Appears in the 20th Annual International Symposium on Computer Architecture, May 16-19, 1993, San Diego CA.

Parity Logging
Overcoming the Small Write Problem in Redundant Disk Arrays

Daniel Stodolsky, Garth Gibson, and Mark Holland

School of Computer Science and Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890
Daniel.Stodolsky@cmu.edu

Abstract

Parity encoded redundant disk arrays provide highly reliable, cost effective secondary storage with high performance for read accesses and large write accesses. 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 cost 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.

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 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].

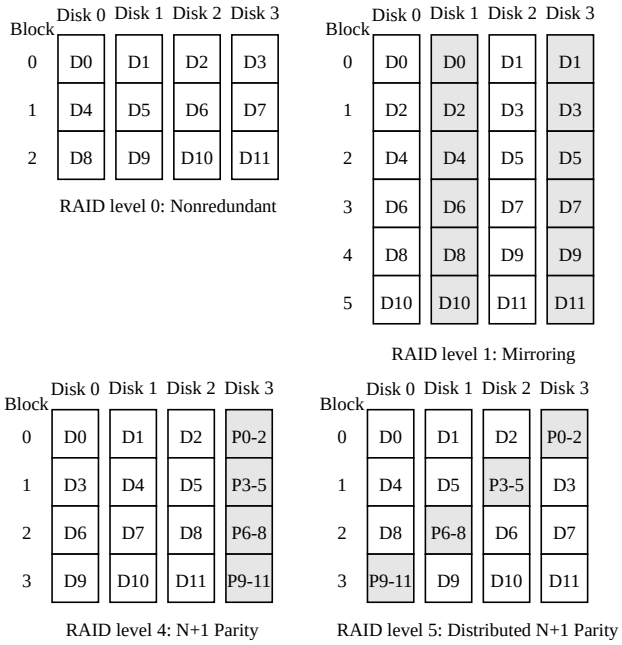


Figure 1 ***Data Layouts****. In nonredundant disk arrays, data units are simply interleaved across the array. RAID level 1 arrays duplicate every user data unit. RAID level 4 arrays interleave user data blocks across all disks except one. Blocks on the final disk hold the parity (bitwise xor) of the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

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 commonly thought to possess at least one critical limitation: their 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

TPC Benchmark	Scaling to X transactions per second
get request from terminal	
begin transaction	X*100k account records (100 bytes each)
update account record	X*10 teller records (100 bytes each)
write history log	X branch records (100 bytes each)
update teller record	X*10 terminals (1/10 TPS each)
update branch record	X*30K history records (50 bytes each)
commit transaction	>X*11.5 MB total online storage
respond to terminal	

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 15% of the time from a different branch. 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 3GB of storage must concurrently provide more than 250 account record reads and writes per second.*

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* [Gibson92].

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 continue to 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 introduces our simulation system, describes implementations of parity logging and alternative organizations, and contrasts their performance on a workload of small random writes. Section 6 discusses extensions to multiple failure tolerating systems. Section 7 reviews related work by Bhide and Dias [Bhide92]. Section 8 closes with a summary of current and future work in redundant disk arrays for small write intensive workloads.

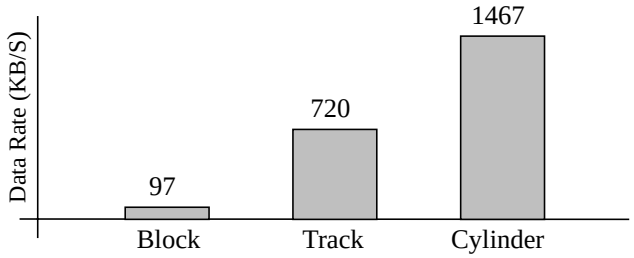


Figure 3 ***Peak I/O Bandwidth.*** *The figure shows the total kilobytes per second that can be read from or written to a drive using random one block (2KB), one track, and one cylinder access on an IBM 0661 drive (see Figure 12 for disk parameters).*

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. The model is introduced in terms of a simple, but impractical RAID level 4 scheme, then refined to the realistic implementation used in the 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 update accesses to log and parity blocks.

Logically, our scheme can be developed beginning with Figure 4. A RAID level 4 disk array (Figure 1) 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 fault tolerant buffer[1]. When enough (one or more tracks) parity update images are buffered to allow for an efficient disk transfer, 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 rewritten with large sequential writes. When this completes, the log disk is

1. The specific characteristics of the fault tolerant buffers depends on the expected failure modes. If simultaneous controller memory and disk loss is considered to be a single failure, then the fault tolerant buffers must be nonvolatile to provide single failure tolerance. If, however, array controller memory loss and disk failure are independent of each other, then the array can be single failure tolerating without nonvolatile controller buffers. In either case, the software fault tolerance needed to protect these buffers against corruption resulting from software failures is beyond the scope of this paper.

