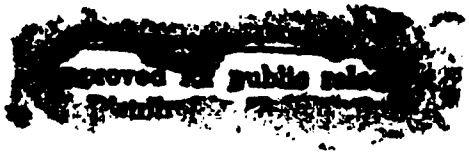
Computer Science

A Redundant Disk Array Architecture for Efficient Small Writes

Daniel Stodolsky, Mark Holland, William V. Courtright II,
and Garth A. Gibson

October 20, 1993
CMU-CS-93-200

Carnegie Mellon



A Redundant Disk Array Architecture for Efficient Small Writes

Daniel Stodolsky, Mark Holland, William V. Courtright II,
and Garth A. Gibson

October 20, 1993
CMU-CS-93-200

School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213-3890

Submitted to *Transactions on Computer Systems*

Abstract

Parity encoded redundant disk arrays provide highly reliable, cost effective secondary storage with high performance for reads and large writes. Their performance on small writes, however, is much worse than mirrored disks — the traditional, highly reliable, but expensive organization for secondary storage. Unfortunately, small writes are a substantial portion of the I/O workload of many important, demanding applications such as on-line transaction processing. This paper presents parity logging, a novel solution to the small write problem for redundant disk arrays. Parity logging applies journalling techniques to substantially reduce the cost of small writes. We provide a detailed analysis of parity logging and competing schemes — mirroring, floating storage, and RAID level 5 — and verify these models by simulation. Parity logging provides performance competitive with mirroring, the best of the alternative single failure tolerating disk array organizations. However, its overhead is close to the minimum offered by RAID level 5. Finally, parity logging can exploit data caching much more effectively than all three alternative approaches.

This research was supported by the ARPA, Information Science and Technology Office, under the title "Research on Parallel Computing", ARPA Order No. 7330, the National Science Foundation under contract NSF ECD-8907068, and by IBM and NCR graduate fellowships. Work furnished in connection with this research is provided under prime contract MDA972-90-C-0035 issued by ARPA/CMO to CMU. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

Keywords: Redundant disk arrays, RAID level 5, Parity logging.

Section 1. Introduction

The market for disk arrays, collections of independent magnetic disks linked together as a single data store, is undergoing rapid growth and has been predicted to exceed 7 billion dollars by 1994 [Jones91]. This growth has been driven by three factors. First, the growth in processor speed has outstripped the growth in disk data rates, requiring multiple disks for adequate bandwidth. Second, arrays of small diameter disks often have substantial cost, power, and performance advantages over larger drives. Third, low cost encoding schemes preserve most of these advantages while providing high data reliability (without redundancy, large disk arrays have unacceptably low data reliability because of their large number of component disks). For these three reasons, redundant disk arrays, also known as Redundant Arrays of Inexpensive Disks (RAID), are strong candidates for nearly all on-line secondary storage systems [Gibson92].

Figure 1 presents an overview of the RAID systems considered in this paper. The most promising variant employs rotated parity with data striped on a unit that is one or more disk sectors [Lee91]. This configuration is commonly known as the RAID level 5 organization [Patterson88].

RAID level 5 arrays exploit the low cost of parity encoding to provide high data reliability [Gibson93]. Data is striped over all disks so that large files can be fetched with high bandwidth. By rotating the parity, many small random blocks can also be accessed in parallel without hot spots on any disk. While RAID level 5 disk arrays offer performance and reliability advantages for a wide variety of applications, they are thought to possess at least one critical limitation: their

Author's addresses: Daniel Stodolsky, Garth A. Gibson, School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213; Mark Holland, William V. Courtright II, Department of Electrical & Computer Engineering, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213. Electronic mail: {daniel.stodolsky, garth.gibson, mark.holland, william.courtright}@cmu.edu.

Portions of this work has been previously published under the title "Parity Logging: Overcoming the Small Write Problem in Redundant Disk Arrays," *Proceedings of the 20th Annual International Symposium on Computer Architecture*, 1993, pp. 64-75.

throughput is penalized by a factor of four over nonredundant arrays for workloads of mostly small writes. A small write may require prereading the old value of the user's data, overwriting this with new user data, prereading the old value of the corresponding parity, then overwriting this second disk block with the updated parity. In contrast, mirrored disks simply write the user's data on two separate disks, and therefore, are only penalized by a factor of two [Bitton88]. This disparity, four accesses per small write instead of two, has been termed the *small write problem*.

Unfortunately, small write performance is important. The performance of on-line transaction processing (OLTP) systems, a substantial segment of the secondary storage market, is largely determined by small write performance. The workload described by figure 2 is typical of OLTP but nearly the worst possible for RAID level 5; a read-modify-write of an account record will require four or five disk accesses. The same operation would require three accesses on mirrored disks, and only two on a nonredundant array. Because of this limitation, many OLTP systems employ the much more expensive option of mirrored disks.

This paper describes and evaluates a powerful mechanism, *parity logging*, for eliminating this small write penalty. Parity logging exploits well understood techniques for logging or journalling events to transform small random accesses into large sequential accesses. Section 2 of this paper develops the parity logging mechanism. Section 3 introduces a simple model of its performance and cost. Section 4 describes alternative disk system organizations, develops comparable performance models and contrasts them to parity logging. Section 5 provides an analysis of small write overhead in parity logging with respect to various configuration and workload parameters, while section 6 analyzes potential bottlenecks. Section 7 analyzes the reliability of parity logging disk arrays and contrasts it to that of traditional storage systems. Section 8 introduces our simulation system, describes implementations of parity logging and alternative organizations, and contrasts their performance on workloads of small random writes and a more general OLTP workload. Section 9 discusses extensions to multiple failure tolerating systems. Section 10 reviews closely related work. Finally, section 11 closes with a summary of current and future work in redundant disk arrays for small-write intensive workloads.

Section 2. Parity Logging

This section evolves the parity logging modification to RAID level 5. Our approach is motivated by the much higher disk bandwidth of large accesses over small. A parity logging disk array accumulates small parity updates until sufficiently large accesses can be used to apply these updates efficiently. Our model is introduced in terms of a simple, but impractical RAID level 4 scheme, then refined to the realistic implementation used in our simulations.

A disk access can be broken down into three components: seek time, rotational positioning time, and data transfer time. Small disk writes make inefficient use of disk bandwidth because their data transfer time is much smaller than their seek and rotational positioning times. Figure 3 shows the relative bandwidths of random block, track and cylinder accesses for a modern small-diameter disk [IBM0661]. This figure largely bears out the lore of disk bandwidth: random cylinder accesses move data twice as fast as random track accesses which, in turn, move data ten times faster than random block accesses. Parity logging exploits this relationship by replacing many random small parity update accesses with a few large accesses to log and parity blocks.

Logically, we develop our scheme beginning with figure 4 in which a RAID level 4 disk array is augmented with one additional disk, a log disk. Initially, this log disk is considered empty. As in RAID level 4, a small write prereads the old user data, then overwrites it. However, instead of similarly updating parity with a preread and overwrite, the parity update image (the result of XOR'ing the old and new user data) is held in a log buffer. When enough parity update images are buffered to allow for an efficient disk transfer (one or more tracks), they are written to the end of the log on the log disk.

When the log disk fills up, the out-of-date parity and the log of parity update information are read into memory with large sequential accesses. The logged parity update images are applied to an in-memory image of the out-of-date parity, and the resulting updated parity is written with large sequential writes. When this completes, the log disk is marked empty and the logging cycle begins again.

Because only parity updates (not data changes) are deferred, this scheme preserves single failure tolerance. If a data disk failure occurs, the log disk (and any buffered parity updates) are first applied to the parity disk, which can then be used to reconstruct the lost data in the same manner as RAID level 5. If the log or parity disk fails, the system can simply recover by reconstructing parity from its data and installing a new empty log disk. If the controller fails, its buffered parity updates are lost, and parity can be rebuilt in the same way as if a log disk had been lost.[1]

The addition of a log disk allows substantially less disk arm time to be devoted to parity maintenance than in a comparable RAID level 4 or 5 array. This can be shown by computing the average disk busy time devoted to parity updates. Assume there are D data units per track, T tracks per cylinder, and V cylinders per disk (see the glossary in figure 5). First, every D small writes issued to the array cause one track write to the log to occur. Next, every TVD small writes issued cause the log disk to fill up, which must then be emptied by updating the parity. This requires three full disk accesses, which should occur at cylinder transfer rates. On average, then, for every TVD small writes there are TV sequential track accesses, and $3V$ cylinder accesses for maintenance of the parity information. Track accesses are D times larger than a random small write but about 10 times more efficient. Cylinder accesses are twice as efficient and T times larger than track accesses. Thus parity maintenance for TVD small writes consumes about as much disk time as

$$TV(D/10) + 3V(T/2 \times D/10) = TVD/4$$

random small accesses. In a standard RAID level 4 or 5 disk array, parity maintenance for TVD small writes would consume about as much disk time as TVD pairs of random block reads and writes. Thus, by logging parity updates, we have reduced the disk time consumed by parity update by about a factor of eight.[2]

1. Our failure model treats disk and controller failures as independent. If concurrent controller and disk failures must be survived, controller state must be partitioned and replicated [Schulze89, Gibson92, Cao93].

2. Notice that we make no attempt to reduce the cost of the preread and overwrite of the target data block. Additional savings are possible if data writes and deferred and optimally scheduled [Solworth90, Orji93].

2.1. Partitioning the Log Into Regions

As stated, however, this scheme is completely impractical: an entire disk's capacity of random access memory is required to hold the parity during the application of the parity updates. Figure 6 shows how this limitation can be overcome by dividing the array into manageably sized regions. Each region is a miniature replica of the array proposed above. Small user writes for a particular region are journalled into that region's log. When a region's log fills up, only that region's log is required to update the region's parity. This reduces the size of the controller memory buffer needed during parity reintegration from the size of a disk to a manageable fraction of a disk. Our models and simulation will use approximately 100 regions per disk (about 3MB per region).

Now, however, each region requires a log buffer. Each buffer holds K tracks of parity update images. When one of these buffers fills up, the corresponding region's log is appended with an efficient track (or multitrack) write. Thus, the sequential track writes of the single log scheme are replaced with random track writes in the multiple region layout. While random track writes are more expensive than sequential track writes, section 3 will show that this more practical implementation still has dramatically lower parity maintenance overhead than RAID levels 4 or 5.

To avoid deadlock while maintaining a minimum log buffer transfer size of K tracks, a parity logging disk array controller requires K tracks of buffer per region. The simplest buffer management scheme statically assigns to each region K tracks of buffer. When this buffer is exhausted, additional writes to the region are blocked. Obviously, such an approach leads to degraded performance if a single region becomes a hot spot. A slightly more sophisticated approach overcomes this limitation. Instead of statically assigning buffers to regions, all unused buffers are placed in a global free pool. When K tracks of buffer have accumulated for a single region, a log write is initiated, but additional records are allowed to accumulate in the global buffer pool. Now writes are only blocked if the entire buffer pool is exhausted, and degraded performance due to buffer contention is effectively avoided. While more sophisticated approaches could use free buffers to

improve performance (such as a water-mark based flush that attempts to write a multiple of K tracks at once), this simpler approach is assumed in the analysis and simulations that follow.

2.2. Striping Log and Parity for Parallelism

Similarly to the case of RAID level 4, the log and parity disks may become performance bottlenecks if there are many disks in the array. In particular, the disk bandwidth to all logs is just the bandwidth of single disk. This limitation can be overcome by distributing parity and logs across all the disks in the array, as indicated in figure 7. Now the aggregate log bandwidth equals the bandwidth of the array.

The log and parity bandwidth for a particular region, however, is still that of a single disk. Following the example of RAID level 5, figure 8 shows the parity for each region is block-striped across the array to increase bandwidth. This also decreases the latency of reintegrating parity updates for a particular region. The log, however, remains a potential bottleneck.

The log bottleneck may also be eliminated by distributing the log for each region over multiple disks. Figure 9 shows a parity logging array with the log for each region striped across two disks. Since the parity log is logically part of the parity, it cannot be placed on the same disks as the data is protects. Thus, log striping reduces the number of disks on which data for a particular region may be placed. This reduction in data striping in a region increases the disk space overhead as follows. Each region contains C_L cylinders of log and C_P cylinders of parity where $C_L = C_P$. The number of data cylinders per region, C_D, is related to the size of the parity, C_P, according to the standard RAID level 4 and 5 rule for data striped over $N-L$ disks: $C_D = (N-L-1)\,C_P$. Hence, the disk space overhead

$$(C_P + C_L) / (C_L + C_P + C_D) = 2/(N-L+1)$$

rises as the degree of log striping, L, rises (figure 10). As will be shown in section 8, however, the performance advantages of log striping are substantial.

The buffer memory overhead for this mechanism is modest. With B regions, the controller

requires KB track buffers and another buffer that is VT/B tracks in size for the parity reintegration. If a single log track is buffered for each of 100 regions, an array of 22 IBM 0661 disks requires 5592KB of buffer space. If memory is assumed to cost 20 times as much as disk per byte, this buffer space costs the equivalent of about 35% of one disk — roughly 2% of the 22 disk array.

2.3. The Impact of Varying Log Length

The previous subsection assumed that the same amount of disk space was allocated for log and parity in each region because our introduction assumed exactly one log disk was added to an array containing one parity disk. Given the more flexible striped log and parity model of figure 9, the efficiency and space overheads of parity logging can be altered by increasing or decreasing the amount of log allocated per region. Let A be the ratio of log to parity ($A = C_L/C_P$) in each region. The disk space overhead becomes

$$\frac{C_L + C_P}{C_L + C_P + C_D} = \frac{AC_P + C_P}{AC_P + C_P + (N-L-1)C_P} = \frac{1+A}{N-L+A}$$

The log for each region now fills up after ATC_PD small writes. Updating the parity still requires reading the parity, reading the log, and writing the parity disk, so the parity maintenance work for ATC_PD small writes is

$$ATC_P(\frac{D}{10}) + (2+A)C_P(\frac{T}{2} \times \frac{D}{10}) = (\frac{3}{20} + \frac{1}{10A})ATC_PD$$

random small accesses, or an overhead of $(3/20 + 1/10A)$ random small accesses per small write. Performance can therefore by traded for space, as in shown in figure 11. For instance, allocating twice as much log as parity ($A = 2$) increases the space overhead to 13.5% but decreases the parity maintenance overhead to 10% of that of RAID level 5, which does two parity accesses for each small write. Halving the amount of log ($A = 0.5$) decreases the disk space overhead to 6.75% while only increasing the parity maintenance work to 18% of RAID level 5.

