Parity Striping of Disc Arrays: Low-Cost Reliable Storage with Acceptable Throughput

Jim Gray, Bob Horst, Mark Walker
Tandem Computers Inc., 19333 Vallco Parkway, Cupertino, CA. 95014

Abstract
An analysis of mirrored discs and of RAID5 shows that mirrors have considerably better throughput, measured as requests/second on random requests of arbitrary size (up to 1MB). Mirrors have comparable or better response time for requests of reasonable size (less than 100KB). But mirrors have a 100% storage penalty: storing the data twice. ***Parity striping*** is a data layout that stripes the parity across the discs, but does not stripe the data. Parity striping has throughput almost as good as mirrors, and has cost/GB comparable to RAID5 designs -- combing the advantages of both for high-traffic disc resident data. Parity striping has additional fault containment and software benefits as well. Parity striping sacrifices the high data transfer rates of RAID designs for high throughput. It is argued that response time and throughput are preferable performance metrics.

1.Introduction

Disc arrays have traditionally been used in supercomputers to provide high transfer rates by reading or writing multiple discs in parallel [Kim]. Rather than getting 2MB/s from a single disc, applications are able to read or write *N* discs in parallel by striping data across the discs thereby getting a transfer rate of 2*N*MB/s.

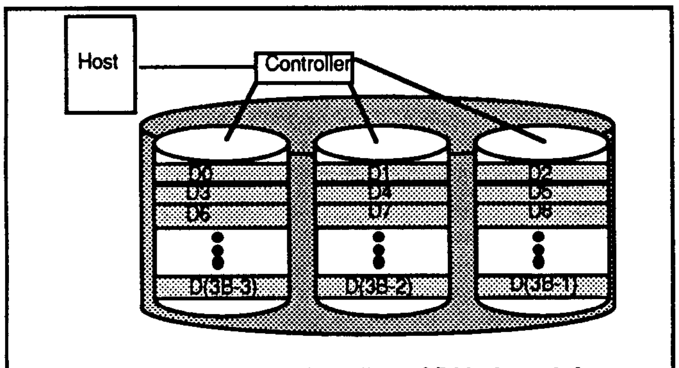


Figure 1: Striping data across three discs of *B* blocks each forms one large logical disc of *3B* blocks. A sequential read or write of data D0, D1, D2 can proceed in parallel at three times the data transfer rate of a single disc.

Proceedings of the 16th VLDB Conference
Brisbane, Australia 1990

The striping unit can be a bit, a byte, a sector, a page, a track, or any larger granule. If the striping unit is a block, then the ***ith*** logical block maps to physical block $\lfloor i/N \rfloor$ of disc *i mod N*. The whole array of *N* discs is treated as a single large fast disc. Reading or writing the group of *N* blocks $\{D_{Ni}, D_{Ni+1}, \ldots, D_{N(i+1)-1}\}$ can be done in parallel using a single disc rotation. If the read is not aligned to an *N* block boundary, or if the read involves more than *N* tracks, then multiple disc rotations will be required to complete the read.

In the last five years, the idea of using part of the array capacity to mask discs failures has become quite popular. The most common example of this parity approach is found in the IBM AS400 [AS400]. The idea is most clearly explained in [Patterson] which coined the term RAID (Redundant Arrays of Independent Discs), and discussed several design alternatives. A typical data layout of a RAID5 disc array is as follows (see Figure 2):

- Sacrifice $\left(\frac{1}{N+1}\right)$th of the disc space to parity by acquiring $N+1$ discs of *B* blocks each.
- Logical block *i* maps to physical block $\lfloor i/N \rfloor$ of disc $(i \bmod (N+1))+j)$, for $i = 0,1,\ldots, NB-1$ where $j = 0$ *if* $i \bmod N < i \bmod (N+1)$ *else* $j = 1$.
- The parity block P_i for logical blocks $\{D_{Ni}, D_{Ni+1}, \ldots D_{N(i+1)-1}\}$ is block *i* of disc $i \bmod (N+1)$.[1]

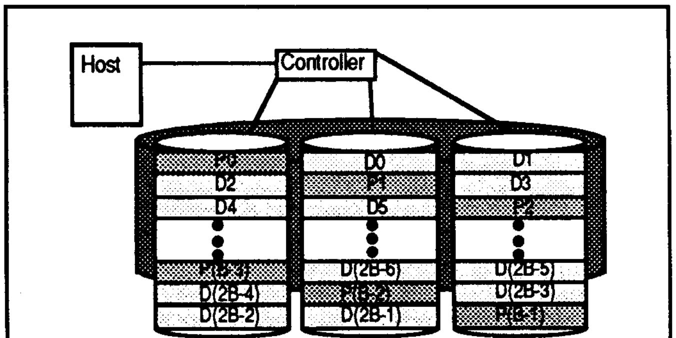


Figure 2: The RAID5 approach to striping data and parity on three discs of *B* blocks each. The parity blocks are labeled by P0, P1,...while the data blocks are labeled D0, D1,..., D(2B-1). The resulting logical disc has *2B* data blocks protected by *B* parity blocks. Parity block P1 is maintained as D2 XOR D3. A read of data D0, D1, D2 can proceed in parallel at three times the data transfer rate of a single disc, while a write of D0 and D1 can proceed at twice the rate while writing P0 = D0 XOR D1 in parallel.

[1] This strips the parity from upper left to lower right. Garth Gibson uses the equations: logical block *i* is physical block $\lfloor i/N \rfloor$ of disc $i \bmod (N+1)$, and its parity block is physical block $\lfloor i/N \rfloor$ of disc $(-1-i) \bmod (N+1)$ which stripes the data from upper right to lower left. The two designs are equivalent. Gibson's equations avoid the "*j*" term in the text.

This mapping creates a helical pattern of parity running through the disc array (see Figure 2). Requests to the logical disc are spread among the $N+1$ physical discs. Small requests involve only one or two discs, while multi-block requests may involve several discs and benefit from the sum of their bandwidth.

A RAID disc controller protects against damaged blocks and disc failures as follows (see Figure 3):

- When reading the logical group of blocks $\{D_{Ni}, D_{Ni+1},...,D_{N(i+1)-1}\}$, if any single block is bad (based on ECC or device error), that block can be reconstructed by the XOR (exclusive-or) of the good blocks with the corresponding parity block.
- When writing any subset of the logical group of blocks $\{D_{Ni}, D_{Ni+1},...D_{N(i+1)-1}\}$, the corresponding new parity block must also be computed (XOR of the logical blocks) and written.

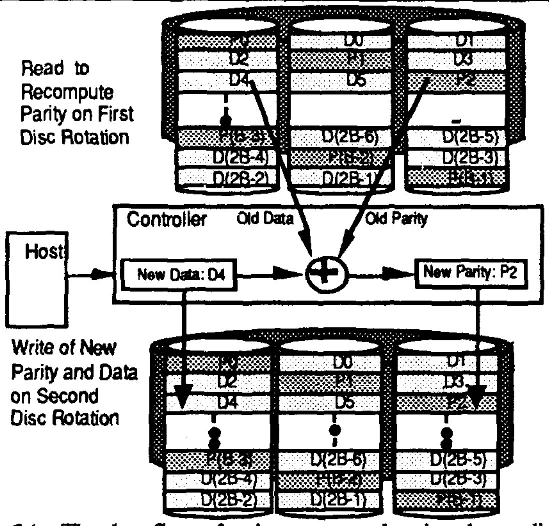

Read to Recompute Parity on First Disc Rotation
Controller
Old Data
Old Parity
Host
New Data: D4
New Parity: P2
Write of New Parity and Data on Second Disc Rotation

Figure 3A : The data flow of writes to RAID showing the reading of old parity and old data from disc to compute the new parity. This can easily be done in one rotation, as the discs rotate if the spindles are synchronized. If the spindles are not synchronized, the data must be buffered, but the parity can still be computed in one rotation. The new data and parity are then written during the second rotation.