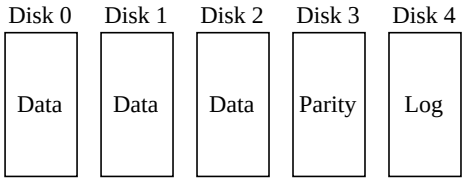


Figure 4 ***Basic Parity Logging Model****. A RAID level 4 disk array is augmented with a log disk. Parity update records are written sequentially to the log disk at track rates. A full log disk triggers a read of the log and parity disks, computation of the current parity, and a rewrite of the parity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

marked empty, and the logging cycle begins again.

It is straightforward to verify that this scheme preserves data reliability. If a data disk failure occurs, the log disk (and any records in the fault tolerant memory) are first applied to the parity disk, which can then be used to reconstruct the lost data. 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.

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 blocks on a track, T tracks per cylinder, and V cylinders on a disk (see the glossary in Figure 6). 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 occur at cylinder data 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 fast and T times larger than track accesses. Thus parity maintenance for TVD small writes consumes about as much disk time as

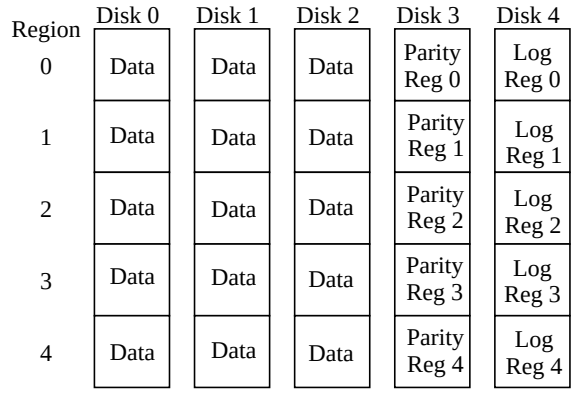


Figure 5 ***Parity Logging Regions****. Dividing each disk into regions dramatically reduces the required amount of controller buffer space. Each region requires a fault tolerant track buffer to hold its unwritten log records. When a track buffer fills up, the track is written into its regions log with a full track write.*

S	Average seek time
R	Average rotational delay (1/2 disk rotation time)
H	Head switch time
M	Single track seek time
T	Tracks per cylinder
V	Cylinders per disk
N	Disks in the array
K	Tracks buffered per region
C	Cylinders per region
D	Data units per track
L	Log Striping Degree

Figure 6 ***Model Parameters****. The bandwidth utilization model of Section 3 is presented in terms of these parameters. The majority of the parameters are based on disk geometry. The remainder come from the application or array configuration. The left hand column indicates the symbol used in this text. The same notation also used in Sections 3 and 4.*

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

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

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 5 shows how this limitation can be overcome by dividing the array into 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 100 regions per disk (about 3MB per region).

Now, however, each region requires a fault tolerant buffer. Each buffer holds a track (or a few 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, this more practical implementation still has dramatically lower parity maintenance overhead than RAID levels 4 or 5, as will be shown in the next section.

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 log regions 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.

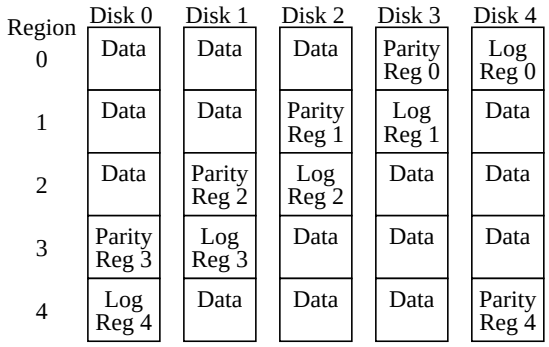


Figure 7 ***Log and Parity Rotation****. Spreading the log and parity over the entire array increases the parity and log bandwidth to the entire bandwidth of the array. An individual region may still be a hot spot.*

The log and parity bandwidth for a particular region, however, is still that of a single disk. Following the example of RAID level 5, the parity for each region is block striped across the array to increase bandwidth (Figure 8). 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 parity 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

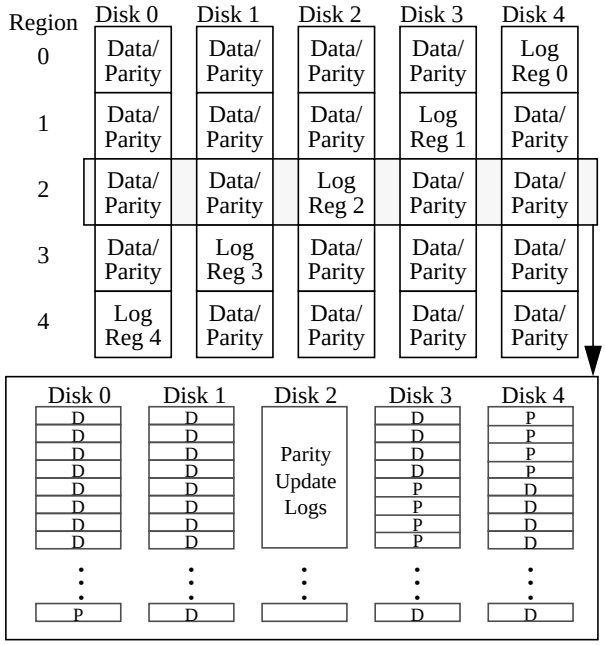


Figure 8 ***Block Parity Striping****. Parity and data are distributed over all but one disk in each region. The remaining disk contains the parity log. A contiguous layout of parity on each disk allows efficient cylinder rate transfers, while distribution reduces the latency of parity reintegration.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