2.4. Number of Regions

The selection of the number of regions has an important impact on controller memory requirements and cost. Recall that the array controller requires two types of buffer: a single parity reintegration buffer and per-region log buffers. The cost of this additional memory is $X[TV/B + KB]$, where B is the number of regions and X is the cost of a track of memory. The cost of this additional memory can be minimized by choosing the number of regions, B, to equal $\sqrt{TV/K}$. Selecting this value for B makes the cost of the additional memory $2X\sqrt{TVK}$. However, to maintain transfer efficiency, the sublogs, parity, and user data for each region should all be an integral number of tracks, and the user data per region must be exactly $(N-L-1)$ times as long as the corresponding parity for that region. A layout that meets these criteria — that begins with a calculated number of regions then truncates the sizes of each component to an integral number of tracks while maintaining the $(N-L-1)$ ratio of data and parity — may waste a considerable fraction of every disk. One way to reduce wasted space is to consider a range of numbers of regions near the optimum as calculated above, looking for a value which yields a better track discretization. We have found that a range of $\pm 10\%$ around the optimum number of regions leads to a choice which wastes less than 1% of the array's disk space.

If, however, the per-region sublogs, parity and user data are not required to be an integral number of tracks, fragmentation is substantially reduced. Relaxing this discrete tracks condition will cause additional head switches and single cylinder seeks to occur during log writes, thereby increasing overhead. Fortunately, for disks similar to the IBM Lightning drive (figure 12), this added overhead should reduce performance by less than 3%.

2.5. Accommodating the RAID Level 5 Large Write Optimization

In parity-based disk arrays, a large write operation, which is defined as a write that updates all the data units associated with a particular parity unit, can easily be serviced more efficiently than a small write operation. Since all data units in the stripe are updated, the new parity can be com-

puted in memory from the new data and written directly to the parity unit. This "large write optimization" avoids the preread of data and parity associated with small writes, improving write performance by about a factor of four [Patterson88].

This optimization can not be applied directly to parity logging disk arrays as we have described them so far because there may exist outstanding (not yet reintegrated) logged updates for a particular parity unit at the time when a large write overwrites that parity unit. If these logged updates were ignored and a parity overwrite were done, the parity could be erroneously updated with the stale logged updates when reintegration occurs. This problem can be corrected by placing the new parity into the log instead of writing it directly to disk.[3] Parity placed in the log by a large write operation is marked as a special "overwrite" record, and the reintegration process, which normally XORs each log record into the corresponding parity unit, now distinguishes between a normal "update" log record and the new overwrite record. Update records are XORed into the accumulating parity unit, while overwrite records are simply copied in.

This approach has the disadvantage of *forcing the log* to be processed sequentially rather than concurrently. If the log were guaranteed to contain only update records, the log records could be applied to the parity image in any order, increasing parallelism. The existence of overwrite records forces the reintegration process to determine the sequence in which the log updates occurred and to apply the log records accordingly.

This new sequentiality constraint potentially lengthens the reintegration time, which, as section 8 will show, can substantially degrade performance at high loads. In the simplest case, a region's logs must be in read in the order they were written and merged to produce a update/overwrite image before any of the parity is processed. Given sufficient buffer memory for a region's parity and log, full parallelism could be achieved during the log and parity reads, but the applica-

3. An alternative way to correct the problem is to write the new parity directly to disk and place a "cancel" record in the log. The reintegration process would then discard all previous log entries for the identified parity unit when it detects a cancel record. This solution has the potential to reduce the log traffic by making cancel records only a few bytes in size.

tion of logs to parity would still have to be deferred until these reads complete. At this point, a sequential in-memory reintegration could be performed. However, as long as log buffers are written to sublogs in a round-robin fashion, it is reasonable to assume that parallel sublog reads will return parity records in nearly sequential order. Based on this observation and because overwrite records eliminate all prior information, the following highly parallel algorithm can be used. Each block in the reintegration buffer is initially zeroed and marked "non-overwrite". Parity and log for the target region are all read in parallel. A parity block is applied if the corresponding buffer is marked "non-overwrite," and discarded if the buffer is marked "overwrite." If a logged record is an update and the block is "non-overwritten", the record is XORed in, but is buffered until all earlier log records have been processed. If a log record is an overwrite, the target block is overwritten and marked as "overwritten by record X." Any buffered updates that have already been applied should occur after this overwrite are reapplied. Overwrite or update records preceding X are not applied to a block marked "overwritten by X." As long as parallel reads on different sublogs proceeded at nearly the same rate, this algorithm will not consume much extra buffer space. If buffer exhaustion occurs, the algorithm can simply serialize. Parity is computed with a similar approach in the ADP-93-02 disk array controller [NCR93].

In summary, parity logging buffers parity updates until they can be written to a log efficiently. It then further delays their reintegration into the a redundant disk array's parity until the size of the log makes a complete revision of the parity efficient. To accommodate limited memory for reintegration of parity records, the disk arrays is partitioned into regions with per-region logging. Then, to avoid bandwidth bottlenecks, parity and log information is striped over multiple disks. This parity logging scheme reduces the extra work done by RAID level 5 arrays for small random writes to little more than is done in the much more expensive, traditional mirrored approach.

Section 3. Analytical Modeling for Parity Logging

In this section we present a utilization-based analytical model of a parity logging redundant disk array. This model predicts sustained array performance in terms of achieved disk utilization,

disk geometry, and access size. The variables used in this model are defined in figure 5.

Consider a single small user write in a parity logging array. The user data must be preread, then overwritten. This is done in an I/O which seeks to the cylinder with the user's data, waits for the data to rotate under the head, reads the data, waits for the disk to spin around once, then updates the data.[4] On average, such an access will take

$$\underbrace{(S+R)}_{\text{Seek and rotational delay}} + \underbrace{2R/\mathrm{D}}_{\text{Data preread}} + \underbrace{(2\mathrm{R}-2R/\mathrm{D})}_{\text{Rotational delay}} + \underbrace{2R/\mathrm{D}}_{\text{Data write}}$$

disk seconds, which may be simplified to $t_{rmw} = S + (3 + 2/\mathrm{D})\,R$.

In many cases, it may be possible to predictably avoid prereading user data. For example, in the TPC benchmark, the update of a customer account record is a read-modify-write operation; an account record is read, modified in memory, then written back to disk. In these cases, the old data value is usually known (cached) at the time of the write and the preread of the data may be skipped. Without prereading, the disk busy time needed for a small write access is $t_w = S + (1 + 2/\mathrm{D})\,R$.

Each region has K tracks worth of log buffers. On average, for every KD small user writes, one region's buffers will fill and be written to the region's log in a single K track write. The number of disk seconds needed to do this is

$$\underbrace{(S+R)}_{\text{Seek and rotational delay}} + \underbrace{2RK}_{\text{Data transfer time}} + \underbrace{(K-1)\,H}_{\text{Head switch time}}$$

assuming all K tracks are on the same cylinder[5]. This may be rewritten as

4. This single access could be separated into two accesses each taking $S+R+2R/D$ for a total of $2S+(2+4/D)R$. For most modern disks S is about twice R, so the single access is more efficient.

5. Disks that support zero-latency writes [Salem86] can eliminate the initial rotational positioning delay. If only a single track is buffered (K=1) this can reduce the I/O time by 26% in drives such as the IBM 0661 (which does not support this feature).

$$t_{Ktrack} = S + (2K+1)R + (K-1)H.$$

Finally, on average, for every DTC_L small user writes one region of logged parity must be reintegrated. First, consider the case of an array that does not stripe its log (figure 8). The reintegration consists of three steps: a sequential read of C_L cylinders (one region) from the log, a striped read of the parity from $N-1$ disks, and a striped write of the parity back onto $N-1$ disks. The sequential log read requires

$$\underbrace{(S+R)}_{\text{Seek and rotational delay}} + C_L \underbrace{(2RT + (T-1)H)}_{\text{Read time for 1 Cylinder}} + M \underbrace{(C_L - 1)}_{C_L\text{-1 single cylinder seeks}}$$

disk seconds, and may be rewritten as

$$t_{C_L} = S + (2TC_L + 1)R + (T-1)HC_L + (C_L - 1)M.$$

The striped parity accesses each consist of $N-1$ sequential transfers of $C_P/(N-1)$ cylinders. Each of these sequential transfers takes

$$\underbrace{(S+R)}_{\text{First seek and rotational delay}} + \underbrace{(C_P/(\text{N-1}))}_{\text{Cylinders per subaccess}} \underbrace{(2RT + (T-1)H)}_{\text{Read time for 1 cylinder}} + \underbrace{(C_P/(\text{N-1})\text{-1})M}_{\text{Single track seeks per subaccess}}$$

Rewriting, the total striped access takes t_{C_P} disk seconds

$$t_{C_P} = (N-1)(S+R) + C_P(2RT + (T+1)H) + M(C_P - N + 1).$$

Thus, on average, every small user write utilizes disks for

$$t_{amw} = t_{rmw} + \underbrace{\frac{1}{KD}[t_{Ktrack}]}_{\text{Log write}} + \underbrace{\frac{t_{C_L}}{DTC_L}}_{\text{Log read}} + \underbrace{\frac{2t_{C_P}}{DTC_L}}_{\text{Parity read and write}}$$

Figure 13 shows the contributions to disk busy time of the various terms after t_{rmw} in the above equation for the example disk array given in figure 12.

The analysis for a parity logging disk array with a striped log such as that shown in figure 9 is similar. When a region's log buffers fill, these buffers will be written to one of the regions sublogs in a single K track write. The cost of this operation is the same as in the unstriped case. Log reintegration still occurs every DTC_L small user writes, but now consists of three striped I/Os: a striped (over L disks) read of the log, and a striped (over N disks) read and write of the parity. Each of L accesses in the striped log read costs

$$\underbrace{(S+R)}_{\text{First seek and rotational delay}} + \underbrace{(C_L/L)}_{\text{Cylinders per subaccess}} \underbrace{(2RT+(T-1)H)}_{\text{Read Time for 1 Cylinder}} + \underbrace{(C_L/L-1)M}_{\text{Single track seeks per subaccess}}$$

for a total of

$$L(S+R) + C_L(2RT+(T-1)H) + (C_L-L)M$$

disk seconds. Similarly, the striped parity reads and writes will consume

$$N(S+R) + C_P(2RT+(T-1)H) + (C_P-N)M$$

disk seconds. Thus striping introduces an additional overhead of $L(S+R-M)$ disk seconds to the log reintegration. This increases the parity maintenance overhead per small write by

$$\frac{L(S+R-M)}{DTC_L}.$$

As section 8 will show, this increase in parity maintenance work is worthwhile because it reduces long reintegration periods during which disk queues grow, the system becomes underutilized, and maximum performance falls far short of expectations.

Section 4. Analytical Models for Alternative Schemes

Few other authors have addressed the problem of high performance yet reliable disk storage for small write workloads. The most notable of these is floating data and parity [Menon92]. This section reviews and estimates the performance of four configurations: mirrored disks (RAID level 1), nonredundant disk arrays (RAID level 0), distributed N+1 parity (RAID level 5), and

floating data and parity. The notation and analysis methodology are the same as used in section 3.

In nonredundant disk arrays, a small write requires a single disk access which consumes

$$\underbrace{(S+R)}_{\text{Seek and rotational delay}} + \underbrace{2R/D}_{\text{Data write}}$$

disk-arm seconds. No long term storage is required in the controller.

In mirrored systems, every data unit is stored on two disks, and all write requests update both copies. Each access takes as much time as a small write in a nonredundant disk array, $S + (1 + 2/D)\,R$. Hence, each small user write utilizes disks for a total of $2S + (2 + 4/D)\,R$ seconds. While mirrored disks are more efficient than RAID level 5, half of their capacity is devoted to redundant data. Controllers for mirrored disk arrays do not require long term buffer.

Small writes in RAID level 5 disk arrays require four I/O's: data preread, data write, parity read, parity write. These can be combined into two read-rotate-write accesses, each of which takes

$$\underbrace{(S+R)}_{\text{Seek and rotational delay}} + \underbrace{2R/D}_{\text{Data preread}} + \underbrace{(2R-2R/D)}_{\text{Rotational delay}} + \underbrace{2R/D}_{\text{Data write}}$$

disk seconds for a total disk busy time of $2S + (6 + 4/D)\,R$. Again, no long term controller storage is required.

The *floating data and parity* modification to RAID level 5 was proposed by Menon and Kasson [Menon92]. In its most aggressive variant, this technique organizes data and parity into cylinders that contain either data only or parity only. As illustrated in figure 14, by maintaining a single track of empty space per cylinder, floating data and parity effectively eliminates the extra rotational delay of RAID level 5 read-rotate-write accesses. With floating data and parity, the rotational term $2R - 2R/D$ in the RAID level 5 disk arm busy time expression above is replaced with a head switch and a short rotational delay. Using disks similar to those in our sample array, Menon and Kasson report an average delay of 0.76 data units. So, the expected disk busy time for each access in a floating data and parity array is

$$S + R + S + R + 2R/D + H + 0.76\,(2R/D) + 2R/D$$

which may be rewritten as $S + (1 + 5.52/\mathrm{D})\,R + H$. Hence, the total disk busy time for a small random user write in a floating data and parity array is $2S + (2 + 11.04/D)\,R + 2H$. Note that if D is large and H is small, this is close to the performance of mirroring.

Even with a spare track in every cylinder, floating data and parity arrays still have excellent storage overheads. For an N disk array, floating data and parity has a storage overhead of $(T + N - 1) / (TN)$.[6] Floating data and parity arrays, however, require substantial fault-tolerant storage in the array controller to keep track of the current location of data and parity.[7] For each cylinder, an allocation bitmask is maintained. This requires DT bits per cylinder. In addition, a table of current block locations for each cylinder is required. This consumes $D\,(T-1)\lceil \log\,(DT)\rceil$ bits per cylinder. Thus, a total of $VD\,(T + (T-1)\lceil \log\,(DT)\rceil)$ bits of fault-tolerant controller storage are required. For the disks in figure 12, this is 1,343,784 bits (164 KB) per disk. The total controller storage is about 3608KB, roughly comparable to parity logging.

While floating data and parity substantially improves the performance of small writes relative to RAID level 5, its performance for other types of accesses is degraded. Within a cylinder, logically contiguous user data units are not likely to be physically contiguous. In the worse case, two consecutive data units may end up at the same rotational position on two different tracks, requiring a complete disk rotation to read both. In addition, the average track has only $D\,(T-1)\,/T$ valid data units. Thus, even on disks with zero-latency reads, the maximum sequential read bandwidth is reduced, on average, by $(T-1)\,/T$.

Figure 15 compares the model's estimates for maximum throughput of the example arrays

6. Each disk gives up $1/T$ of its capacity for free space and the array gives up $1/N$ of the remaining space for parity. Thus the array storage efficiency is $(T\text{-}1)(N\text{-}1)/TN$ and the array storage overhead is $1\text{-}(T\text{-}1)(N\text{-}1)/TN = (T+N\text{-}1)/TN$.

7. The nature of fault tolerance in a storage controller depends on an underlying failure model. If only power failure is of concern, then nonvolatile storage will suffice, while other fault models require redundant storage.

based on figure 12. Throughput at lower utilizations may be calculated by scaling the maximum throughput numbers by the disk utilization. Figure 15 predicts that parity logging and floating data and parity will both substantially improve on RAID level 5, approaching the performance of mirroring, for small random writes.

Section 5. Sensitivity Analysis

Figure 13 shows the amortized write overheads for parity logging for our example array with an unstriped log. This section analyzes the sensitivity of these overheads to workload parameters. disk geometry parameters, disk performance parameters, and controller configuration.