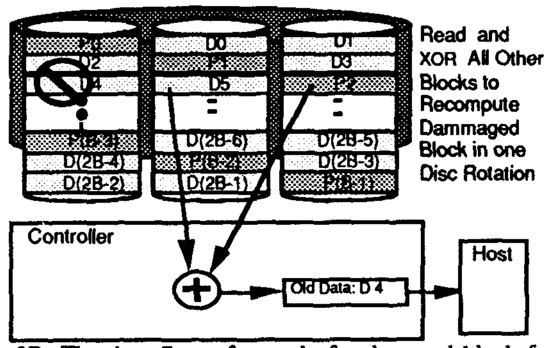

Read and XOR All Other Blocks to Recompute Dammaged Block in one Disc Rotation
Controller
Old Data: D 4
Host

Figure 3B: The data flow of a read of a damaged block from a RAID. The value of damaged block D4 can be reconstructed from the other discs (D4 = D5 XOR P2).

Traditionally, fault tolerant-disc storage has been implemented using duplexed discs (aka. mirrors (Tandem) or shadows (DEC)) [Katzman]. The idea of mirroring is to tolerate any single fault by using dual ports, dual controllers, dual paths, and dual discs which store exactly the same data. When data is written, it is written to both discs. When data is read, it is read from either disc. If that read fails, the other disc is read and the bad-spot on the first disc is spared and rewritten.

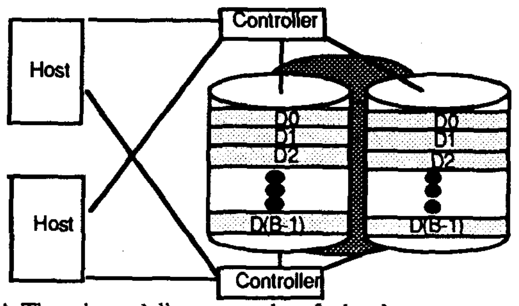


Figure 4: The mirrored disc approach to fault-tolerant storage places identical data on a pair of discs, accessible from fault-tolerant hosts via dual disc controllers and via four data paths. This gives a single logical disc of B blocks. Reads go to either copy of the data, writes go to both copies.

Mirrored discs have a major drawback: cost. If you buy $2N$ discs of B blocks each, you can only store NB blocks of data, a 100% storage overhead. In addition, a write intensive application must write each update to two discs and so pays a 100% write penalty. Actually, it is slightly worse than 100% since one must pay for the longest seek of two disc arms. These arguments seem a high price to pay for reliable storage, and explain the interest in RAID systems.

There are some mitigating circumstances that make mirrors slightly more attractive: random reads of blocks Bi and Bj can seek, rotate, and transfer in parallel. So, for read intensive applications mirrored discs give approximately twice the throughput of a single disc. In fact, due to the shortest-seek optimization, mirrored discs may give slightly better than twice the performance of a single disc on read intensive applications [Bitton1].

Figure 2 paid no attention to processor failures, path failures, or controller failures. But controllers are no more reliable than discs these days. In fact, a truly fault-tolerant RAID design should look like Figure 5. In order to tolerate single controller failures, the host must be able to ask the second controller to retry the write. The issue of controller failures has not been discussed in the literature, but it is essential to making a fault-tolerant store. In addition, fault tolerant disc arrays are generally configured with a spare drive which receives a reconstructed copy of the failed drive within a few hours of the failure -- this is generally called an $N+2$ array scheme. The standby spare minimizes the repair window, and so improves the array mean time to data loss [Schulze]. With a correct implementation of these two issues, an $N+2$

disc array offers fault-tolerant storage comparable to mirrored discs but with high data transfer rate and approximately a 40% cost savings measured in $/GB (for a 10+2 array).

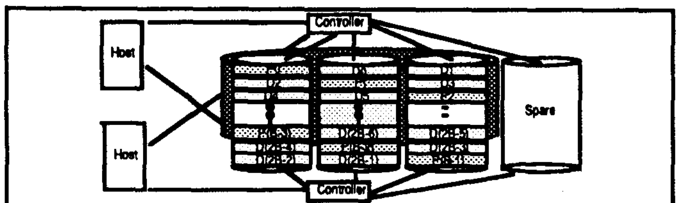
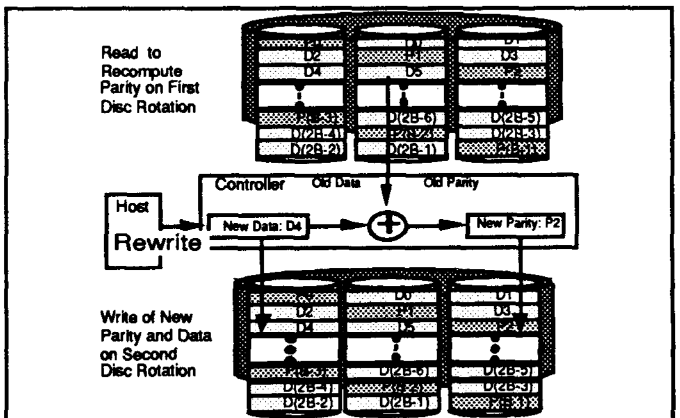


Figure 5: The RAID5 approach configured for single-fault tolerance. This includes dual processors and dual controllers along with four paths to each disc so that there is no single point of failure. In addition, to deal with controller failures, the controller must have a "retry interface" that computes new parity from the new data and from unaffected data. A spare disc is configured so that a failed disc can be quickly reconstructed. Otherwise, the failure of a second disc in the array will result in lost data. Generally, arrays are configured with eight or more drives to amortize the cost of storing the parity across many drives and the high fixed cost of the dual controllers. In this article, we assume a 12-drive complex.

The retry logic to deal with controller and path failures is best described by the case of writing a single block D_i.[2] If the write fails, the disc contents of block D_i and its parity block P_j are suspect (may have been partially written). The host asks the second controller (Figure 5) to *retry* the write of D_i. Retry is a special controller operation which computes the new parity by reading all other blocks of the stripe (all except old D_i and old P_j) and XORing them with with the new data block D_i to produce the new parity block P_j (see Figure 6). During the second rotation, the controller writes the new data and new parity blocks (D_i and P_j.). This idea easily generalizes to multi-block writes.


Read to Recompute Parity on First Disc Rotation
Controller
Old Data
Old Parity
Host
Rewrite
New Data: D4
New Parity: P2
Write of New Parity and Data on Second Disc Rotation
Figure 6: The retry logic for RAID and parity stripe on path and controller failures. The second controller reads and XORs the undamaged blocks with the new data to compute the new parity; all in one rotation. Contrast this to Figure 3.A which reads old D4 and old parity.

[2] This retry logic is new. It works for parity stripe and RAID5 arrays. Making RAID writes atomic (all or nothing) has been a major sticking point for some database applications.

2. Why Striping and RAID Are Inappropriate for OLTP Systems

The RAID idea has caused most computer designers to reexamine their disc subsystem architecture. As a result the classic disc striping idea is excellent for supercomputers and has been added as an option for the scientific community in IBM's MVS, Amdahl's Unix, and DEC's VMS. But the surprising result is that the business applications community (e.g. databases) have generally concluded that RAID is not appropriate for their applications because they don't need the bandwidth, they don't need the extra storage capacity, and they cannot afford to use several disc arms to service a single request. These three surprising observations are elaborated in the next paragraphs.

Why they don't need the space: As Gelb points out [Gelb], most IBM disc farms are 50% empty: 25% is unused to allow files to grow, but another 25% is unused because putting too much data under a disc arm results in long queues of requests for that data. If these customers could buy infinite capacity discs for the same price, most would not be able to put more than a gigabyte of data under each disc arm. So that is why they do not need extra space -- they can't use it[3].

Why they don't need the bandwidth: Supercomputers may be able to absorb data at 40MB/s, but most computers cannot. First, the IO channels of most computers run at 1MB/s to 5MB/s burst rates, and actual data rates are typically half that. So the array controller cannot deliver data to the host or application very quickly. One way to circumvent this is to do the striping in the host: the processor reads via multiple channels in parallel. This is how the IBM, Amdahl, and DEC implementations of striping work. In such a multi-channel design the host becomes the RAID controller and does the parity work. Having the host compute the XOR of the data is expensive in host processing cycles. In fact the host implementations mentioned above do pure striping for bandwidth rather than maintain RAID parity. Perhaps more to the point, most applications cannot scan structured data at 40MB/s. Scans, sorts, and other structured access to data typically process a few thousand records per second [Schneider]. At 100 bytes per record and 1k instructions to process each record, a 10MIP processor consumes data at 1MB/s -- well below current device speeds of 4MB/s. So, the bottleneck for Cobol and SQL applications is not disc transfer rate, unless they are running on processors of 50MIPS or more, and have IO channels rated in excess of 5MB/s. Device speeds are likely to improve as processors become faster, so only limited degrees of striping will be needed.[4]

[3] Since this study was done (1984), IBM has twice doubled the storage capacity under each disc arm. Presumably, this extra capacity has gone unused in many applications.