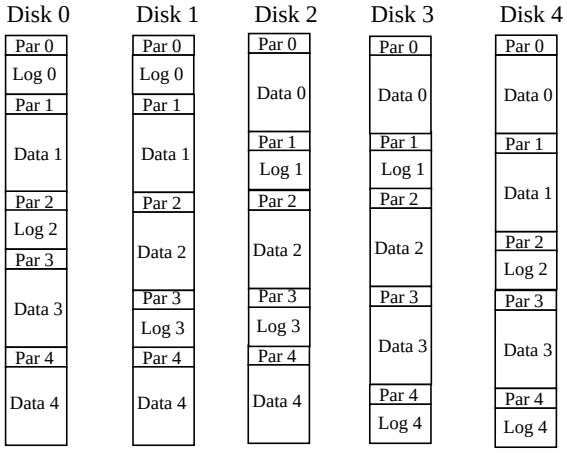


Figure 9 ***Distributed Parity Logs****. To increase the log bandwidth for each region, the log for each region is striped. In this example, each log region is striped over 2 disks. As before, the parity is still spread over on all disks. To preserve single fault tolerance, a parity sublog for a region cannot reside on the same disk as any data for that region. Thus while striping reduces the time for log application 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 is beneficial to performance with a small cost increase.*

of disks on which data for a particular region may be placed. Since the disk space overhead is proportional to the number of disks over which data is placed, striping the log increases the disk space overhead. Figure 10 shows the dependence of disk space overhead on the striping degree. As will be shown in Section 5, however, the performance advantages of striping are substantial. The selection of the number of disks over which to stripe the log, the striping degree (L), will also be examined in that section.

The controller memory overhead for this mechanism is fairly modest. With r regions, the controller requires Kr track buffers and another buffer that is VT/r tracks large for the parity reintegration. If a single 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 40% of one disk, or about 2% of the 22 disk array.

Section 3: Analytical Modeling

In this section we present a utilization-based analytical model of a disk array. This model predicts sustained array performance in terms of achieved disk utilization, disk geometry, and access size. The parameters and symbols used in this model are listed in Figure 6.

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

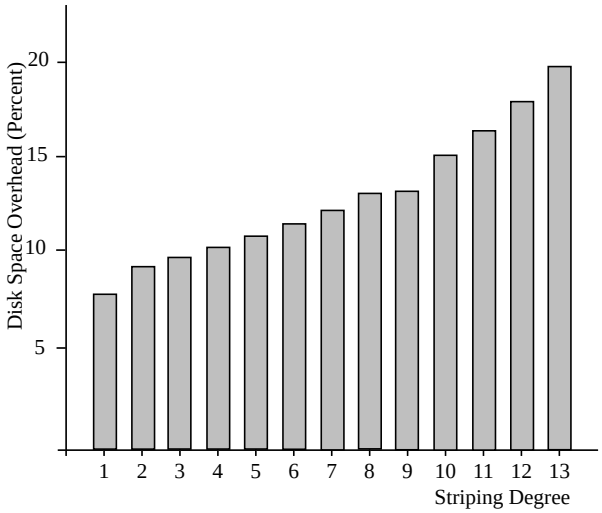
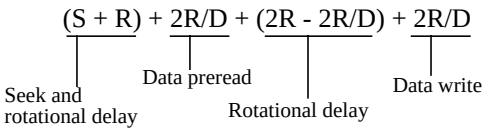


Figure 10 ***Disk Storage Overheads****. While increasing the log striping degree improves array performance, the storage overhead increases. Shown above is the percent of the total disk capacity devoted to storing redundant data in an array with 22 disks. In general, the storage overhead is 2/(N+1-L),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

In addition to these disk space overheads, parity logging also requires fault tolerant memory buffer space. With the example disk array of Figure 12, this amounts to 5592 KB, roughly equivalent in cost to 2% of the disk array.

data, waits for the disk to spin around once, then updates the data[2]. On average, such an access will take

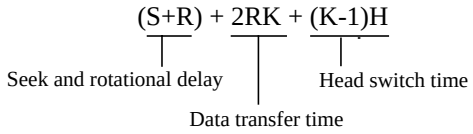


disk seconds, which may be simplified to

$$S + (3 + \frac{2}{D}) R = A_0$$

In many cases, it may be possible to predictably avoid prereading user data. For example, in the TPC benchmark the updates of a customer account record is a read modify write operation; a data 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 an additional preread of the data may be skipped. Without prereading, the disk busy time needed for a small write access is $S + (1 + 2/D) R$.