5.1. Workload Parameters

Of the workload parameters, the number of data units per track, D, has the greatest impact on performance. Parity logging transfers each data unit two more times than RAID level 5 and four more times than mirroring. If the transfer time of a unit is small, parity logging will be efficient. Figure 16 shows the relative performance of parity logging (when data caching is ineffective; that is, blind writes), mirroring and RAID level 5 for different values of D for our example array. The performance of mirroring exceeds that of parity logging with 13 or fewer data units per track, and RAID level 5 performance exceeds that of parity logging with the unlikely case of 2 data units per track. Industry estimates, however, show that track capacity within a given form factor is increasing at over 20% a year. Consequently, it is reasonable to assume that the number of data units per track may not decrease even as database account record sizes grow with new value-added features. Additionally, the large capacity of the outer cylinders in zone bit recording [Seagate92, Hewlett-Packard93] disks offers the potential to increase the efficiency of sequential transfers in parity logging by locating the log and parity in this area.

5.2. Disk Geometry Parameters

Zero latency write support has a small impact even though it eliminates the rotational delay

for track sized writes. Figure 17(a) shows the contribution of rotational delay to the log write overhead. Even when only a single track is buffered per region, the throughput improvement due to zero latency support is less than 2.5%.

Parity logging performance is also insensitive to the number of cylinders in the disk, V. The number of tracks per cylinder, T, measurably impacts the relative performance of sequential transfer and random I/O: head switches are replaced with single cylinder seeks. Single cylinder seeks are typically between 1 and 3 times as expensive as head switches, so decreasing T decreases the efficiency of long transfers. Each small write is involved is three long transfers: log read, parity read and parity write. For a hypothetical Lightning drive with a single track per cylinder, the contribution of track switches climbs from 3(13(1.16 ms)+2 ms)/14(12) = 0.305 ms to 3(2.0 ms)/12 = 0.5 ms, increasing the overhead by roughly 3% (0.2 ms out of 6.6 ms). Thus, while sequential transfer performance is sensitive to T, parity logging is not.

5.3. Disk performance parameters

The ratio of average seek time to rotational latency has a substantial impact on the performance of parity logging disk arrays. Figure 18 plots the relative performance of parity logging, RAID level 5 and mirroring versus seek time. The performance of mirroring achieves as much benefit from decreased seek time as nonredundant arrays because its two accesses are each equivalent to the single nonredundant access. RAID level 5 and parity logging, however, do more rotational work for each seek so decreasing seek time relative to rotational latency hurts their performance relative to nonredundant arrays. Parity logging does extra rotational work to avoid the parity write seek of RAID level 5. Consequently, the relative advantage of parity logging over RAID level 5 decreases as the seek time to rotational latency ratio decreases. This ratio, however, is nearly unity for all modern drives, and shows no particular trend in any direction.

5.4. Array Controller Parameters

There are three array controller parameters that influence the performance of parity logging:

the number of regions, B, the number of disks, N, and the number of tracks buffered per region, K. The performance of parity logging is insensitive to B, as long as B is kept small enough to maintain the efficiency of the long log read and parity read/write operations. Practically, this means the number of regions should be less than half the number of cylinders per drive. The number of disks has almost no impact on performance, unless the number of disks become sufficiently large that the parity and log become too finely striped or that the array controller buffers become the bottleneck. The number of tracks buffered per region, K, however, has a measurable if small impact on overhead. As K is increased, the overhead of the log write approaches that of a long sequential transfer. Figure 17(b) shows the log write overhead for various values of K and the corresponding performance of the array.

Section 6. Bottleneck analysis

The analytical model presented in section 3 provides an estimate of the performance of parity logging disk array in terms of the disk arm service time. Such a model will only be an accurate predictor if (1) the disks are equally loaded and (2) controller resources are not exhausted. These assumptions are examined in this section.

The model in section 3 assumes the user request stream is spread uniformly throughout the disk array. While this assumption approximately holds for many OLTP workloads, other workloads exhibit substantial locality. In particular, consider the extreme situation in which all user I/O is concentrated within one region. Choosing an appropriate data stripe unit [Chen90] will balance the user I/O across the actuators that contain data for this region; however, log and data traffic are partitioned over non-overlapping disks. Recall that every DTC_L small user writes to a particular region will cause TC_L/K K-track log writes, a full log read and a full read and full write of the parity for that region. Parity reads and writes are spread out over the entire array, so a uniform load is maintained even for this extremely localized workload if the work per data disk equals the work per sublog disk. That is,

$$\frac{DTC_L t_z}{N-L} = \frac{(TC_L/K)\, t_{Ktrack} + t_{C_L}}{L}$$

where t_{Ktrack}, and t_{C_L} are the service times for a K track write and a full log read, respectively, and where t_z is the service time for a small user write — t_w when data caching is effective, or t_{rmw} when it is not. Solving for L, one obtains

$$L = N\frac{(TC_L/K)\, t_{Ktrack} + t_{C_L}}{(DTC_L)\, t_z + (TC_L/K)\, t_{Ktrack} + t_{C_L}}.$$

Strictly speaking, t_{C_L} is a function of L, so the relation above does not immediately provide a value for L that will balance the workload. However, as a long as the transfer time for a sublog is large relative to the average seek time, t_{C_L} can be accurately approximated by the service time for an unstriped log. Using this approximation and the disk array parameters from figure 12, one obtains $L \approx 0.16N$ for blind writes ($t_z = t_{rmw}$) and $L \approx 0.11N$ for the write hit case ($t_z = t_w$). Thus, to balance the load in this highly localized workload, the example 22 disk array must have two to four sublogs per region.

Another potential bottleneck is the log and parity reintegration buffer. Recall that log reintegration requires two disk utilizing steps: reading the log and parity, and writing the parity. The reintegration buffer must be held from the start of the reads until the successful completion of the writes. While reintegration involves every disk in the array, the disk utilization during reintegration is not 100% because reintegration places unequal loads on the disks. Thus, during one log reintegration, another log could fill and block on the reintegration buffer. By computing the latency of log application an estimate of vulnerability to this kind of underutilization can be obtained.

The first step of reintegration involves a simultaneous read of the log and parity. Of the N disks in the array, $(N-L)$ of them will contain only parity, while the remaining L contain both parity and log. Since the parity and log for a region are consecutive (see figure 9), the log and par-

ity may be transferred in one sequential access. Reintegration, therefore, requires $(N-L)$ disks to sequentially transfer C_P/N cylinders, and the remaining L disks to transfer $C_L/L + C_P/N$ cylinders. If the disks arms are in random positions, it can be shown [Bitton88] that the expected longest seek of these N parallel transfers is

$$I_N = 1 - \left[\frac{2n}{2n+1} \times \frac{2n-2}{2n-1} \times \ldots \times \frac{2}{3}\right]$$

of the disk. For most disks, a maximum (full stroke) seek is roughly twice the average seek, and the seek time distribution between the average (1/3 stroke) and full stroke is roughly linear, so we approximate the maximum parallel seek time by

$$S\left[\frac{1}{2} + \frac{3}{2}\alpha\right]$$

where α is the fraction of the stroke seeked over. Assuming the worst case — that a disk with log records on it has the longest seek — the latency for reading the log and parity into memory on an idle array (or with preemptible I/Os) is

$$T_{load} = S(1/2 + 3I_N/2) + R + [C_L/L + C_P/N]\ (2RT + (T-1)H + M) - M.$$

After reading the log and parity, the controller will take some time to finish integrating the log. With an aggressive algorithm, most of the log reintegration should be overlapped with the transfer process. The examination of how much processing is not overlapped by transfer is beyond the scope of this paper, and we simply denote this time as T_{cpu}. In the calculations below, T_{cpu} is assumed to be negligible.

After the completion of reintegration, the modified parity is written out. Writing out the parity requires that each drive sequentially write C_P/N cylinders of parity. The latency for the longest of these writes is expected to be

$$T_{write} = S(1/2 + 3I_N/2) + R + [C_P/N]\ (2RT + (T-1)H + M) - M,$$

assuming that intervening I/Os have randomized the disk arm positions and that none of the rein-

tegration accesses are delayed by queueing. Thus, the total time to reintegrate the parity for one region is

$$T_{load} + T_{cpu} + T_{write} =$$

$$T_{cpu} + 2\,[S\,(1/2 + 3I_N/2) + R - M] + [2C_P/N + C_L/L]\;(2RT + (T-1)\,H + M)$$

In a saturated, balanced system, the frequency of reintegration operations is simply $DTC_L \times t_{amw}/\mathrm{N}$, where t_{amw} is the amortized disk arm time per small write calculated in section 3, so the mean time between reintegrations is at least $N/(t_{amw}DTC_L)$. Figure 19 plots the reintegration time for the example array for various degrees of log striping. As we have suggested previously, the reintegration latency is quite sensitive to the degree of log striping.

Section 7. Reliability Analysis

The traditional method for improving the reliability of disks has been mirroring. A drawback of mirroring is the high overhead required for redundancy (50% of the total disk capacity is lost to redundancy). Parity logging provides a low-overhead alternative. This section compares the overhead and reliability of mirroring, parity logging, and nonredundant arrays.

Independent disk failures in single fault tolerant disk arrays can be characterized by a Mean Time To Data Loss ($MTTDL$) and a reliability function $R\,(t)$, the probability of surviving a lifetime of t, of

$$MTTDL = \frac{(2N-1)\,MTTF_{disk}MTTR_{disk} + MTTF_{disk}^2}{GN\,(N-1)\,MTTR_{disk}}$$

$$R(t) = exp\,(-t/MTTDL)$$

where G represents the number of redundancy groups, N the number of drives per group, $MTTF_{disk}$ the mean time for a disk to fail, and $MTTR_{disk}$ is the mean time required to repair a single failed disk [Gibson93]. For modern disks, a $MTTF_{disk}$ of 300,000 hours is not unreasonable [Hewlett-Packard93] and repair times of 1 hour are easily achieved with an on-line spare

[Holland93]. As shown in figure 20, both parity logging and mirroring offer extremely reliable storage.

Section 8. Simulation

To validate the models presented in sections 3 and 4 and to explore response time for these arrays, we simulated the example array described in figure 12 under five different configurations: nonredundant, mirroring, RAID level 5, floating data and parity, and parity logging. Parity logging was simulated with a single track of log buffer per region for several different degrees of log striping. The RAIDSIM package, a disk array simulator derived from the Sprite operating system disk array driver [Ousterhout88, Lee91], was extended with implementations of parity logging and floating data and parity.

In each simulation, a request stream was generated by 66 user processes (i.e, three per disk). Each process requests a small write from a disk selected at random, then waits for acknowledgment from the disk array. Process think time has a Gaussian distribution, but the mean is dynamically adjusted until the desired system throughput is achieved. If the disk array is unable to sustain the offered load, think time is driven to zero. Simulations were run until the 95% confidence interval of the response time became less than 5% of the mean.

8.1. The Need for Log Striping

Figure 21 shows peak throughput, response time[8] and response time variance as the degree of log striping (L) is varied from 1 (unstriped) to 13. As section 6 and figure 19 suggest, when the log is striped over a small number of disks, performance is substantially lower than in other configurations with more log striping. This behavior results from a "convoy effect" in which processes back up behind very long sublog read accesses. Figure 22 shows sublog read times for low

8. The simulations reported herein consider a user write in a parity logging array complete when the user data is on disk and the parity update record has been buffered. The alternatives (nonredundant, mirroring, floating data and parity, and RAID level 5) consider a user write complete when data and parity are on disk.

degrees of log striping. While these long I/O's are efficient, they completely tie up a disk for seconds. During this period, any access to the disks involved in the striped log read will block, reducing the effective concurrency in the system. This concurrency reduction causes other disks in the array to become idle until the log read completes, reducing peak throughput and utilization. This convoy effect also has a substantial impact on response time; I/O requests that block behind these long read requests will have very long response times, leading to an increase in both average response time and response time variance. Fortunately, a modest degree of striping eliminates the convoy effect. Striping the log over six disks achieves most of the available performance without greatly increasing disk space overhead. Figure 23 compares the performance of a parity logging array with one log buffer per region and a log striping degree of six against the alternative organizations presented in section 4: nonredundant, mirroring, RAID level 5, and floating data and parity. Figures 23(a)-(b) present response time statistics as a function of throughput for simulations that preread (data cache miss) user data, and (c) presents the corresponding data for the no preread (data cache hit) case.

The simulation response time results may be summarized as follows. Nonredundant disk arrays perform a single disk access per user write, so they have the lowest and most slowly growing response time. Mirroring shows a similar behavior, but is driven into saturation with half as much load. In contrast, each small user write in RAID level 5, in the user data preread case, completes two slow read-rotate-write accesses sequentially.[9] Unloaded system response time is thus quite high and queuing effects cause it to grow quite rapidly with load. While the response time for parity logging on a lightly loaded system is approximately 14 ms higher than mirroring,[10] the peak sustainable I/O rate and response time are quite similar. Similarly to RAID level 5, floating

9. In a highly aggressive implementation, it would be possible to initiate the parity read-rotate-write access after the preread of the old user data completes, but we assume status is only returned after the completion an entire read-rotate-write access.

10. In parity logging arrays that are not driven into saturation, prioritizing the log accesses to allow preemption by user accesses will reduce response time and response time variance.

data and parity arrays require two read-rotate-write accesses per user write. But by removing the rotational delays, floating data and parity achieves peak IO rates similar to parity logging and mirroring. Response time, however, is significantly longer.

Figure 23(c) shows the performance of all configurations without data prereads As expected, this has no effect on mirrored or nonredundant systems and the performance of the other three configurations improves. RAID level 5 benefits substantially from the elimination of the full rotation delay incurred by a data preread. In addition, a user's data write and parity update can be issued concurrently, further improving the response time and array utilization. Floating data and parity achieves a lesser benefit from elimination of the preread because its preread overhead is much less. Response time does drop, however, because of the ability to issue user write and parity update accesses simultaneously. The response time of parity logging improves by a full rotational delay (13.9 ms) due to the elimination of the preread rotate, providing an unloaded response time comparable to a nonredundant array. This also reduces the actuator time per I/O by nearly one third, allowing its I/O rate and response time to improve proportionately.

The variance in user response time, however, is larger with parity logging than with mirroring or floating data and parity, although it is not as large as with RAID level 5. This results from the basic structure of parity logging: most accesses are fast because inefficient work is delayed. However, some accesses see long response times as delayed work is (efficiently) completed. With this higher variance in mind, we conclude that the response time estimates in figure 23 show that parity logging is a viable and much lower cost alternative to mirroring for small-write intensive workloads.

8.2. Analytic Model Agreement with Simulation

The analytical model estimates in figure 15 predict the vertical asymptotes (saturation throughputs) of figure 23(a) and (c). A direct comparison, however, will display significant discrepancies because of the relatively small number of simulated processes. Because the number of these requesting processes is fixed, one overloaded disk can cause other disks to be underutilized.