[4] Software parallelism is often used to exceed these rates. Examples of this are Teradata, Gamma [Schneider], and NonStop SQL.

Why they can't afford to use several disc arms on a single request: Disc service time on typical commercial, timesharing, and transaction processing applications is 50% queueing, 17% seek, 17% rotation, and 17% transfer [Scranton]. A RAID slaving all the disc arms together reduces the transfer time, leaves the seek almost unchanged, doubles the rotation time on writes, and makes the queueing *much* worse (since there is only one service center rather than $N+2$ service centers). As pointed out above, most commercial applications are disc-arm limited; customers buy discs for arms rather than for giga-bytes. If , as in RAID5, the array does not slave the arms together and allows small transfers, then the array still consumes more arm resource because a RAID5 seek involving M of the N arms is much slower than a 1-arm seek (see [Bitton] or Figure 9). More importantly, RAID5 writes require an extra rotation; thereby adding 34% (17ms) to write service times and driving up device utilization and queueing [Scranton]. Figures 10, 11, and 12 quantify this argument in terms of requests/second processed by a disc array vs the same hardware configured as a mirrored array.

In fairness, this discussion focuses on traditional applications (ones that access structured records), rather than applications that simply move data in bulk (like image processing, real time video, and so on). In addition, it ignores utility access such as disc-to-tape copy and operating system program loading, dumping, and swapping. Each of these applications simply move the data and so are not processor limited; rather, they are limited by channel and device speeds. If the channels ran at more than 10MB/s, then these applications would benefit from the high transfer rate of stripe and RAID schemes. In fact, the software implementations of striping are currently being used primary by scientific applications to quickly load images and tables into memory, and to swap large address spaces.

In addition, we are assuming medium capacity discs (say 1GB/drive), and consequently high activity on the disc arms. If we assumed four times smaller discs (say 250MB/drive), then the request rate per drive would be reduced by a factor of four and our arguments about buying discs for arms rather than for giga-bytes would be incorrect. If four small (3.4inch) discs and their associated power, controllers, and cabinetry have a price comparable to a single "large" (5.25 inch) disc and its support logic, power and cabinetry, then the arm contention arguments above do not apply. However, we do not forecast the necessary 4:1 price advantage for small capacity discs -- both device categories are likely to have small form factors (5.25 inch or less), and are likely to be commodity items.

3. Parity Striping: Cheap Reliable Storage Plus High Throughput

As explained above, many applications would be willing to pay 20% disc space penalty for reliable storage but they cannot afford to spend disc arm time. Parity striping is a compromise devised for such applications. A parity stripe system involves $N+2$ drives and involves parity much as the RAID schemes do. But the parity is mapped as large contiguous extents, and data is not striped across the discs at all. The basic idea is that an $N+2$ array of discs looks like $N+1$ logical discs plus a spare (in a RAID5 scheme it looks like one logical disc with many independent arms).

In a *parity-striped disc array*, if each disc has B blocks, the last $P=B/N$ blocks of each disc are reserved for parity, the other blocks hold data (see Figure 7). So each disc has $D=B-P$ logical blocks and P parity blocks. The data is mapped as:

- Logical block i of disc j is physical block i of disc j for $i = 0,\ldots,D-1;\ j = 0,\ldots,N$.
- The parity block for block i of disc j is block $D + (i \bmod P)$ of disc $(\lfloor i/P \rfloor + k) \bmod (N+1)$ where $k = 0$ if $\lfloor i/P \rfloor < j$ else $k = 1$.

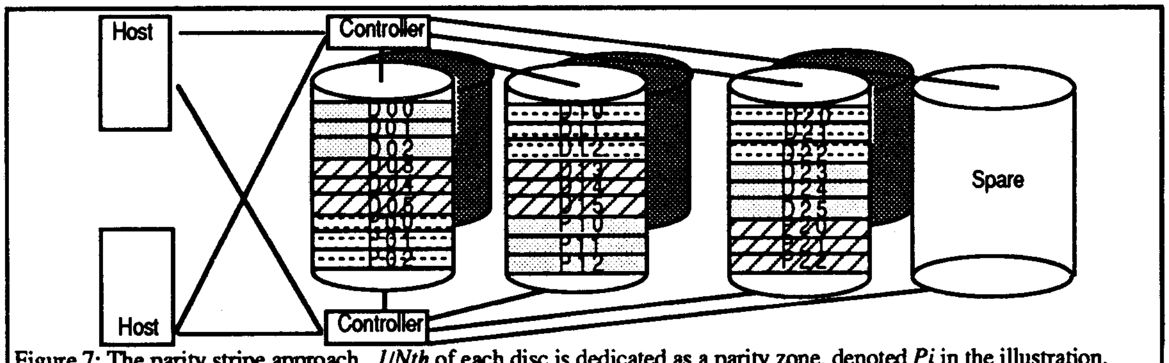


Figure 7: The parity stripe approach. *1/Nth* of each disc is dedicated as a parity zone, denoted *Pi* in the illustration, leaving *N* slightly smaller logical discs containing *N* data zones. Each data zone maps to a distinct parity zone on some other disc -- the color coding shows this mapping. Read requests to a logical disc involve only a single physical disc unless there is a failure. Write requests typically involve only one logical zone and so only two physical discs: the data zone and its parity zone disc. So parity striping has disc utilization similar to mirroring. Each parity zone contains the XOR of all zones which map to it. As with RAID, a spare disc and multiple controllers are configured so that a failed disc can be quickly reconstructed.

The complication of k in the above equation is needed to avoid a disc containing one of its own parity blocks -- if disc j fails, its parity must reside on the remaining N discs[5].

In the normal case all discs are available. When a read request arrives, it goes to a single logical disc and a single physical disc. When a write request arrives, it also goes to a single logical disc. That logical disc is represented by one of the $N+1$ disc data areas and by the parity areas on the other N discs. The number of blocks on a logical disc (D) and and the number of blocks in the parity area (P) are large (typically 10^6 and 10^5 respectively) compared to the number of blocks in a request (typically less than 10.) So most (99.9%) of the requests involve only one parity area. This means that virtually all write requests use only two disc arms -- much as writes to mirrored discs do. So parity striping gives the low-cost/GB of RAID with the low device utilization and consequent high throughput of mirrors -- the only penalties being the extra revolution needed for the writes to compute parity, and the more complex controller to compute the parity.

As Chen argues [Chen], one can configure RAID5 with very large striping units, say a cylinder of 1MB. In that case, almost all read requests to the RAID5 array will involve only one disc, and almost all writes will involve only two discs. But if smaller striping units are used , for example a 32KB disc track, then many multi-block requests will intersect the 32KB boundaries and so will involve more than two discs. This logic seems to force a stripe size at least ten times the typical request size. Such coarse RAID5 configurations will have the same throughput characteristics as parity striping. But such configurations have almost completely abandoned the high parallel transfer rates, and have none of the other advantages of parity striping described later (fault containment, smaller logical discs, software migration).

4. An Analysis of Mirrors, RAID5, and Parity Stripe Performance

The following is a fairly tedious analysis of the three designs. It is summarized in of Figures 10, 11, and 12. Analytic formulae are developed for the average device utilization and average zero-load response time of read and write operations. The analysis covers the no-failure case. It assumes an $N+2$ array of discs configured as a RAID5 with spindle synchronization, vs the same discs configured as a parity stripe array with a spare vs the same $N+2$ disc complex configured as $\frac{N+2}{2}$ mirrored pairs. The next section presents the performance of these three reliable storage configurations on a 10+2 array and compares them to the same 12 discs configured as a standard disc farm.

[5] To make the arithmetic simple, we place the parity stripe at the end of the disc. Since it is frequently written, an organ-pipe distribution would probably placed it in the center cylinders to minimize seek time [Hardy].