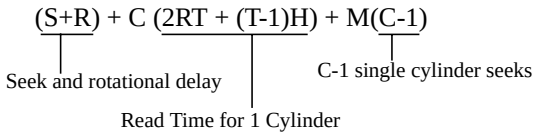
Each region has K tracks worth of fault tolerant buffers. Thus, on average, for every KD small user writes, one region's buffers will fill and be written to the region's log in a single K track write. The number of disk seconds needed to do this is



assuming all K tracks are on the same cylinder[3]. This may be rewritten as

$$S + (2K + 1) R + (K - 1) H = A_1.$$

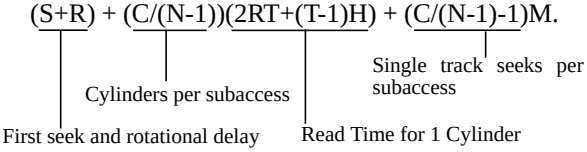
Finally, on average, for every DTC small user writes one region of logged parity must be reintegrated. First, consider the cast of an array that does not stripe its log (Figure 8). The reintegration consists of three steps: a sequential read of C cylinders (one region) from the log, a striped read of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



disk seconds, and may be rewritten as

$$+ (2TC + 1) R + (T - 1) HC + (C - 1) M = A_2.$$

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



Rewriting, each striped access takes A_3 disk seconds:

$$(N - 1)(S + R) + C(2RT + (T - 1) H) + M(C - N + 1) = A_3$$

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

$$A_0 + \frac{1}{KD} A_1 + \frac{1}{DTC} [A_2 + 2A_3] .$$

Figure 11 shows the contributions to disk busy time of the various terms after A_0 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 fault tolerant buffers fill, the 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.

2. 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.

3. 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%.

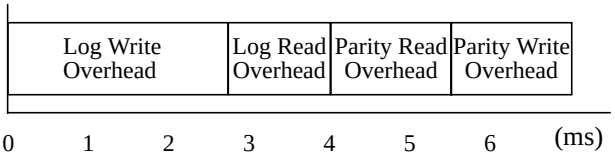
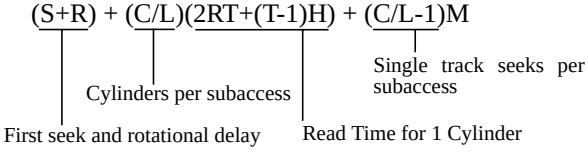


Figure 11 ***Parity Logging Overheads****. The amortized overhead cost of extra I/Os done in our example parity logging array is shown above. The log writes contribute approximately 40% of the overhead, while the cylinder rate log reads, parity reads and arity writes each contribute about 20%. In contrast, the extra /Os done by RAID level 5 cost nearly 35 milliseconds per small write.*

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 cyls, 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 typical of 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]. 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 traditional, 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.*

Log reintegration still occurs every DTC small user writes, but now consists of three striped I/Os: a striped (over L disks) read of the log, and a striped read and write of the parity (striped over N disks). The striped log read costs



for a total of

$$L\,(S+R)\;+C\,(2RT+\;(T-1)\,H)\;+\;(C-L)\,M$$

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

$$N\,(S+R)\;+C\,(2RT+\;(T-1)\,H)\;+\;(C-N)\,M$$

disk seconds. Thus striping introduces an additional overhead of $(L+1)\;(S+R-M)$ disk seconds to the log integration. This increases the parity maintenance overhead per small write by

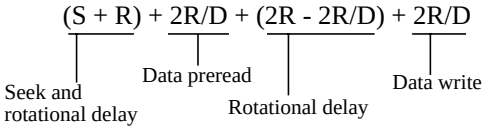
$$\frac{(L+1)\;(S+R-M)}{DTC}$$

This increase in parity maintenance work is worthwhile because it reduces long reintegration periods when disk queues grow until the system becomes underutilized which causes maximum performance to fall far short of expectations.

Section 4: Alternative Schemes

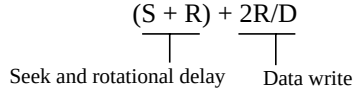
Few other authors have addressed the problem of high performance yet reliable disk storage for small write workloads. The most notable of these is floating data and parity [Menon92]. This section reviews and estimates the performance of four configurations: mirrored disks (RAID level 1), nonredundant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

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



disk seconds for a total disk busy time of $2S+\;(6+4/D)\,R$. No fault tolerant controller storage is required.

The traditional solution to reliable disk storage has been mirroring. In mirrored systems, every data unit is stored on two disks, and all write requests update both copies. No preread is required, however, so each access takes



Hence each small user write utilizes disks for $2S+\;(2+4/D)\,R$ seconds. While mirrored disks are more efficient than RAID level 5, half their capacity is devoted to redundant data, making them expensive. Similarly to RAID level 5, controllers for mirrored disk arrays do not require

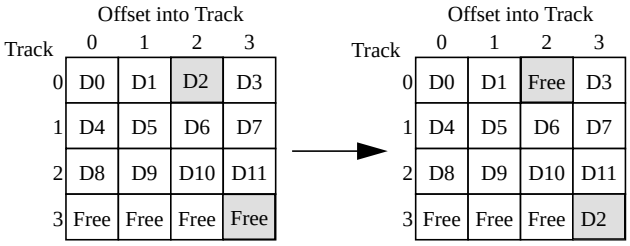


Figure 13 ***Floating Data/Parity****. When updating block D2, the controller searches for a free block within the cylinder that is rotationally close to block D2. In this case, it finds the block at offset 3 into track 3. Immediately following the preread of block D2, the controller writes the new block 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 time of one access.*

fault tolerant storage.