The impact of this effect on peak disk utilization varies from configuration to configuration. Figure 24 shows the disk utilizations at peak load for the configurations simulated. Parity logging, floating data and parity, RAID level 5, and nonredundant disk arrays are about equally affected since each system presents only one disk access request at a time per process. Mirroring, on the other hand, presents two write requests simultaneously and is therefore impacted the least.[11] Nonetheless, figure 25 shows that simulation agreement with the model is good (within ± 5%) when the model results of figure 15 are scaled by the achieved disk utilizations of figure 24.

8.3. Performance in More General Workloads

Up to this point, the results in this section have applied only to workloads whose accesses are 100% small random writes. This section examines a mixed workload, defined in figure 26, modeled on statistics taken from an airline reservation system [Ramakrishnan92]. With this more general workload, the results of the earlier sections are modified by two important effects: reads and medium to large writes. The issues encountered in extending floating data and parity to handle variable sized access are complex and it is neglected from this section. For the other array configurations reads will perform identically. This will have the effect of compressing the performance differences between configurations. Writes that are not small, however, will hurt the performance of parity logging as is discussed in section 5.1.

Figure 27 presents the results of simulations of four of the array configurations — nonredundant, mirroring, RAID level 5, and parity logging — on this more realistic OLTP workload. With FIFO disk scheduling, used throughout the rest of this paper, parity logging is substantially superior to RAID level 5 and equivalent to mirroring when data caching of writes is effective. With CVSCAN, all configurations delivery higher throughput with lower average response times, but mirroring and nonredundant arrays benefit most. Nonetheless, parity logging remains superior to

11. In many systems, writes to mirrored disks are serialized. One disk in each pair is considered primary, and the write to that disk must complete before the write to the second disk begins. Such serialization would reduce mirroring's disk utilization to the same as the nonredundant case and possibly double response time.

RAID level 5 and is quite close to mirroring when data caching of writes is effective.

Section 9. Multiple failure tolerating arrays

A significant advantage of parity logging is its efficient extension to multiple failure tolerating arrays. Multiple failure tolerance provides much longer mean time to data loss and greater tolerance for bad blocks discovered during reconstruction [Gibson92]. Using codes more powerful than parity, RAID level 5 and its variants can all be extended to tolerate f concurrent failures. Figure 28 gives an example of one of the more easily understood double failure tolerant disk array organizations. This two dimensional parity and the more familiar one dimensional parity used in the rest of this paper are called *binary codes* because a particular bit of the parity depends on exactly one bit from each of a subset of the data disks. If, instead, generalized parity (check information) is computed as a multiple-bit symbol, dependent on a multiple-bit symbol from each of a subset of the data disks, then the code is a *non-binary code* [Macwilliams77, Gibson92]. Non-binary codes can achieve much lower check information space overhead in a multiple failure tolerating array. In particular, a variant of a Reed-Solomon code called "$P+Q$ Parity" has been used in disk array products to provide double failure tolerance with only two check information disks [ATC90].

This paper is not concerned with the choice of codes that might be used for f failure tolerance, except to note that the best of these codes all have one property important to small random write performance [Gibson89]: each small write updates $(f+1)$ disks — f disks containing check information (generalized parity) and the disk containing the user's data. This check maintenance work, which scales up with the number of failures tolerated, is exactly the work that parity logging is designed to handle more efficiently.

Multiple failure tolerating parity logging disk arrays arise as a natural extension of multiple failure tolerating variants of RAID 5. As with single failure tolerating parity logging, the underlying disk array is augmented with a log. However, to maintain f failure tolerance, the log itself must be $f-1$ failure tolerant. One way to achieve $f-1$ failure tolerance is to replicate the log f

times. Figure 29 shows one region of a double fault tolerant parity logging disk array based on a nonbinary code such as "P+Q Parity."

The log management cycle is quite similar to that of a single fault tolerant parity logging disk array. When a region's log buffers fill up, the corresponding parity update records are written once into each of the f logs. When these logs fill up, one copy of the log is read into the reintegration buffer, along with the check information for the region. The updated check information is then rewritten, all log copies are truncated, and the logging cycle starts again.

Mirroring and floating data and parity also extend to multiple failure tolerance in straightforward manner. Mirroring becomes f-copy shadowing [Bitton88]. Floating data and parity becomes floating data and check, requiring f "floated" read-rotate-write accesses per blind write.

Relative to these other schemes, parity logging has better performance because of its lower nonpreread overhead. The overhead associated with maintaining check information can be divided into two components: preread bandwidth overhead and nonpreread bandwidth overhead. The bandwidth needed to preread the old copy of the user's data is independent of the number of failures to be tolerated. Nonpreread bandwidth, the disk work done to update the check information given a data change, grows linearly with the number of failures to be tolerated. Parity logging has the smallest cost for this latter, linearly growing component of check maintenance overhead because all check information accesses (log and generalized parity) are done efficiently.

Figure 30 shows the maximum rate that small random writes can be completed in zero, single, double, and triple failure tolerating arrays using mirroring, RAID level 5, floating data and parity and parity logging. This data is derived from the models of sections 3 and 4 and applied to the example disk array of figure 12.

The maximum I/O rate of the parity logging array declines more slowly than the other configurations because parity logging has a substantially lower nonpreread overhead. For example, while triple failure tolerating parity logging arrays should sustain about 35% of the I/O rate of nonredundant arrays for random small writes, quadruplicated storage (triple failure tolerating mir-

roring disk arrays) will sustain only 25%.

Section 10. Related Work

Bhide and Dias [Bhide92] have independently developed a scheme similar to parity logging. Their LRAID-X4 organization maintains separate parity and parity-update log disks, and periodically applies the logged updates to the parity disk. In order to allow writes from the user to occur in parallel with log reintegration, they duplicate both the parity and the parity log for a total of four overhead disks. This double-buffering scheme, while expensive in disks, can support a fairly large number of data disks without saturating the parity and log disks, so LRAID-X4 does not distribute parity or log information. Instead of breaking down the log disk into regions to reduce the required storage in the controller, LRAID-X4 sorts buffered parity updates in memory according to the parity block to which they apply. This allows LRAID-X4 to write a "run" of updates for ascending parity blocks to a log disk. When this log disk is full, further updates are sorted into runs and written to the second log disk while the first log disk reintegrates its updates with the parity by reading from one parity disk and writing to the other. The reintegration of a full log disk uses an external sorting algorithm to collect subsequences applying to one area of parity from each run on the log disk. If this area is large, all log reads and parity reads and writes will be efficient.

The model derived by Bhide and Dias assumes user data does not need to be preread. It shows that throughput is limited by the rate at which subsequences of runs are collected for reintegration with the parity. In a 100% write workload, the peak throughput is $1/Tseqr$, where $Tseqr$ is the amortized time taken to read a block in a subsequence of a run on the log disk. Bhide and Dias approximate this by

$$(Tfewtrackseek + R + tracksr\,(2R + H)\,) \,/\, (tracksr \times D)$$

where $Tfewtrackseek$ is the time to seek across 5 to 10 tracks and $tracksr$ is the average size in tracks of a subsequence (constrained to one cylinder). While $tracksr$ is dependent on the amount of controller memory, their array achieves about 80% of its maximum throughput with about 2%

of a disk's worth of memory. With this much memory, *tracksr* is 2.4. Using the array parameters in figure 12 and taking *Tfewtrackseek* to be the time of a 5 track seek, one obtains a peak throughput of 624 accesses per second, or an average of 28.4 I/Os per second per disk. With 5% of a disk's worth of memory, LRAID-X4 achieves its maximum of 760 I/Os, or 34.5 I/Os per disk per second. However, LRAID-X4 reaches this performance maximum with 20 disks (16 data, 2 parity, 2 log) for a 100% write workload. Additional disks do not increase performance. In comparison, the parity logging disk array simulated in section 8, whose controller requires about 2% of a disk's worth of memory, is predicted to achieve 36.7 I/Os per disk per second in section 3 on the same workload, and its performance continues to increase with increasing numbers of disks.

The class of less closely related research efforts can be characterized by their use of three techniques that are frequently exploited to improve throughput in disk arrays: write buffering, write-twice, and floating location.

Write buffering delays users' write requests in a large disk or file cache to achieve deep queues, which can then be scheduled to substantially reduce seek and rotational positioning overheads [Seltzer90, Solworth90, Rosenblum91, Polyzois93]. Fault tolerance is at risk in these systems unless fault-tolerant caches are used (nonvolatility is a minimum requirement).

The write-twice approach attempts to reduce the latency of writes without sacrificing the durability advantages of disk storage. Similar to the floating data and parity technique, several tracks in every disk cylinder are reserved, and an allocation bitmap is maintained. When a write is issued, the data is immediately written (typically in a self-identifying manner) to a rotationally close empty location in a reserved track, making the data durable. The write is then acknowledged, but the data is retained in the host or controller and eventually written to its fixed location. When the data has been written the second time, the corresponding bit in the allocation bitmap is cleared. While significant memory may be required for the allocation bitmaps and write buffers, this storage is not required to be fault tolerant, limiting controller cost. Write-twice is typically combined with one of the write buffering techniques to improve the efficiency of the second write. This technique has been pursued most fully for mirroring [Solworth91, Orji93].

The floating location technique improves the efficiency of writes by eliminating the static association of logical disk blocks and fixed locations in the disk array. When a disk block is written, a new location is chosen in a manner that minimizes the disk arm time devoted to the write, and a new physical-to-logical mapping is established. We have described one such scheme, floating data and parity [Menon92], in this paper. An extreme example of this approach is the log structure filesystem (LFS), in which all data is written in a segmented log, and segments are periodically reclaimed by garbage collection [Rosenblum91]. This approach converts all writes into long sequential transfers, greatly enhancing write throughput. However, because logically nearby blocks may not be physically nearby, the performance of LFS in read intensive workloads may be degraded. The distorted mirror approach [Solworth91] uses the 100% storage overhead of mirroring to avoid this problem: half of each mirror writes data in fixed locations, while the other half is used for floating storage, achieving higher write throughput while maintaining data sequentiality. In doubly distorted mirrors, write buffering, write-twice and floating location are all combined to provide high write throughput while maintaining data sequentiality [Orji93]. However, all floating location techniques require substantial host or controller storage for mapping information and buffered data.

Section 11. Concluding Remarks

This paper presents a novel solution to the small write problem in redundant disk arrays based on a distributed (and possibly replicated) log. Analytical models of the peak bandwidth of this scheme and alternatives from the literature were derived and validated by simulation. The proposed technique achieves substantially better performance than RAID level 5 disk arrays. When data must be preread before being overwritten, parity logging achieves performance comparable to floating parity and data without compromising sequential access performance or application control of data placement. Performance is superior to floating parity and data, and on some workloads superior to mirroring as well, when the data to be overwritten is cached. This performance is obtained without the 50% disk storage space overhead of mirroring. For extremely reli-

able environments, the advantage of parity logging systems is shown to be even more pronounced.

While the parity logging scheme presented in this paper is effective, several optimizations should be explored. Detailed simulations of multiple failure tolerating configurations should be undertaken. More dynamic assignment of controller memory should allow higher performance to be achieved or a substantial reduction in the amount of memory required. Application of data compression to the parity log should be very profitable. A detailed comparison of the log structured filesystem [Rosenblum91], which completely avoids small writes, and parity logging should be undertaken. The interaction of parity logging and parity declustering [Holland92] merits particular exploration. Parity declustering provides high performance during degraded-mode and reconstruction while parity logging provides high performance during fault-free operation. The combination of the two should provide a particularly cost effective system for OLTP environments.

References

[ATC90] Array Technology Corporation. *Product Description, RAID+ Series Model RX, Revision 1.0,* Array Technology Corporation, 1990.

[Bhide92] Bhide, A., and Dias, D. RAID Architectures for OLTP. Computer Science Research Report RC 17879, IBM Corporation, 1992.

[Bitton88] Bitton, D., and Gray, J. Disk Shadowing. In *Proceedings of the 14th Conference on Very Large Data Bases* (1988), 331-338.

[Cao93] Cao, P., Lim, S. B., Venkataraman, S., and Wilkes, J. The TickerTAIP Parallel RAID Architecture. In *Proceedings of the 20th Annual International Symposium on Computer Architecture* (May 1993). IEEE, San Diego, 52-63.

[Chen90] Chen, P. M., and Patterson, D. A. Maximizing Performance in a Striped Disk Array. In *Proceedings of the 17th Annual International Symposium on Computer Architecture* (1990), 322-331.

[Gibson89] Gibson, G., Hellerstein, L., Karp, R. M., Katz, R. H., and Patterson, D. A. Coding

Techniques for Handling Failures in Large Disk Arrays. In *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)* (1989). ACM, 123-132.

[Gibson92] Gibson, G. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*, MIT Press, 1992.

[Gibson93] Gibson, G., and Patterson, D. Designing Disk Arrays for High Data Reliability. *Journal of Parallel and Distributed Computing* (January, 1993), 4-27.

[Geist87] Giest, R. M., and Daniel, S. A Continuum of Disk Scheduling Algorithms. *ACM Transactions on Computer Systems*, 5, 1 (1987), 77-92.

[Gray90] Gray, J., Horst, B., and Walker, M. Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput. In *Proceedings of the 16th International Conference on Very Large Databases (VLDB)* (1990), 148-161.

[Hewlett-Packard93] Hewlett-Packard Company, *HP C2247 3.5-inch Disk Drive Product Brief, 5091-2788E.* Hewlett-Packard Company, 1993.

[Holland92] Holland, M., and Gibson, G. Parity Declustering for Continuous Operation in Redundant Disk Arrays. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)* (1992). ACM, 23-35.

[Holland93] Holland, M., Gibson, G. A., and Siewiorek, D. Fast, On-line Recovery in Redundant Disk Arrays, *Proceedings of the International Symposium of Fault Tolerant Computing* (1993), 422-431.

[IBM0661] IBM Corporation, *IBM 0661 Disk Drive Product Description, Model 370, First Edition, Low End Storage Products, 504/114-2.* IBM Corporation, 1989.

[Jones91] J. Jones, J. Jr., and Liu, T. RAID: A Technology Poised for Explosive Growth. *Montgomery Securities Industry Report.* Montgomery Securities, San Francisco, 1991.

[Lee91] Lee, E., and Katz, R. Performance Consequences of Parity Placement in Disk Arrays. In *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IV)* (1991). ACM, 190-199.

[MacWilliams77] MacWilliams, F. J., and Sloane, N. J. A. *The Theory of Error-Correcting Codes*, North-Holland Mathematical Library, Volume 16, Elsevier Science Publishing Company, New York, NY, 1977.

[Menon92] Menon, J., and Kasson, J. Methods for Improved Update Performance of Disk Arrays. In *Proceedings of the Hawaii International Conference on System Sciences* (1992), 74-83.

[NCR93] NCR Corporation, *NCR ADP 93-02 Disk Array Controller Manual, MS-0025*, NCR Corporation, 1993.

[Orji93] Orji, C. U., and Solworth, J. A. Write-only disk caches. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (May 1993). ACM, 307-316.

[Ousterhout88] J. Ousterhout, et. al. The Sprite Network Operating System. *IEEE Computer* (February 1988). IEEE, 23-36.