In order to compare the three designs, we assume discs have the properties of a "modern disc" as described in Table 1. Later we will consider how the conclusions change when a "future disc" is introduced.

Table 1 Performance characteristics of hypothetical discs.

	Modern Disc [Chan]	Future Disc [Kryder]
capacity:	1GB	1GB
cylinders:	1000	7000
block size:	1KB	1KB
max_seek:	30ms	20ms
rotate:	3600rpm	6000rpm
transfer_rate:	2MB/s	6MB/s
device utilization:	50% avg	50% avg

The analysis assumes infinitely fast processors, channels, and controllers (so that the array parity computation time is zero), and assumes uniform access to data pages. At the beginning of each request, the arms are randomly arranged. This models a typical multiprogramming mix.
Disc seek time is a function of distance (cylinders traveled). For the modern disc traveling across 20% of the cylinders, the arm is constantly accelerating or decelerating and so is modeled by a square root function. Beyond 20%, it is modeled by a linear function. To within 5%, the seek time of Tandem's XL80 discs is approximated by the formula:

$$seek\_time\left(\frac{distance}{cylinders}\right) =$$
$$\text{if } distance < cutoff \quad \text{then } 5 + .64\sqrt{distance}$$
$$\text{else } 14 + \frac{distance - cutoff}{50}$$

where cutoff is 20% of the disc cylinders. Figure 8 shows how well the equation fits the observed seek behavior of these drives [Chan].

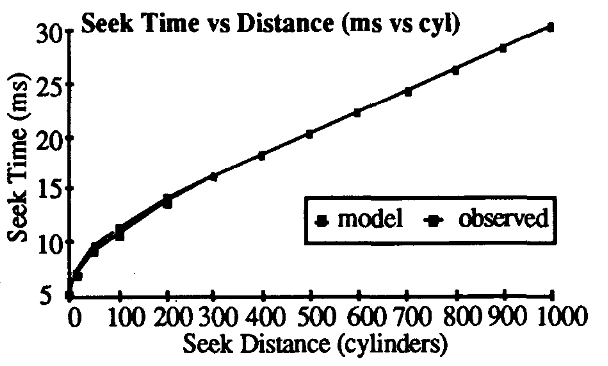


Figure 8: Comparison of analytic seek time model and observed. A maximum error (5%) occurs near 100 cylinders, otherwise the curves are almost identical.

It is convenient to have a formula for the expected seek distance of A arms to one particular cylinder. Assuming each arm is initially randomly positioned [Bitton1] derives:

$$seek(A) = cylinders \bullet \left(1 - \frac{2}{3} \bullet \frac{4}{5} \bullet \frac{6}{7} \bullet \bullet \bullet \frac{2A}{2A+1} \right) \qquad (1)$$

The table and chart for *seek(A)* on the "modern disc" of Table 1 is:

EXPECTED SEEK TIME VS ARMS		
arms	expected seek cylinders	seek time (ms)
1	333	17
2	467	19
3	543	21
4	594	22
5	631	23
10	730	25
15	777	26
20	805	26

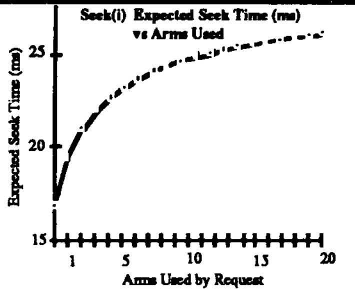


Figure 9: A table and graphs showing the seek time penalty of using multiple disc arms in a request.

4.1. Mirrored Discs: The analysis of access to mirrored discs is taken from [Bitton1]. A read must seek one disc to the target cylinder. Ordinarily this would be *seek(1)*, but using the closest arm reduces seeks to approximately 1/6 of the surface rather than the 1/3 typical of unmirrored discs. This takes approximately 13ms rather than the 17ms seek of a single arm disc. This is modeled as *.8•seek(1)* here, but the real equation is used in the spreadsheet and graph. After the seek, the disc must rotate half a revolution on average before the desired data comes under the disc read head. Finally, the transfer begins at the device *transfer_rate* and lasts for *request_size/transfer_rate* seconds. So the response time of a mirrored disc read request is:

mirror read time (elapsed seconds): *(2)*

$$.8 \bullet seek(1) + rotate/2 + request\_size/transfer\_rate$$

Since only one disc is involved in the transfer, this is also the device-busy time.

mirror read cost (disc seconds): (3)

$$.8 \bullet seek(1) + rotate/2 + request\_size/transfer\_rate$$

Mirrored writes must seek both disc arms, then must wait half a rotation and then transfer.

mirror write time (elapsed seconds): *(4)*

$$seek(2) + rotate/2 + request\_size/transfer\_rate$$

Since two devices (discs) are occupied during this time, the device-busy time is:

mirror write cost (disc seconds): *(5)*

$$2 \bullet (seek(2) + rotate/2 + request\_size/transfer\_rate))$$

The analysis assumes that the two writes are done in parallel on spindle synchronized discs (rather than writing the first disc and then the second, sometimes called serial writes).

4.2. Parity Stripe Discs: For parity striping, the read equations are almost exactly the same as for mirroring -- except that parity stripe reads do not get the shortest seek optimization of mirrors. Parity stripe reads seek 1/3 of the disc rather than the 1/6 typical of mirrored disc reads [Bitton1]. The read equations are:

parity stripe read time (elapsed seconds): *(6)*

$$seek(1) + rotate/2 + request\_size/transfer\_rate$$

Since only one disc is involved in the transfer, this is also the device-busy time.

parity stripe read cost (disc seconds): *(7)*

$$seek(1) + rotate/2 + request\_size/transfer\_rate$$

The parity stripe write equations are more complex because the relevant parity disc(s) must first be read, the old data XORed out, the new data XORed in, and then the new data and new parity can be written in parallel as a unit. As argued before, all but .1% of the write requests involve only one parity stripe. So the analysis here just accounts for the common case. The write seek must move both arms as in the mirrored read. Then the disc must rotate .5 revolutions before the data begins. Then the disc must rotate one revolution to compute the new parity (read the old parity and data from disc and XOR them with the new data). So there is a 1.5 revolution wait before the write transfer can begin.

parity stripe write time (elapsed seconds): *(8)*

$$seek(2)+1.5 \bullet rotate+ request\_size/transfer\_rate)$$

and since two devices are used, the device busy-time is:

parity stripe write cost (disc seconds): *(9)*

$$2 \bullet (seek(2)+1.5 \bullet rotate+ request\_size/transfer\_rate)$$

4.3. RAID5: For RAID5 discs, a read of *S* striping units involves *A* discs, where *S = request_size/block_size* and *A = min(S, N+1)* . These discs must all seek, then rotate, and then the read transfer can begin. The seek time is the max of the seek times of each disc. Once the seek completes, the read waits for an average rotation and then can transfer at a rate of *A • transfer_rate*. So the response time for a RAID5 read is:

RAID5 read time (elapsed seconds): *(10)*

$$seek(A) + \frac{rotate}{2} + \frac{request\ size}{A \bullet transfer\_rate}$$

Since *A* discs are involved in the transfer, the device-busy time is *A* times the service time:

RAID5 read cost (disc seconds): *(11)*

$$A \bullet (seek(A) + \frac{rotate}{2} + \frac{request\ size}{A \bullet transfer\_rate})$$

For writes, an extra disc is involved if *A < N+1*, so define *A' = min(S+1,N+1)*. The A' discs must all seek, rotate to the parity, and then the parity must be read and the new parity computed during a rotation. Then a write of the data can be done at the high transfer rate. The equations are:

RAID5 write time (elapsed seconds): *(12)*

$$seek(A') + 1.5 \bullet rotate + \frac{request\ size}{A \bullet transfer\_rate}$$

RAID5 write cost (disc seconds): *(13)*

$$A' \bullet (seek(A') + 1.5 \bullet rotate + \frac{request\ size}{A \bullet transfer\_rate}).$$

This analysis ignores several important phenomenon:

- The parity areas are likely to be warmspots; they get N times the update traffic of other parts of each disc. Half of all the write traffic goes to the parity area of each disc -- in a 10+2 array, 50% of the writes go to 9% of the data. So there may be queueing on writes to the parity areas in the parity stripe and RAID5 schemes.
- In the RAID5 schemes, a request is for a particular sequence of A of the N discs. This tends to make RAID5 reads and writes of more than 2 discs interfere with one another much more than they would in a mirrored or parity stripe scheme. So it may be difficult to run each disc at 50% utilization.
- The analysis is for the no-fault case. In case a block or disc fails, the load on each other disc approximately doubles (driving them to 100% utilization if they were 50% utilized before!). In a mirrored disc scheme, the failure of one disc causes the load on its mirror to increase by the read traffic. So a mirrored pair running with ten logical reads and ten logical writes per second places a 15 request/second load on each drive. When one drive fails, the other now must carry all 10 reads and so gets a 20 request/second load, a 33% increase rather then the 100% increase of disc arrays. Using the "modern disc" numbers of Table 1 on 16KB requests, this would move the mirror from 44.5% utilized to 54% utilized.