The *floating data and parity* modification to RAID level 5 was proposed by Menon and Kasson [Mennon92]. This technique organizes data and parity into cylinders that contain either data only or parity only. As illustrated in Figure 13, 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. Recall that for RAID level 5, the disk busy time for each data and parity update is

$$S + R + 2R/D + (2R - 2R/D) + 2R/D.$$

With floating data and parity, the rotational term $2R - 2R/D$ 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

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

which may be rewritten as $S + (1 + 5.52/D)\,R + H$. 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 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[4] of $(T + N - 1)/(TN)$. 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. 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,

4. 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-1)(N-1)/TN and the array storage overhead is 1-(T-1)(N-1)/TN = (T+N-1)/TN.

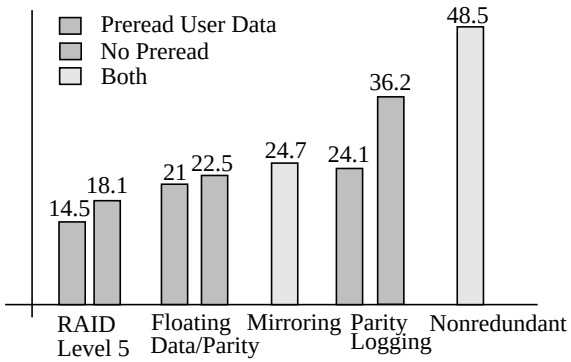


Figure 14 ***Model Estimates****. I/Os per second per disk as predicted by the bandwidth models of Sections 3 and 4. These predictions assume 100% disk utilization, FIFO disk arm scheduling and an unbounded number of requestors. Raid level 5 and parity logging disk arrays both benefit substantially from 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. The parity logging estimates are insensitive to the degree of striping.*

roughly comparable to parity logging.

While floating data and parity substantially improves the performance of small writes, 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 by $(T-1)/T$.

Figure 14 compares the model's estimates for maximum throughput of the example arrays based on Figure 12. Throughput at lower utilizations may be calculated by scaling the maximum throughput numbers by the disk utilization. Figure 14 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: Simulation

To validate the models presented above and to explore response time for these arrays, we simulated the example array in Figure 12 under five different configurations: nonredundant, mirroring, RAID level 5, floating data and parity, and parity logging. Parity logging was simulated for several different degrees of log striping[5]. 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 float parity and data.

In each simulation, the request stream was generated by 66 processes (i.e, three per disk). Each process requests a

5. A single track was buffered per region in all parity logging simulations.

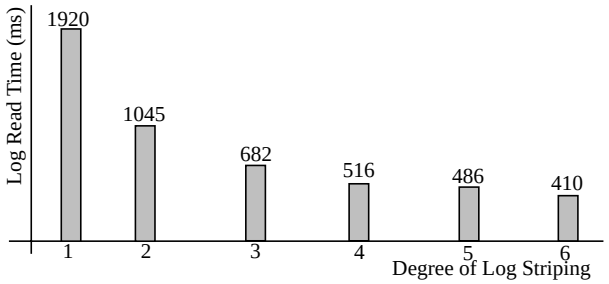


Figure 15 ***Sublog Read Times****. This figure presents the sublog read time for low degrees of log striping for the example disk array. When the sublog reads are very long, many user requests queue behind the read, increasing response time and decreasing array utilization.*

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 90% confidence interval of the response time is less than 5% of the mean.

Figure 16 shows peak throughput, response time[6] and response time variance as the degree of log striping (L) is varied from 1 (unstriped) to 13. When the log is striped over a small number of disks, performance is substantially lower than other configurations. This behavior can be explained in terms of a "convoy effect". The length of the sublog read I/Os is the basis of the convoy effect. Figure 15 shows sublog read times for low log striping degrees. While these long I/O's are efficient, they completely tie up a disk for seconds. During this period, any access to the disk involved in the 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.

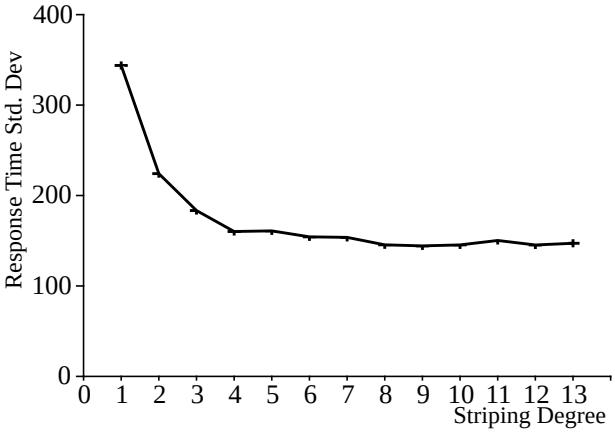


Figure 16(c): Response Time Variance at Peak Load

6. The simulations reported herein consider a user write in a parity logged array complete when the user data is on disk and the parity update record has been committed to fault tolerant storage. The alternatives consider a user write complete when data and parity are on disk.

