The Performance of Parity Placements in Disk Arrays

Edward K. Lee Randy H. Katz

Computer Science Division
Electrical Engineering and Computer Science
University of California at Berkeley
Berkeley, California 94720

Abstract

Due to recent advances in CPU and memory system performance, I/O systems are increasingly limiting the performance of modern computer systems. Redundant Arrays of Inexpensive Disks (RAID) have been proposed by Patterson et al. [1] to meet the impending I/O crisis. RAIDs substitute many small inexpensive disks for a few large expensive disks to provide higher performance (both transfer rate and I/O rate), smaller footprints and lower power consumption at a lower cost than the large expensive disks they replace. Unfortunately, with so many small disks, media availability becomes a serious problem. RAIDs provide high availability by using parity encoding of data to survive disk failures. As will be shown, the way parity is distributed in a RAID has significant consequences for performance. In particular, we investigate the performance of eight different parity placements via simulation and show that these eight parity placements span a limited space of parity placements. We show that although for small request sizes the choice of parity placement does not have a significant effect on performance, for relatively large request sizes of hundreds of kilobytes at low loads, the choice of parity placement significantly affects performance (up to 20-30 percent for the typical disk array configurations that are common today). We also propose properties that are generally desirable of parity placements.

1 Motivation

In recent years, improvements in CPU and memory system performance has greatly outpaced improvements in I/O performance. If the trend continues, future improvements in CPU and memory system performance will be wasted as computer systems become increasingly I/O bound. To overcome the impending I/O crisis, Patterson et al. have proposed Redundant Arrays of Inexpensive Disks (RAID) [1–3].

RAIDs substitute many small inexpensive disks for a few large expensive disks to provide higher

performance (both transfer rate and I/O rate), smaller footprints and lower power consumption at a lower cost than the large expensive disks they replace. Unfortunately, with so many small disks, media availability becomes a serious problem. RAIDs provide high availability by using parity encoding of data to survive disk failures. Patterson [1] and Chen [4] define six different RAID organizations:

RAID level 0 Non-redundant disk array. Only data striping is supported.

RAID level 1 Mirrored disk array. Data is duplicated for reliability.

RAID level 2 Hamming-coded disk array.

RAID level 3 Parity-protected disk array with byte-interleaved data. Reads access all disks except the parity disk and writes access all disks. Only one I/O request may be serviced per parity disk at a time.

RAID level 4 Parity-protected disk array with block-interleaved data. Small reads access a single data disk and small writes access a data disk and a parity disk. Several reads and a single write per parity disk may be serviced concurrently.

RAID level 5 Parity-protected disk array with block-interleaved data and distributed parity. Similar to RAID level 4 except that the parity is distributed across all disks. Several reads and writes per parity disk may be serviced concurrently.

This paper will investigate the performance implications of different ways of distributing parity (*parity placements*) in RAID level 5 disk arrays.

Previous work in the field has been limited, dealing primarily with parity placements for specific applications or specific optimizations for currently known parity placements. Dibble [5] has investigated parity placements under the restriction that all writes are small. A large write is, therefore, broken into many independent small writes. We do not assume such a restriction. Gray [6] has proposed a parity placement scheme in which parity is striped but the data is not. Menon [7] has proposed that the exact placement of the parity and data be varied dynamically in order to reduce the rotational latency. This paper differs from those mentioned above by looking at parity placements within a more general framework.

Due to the relatively recent surge of interest in the area, most of the papers written about RAID level 5 disk arrays have either been unaware of the performance consequences of parity placements

or have assumed that all parity placements result in equivalent performance. The result has been a dissemination of ad-hoc parity placements in the literature, encouraging the adoption of arbitrary parity placements by commercial vendors without performance justifications. This is regrettable given the small difference in complexity of implementing one parity placement over another. Thus, if one initially knew the performance trade-offs involved in selecting parity placements, immediate performance benefits could be achieved at negligible cost. Specifically, we will investigate a limited space of parity placements and show that at relatively large request sizes that are typical of scientific applications [8], the choice of parity placement results in significant differences in both read and write I/O performance (20-30 percent for the typical disk array configurations that are common today). Based on our studies, we will also propose properties that are desirable of parity placements in general. Finally, we will conclude by making general recommendations for the selection of a parity placement for a given system.