All these shortcomings in the analysis tend to give optimistic estimates of RAID and parity stripe throughputs compared to mirrored schemes. The only assumption here unfavorable to RAID is the assumption that requests are not aligned to the array geometry. If requests are aligned to $N \bullet B$ boundaries and are in units of $N \bullet B$ bytes, then no extra rotations are needed to compute parity. On the other hand, if the transfer is very large and involves multiple rotations, the analysis here assumes the subsequent aligned transfers do not require a parity read, and indeed the analysis does not charge for the extra rotation that will generally be required to write the unaligned suffix of such a large transfer. We believe it unreasonable to expect applications to be aware of disc geometry.

5. Applying the Analysis to a 10+2 array

Equations (7)-(13) can be used to compute the minimum response time (zero load response time) for an array of N discs configured as mirrors, parity stripe, or RAID5. In this analysis, the number of discs is held constant and the request size is varied from small (one block unit = 1KB) to large (1MB) in powers of 2. For simplicity, only the pure read case (only read requests), and the pure write case (only writes) are analyzed, rather than a mix of reads and writes. Also, rather than looking at a mix of request sizes, the analysis looks at workloads of constant request size and simply grows the request size by powers of two. These simplifications make the analysis tractable (trivial). We believe that elaborating the model to include these refinements would not change the conclusions.

A second issue is how the array behaves under load. Assuming that the discs are run at 50% utilization, a 10+2 disc array will use 50% of 11 discs (recall that the spare disc in the array does not provide service) giving 5.5 disc seconds of service per second; while mirrored disc will use all 12 discs and give 6 seconds of service per second. So equations (7), (9), and (13) imply the array throughput in each case.

In looking at the following figures, realize that application disc requests are typically less than 10KB today, while utilities issue requests in the 50KB range. Larger transfers are rare, because they monopolize the IO subsystem with little benefit. Rather, large transfers are typically broken into multiple "small" 50ms transfers so that truly small transfers can be serviced during the gaps between the large transfers. This allows a kind of priority scheduling of devices and paths, and at 50% utilization suggests that the average response to a disc request will be about 100ms. A virtue of striping is that it can transfer much more data in these 50ms windows by doing parallel transfers from each disc. But, to repeat: current software typically operates in the first columns of these graphs (less than 10KB requests), next generation software is likely to be in the second columns (10KB to 100KB). The high transfer rate of RAID5 only begins to dominate at the high-end of this range, and so is a poor design for traditional applications.

Also realize that the fine (1KB) striping unit was chosen to show the extreme case. If we had chosen a 1MB striping unit, then RAID5 and parity stripe would have virtually identical performance curves. So, the real point of this section is that fine granularity striping is a bad tradeoff for OLTP systems. This echos the conclusions of [Chen]. Given that parallel transfers are not used by RAID5 in OLTP applications, the next section argues the merits of parity striping over RAID5 in terms of benefits other than performance.

5.1. The Read-Only Case

Figure 10 shows the read performance of RAID5, mirrors, and disc striping. An array of 12 standard discs has approximately the same read performance as mirrors, and so is not shown.

Figure 10 indicates that for small transfers (less than 32KB) the shortest-seek benefit of mirrors gives them a distinct advantage over striping which must seek over 1/3rd of the surface rather than 1/6th as in mirrors [Bitton1]. After the request size exceeds about 24KB, the high transfer rate of RAID begins to dominate -- and RAID5 shows considerable response time benefit. The throughput of disc arrays is not attractive in any range because each large request consumes seek and rotation time on multiple discs. The RAID5 throughput curve drops steeply until all discs are involved in each request and then holds approximately constant for request sizes between 11KB and 256KB. *In this range, the array throughput is typically five times worse than either the mirror or parity stripe throughput.*

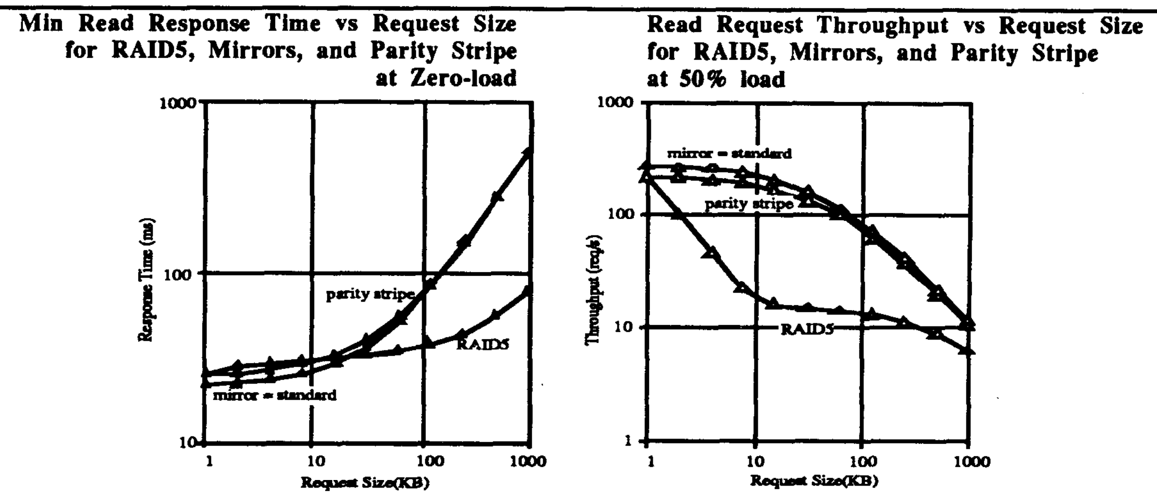


Figure 10: Log-log plots of the read performance of the three disc architectures vs request size. The RAID5 striping unit is assumed to be 1KB, the discs are assumed to be spindle synchronized, the requests are assumed uniformly distributed to all locations. The discs are run at 50% utilization to compute the throughput. A conventional array of 12 discs would have read performance similar to mirrors.

Throughout the range, mirrors have a slight advantage over parity stripe because they benefit from the shortest seek optimization, and because they can use all 12 disc arms.

The model does not include the obvious optimization form mirrors: splitting large reads in half and sending half of the read to each arm. Using this optimization for reads of 64KB or more would slightly degrade throughput but would give mirrors some of the response time advantages of the RAID5 scheme. The Tandem sequential scanning software includes this optimization.