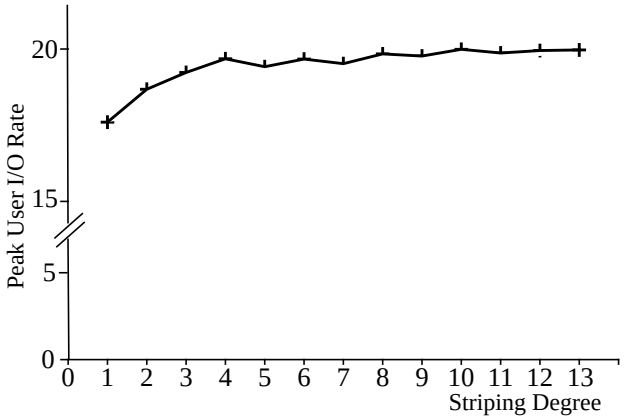


Figure 16(a): Peak User I/Os

Figure 16 ***Striped Parity Logging****. Figure 16(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 1 (no striping) to 4. 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 log reads are long (see Figure 15), 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, reducing this variance.

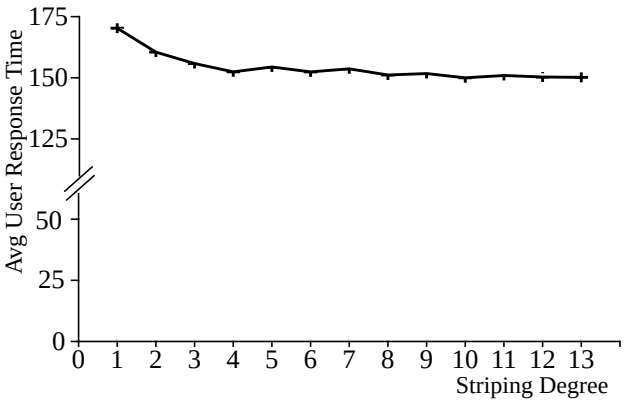


Figure 16(b): Response Time at Peak Load

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 both average response time and response time variance.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 17 compares the performance of this configuration against the alternative organizations presented in Section 4: nonredundant, mirroring, RAID level 5, and floating data and parity. Figure 17(a)-(b) present response time statistics as a function of throughput for simulations that preread user data, and (c) presents the corresponding data for the no preread case.

Because of the relatively small number of simulated pro-

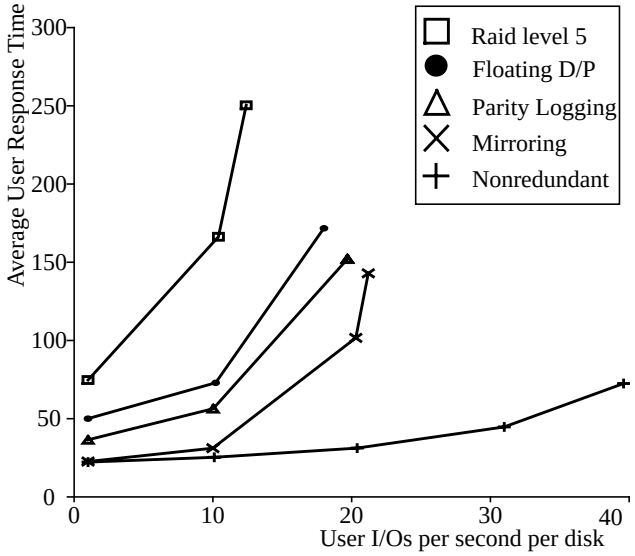


Figure 17(a): Response Times

Figure 17 ***Response Times and Utilization****. Figure 17(a)-(c) resent 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. Figure 17(a) and (b) present the results when the user data must be preread, while the results in Figure 17(c) assume the user data was cached, making the reread 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 17(b).*

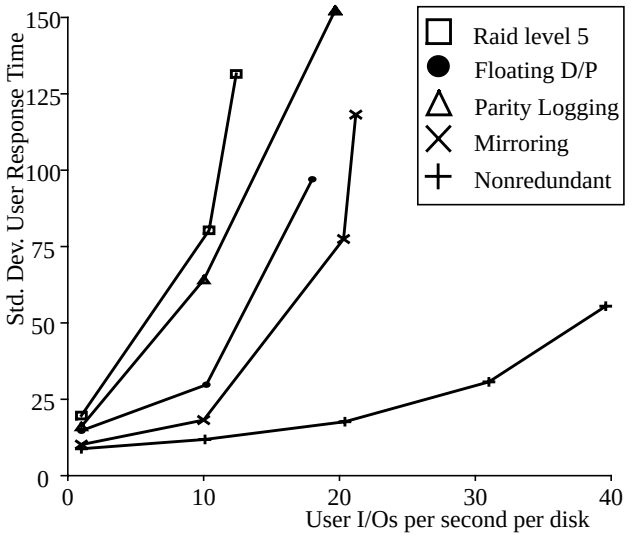


Figure 17(b): Response Time Standard Deviation

cesses, the array saturates while some disks are less than fully utilized. That is, because the number of requesting processes is fixed, one overloaded disk can cause other disks to be underutilized. The impact of this effect on peak