This paper is an extension of work previously reported [9]. Section 2 defines terms related to RAID that will be used throughout the paper and serves as a brief introduction to RAID. Section 3 defines eight parity placements and points out, for the first time, an important distinction between physical and logical parity placements. Section 4 compares the simulated performance of the previously defined parity placements for homogeneous workloads consisting either of reads or writes. For the two most promising parity placements, workloads consisting of a mixture of reads and writes are examined for the first time. Based on the simulation results, we then propose properties that are desirable of good parity placements. Section 5 argues that the eight parity placements examined are the only interesting placements from a space of parity placements. Finally, we conclude by summarizing our main results and making general recommendations in the selection of parity placements.

This work was heavily influenced by experiences gained in designing, developing, using and analyzing RAID systems. RAID-I, our first RAID prototype was built using off-the-shelf hardware and has been in experimental use for the past year. The hardware components of RAID-II, a much more ambitious second prototype using custom hardware, is currently operational and is undergoing software integration. RAID-II should be fully operational in the next six months.

2 Overview

This section introduces concepts and defines terms that will be used throughout the paper.

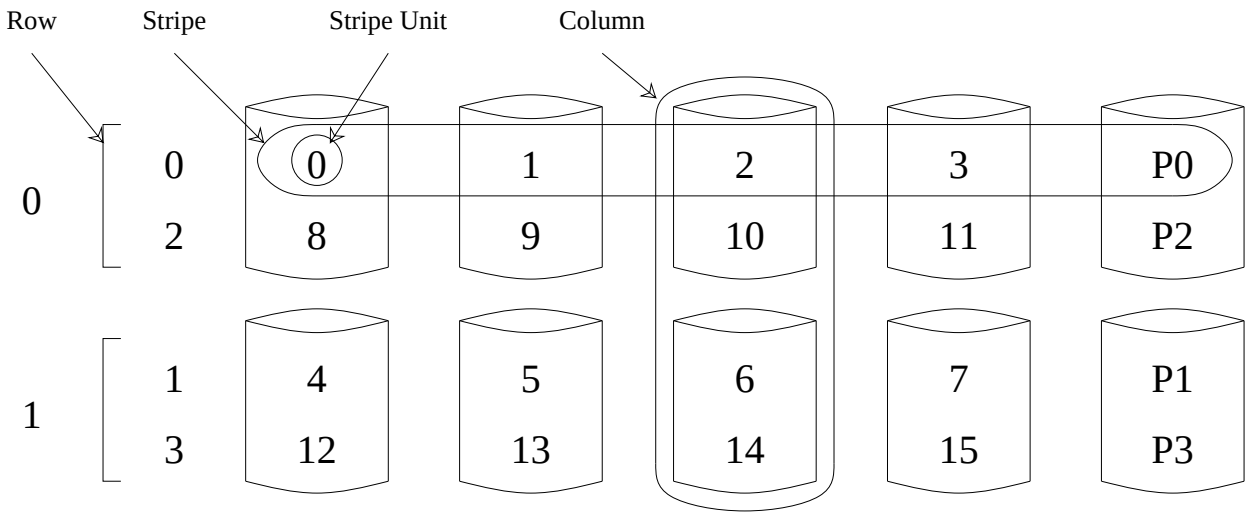

Fig. 1. Data Mapping. *This figure illustrates the data mapping entities,* stripe unit, stripe, row *and* column *in the context of a RAID level 4 disk array.*

2.1 Data Mapping

Ignoring for now the placement of parity, the following data mapping entities are considered. Figure 1 illustrates each mapping entity.

The **stripe unit** is the unit of data interleaving, the amount data that is placed on a given disk before placing data on a different disk.

The **parity stripe** is the minimal collection of stripe units over which parity is computed.

The **row** is the minimal collection of disks over which a parity stripe can be placed.