5.2. The Write-Only Case

The picture for writes is not much different. For small transfers (those less than 64KB), mirrors have better response time and throughput than do the other kinds of arrays (because they avoid the extra rotation needed for parity stripe and RAID). Looking at throughputs (requests/second), mirrors consistently outperform any kind of striping because of striping's rotational penalty. In the 10kb to 100kb range, mirrors service five times as many requests as RAID5 discs. These charts show the virtue of parity striping. At a *relatively* minor cost (50% more response time and ~30% less throughput), parity striping gives the the low $/GB cost of RAID5 without the 500% r/s throughput penalty. Using the array as a standard disc farm (no reliable storage) gives approximately the same minimum response time as mirrors (the mirror seek time *seek(2)* of Figure 9 rather than *seek(1)*), but standard discs give twice the throughput since they use only one arm per write. At 16KB requests: the write throughput of a mirrored drive is 46% of a standard drive, the write throughput of parity striped discs is 29% of standard, and the write throughput of RAID5 is 9% of standard

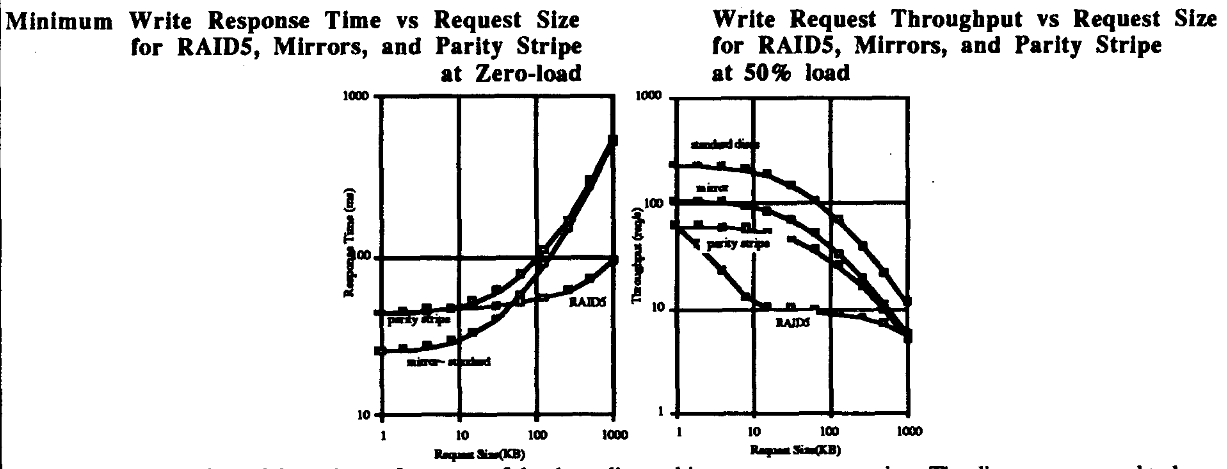


Figure 11: Log-log plots of the write performance of the three disc architectures vs request size. The discs are assumed to be spindle synchronized, the requests are assumed uniformly distributed to all locations. The discs are run at 50% utilization to compute the throughput. Standard shows the throughput of the array used as 12 independent discs. Standard disc writes have approximately the same minimum response time as mirrored writes.

5.3. Analyzing A High Performance Disc

One might object that these results are dependent on the disc technology; that the conclusions would be very different if the discs were much higher performance, or much lower performance. Examination of equations (2)-(13) shows this is not so. We have looked a many kinds of discs, and include here the curves for the "future disc" of Table 1 which seeks and rotates about 70% faster, and has three times the data transfer rate. The curves for that disc are given in Figure 12 and show the same pattern: mirrors have the best response time and throughput below 32KB, fine granularity striping to get parallel transfers is a poor response-time tradeoff, and so coarse striping in the style of parity striping or [Chen] is very attractive if the goal is low cost per reliable GB. At 16kb request sizes, RAID5 has about a 1000% throughput penalty compared to mirrors, and parity striping represents a compromise -- providing reliable storage with throughput only 40% worse than mirrors.

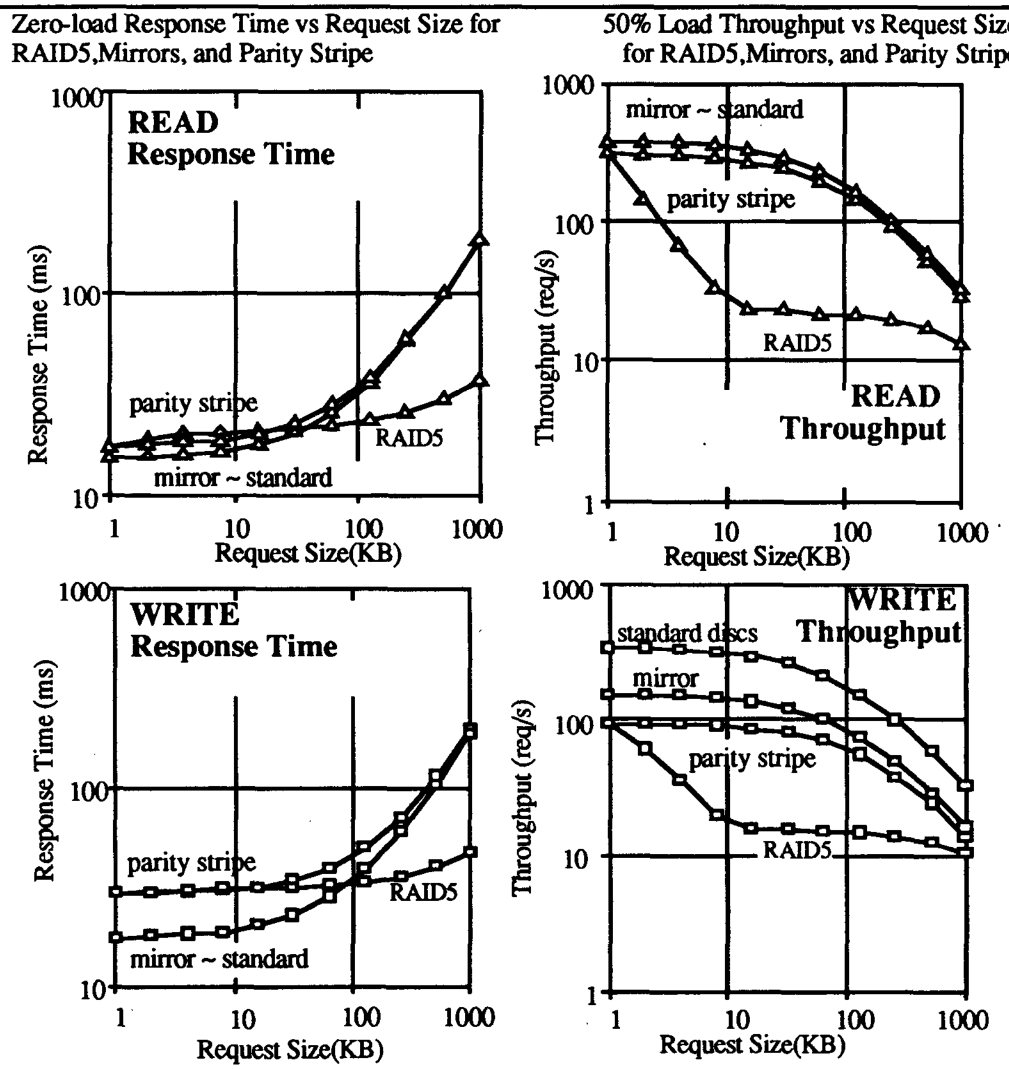


Figure 12: Log-log plots of the performance of the three disc architectures vs request size. The discs are assumed to be higher performance (12ms seek, 6000rpm, and 6MB/s transfer) than the discs in the previous charts. These charts still indicate a preference for mirrors over RAID5 if throughput is the goal.

6. Other Benefits of Parity Striping

The argument made so far for parity striping over RAID5 has been in terms of performance one. But one could simply use RAID5 with a stripe size of $\frac{B}{N+1}$ and get the same effect. [Chen]. So why bother with parity striping? Parity Striping has several advantages over a traditional RAID5 design -- even when it is configured with large (say 1MB striping units). In fact the idea of parity striping grew out of attempts to fit a RAID5 into a Tandem system. Parity striping circumvents some of the problems we encountered with a traditional RAID5 approach. These problems include:

1. **Archiving a Giant Disc**: Since the data is spread across all the discs, archiving software must copy the contents of the entire array, rather than just one spindle.
2. **Operations with a double failure**: When a second disc in the array fails during the repair window, the entire array is damaged and must be restored from the archive (see 1 above). The restoration of a giant disc (say 11GB) from a tape drive at 3MB/s will take an hour vs the five minutes needed to restore a single drive. Agreed, this only happens rarely, but when it happens it is a big event.
3. **Load balancing:** Load balancing of requests across a RAID array is not controllable: in particular one cannot place two different files on two different discs unless the file system does some very complex arithmetic. Rather, one must hope that striping does not cause bad interference. This is the converse of most RAID arguments which point to the automatic load balancing that comes from spreading all files across all discs. If coarse striping units (say 1MB) are chosen for RAID5, then the load balancing benefits claimed for RAID disappear since hotspots tend to be of that size, but the ability to manually control the location of files does not return.
4. **The software problem**: Introducing a new disc which is an order of magnitude larger and which processes ten times as many requests per second will break most software designs. Here is a partial list of Tandem software problems presented by a 10+2 array of modern drives. We believe that MVS, VMS, and most Unix implementations have similar problems.
   - The disc server software uses 31 bit addressing for the disc cache (disc buffer pool) -- this limits the disc cache to 2GB. So a disc array of will have a limited cache size compared to six mirrored disc servers with a 12GB aggregate cache. In fact, due to fault tolerance, and to some data structures the current Tandem disc cache is limited to 56MB per logical drive so this problem is quite extreme. Main memory databases will only be able to use 56MB of any size array.
   - The software uses 32 bit addressing for files, and uses partitioning of files across discs to get file sizes up to a Terabyte. If the array appears to be a single logical disc, it will be impossible to fill it with one large file.
   - The software assumes that there are *frequent* operations (e.g. read and write) and *rare* operations (e.g. b-tree split, update directory, and so on). Frequent operations run in parallel and typically execute in cache. Rare operations acquire a semaphore to simplify concurrency and fault tolerance. With eleven discs viewed as one, rare operations will be eleven times more common. This will likely cause bottlenecks on the semaphores. In particular: the disc directory is stored as a single file and updates to it are covered by a semaphore and some elaborate fault-tolerance logic. If the directory update rate increases by an order of magnitude, the current logic will bottleneck and will have to change.
5. **The bandwidth problem**: Building controllers and channels that can run at 100MB/s is non-trivial. Current controllers and channels run one or two orders of magnitude slower than this. For the uninitiated, IO channels are like LANs but operate at mega-BYTES-per-second rather than mega-BITS-per-second. A 100MB/s channel is about 1000 times faster than LANs like Ethernet. In addition, many applications can't use the high bandwidth until processors of 100MIPS or more are commonplace.
6. **Exotic Controllers**: Rather than using standard controllers (as with mirrors), disc arrays depend on exotic controllers and spindle synchronized discs. In addition, they require complex controller logic (software) to retry the operation via a second controller if the first controller fails. Such exotic controllers will not be cheap and may adversely affect the price advantage of disc arrays when compared to mirrors using standard controllers.
7. **Performance with a single failure**: When a single disc in the array fails, the load on the remaining discs doubles. With mirrors, when one disc fails the read load on the mirror doubles, but the write load is not changed. So the net change in load is typically a 33% increase on one drive rather than 100% increase on 10 drives. The real story is even worse than this since the reconstruction of the lost disc on the spare disc will add to the load.
8. **The parity hotspot problem**: Half the update traffic of each disc is parity updates. In the 10+2 array, half of the updates go to 10% of the data. This may make the parity areas hotspots, further exaggerating the load balancing issue (problem 3 above).

Parity striping exploits problem 5, sacrificing bandwidth to solve problems 1, 2, 3, and 4. We have no answer for problems 6, 7, and 8. Perhaps experience will show that these are not really problems after all. After all, problem 4 is just smop (a simple matter of programming).

Parity striping solves the giant disc problem (1 above) by making each physical disc a smaller logical disc. So the 10+2 array looks like eleven logical discs each containing 1GB. The data of these logical discs can be archived and restored independently. The parity can be reconstructed from the other discs and so need not be archived.

If two discs of a parity stripe array fail, then the data of those two discs must be restored from the archive. But the data on the other *N-1* discs is still available for reading and writing (solving problem 2 above). In particular the file directory and B-trees of each surviving logical disc are still intact. So parity striping has better fault containment than RAID5 designs. Double failures are a rare event if everything goes well (once in 500 years according to [Schulze]). But when such failures happen they will be an big event.

Ignoring the parity hotspot problem (problem 8), load balancing an *N+1* parity stripe disc is just like load balancing a *N+1* array of standard or mirrored discs (problem 3 above).

A parity striped disc array looks like *N+1* modern discs to the software, and so should introduce minimal software disruption beyond the need to retry via the alternate controller if the first controller fails. This is a relatively minor and isolated change to the software. So parity striping solves many of the software problems posed by reliable disc arrays (problems 4 above).

7. Summary

Perhaps a better way to look at this whole argument is:

1. Parity striping is just a variation of RAID5. The parity techniques, recovery techniques, reliability analysis, and hardware requirements are the same for both schemes. All that differs is the way the data is mapped to the disc array.
2. For a large class of applications, a large stripe size (say $\frac{B}{N+1}$) is appropriate [Chen].
3. Given (2), the high parallel transfer rate of disc arrays is lost, and the automatic load balancing claims of RAID do not apply.
4. Current software will have a difficult time with giant discs implied by RAID5.
5. So, rather than map the array as one big logical disc, parity striping maps it as *N+1* conventional (but very reliable) discs. Each logical disc maps to most of one physical disc. This has fault containment, load balancing, and software benefits.

Previous analysis of disc arrays used for reliable storage focused on their attractive cost/GB and their high data transfer rate. The discussion here focused on response time and throughput. With that point of view, mirrored discs are the best reliable storage choice for applications which are disc-arm limited and which cannot absorb data at current device speeds. Parity striping offers the low cost/GB of disc arrays, while sacrificing the high transfer rate of RAID5 schemes, and accepting a 40% reduction in throughput compared to mirrored schemes. Perhaps the best way to see these differences is to look at the price and performance of a single modern disc in an array configured in the four different ways: This table shows that parity striping provides an attractive compromise between RAID5 and mirrors. As argued in the previous section, parity striping has some additional advantages over RAID5: it has preferable fault containment and operations features. Perhaps most importantly, it causes minimal software disruption. Its major drawback when compared to RAID5, is the reduced data transfer bandwidth -- 2MB/s rather than 22MB/s with current discs and 6MB/s rather than 66MB/s with future discs. For many applications, only a small part of disc response time is data transfer time, so this bandwidth advantage at the cost of increased queueing is a false economy (see Table 2 above).

Given this point of view, we recommend that anyone implementing an array controller should support standard discs (no parity), mirrored discs, RAID5, and parity stripe. Standard discs give the best performance and cost/GB, mirrors give the highest throughput reliable storage, RAID5 gives high-transfer rates to and from reliable storage, and parity stripe gives the reliable storage with cost/GB of RAID5, but has additional benefits. The marginal cost of supporting all these options should be small since the hardware requirements for RAID5 and parity stripe are identical (parity and spindle synchronization logic), and since the software to do the different data mappings is literally a few lines of code.

Table 2: Comparison of the four designs on a 10+2 array with 16KB requests.

Configuration	Cost/GB[6]	Throughput @50% utilization requests/arm/s		Minimum Response Time ms/request	
		read	write	read	write
Standard	1.0k$/GB	15.2r/s	15.2r/s	33ms	33ms
Mirrors	2.0k$/GB	16.9r/s	7.0r/s	30ms	36ms
Parity Stripe	1.2k$/GB	13.8r/s	4.4r/s	33ms	52ms
RAID5	1.2k$/GB	1.3r/s	0.8r/s	32ms	49ms

[6] The cost of the controllers is ignored here. Standard and mirror configurations can use standard commodity controllers, while parity stripe and RAID5 designs require specialized and high-performance controllers to do the parity computation, spindle synchronization, and data reconstruction. So the price advantages of parity stripe and RAID5 are overstated here.

8. Acknowledgments

The idea of parity striping grew out of discussions with Andrea Borr, Franco Putzolu, and Todd Sprenkel. We were all trying to understand how a RAID5-style array would fit into the Tandem environment. These discussions defined the problems. The parity stripe solution was easy once the problems were stated. Although not highlighted, the retry logic in this paper is new. It works for both parity stripe and RAID5 arrays. Peter Chen and Garth Gibson have been a constant source of ideas and have patiently listened to ours. We were especially interested by Peter Chen's recent paper [Chen] which suggests coarse striping. Wing Chan provided us with empirical measurements of a modern disc. The analysis here benefited considerably from papers by and discussions with Dina Bitton. The paper also benefited from a spirited debate when it was presented at the RAID systems seminar at UC Berkeley. Garth Gibson, Randy Katz, Dave Patterson, and Mike Stonebraker each made detailed and specific comments to improve the paper -- we tried to include their advice.

9. References

[AS400] *AS400™ Programming: Backup and Recovery Guide*, IBM Form No. SC21-8079-0, June 1988. Note: parity discs are called "check disks" in the AS400.