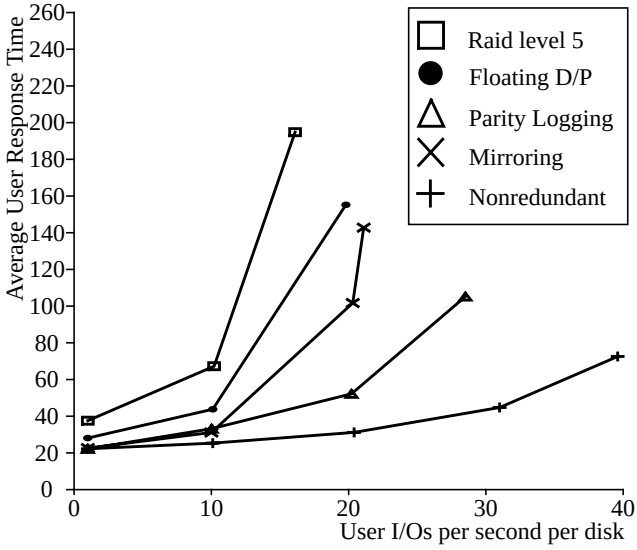


Figure 17(c): Response Times without prereads

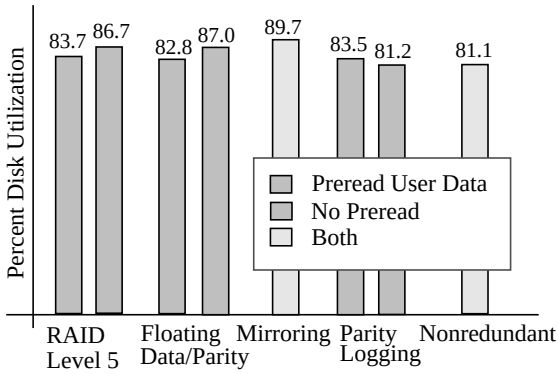


Figure 18 ***Disk Utilization at Peak Load****. The figure above presents the average disk utilization at maximum load for the array simulated in Figure 17. In every configuration disk utilization grew linearly with throughput.*

disk utilization varies from configuration to configuration. Figure 18 shows the disk utilization 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[7]. Nonetheless, Figure 19 shows that simulation agreement with the model is good when the model results (Figure 14) are scaled by the achieved disk utilizations in Figure 18.

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

7. 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 while approximately doubling response time.

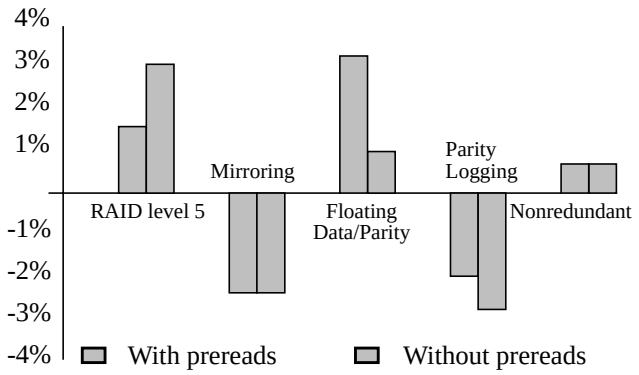


Figure 19 ***Model errors****. The figure shows the percent error between the models and the simulations. The model predictions have been scaled by the achieved disk utilizations. In all cases, the disagreement between the simulation and the models is less than four percent. Note that the 90% confidence interval of the simulation response time is* $\pm 5\%$ *of the mean.*

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, must complete two slow read-rotate-write accesses sequentially. 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 20ms higher than mirroring, the peak sustainable I/O rate and response time are quite similar. Similar to RAID level 5, floating 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.[8]

Figure 17(c) shows the performance of all configurations without data preread. 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 the data preread. In addition, the user 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 the 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 a unloaded response time comparable to a nonredundant array. This also reduces the actuator time per I/O by nearly one third, and the I/O rate and response time 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. Nonetheless, the response

8. In parity logging arrays that are not driven into saturation, making the log accesses preemptible by user access should substantially improve response time and response time variance.

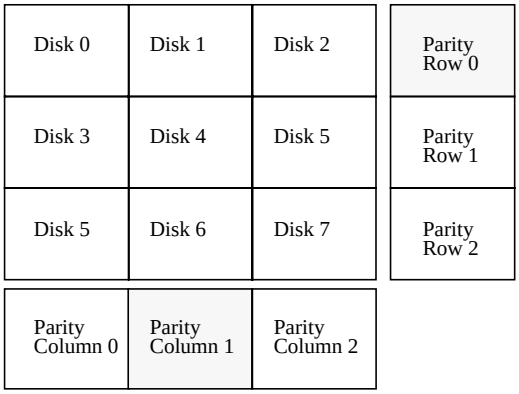


Figure 20 ***Two dimensional parity****. One disk array organization that achieves double failure tolerance is two dimensional parity. Parity disks hold the parity for the corresponding row or column. In the example above, the parity disk for column 0 holds the parity of disks 0, 3 and 5. Whenever a data disk is written, the corresponding 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.*

time estimates show that parity logging is a viable and much lower cost alternative to mirroring for small write workloads.