[Patterson88] Patterson, D., Gibson, G., and Katz, R. A Case for Redundant Arrays of Inexpensive Disks (RAID). In *Proceedings of the ACM SIGMOD Conference* (1988). ACM, 109-116.

[Rosenblum91] Rosenblum, R. and Ousterhout, J. The Design and Implementation of a Log-Structured File System. In *Proceedings of the 13th ACM Symposium on Operating System Principles* (1991). ACM, 1-15.

[Polyzois93] Polyzios, C. A., Bhide, and A., Dias, D. M. Disk Mirroring with Alternating Deferred Updates. In *Proceedings of the 19th International Conference on Very Large Databases (VLDB)* (1993), 604-617.

[Ramakrishnan92] Ramakrishnan, K. K., et. al. Analysis of File I/O Traces in Commercial Computing Environments. In *Performance Evaluation Review (SIGMETRICS).* 20, 1 (1992). ACM, 78-90.

[Schulze89] Schulze, M. E., Gibson. G. A., Katz, R. H., and Patterson, D. A. How Reliable is a RAID? In *Proceedings of the 1989 IEEE Computer Society International Conference (COMPCON 89)* (1989). IEEE, 118-123.

[Seagate92] Seagate Corporation, *Seagate ST3600N/ND Family SCSI-2 Product Manual, Volume 1, 77738477-A*. Seagate Corporation, 1992.

[Solworth90] Solworth, J. A. and Orji, C. U. Distorted Mirrors. In *Proceedings of the ACM SIGMOD Conference* (1990). ACM, 123-132.

[Solworth91] Solworth, J. A. and Orji, C. U. Write-only disk caches. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems* (Dec. 1991). IEEE, 10-17.

[Salem86] Salem, K., and Garcia-Molina, H. Disk Striping. In *Proceedings of the 2nd IEEE International Conference on Data Engineering* (1986). IEEE, 1986.

[TPCA89] *The TPC-A Benchmark: A Standard Specification*, Transaction Processing Performance Council, 1989.

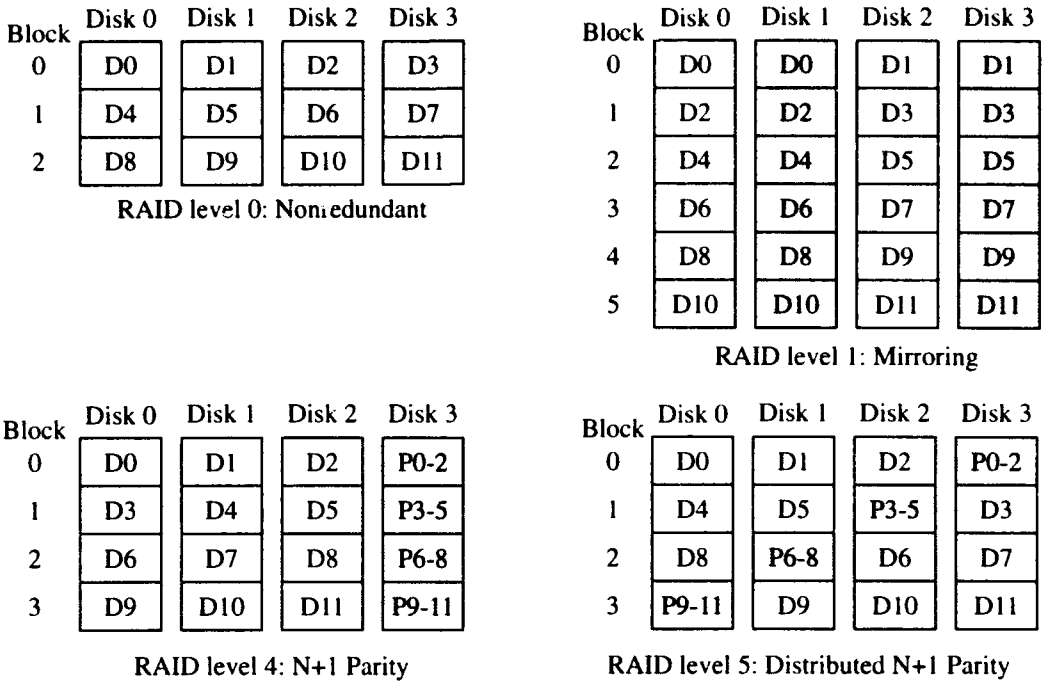


Figure 1 ***Data Layouts****. In nonredundant disk arrays, data units are simply interleaved across the array. RAID level 1 arrays duplicate every user data unit. RAID level 4 arrays interleave user data blocks across all disks except one. Blocks on the final disk hold the parity (bitwise xor) of the corresponding blocks on the other disks. RAID level 5 arrays distribute the parity blocks uniformly across the disk array. Shaded blocks indicate redundant (parity) information.*

TPC Benchmark

get request from terminal
begin transaction
    update account record
    write history log
    update teller record
    update branch record
commit transaction
respond to terminal

Scaling Requirements

Record Type	Minimum Quantity per TPS	Record Size (Bytes)
Account	100K	100
Teller	10	100
Branch	1	100
History	30K	50
Total		11.5 MB

Figure 2 ***OLTP Workload Example****. The transaction processing council (TPC) benchmark is an industry standard benchmark for OLTP systems stressing update-intensive database services [TPCA89]. It models the computer processing for customer withdrawals from and deposits to a bank. The primary metric for TPC benchmarks is transactions per second (TPS). Systems are required to complete 90% of the transactions in under 2 seconds and to meet the scaling constraints listed above. Customer account records are selected at random from the local branch 85% of the time, and from a different branch 15% of the time. Because history record writes are delayed and grouped into large sequential writes and teller and branch records are easily cached, the disk I/O from this benchmark is dominated by the random account record update. For a 250 TPS system, at least 2.8GB of storage must concurrently provide more than 250 account record reads and writes per second.*

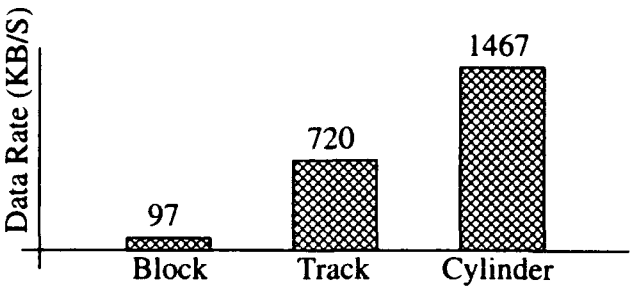


Figure 3 ***Peak I/O Bandwidth.*** *The figure shows the total kilobytes per second that can be read from or written to an IBM 0661 drive using random one block (2KB), one track (24 KB), and one cylinder (336KB) accesses (see figure 12 for disk parameters).*

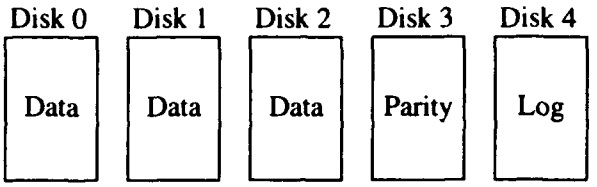


Figure 4 ***Basic Parity Logging Model****. A RAID level 4 disk array is augmented with a log disk. Parity update records are written sequentially to the log disk at track transfer rates. A full log disk triggers a read of the log and parity disks, computation of the current parity, and a rewrite of the parity disk.*

B	Number of regions per disk
C_D	Cylinders of data per region
C_L	Cylinders of log per region
C_P	Cylinders of parity per region
D	Data units per track
H	Head switch time
K	Tracks buffered per region
L	Log striping factor
M	Single track seek time
N	Number of disks in array
R	Average rotational delay (1/2 disk rotation time)
S	Average seek time
T	Tracks per cylinder
V	Cylinders per disk

Figure 5 ***Model Parameters****. The bandwidth utilization models of section 2,3 and 4 are presented in terms of the parameters list above. The majority of these parameters are based on disk geometry, while the remainder come from applications or array configuration. The left hand column indicates the symbol used in this text.*

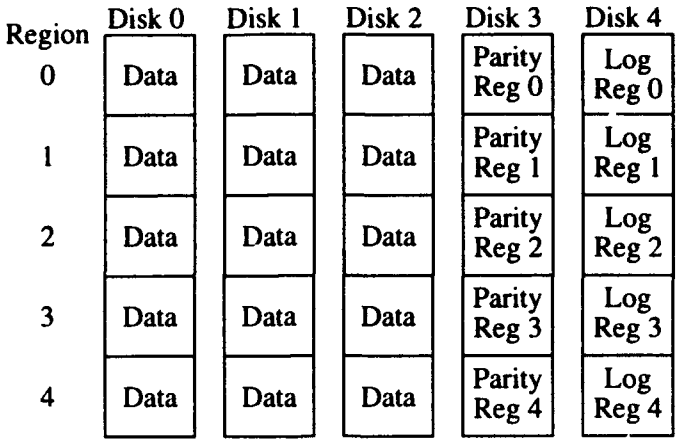


Figure 6 ***Parity Logging Regions****. Dividing each disk into regions dramatically reduces the required amount of controller buffer space. Each region requires a track buffer to hold its unwritten log records. When a track buffer fills up, the track is written into its region's log with a full track write.*

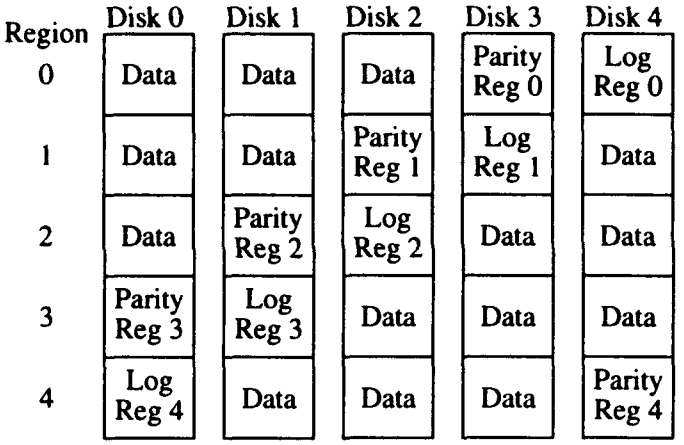


Figure 7 ***Log and Parity Rotation****. Spreading the log and parity over the entire array increases the parity and log bandwidth to the entire bandwidth of the array. An individual region may still be a hot spot.*

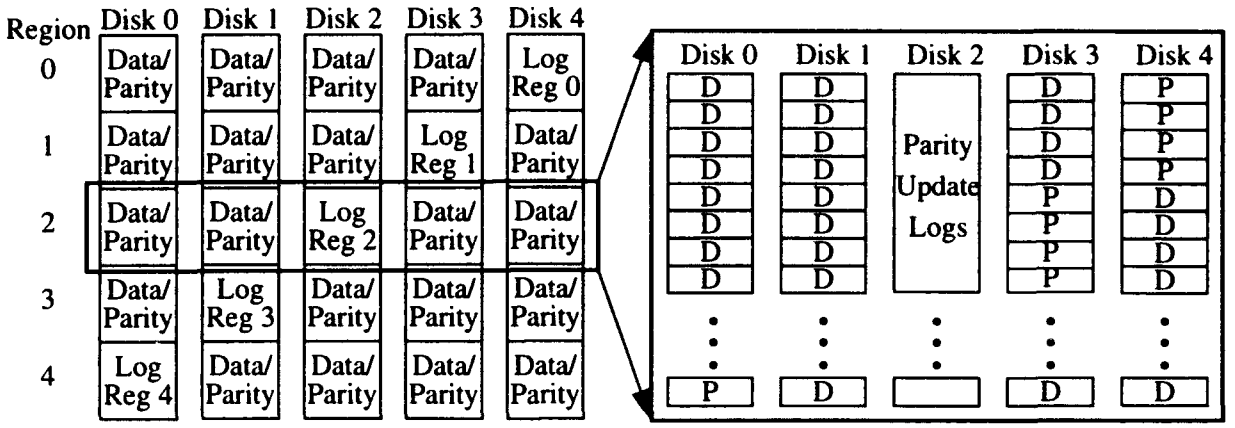


Figure 8 ***Block Parity Striping****. Parity and data are distributed over all but one disk in each region. The remaining disk contains the parity log. A contiguous layout of parity on each disk allows efficient cylinder-rate transfers, while distribution reduces the latency of parity reintegration. The inset shows a detailed layout of a sample region.*

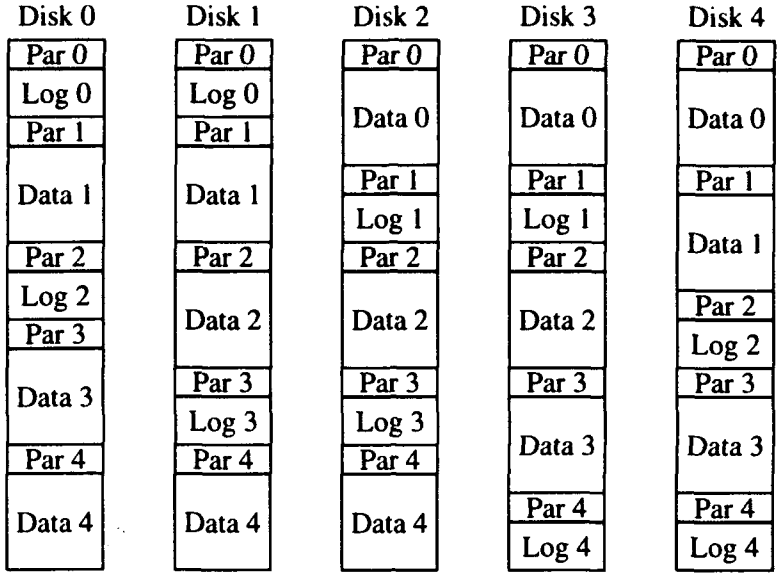


Figure 9 ***Distributed Parity Logs****. To increase the log bandwidth for each region, the log for each region can also striped. In this example, each log region is striped over two disks. As before, the parity is still spread over all disks. To preserve single fault tolerance, a parity sublog for a region cannot reside on the same disk as any data for that region. Thus, while striping reduces the time for log reintegration for a given region, it increases the space overhead. In addition, if the log is striped over too many disks, the sublogs will become too small and access to them will be inefficient, decreasing performance. When the log is not striped, however, many user data requests queue behind the log reads, which degrades throughput and response time. Fortunately, a moderate degree of striping, while increasing cost my a small amount, substantially improves performance.*

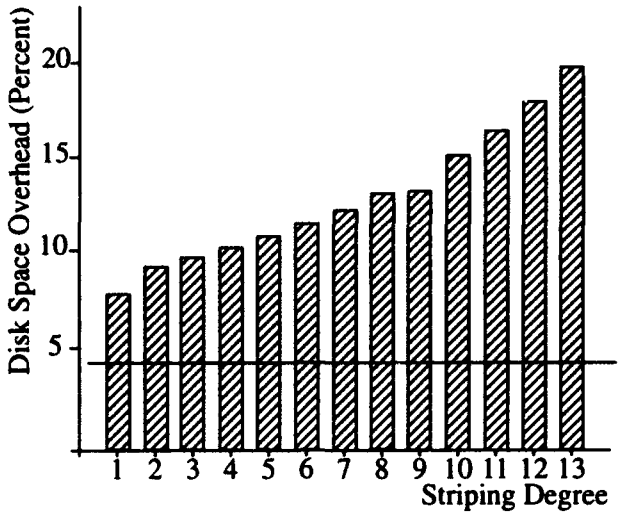


Figure 10 ***Disk Storage Overheads****. While increasing the log striping degree improves array performance, the storage overhead increases. Shown above is the percent of the total disk capacity devoted to storing redundant data in an array with 22 disks. In general, the storage overhead is 2/(N+1-L), where N is the number of disks and L is the log striping degree. Thus, the storage overhead depends only on the total number of disks in the array and the degree of log striping. The horizontal line shows the capacity overhead of a RAID level 5 configuration of the same array.*

In addition to these disk space overheads, parity logging also requires buffer in the host or controller. With the example disk array of figure 12, 5600KB of buffer is required, roughly equivalent in cost to 2% of the disk array.

