-10minSeek + 15avgSeek - 5maxSeek)}{(3\sqrt{numCyl})} \\ b &= \frac{(7minSeek - 15avgSeek + 8maxSeek)}{(3numCyl)} \\ c &= minSeek \end{aligned}$$

The disk drives used in our simulator are IBM 0661 3.5" 320MB SCSI disk drive whose characteristics are given below [6].

Geometry: 949 Cyl., 14 Tracks/Cyl., 48 Sectors/Track
Sector Size: 512 bytes
Revolution Time: 13.9 ms
Seek Time: $2.0 + 0.01 \times dist + 0.46 \times \sqrt{dist}$ ms
Tack Skew: 4 sectors

Since the disk accesses are random, only the seek time and actual block transfer time has been incorporated into our simulator. For rotational delay, uniform distribution has been hypothesized. Head switch time has not been incorporated for simplicity. The SCSI bus, between disks and the controller, is not modeled in our simulator. It is assumed that to avoid bus contention, a controller connected to several disks can issue an I/O request to one disk at a time. Only after that request has been served, the controller can access other disks.

For a cached controller, when the cache is empty, all read accesses are served by the disks, which result in higher transient response time. In our simulator, we allow the the simulation to continue for a certain amount of time, until the cache is nearly full, before the response time is recorded to compute the average response time.

5 Results and Discussions

The effect of write caching is simulated for comparison with parity caching while keeping the total cache size constant. The simulation results show that the parity caching has a better response time than the RAID-5 disk array with only data caching. A small amount of cache can be used for caching parity to avoid reduction in hit ratio in the data cache.

For each simulation run, we have used 10,000 traces after simulator initialization, and the result is averaged over ten independent replicated results to achieve high confidence interval. In our simulation we have used the following data unless otherwise mentioned in the figures.

Disk Array Size:	$4 \times (8 + 1P)$
Work Load Parameters:	
Single Block access Prob, σ:	0.5
Geometric Dist. Parameter, ρ:	0.7
Read ratio:	0.75
Write ratio:	0.25
Read-hit ratio:	0.5
Write-hit ratio:	0.9
block size:	4 (2KB)
I/Os per second:	400
Cache Parameters:	
Block search time:	100 μs
Sector read/write time:	100 μs

Since with only parity caching, the data blocks are updated for every write (involving two disk I/Os), we used data caching along with parity caching to investigate the gain in performance. Fig. 3 shows the average response time with respect to different I/O arrival rate for data cached and data with parity cached

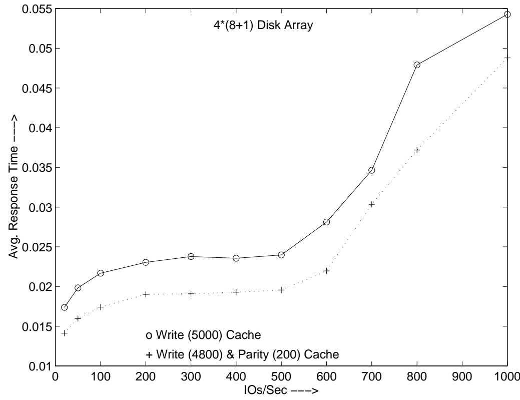

Figure 3: *Average response time for disk array with data cache, and data with parity cache.*

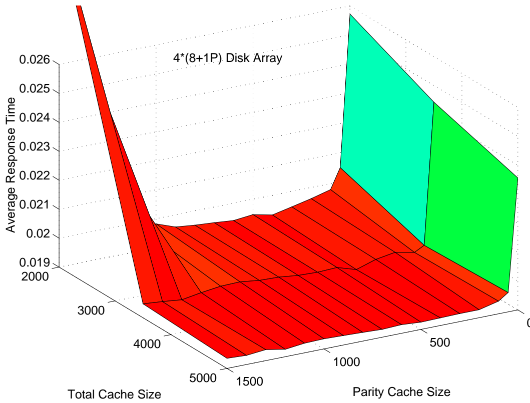

Figure 4: *Effect of different sizes of cache on response time.*

disk arrays. As the arrival rate increases, the response time increases. At a higher arrival rate, the increase in response time is significant as the waiting time in the queue is no longer negligible. This can be seen in Fig. 3 for I/O arrival rate of 600 IOs/sec or more. It can be observed that the data and parity cached array has about 10%-19% better response time than the data cached array at an arrival rate between 500-1000 IOs/sec. At a lower arrival rate, the improvement in response time is even higher.

In Fig. 4, we show the effect of different size of cache on average response time. The total cache size is kept constant while varying the parity cache size. The size of cache used in the simulation is less than what should be used in actual systems. Increasing the cache size makes it difficult to run the simulation within an acceptable time. However, the comparative result presented here is fair for any cache size. It is observed that, as the total cache size increases, the response time does not improve proportionately. So, increasing the cache size beyond a certain size does not yield much improvement in response time. A small size of parity cache helps in reducing the response time by as much as 18% as demonstrated in Fig. 4. But, as the parity cache size is increased, the write cache size decreases and so does the hit ratio. Hence, the response time increases gradually with higher parity cache size. Hence, only a small size of parity cache size should be devoted for caching parity.