Section 6: Multiple failure tolerating arrays

Another 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 [Gibson89]. Using codes more powerful than parity, RAID level 5 and its variants, floating data and parity and parity logging, can all be extended to tolerate f concurrent failures. Figure 20 gives an example of one of the more easily understood double failure tolerant disk array organizations. This paper does not consider the choice of codes that might be used for f failure protection, except to note that 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.

In an f failure tolerating array using parity logging, the single striped log per region is replaced with f striped logs per region, each on a separate set of disks. When a region's fault tolerant buffers fill up, the corresponding parity update records are written to all logs for that region. When a region's logs fill up, one copy of the log is read in, all check data for the region is read in, updated in memory, and rewritten[9].

The other configurations also extend straightforwardly. Mirroring becomes f-copy shadowing. RAID level 5 and

9. Instead of reading all the parity updates from one of the logs, a different subset of the parity update records could be read from each log, effectively further striping the parity update record read.

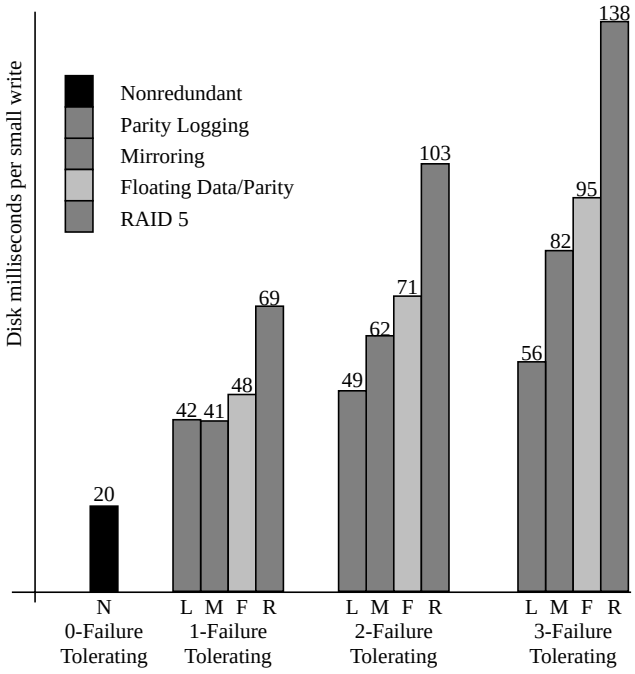


Figure 21 ***Small write costs****. This figure shows the amortized disk arm time consumed by a small write for each of the modeled techniques in arrays that tolerate zero, one, two or three failures. Parity logging is competitive with mirroring in the single failure tolerating case and is substantially better than the other methods in arrays that tolerate two or more failures.*

its floating data and parity version simply store more parity and issue read-modify-write updates to f check blocks with every small 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 access (log and generalized parity) are done efficiently.

Figure 21 shows the total disk time required per small random write 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 much 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 mirroring disk arrays) will sustain only 25%.

Section 7: 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 double buffer 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 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 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 integration 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 the 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 disk per second. 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 5, whose controller requires about 2% of a disk's worth of memory, is predicted to achieve 36.2 I/Os per disk per second on the same workload and its performance increases with increasing numbers of disks.

Section 8: 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 pro-

posed technique achieves substantially better performance than RAID level 5 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 mirroring and floating parity and data when the data to be overwritten is cached. This performance is obtained without the 50% disk storage space overhead of mirroring. For extremely reliable 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. The effects of log length on on-line reconstruction performance should be investigated and detailed simulations of multiple failure tolerating configurations should be undertaken. More dynamic assignment of fault tolerant 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 comparison of the log structured filesystem [Rosenblum 91], 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 reconstruction while parity logging provides high performance during fault free operation. The combination of the two should provide a particularly attractive system for OLTP environments.

Section 9: Acknowledgments

We would like to thank Ed Lee for the original version of Raidsim, and Brian Bershad, Peter Chen, Hugo Patterson, and Jody Prival for early reviews. This research was supported by the Defense Advanced Research Projects Agency monitored by DARPA/CMO under contract MDA 972-90-C-0035 and by an IBM graduate fellowship.

References

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

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

[Gibson89] G. Gibson, L. Hellerstein, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding Techniques for Handling Failures in Large Disk Arrays," *Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS III)*, ACM Press, 1989, pp 123-132.

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

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

[Holland92] M. Holland and G. Gibson, "Parity Declustering for Continuous Operation in Redundant Disk Arrays," *Proceedings of ASPLOS-V*, 1992, pp. 23-35.

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

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

[Lee91] E. Lee and R. Katz, "Performance Consequences of Parity Placement in Disk Arrays," *Proceedings of ASPLOS-IV*, 1991, pp. 190-199.

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

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

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

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

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

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