[Chen] Chen, P., Patterson, D. "Maximizing Performance in a Striped Disc Array", to appear in Proc. ACM SIGARCH 90.

[Gelb] Gelb, J.P., "System-Managed Storage", IBM Sys. J. V28.1 1989. pp. 77-103

[Hardy] Hardy, G.H., Littlewood, J.E., Polya, G., *Inequalities*, Cambridge U. Press. Cambridge 1934. Chap. 10. or see D. Knuth, Art of Computer Programming, V3., Section 6.1 exercise 18. Addison Wesley, 1973.

[Patterson] Patterson, D.A., Gibson, G., Katz, R., "A Case for Redundant Arrays of Inexpensive Discs (RAID)", Proc. ACM SIGMOD June, 1988. pp. 106-109.

[Bitton1] Bitton, D., Gray, J., "Disk Shadowing", VLDB 1988 Proceedings, Morgan Kauffman, Sept 1988. pp. 331-338

[Bitton2] Bitton, D., "Arm Scheduling in Shadowed Disks", COMPCON 1989, IEEE Press, March 1989. pp 132-136.

[Chan] Chan, W., Private communication with Wing Chan of Tandem Computers, Cupertino, CA..

[Katzman] Katzman, J.A., "A Fault Tolerant Computing System," Proc. 11th Hawaii Int. Conference on System Sciences, Honolulu, Hawaii, Jan 1978. pp. 85-102.

[Kim] Kim, M.Y., "Synchronized Disk Interleaving", IEEE TOC, V.3-35.11, Nov 1986. pp 978-988.

[Schneider] Schneider, D.A., DeWitt, D.J., " A Performance Evaluation of Four Parallel Join Algorithms in a Shared-Nothing Multiprocessor Environment", ACM SIGMOD Record, V 18.2, June 1989, pp. 110-121.

[Schulze] Schulze, M., Gibson, G., Katz, R., Patterson, D., "How Reliable is RAID", CompCon 1989. IEEE Press, March 1989. pp. 118-123.

[Scranton] Scranton, R.A., Thompson, D.A., "The Access Time Myth", IBM Research Report RC 10197 (#45223) Sept. 1983.

[Wilhelm] Wilhelm, N.C., "A General Model for the Performance of Disc Systems," JACM, V24.1 Jan. 1977, pp. 14-31.

10. Appendix: Spreadsheets of Graphs 8,9 and 10

Modern Disc: 17ms seek, 3600rpm, 2MB/s.

		device busy time (10+2 drives)							
request	arms	**parity stripe**		**array**		**mirrors**		**standard**	
size	used	**read**	**write**	**read**	**write**	**read**	**write**	**read**	**write**
1	1	26	90	26	90	22	56	26	26
2	2	26	91	56	139	23	57	26	26
4	4	27	93	119	237	24	59	27	27
8	8	29	97	246	433	26	63	29	29
16	11	33	105	349	538	30	71	33	33
32	11	41	121	363	551	38	87	41	41
64	11	57	153	384	571	54	119	57	57
128	11	89	217	420	607	86	183	89	89
256	11	153	345	487	674	150	311	153	153
512	11	281	601	618	804	278	567	281	281
1000	11	525	1089	865	1051	522	1055	525	525
response time (10+2 drives)									
req size		read	write	read	write	read	write	read	write
1	1	26	45	26	45	22	28	26	26
2	2	26	45	28	46	23	29	26	26
4	4	27	46	30	47	24	30	27	27
8	8	29	48	31	48	26	32	29	29
16	11	33	52	32	49	30	36	33	33
32	11	41	60	33	50	38	44	41	41
64	11	57	76	35	52	54	60	57	57
128	11	89	108	38	55	86	92	89	89
256	11	153	172	44	61	150	156	153	153
512	11	281	300	56	73	278	284	281	281
1000	11	525	544	79	96	522	528	525	525
throughput (10+2 drives)									
req size		read	write	read	write	read	write	read	write
1	1	215	61	215	61	271	106	235	235
2	2	211	61	98	40	265	105	230	230
4	4	203	59	46	23	254	101	222	222
8	8	189	57	22	13	234	95	207	207
16	11	166	53	16	10	203	84	182	182
32	11	134	46	15	10	160	69	146	146
64	11	96	36	14	10	112	50	105	105
128	11	62	25	13	9	70	33	67	67
256	11	36	16	11	8	40	19	39	39
512	11	20	9	9	7	22	11	21	21
1000	11	10	5	6	5	12	6	11	11

Dina's formula

arms	formula	distance	seek time
1	0.33	333	17
2	0.47	467	19
3	0.54	543	21
4	0.59	594	22
5	0.63	631	23
6	0.66	659	23
7	0.68	682	24
8	0.70	700	24
9	0.72	716	24
10	0.73	730	25
11	0.74	741	25
12	0.75	752	25
13	0.76	761	25
14	0.77	769	25
15	0.78	777	26
16	0.78	783	26
17	0.79	790	26
18	0.80	795	26
19	0.80	801	26
20	0.81	805	26
Mirrored disc Shortest Seek			
	0.16	167	13

Parameter values

drives	10	_+2
stripe	1	KB
cylinders	1000	cyls
rpm	3600	rpm
transfer_rate	2	MB/s
max_seek	30	ms
settle	5	ms
cutoff	0.2	
slope1	0.64	
slope2	0.01994	
base	14.051	ms

Future disc: 10ms seek, 6000rpm, 6MB/s.

device busy time (10+2 drives)

request	arms	parity	stripe	array		mirrors		standard	
size	used	read		write	read	write	read	write	read
1	1	18	58	18	58	15	38	18	18
2	2	18	59	38	90	16	39	18	18
4	4	18	59	80	153	16	39	19	19
8	8	19	61	165	279	17	41	19	19
16	11	20	63	232	346	18	43	20	20
32	11	23	69	238	351	21	49	23	23
64	11	28	79	247	359	26	59	28	28
128	11	39	101	259	371	37	81	39	39
256	11	60	143	283	394	58	123	60	60
512	11	103	229	327	439	101	209	103	103
1000	11	184	391	410	521	182	371	184	184

response time (10+2 drives)

req size		read	write	read	write	read	write	read	write
1	1	18	29	18	29	15	19	18	18
2	2	18	29	19	30	16	19	18	18
4	4	18	30	20	31	16	20	18	18
8	8	19	30	21	31	17	20	19	19
16	11	20	32	21	31	18	22	20	20
32	11	23	34	22	32	21	24	23	23
64	11	28	40	22	33	26	30	28	28
128	11	39	50	24	34	37	40	39	39
256	11	60	72	26	36	58	62	60	60
512	11	103	114	30	40	101	104	103	103
1000	11	184	196	37	47	182	186	184	184

throughput (10+2 drives)

req size		read	write	read	write	read	write	read	write
1	1	311	94	311	94	389	156	339	339
2	2	308	94	143	61	385	155	336	336
4	4	302	93	69	36	376	152	330	330
8	8	291	91	33	20	361	147	318	318
16	11	272	87	24	16	334	138	297	297
32	11	240	80	23	16	291	123	262	262
64	11	195	69	22	15	231	101	213	213
128	11	141	55	21	15	164	74	154	154
256	11	91	38	19	14	104	49	11	100
512	11	53	24	17	13	60	29	58	58
1000	11	30	14	13	11	33	16	33	33

Dina's formula

write Dina's formula

arms	formula	distance	seek time
1	0.33	333	13
2	0.47	467	14
3	0.54	543	15
4	0.59	594	15
5	0.63	631	16
6	0.66	659	16
7	0.68	682	16
8	0.70	700	17
9	0.72	716	17
10	0.73	730	17
11	0.74	741	17
12	0.75	752	17
13	0.76	761	17
14	0.77	769	17
15	0.78	777	18
16	0.78	783	18
17	0.79	790	18
18	0.80	795	18
19	0.80	801	18
20	0.81	805	18

Mirrored disc Shortest Seek

0.16	167	10

Parameter values

drives	10	_+2
stripe	1	KB
cylinders	1000	cyls
rpm	6000	rpm
transfer_rate	6	MB/s
max_seek	20	ms
settle	2	ms
cutoff	0.2	
slope1	0.64	
slope2	0.01119	
base	11.051	ms