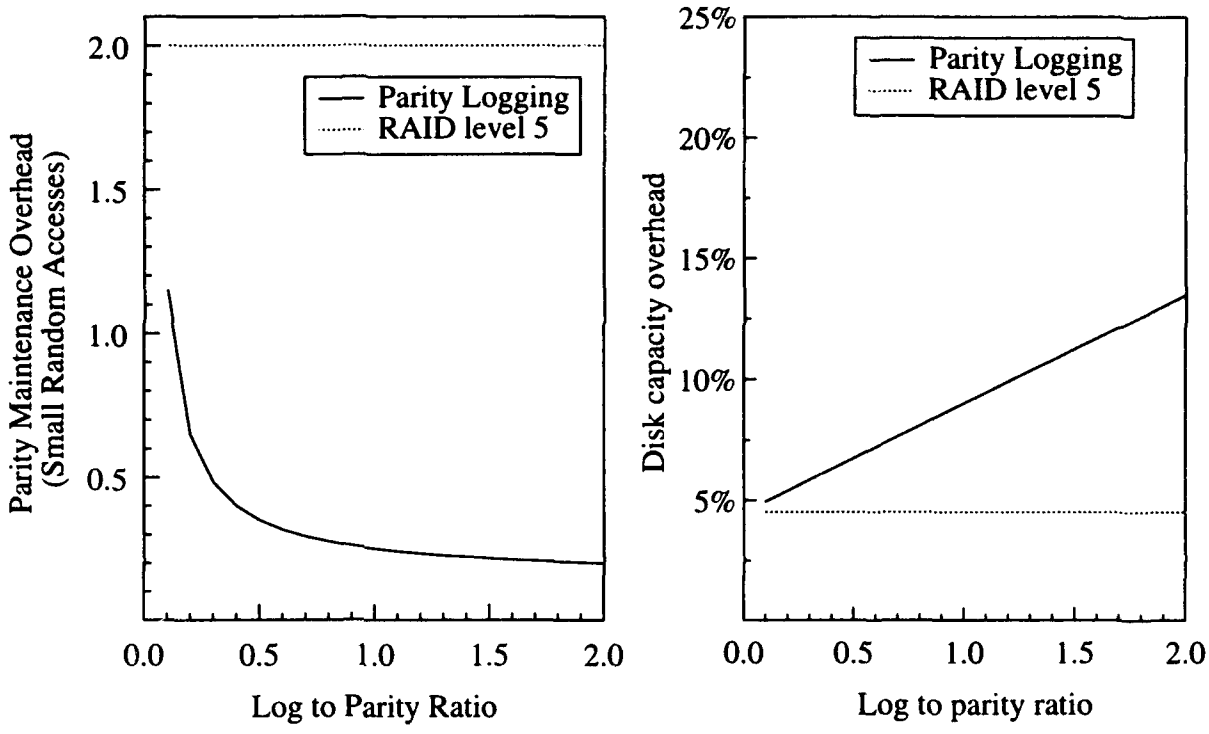


Figure 11 ***Log length and efficiency.*** *As the ratio of log to parity increases, the amount of disk arm time devoted to parity maintenance per small user write decreases. These two graphs show this tradeoff for the 22 disk array described in figure 12. The graph on the left shows the disk arm time overhead per small user write for I/Os devoted solely to parity maintenance for parity logging. Also shown is the overhead of RAID level 5. Parity logging asymptotically approaches an overhead that is 15% of a small random access, and achieves 30% of the overhead of a small random access with a log to parity ratio of 0.7. The graph on the right shows the overhead for parity and log space in parity logging and RAID level 5 as function of the ratio between log and parity space.*

Workload Parameters	
Access size:	Fixed at 2 KB
Alignment:	Fixed at 2 KB
Write Ratio:	100%
Spatial Distribution:	Uniform over all data
Temporal Distribution:	66 closed loop processes Gaussian think time distribution
Array Parameters	
Stripe Unit:	Fixed at 2KB
Number of Disks:	22 spindle synchronized disks.
Head Scheduling:	FIFO
Power/Cabling:	Disks independently powered/cabled
Disk Parameters	
Geometry:	949 cylinders, 14 heads, 48 sectors/track
Sector Size:	512 bytes
Revolution Time:	13.9 ms
Seek Time Model:	$2.0 + 0.01 \cdot dist + 0.46 \cdot \sqrt{dist}$ (ms) 2 ms min, 12.5 ms avg, 25 ms max
Track Skew:	4 sectors
Head Switch Time:	1.16 ms

Figure 12 ***Simulation Parameters.*** *The access size alignment and spatial distribution are not unlikely in OLTP workloads, while a 100% write ratio emphasizes the performance differences of the various techniques. Since the disks have independent support hardware, disk failures will be independent, allowing a single parity group [Gibson92] to be single fault tolerant. Disk parameters are modeled on the IBM Lightning drive [IBM0661]. Note that the* dist *term in the seek time model is the number of cylinders traversed, excluding the destination. As is commonly done in SCSI disks, the track skew is chosen to equal the head switch time, optimizing data layout for sequential multitrack access. These disks do not support zero latency writes.*

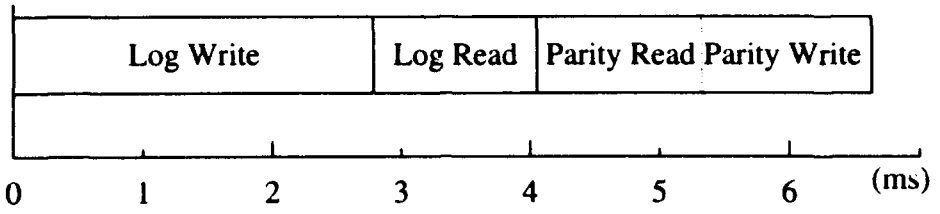


Figure 13 ***Parity Logging Overheads.*** *The amortized overhead cost of extra I/Os done in our example parity logging array is shown above. The log writes contribute approximately 40% of the overhead (2.78 milliseconds), while the cylinder rate log reads, parity reads and parity writes each contribute about 20% (1.29 milliseconds). For comparison, the extra I/Os done by RAID level 5 cost nearly 35 milliseconds per small write.*

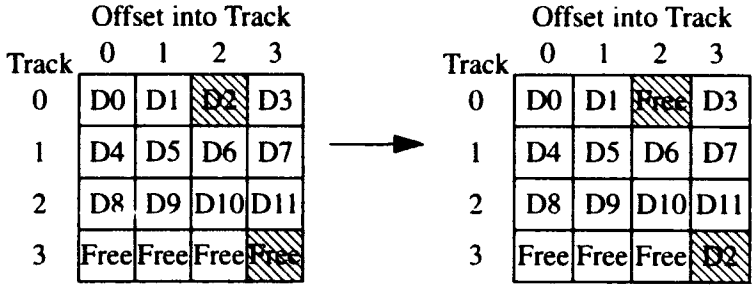


Figure 14 ***Floating Data/Parity****. When updating block D2, the controller searches for a free block within the cylinder that is rotationally close to block D2. In this case, it finds the block at offset 3 into track 3. Immediately following the preread of block D2, the controller writes the new block to the new location, and updates the mapping tables. The preread of the old information and the write of the new are thus effectively done in the time of one access.*

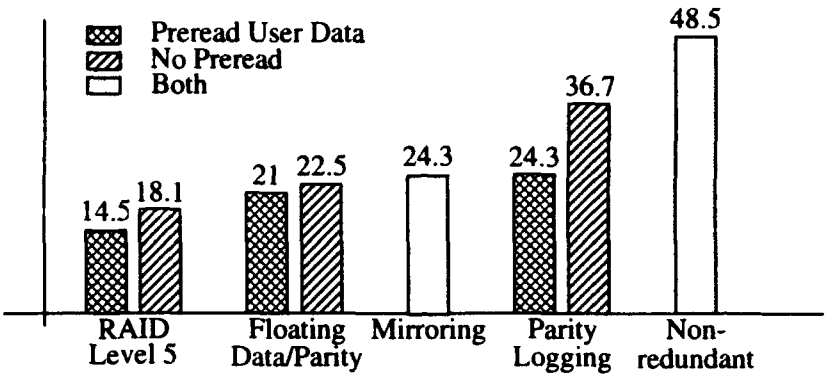


Figure 15 ***Model Estimates****. I/Os per second per disk as predicted by the bandwidth models of section 3 and 4. These predictions assume 100% disk utilization, FIFO disk arm scheduling and an unbounded number of requestors. RAID level 5 and parity logging disk arrays both benefit substantially from not having to preread user data. Floating data and parity substantially reduces the overhead of the user preread and therefore achieves less benefit from its elimination. Mirroring and nonredundant disk arrays do not need to preread user data. Since these predictions assume 100% disk utilization, the parity logging estimates are insensitive to the degree of striping.*

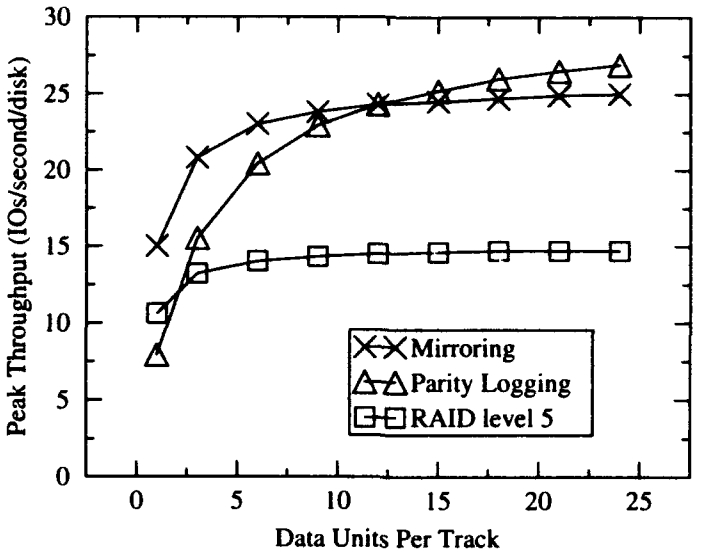


Figure 16 ***Effects of track size on throughput.*** *The performance of parity logging is highly sensitive to the number of data units per track. The figure above shows the performance of mirroring, parity logging (unstriped log), and RAID level 5 on a workload of 100% blind small writes for varying number of data units per track. Parity logging achieves better performance than mirroring when there are at least thirteen data units per track.*

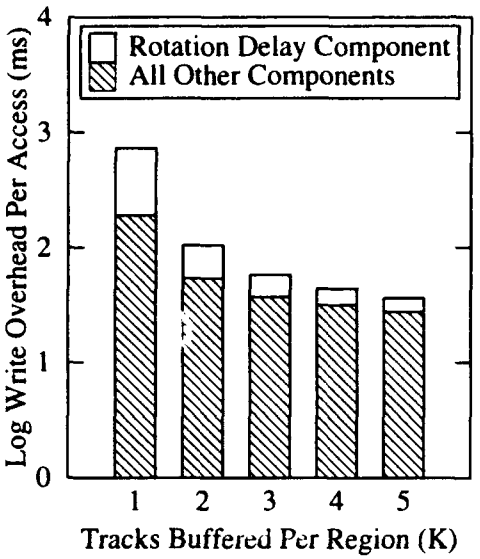


Figure 17(a): Log write overhead

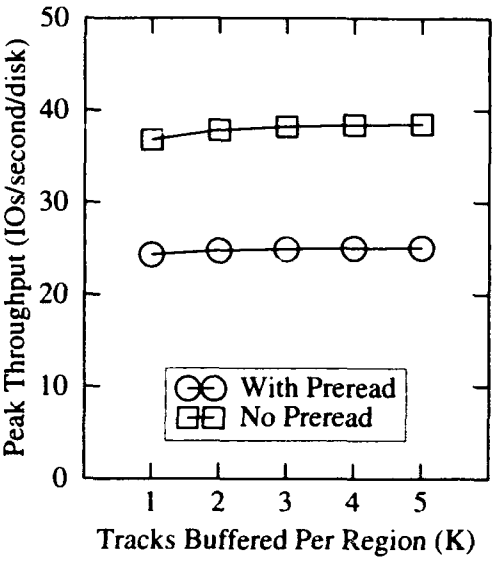


Figure 17(b): Peak throughput

Figure 17 ***Varying the number of tracks buffered per region.*** *The log write overhead declines as the number of tracks buffered per region increases, asymptotically approaching the performance of cylinder rate transfers. While zero latency writes can eliminate the rotational delay component of the log write overhead, this component is negligible if more than a single track is buffered.*

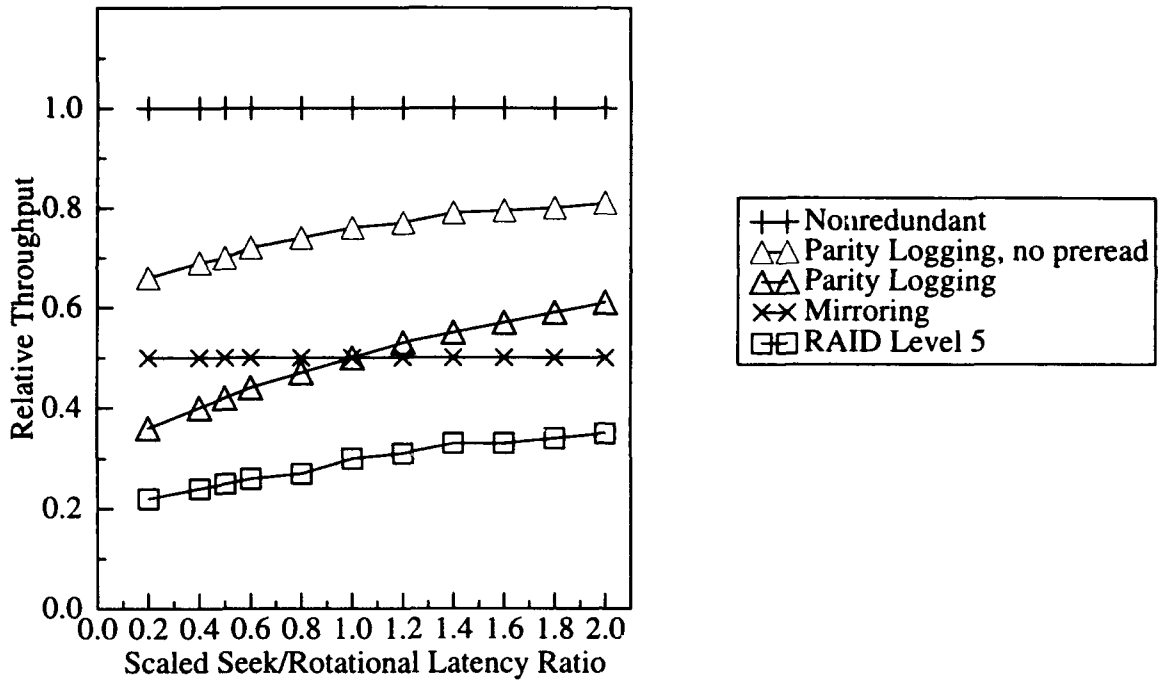


Figure 18 ***Peak throughput normalized to nonredundant array performance as a function of the ratio of average seek time to rotational latency.*** *Altering the ratio of the average seek time to the rotational period of the disk changes the relative performance of mirroring, floating data and parity, nonredundant and parity logging disk arrays. Shown above is the relative performance of these approaches on the example 22 disk array (figure 12) as the average seek time for the drive is varied. The average seek time is varied from 20% of the Lightning average seek to twice that of the Lightning average seek. This parameter range models a large spectrum of drives, from those with very fast positioning to Lightning-like drives spinning at 7200 RPM. When the scaled seek time is 1.0, the ratio of average seek time to rotational latency is the same as the Lightning drive.*

Mirroring spends the same fraction of time seeking during a small write as a nonredundant access, so the relative performance of mirroring is constant with respect to that of the nonredundant array. RAID level 5 and parity logging spend a smaller fraction of time seeking during a small write relative to a nonredundant write. Thus, the relative performance of RAID level 5 and parity logging increases as the ratio of seek time to the rotational period increases.