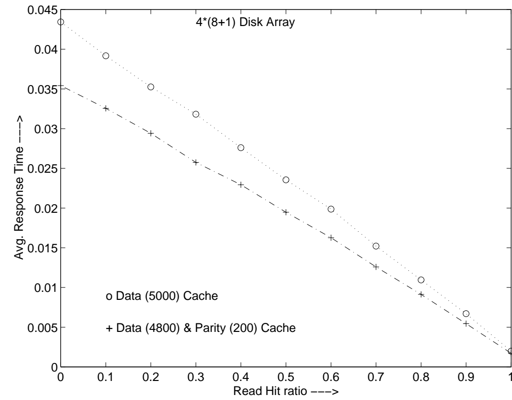

Figure 5: *Effect of read-hit ratio on response time.*

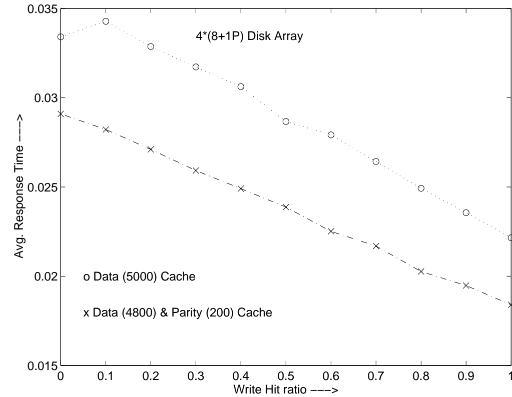

Figure 6: *Effect of write-hit ratio on response time.*

The cached controller's performance is mainly dependent on the hit ratio. This is particularly true for the case of read and write cache. In Fig. 5, we show the effect of read hit ratio on the average response time. With reduced hit ratios, the response time increases considerably. The write-hit ratio is assumed to be 0.90, and I/O arrival rate is kept fixed at 400 IOs/sec. With read-hit ratio near zero, the response

time is fairly high. With read-hit ratio equals to one, the response time reduces to nearly the access time of cache. As the read I/O is predominant in this type of workload (we have assumed read to write ratio of 3:1), with lower read-hit ratio, more blocks need to be destaged, hence results in higher response time. Fig. 6 shows the effect of write-hit ratio. There is a significant reduction in response time with write-hit ratio varying from 0 to 1.0. However, the read-hit ratio plays an important role as any read miss results in a disk access. As the performance degrades rapidly with very low hit ratios, use of cached controller or LRU scheme may not be effective in an environment that does not have much locality of reference in the access pattern.

6 Conclusions

We have presented a simulative study of RAID-5 disk arrays with data caching and parity caching. The results show that RAID-5 disk arrays with data and parity caching have 10%-20% better response time than RAID-5 with data cache alone. The hit ratio is an important deciding factor for the use of cache in RAID-5 disk array for different workload environment. Our study confirms this claim. Our future work is involved with the investigation of the following issues. The write-free schemes should be used for destaging and different replacement policies should be employed for different workloads. As our model uses unified read and write caching, further investigation is required on an efficient use of parity caching in conjunction with separate read and write caching. Also many other latency hiding techniques can be used in conjunction with the cached RAID-5 disk arrays. Further study is required to find an efficient way to combine these techniques along with the caching schemes.

References

[1] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, Vol. 26, No. 2, pp. 145-185, June 1994.

[2] A. L. N. Reddy, "A Study of I/O System Organization," *Proc. of the 19th Annual International Symposium on Computer Architecture*, Vol. 20, No. 2, pp. 308-317, May 1992.

[3] R. Karedla, J. S. Love, and B. G. Wherry, "Caching strategy to improve Disk System Performance," *IEEE Computer Magazine*, pp. 38-46, March 1994.

[4] J. Menon, and D. Mattson, "Performance of Disk Arrays in Transaction Processing Environments," *International Conference on Distributed Computer Systems*, pp. 302-309, 1992.

[5] D. Kotz, and C. S. Ellis, "Caching and Write-back policies in Parallel File Systems," *Journal of Parallel and Distributed Computing*, pp. 140-145, 1993.

[6] D. Stodolsky, G. Gibson, and M. Holland, "Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays," *Proc. of the 20th Annual International Symposium on Computer Architecture*, pp. 64-75, 1993.

[7] J. Menon, J. Roche, and J. Kasson, "Floating Parity and Data Disk Arrays," *Journal of Parallel and Distributed Computing*, pp. 129-139, 1993.

[8] R.H. Patterson, G.A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka, "Informed Prefetching and Caching," *Symposium on Operating System Principles*, pp. , Dec. 1995.

[9] P. S. Yu, K. Wu, and A. Dan, "Dynamic Parity Grouping for Improved Write Performance of RAID-5 Disk Arrays," *International Conference on Parallel Processing*, Vol. 2, pp. 193-196, 1994.

[10] J. Menon, "Performance of RAID5 Disk Arrays with Read and Write Caching," *Distributed and Parallel Databases*, Vol. 2, pp. 261-293, 1994.

[11] J. T. Robinson, and M. V. Devarakonda, "Data Cache Management Using Frequency-Based Replacement," *ACM Performance Evaluation Review*, Vol. 18, No. 1, pp. 134-142, 1990.

[12] K. Treiber and J. Menon, "Simulation Study of Cached RAID5 Designs," *High Performance Computer Architecture*, pp. 186-197, 1995.

[13] K.K. Ramakrishnan, P. Biswas, and R. Karedla, "Analysis of File I/O Tracesin Commercial Computing Environments," *ACM Performance Evaluation Review*, Vol. 20, No. 1, pp. 78-90, June 1992.

[14] E. K. Lee, and R. H. Katz, "Performance Consequences of Parity Placement in Disk Arrays," *ASPLOS-IV*, pp. 190-199, 1991.

[15] E.K. Lee, and R.H. Katz, "An Analytic Performance Model of Disk Arrays," *ACM-SIGMETRICS*, pp. 98-109, 1993.

[16] S. Chen, and D. Towsley, "The Design and Evaluation of RAID 5 and Parity Striping Disk Array Architectures," *Journal of Parallel and Distributed Computing*, pp. 58-74, 1993.