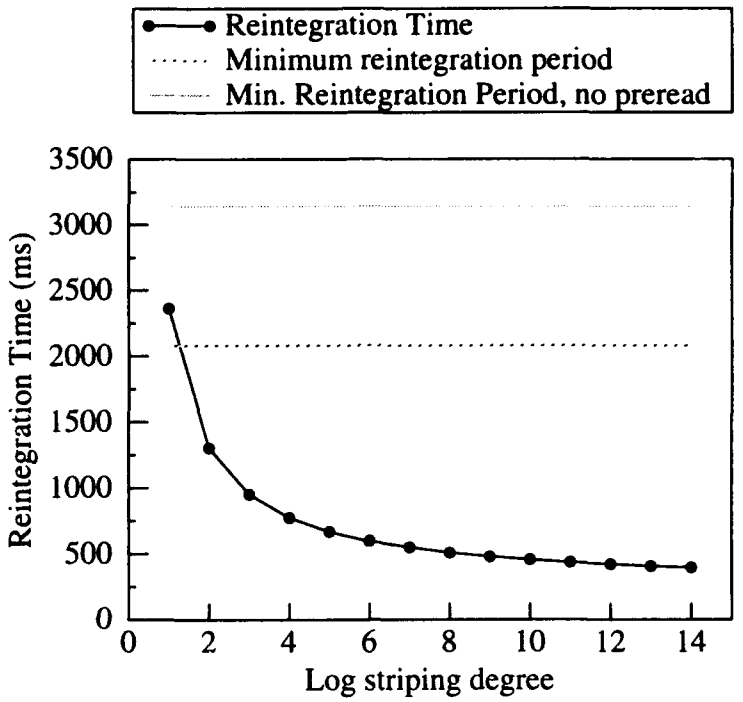


Figure 19 ***Reintegration latency.*** *The time to reintegrate a region's log into its parity depends strongly upon the degree of log striping. The above figure compares reintegration latency for various log striping degrees. Also shown is the minimum average period between reintegrations for workloads of 100% small writes in both the preread required and cached cases assuming 100% disk utilization.*

In particular, consider the case of a parity logging disk array with an unstriped log and effective data caching (the no preread case). Since the reintegration time is longer than the minimum average period between reintegrations, the reintegration buffer can become a performance bottleneck. With an open arrival process (that is, an infinite number of requesting user processes), logs will become full faster than they can be emptied, leading to exhaustion of the controller buffer pool. At this point, all write accesses will be forced to wait, leading to an infinitely growing queue depth. With a closed arrival process (fixed number of processes executing a request/response loop), this bottleneck will manifest by high response time variance, drive underutilization, and degraded throughput.

	Mirroring	Parity Logging	Nonredundant
User capacity	50%	90%	100%
MTTDL	466,683 years	22,226 years	13,636 hours
5 year reliability	99.999%	99.978%	4%
10 year reliability	99.998%	99.955%	less than 2%

Figure 20(a): Large pool of spare disks

	Mirroring	Parity Logging	Nonredundant
User capacity	50%	90%	100%
MTTDL	389,688 years	18,568.4 years	13,636 hours
5 year reliability	99.999%	99.973%	4%
10 year reliability	99.997%	99.946%	less than 2%

Figure 20(b): Single spare disk

Figure 20 ***Disk array reliability.*** *Modern disks have mean times to failure ($MTTF_{disk}$) in excess of a quarter million hours, allowing construction of extremely reliable single fault tolerant arrays. With a $MTTF_{disk}$ of 300,000 hours and a mean repair time ($MTTR_{disk}$) of one hour, the 5 and 10 year reliability of both a parity logging and a mirrored array of 22 disks is greater than 99.95% with a large pool of spare disks. In the more plausible scenario of a single on-line spare and a 3 day (72 hour) spare replacement time, the 10-year reliability of both arrays still exceeds 99.94%. Despite a greater than 34 year mean lifetime of a single disk, the chance of a nonredundant 22 disk array surviving at least 5 years is less than 1 in 20, emphasizing the need for fault tolerant storage.*

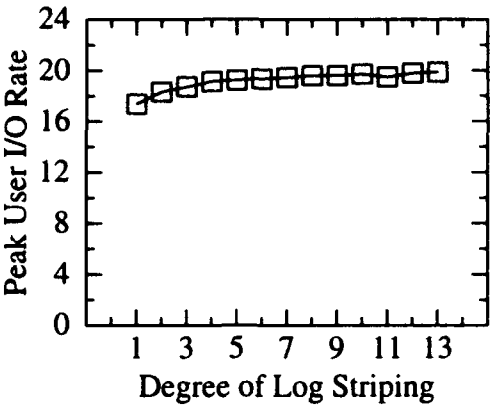


Figure 21(a): Peak user I/Os

Figure 21 ***Striped Parity Logging****. Figures 21(a), (b), and (c) show the achieved user I/Os per disk per second, average user response time, and the standard deviation of the response time under peak load for various degrees of parity log striping. All metrics improve substantially as the striping degree is increased from one (no striping) to four. The difference in performance between striping over 4 to 13 disks is slight, indicating the robustness of the technique.*

The metric with the most dramatic improvement is the response time standard deviation. When unstriped log reads are long (see figure 22), many user requests become queued for that disk, leading to a large variance in the response time. Striping reduces the length of the log reads, thereby reducing this variance.

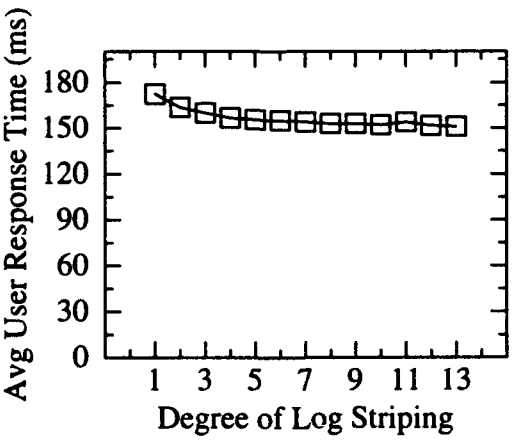


Figure 21(b): Response time at peak load

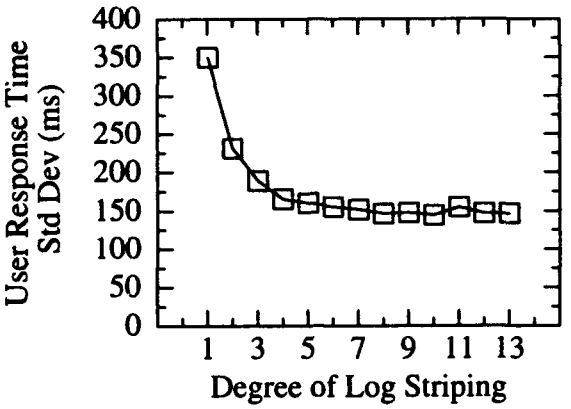


Figure 21(c): Response time standard deviation at peak load

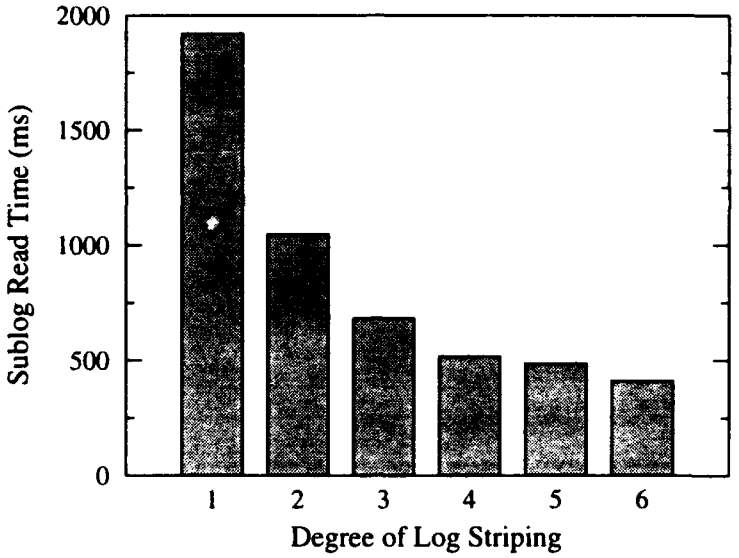


Figure 22 ***Sublog Read Times****. This figure presents the time taken to read a sublog for low degrees of log striping for the example disk array. When the sublog reads are very long, many user requests queue for the disk that is servicing the sublog read, increasing response time and decreasing array utilization and throughput.*

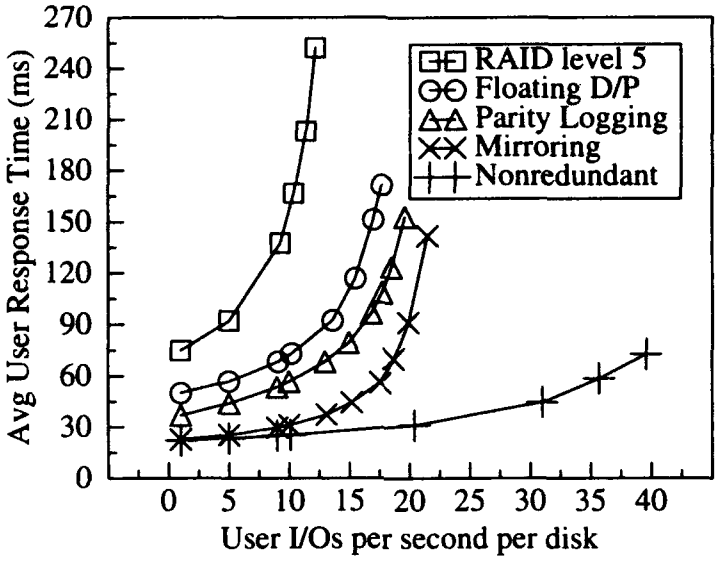


Figure 23(a): Response times

Figure 23 ***Response Times and Utilization.*** *Figures 23(a)-(c) present the average user response times and response time standard deviations as a function of the number of small random writes achieved per disk per second. Figures 23(a) and (b) present the results when the user data must be preread, while the results in figure 23(c) assume the user data was cached, making the preread of the user data unnecessary. In addition to reducing the amount of I/O required, cached user data allows the user write and parity update to occur concurrently, significantly reducing response time for RAID level 5 and floating data and parity. The reported times are in milliseconds. The response time standard deviation for the no preread case is essentially identical to figure 23(b).*

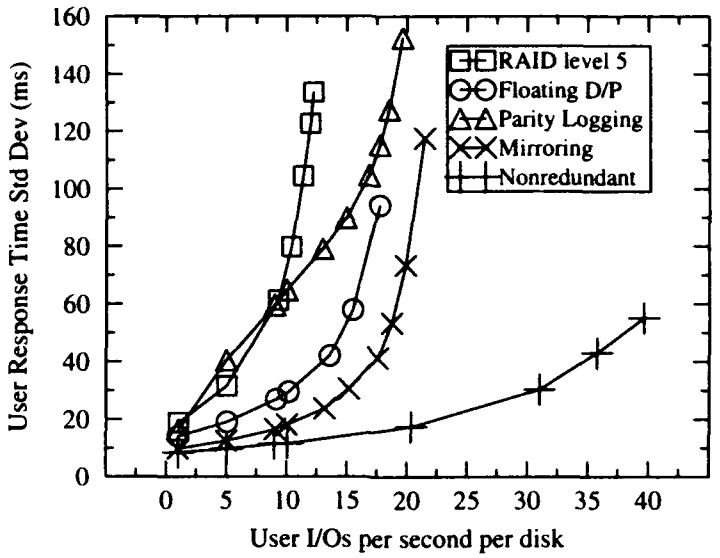


Figure 23(b): Response time standard deviation

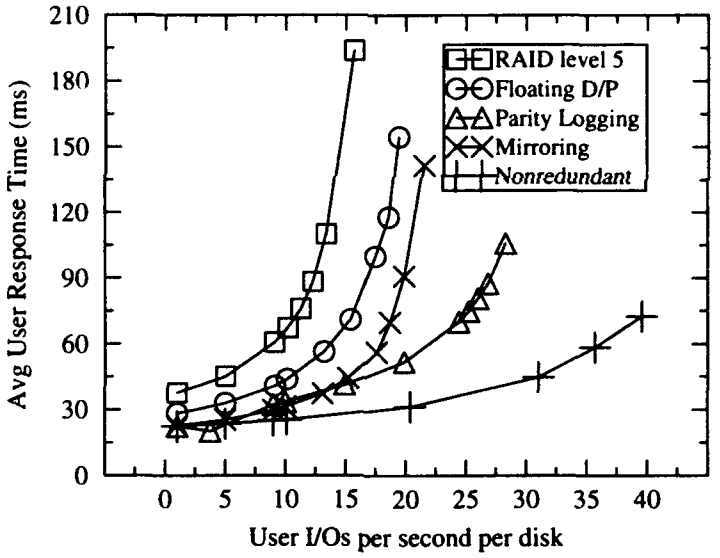


Figure 23(c): Response times without prereads

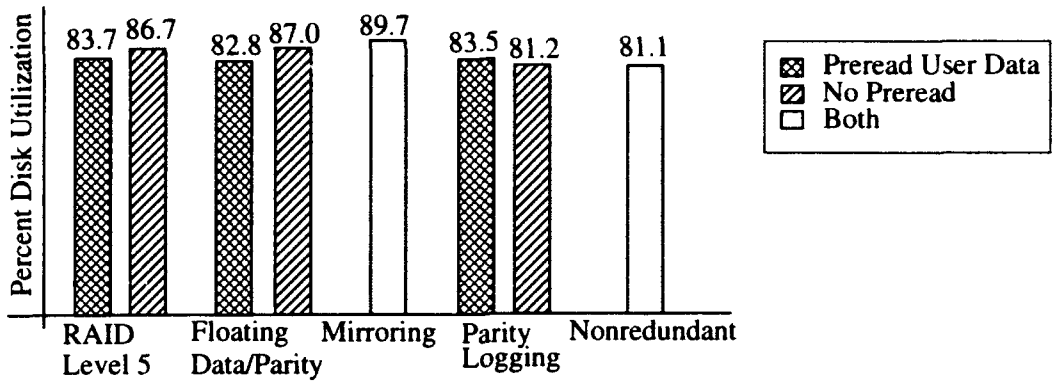


Figure 24 ***Disk Utilization at Peak Load.*** *The figure above presents the average disk utilization at maximum load for the array simulated in figure 23.*

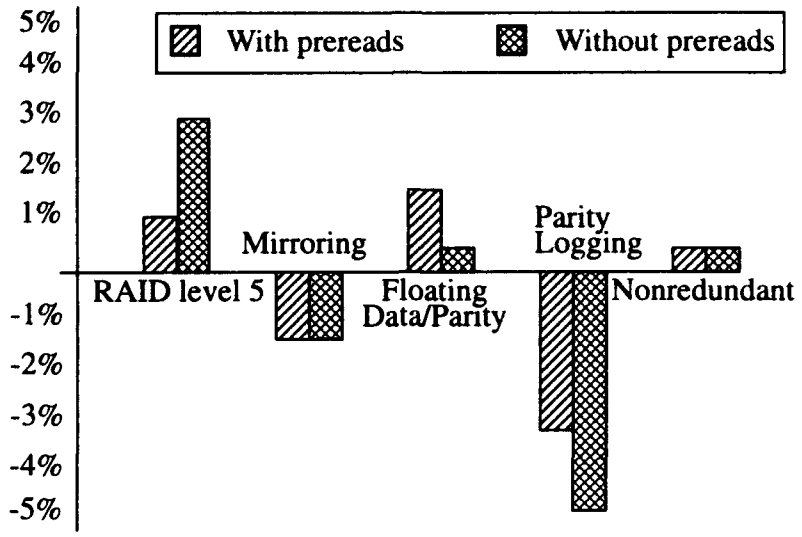


Figure 25 ***Model errors****. This figure shows the percent error between the models of sections 3 and 4 and the simulations of section 8. The model predictions have been scaled by the achieved disk utilizations of figure 24. In all cases, the disagreement between the simulation and the models is less than 5 percent. Note that the 95% confidence interval on the simulation response time is also ±5% of the mean.*

I/O Type	% of Workload	I/O Size (KBytes)
Read	20	1
Read	20	2
Read	33	4
Read	9	24
Write	9	1
Write	7	8
Write	2	24

Figure 26 ***Airline reservation workload.*** *The I/O distribution shown above was selected to agree with general statistics from an airline reservation system [Ramakrishnan92]. This workload is reported as approximately 82% reads, a mean read of 4.61KB, and a median read of 3KB. The mean write size was larger, 5.71 KB, but the median write was smaller, 1.5KB. Locality of reference and overwrite percentages were not reported. All accesses are assumed to occur on their natural boundaries.*

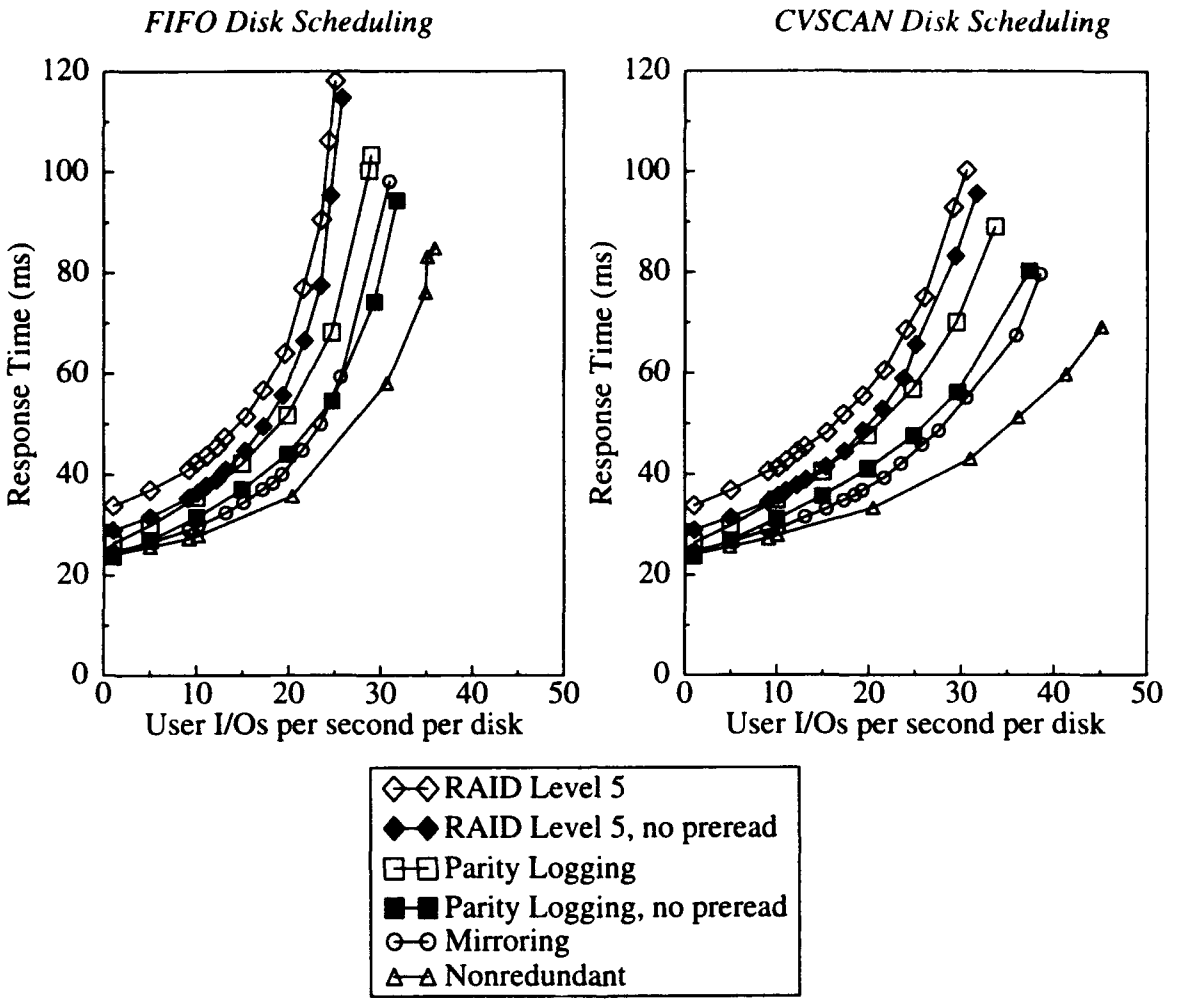


Figure 27 ***Airline reservation simulation.*** *Shown above are the results of simulation using the access size distribution of figure 26. The access distribution is uniform throughout the 22 disk array (figure 12). For all configurations, the data stripe unit was 24KB, so no access spans more than a single drive. For RAID level 5 and parity logging, results are shown both for the case where all writes are blind, and when the old data for all writes is cached (no preread). While CVSCAN [Geist87] scheduling improves throughput and response of all workloads, mirrored and nonredundant disk arrays improve the most, since seek time is a larger proportion of their underlying I/Os.*

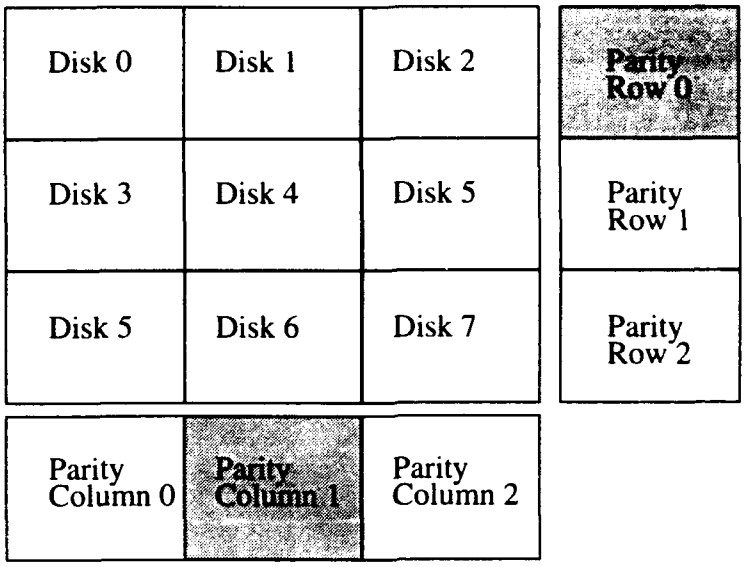


Figure 28 ***Two dimensional parity****. One disk array organization that achieves double failure tolerance is two dimensional parity. Parity disks hold the parity for the corresponding row or column. In the example above, the parity disk for column 0 holds the parity of disks 0, 3 and 5. Whenever a unit in a data disk is written, the corresponding units in both row and column parity disks are also updated. Thus a write to disk 1, in the example above, would require updating the parity on the shaded parity disks, parity row 0 and parity column 1.*

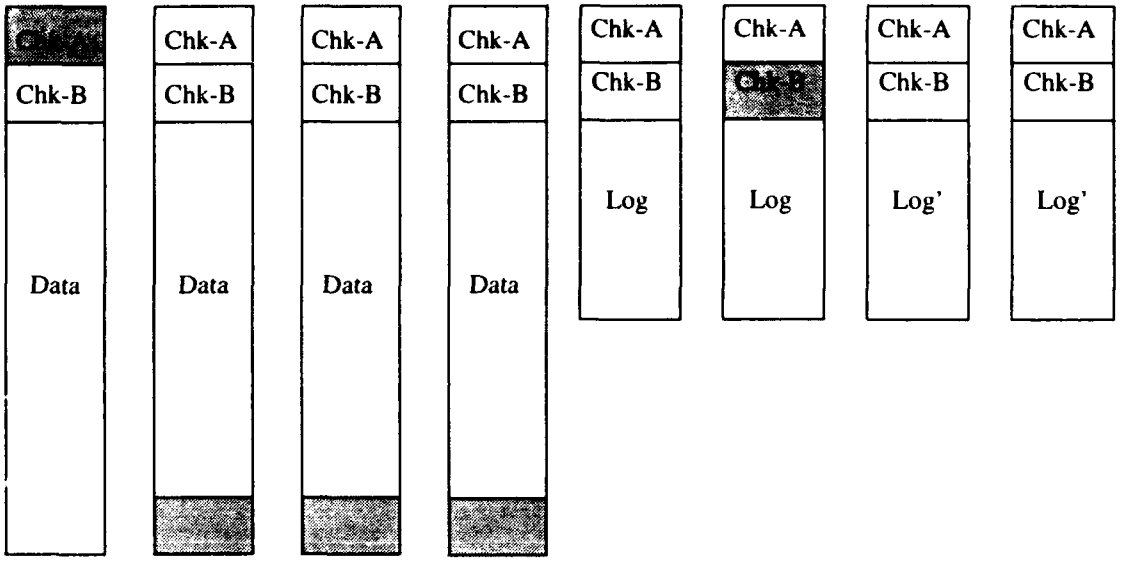


Figure 29 ***Nonbinary coded double failure tolerant parity logging disk array.*** *By using nonbinary codes, disk arrays can achieve double failure tolerance with only two disks of check data. Shown above is a single region of double fault tolerant parity logging disk array with nonbinary check information. The parity of a single fault tolerant array is replaced with two sets of check information. The shaded area shows an example pair of check information blocks and the data blocks that they protect.*

To achieve double fault tolerance in such a parity logging array, the striped log for each region is duplicated. In the picture above, each log is striped over two disks. Note that the contents of this duplicated log are identical and are not associated with a particular copy of the check information.

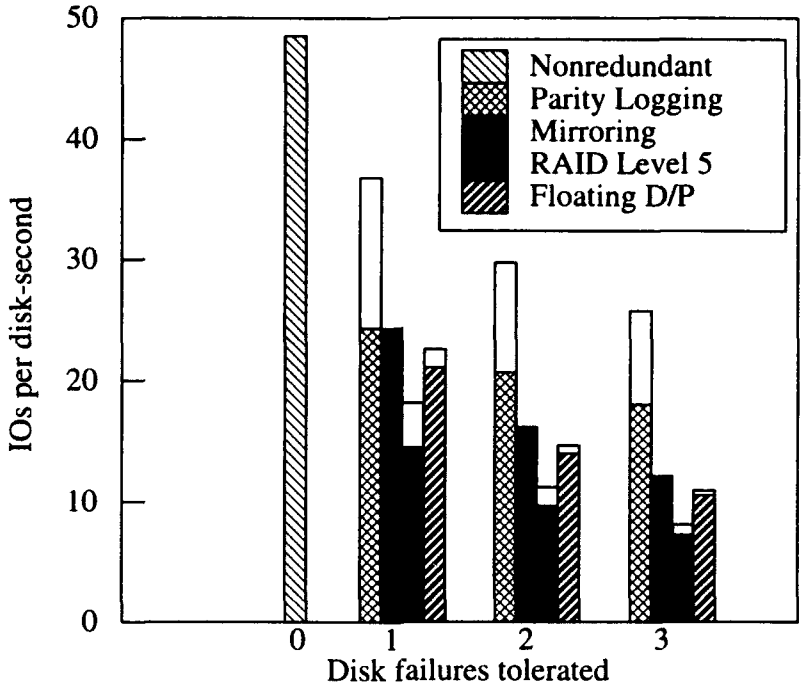


Figure 30 ***Performance of multiple failure tolerating arrays.*** *While the performance of all array configurations declines with the number of failures tolerated, parity logging declines the least, decreasing in performance by about 15% per degree of failure tolerated. The highest performing alternative, mirroring, has a huge disk space overhead, requiring 3 or 4 disks per disk of user data in the double and triple failure tolerating cases, respectively. The performance of RAID level 5 and floating data and parity both decline rapidly, achieving less than 10 user writes per second in the triple failure tolerating case. The shaded portion of each bar shows the performance when the data to be rewritten is not cached, while the full height indicates performance when data is cached.*