The **column** is the collection of disks formed by taking one disk from each row. Note that a disk may not belong to more than one column.

2.2 I/O Request Servicing

Figure 2 illustrates the ways in which I/O requests to RAID level 4 and RAID level 5 disk arrays are serviced when there are no failed disks. A read-modify-write computes the new parity by xoring the new data, old data and old parity. A reconstruct-write computes the new parity by xoring the new data and the data from the part of the parity stripe that is not being written.

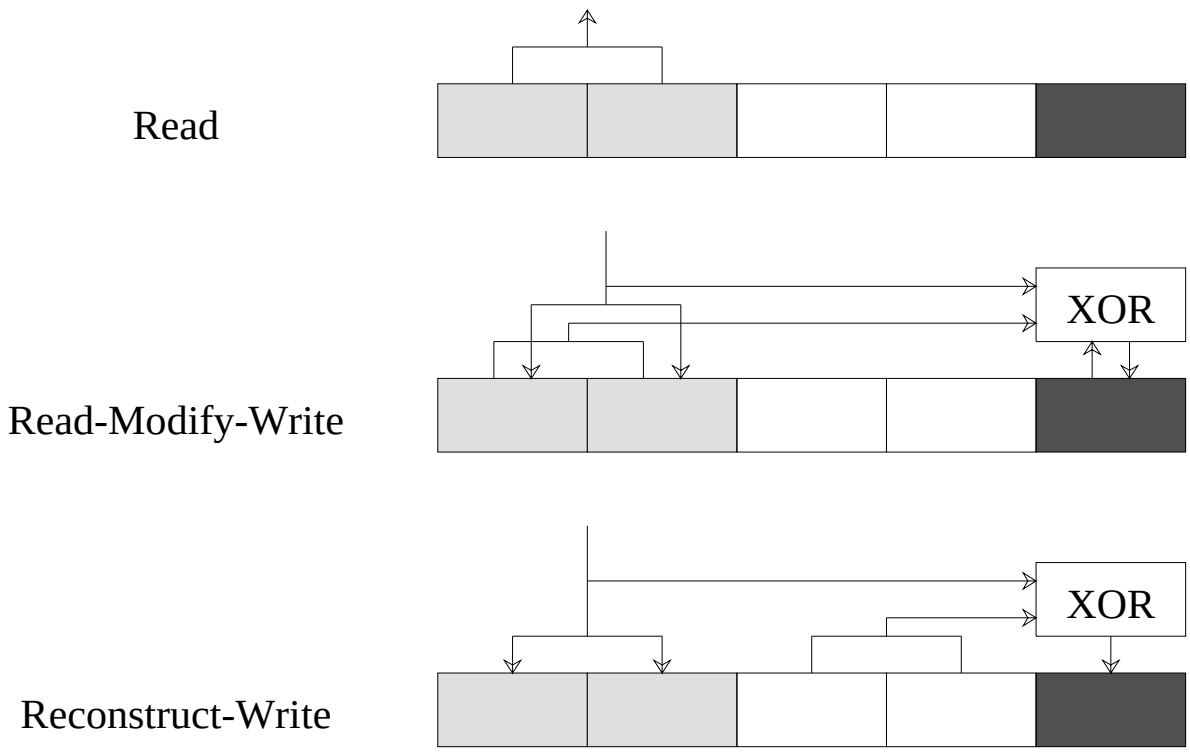

Fig. 2. I/O Request Servicing. *Each rectangle corresponds to a stripe unit. The lightly shaded regions correspond to data that is to be read or written. The dark regions correspond to parity.*

3 Definition of Parity Placements

The number of different ways parity can be placed relative to data is astronomical. We will begin our investigation of parity placements by requiring that they satisfy the following properties:

- Stripe units belonging to the same parity stripe should not map to the same column. (In many RAID systems, the disks within a column have a common failure mode, the string interface for example.) This is referred to as the "orthogonal RAID" property and guarantees that the failure of a single column does not result in data unavailability [10].
- In a RAID with n stripe units per parity stripe, the ith parity stripe unit should correspond to logical stripe unit j such that $j \operatorname{div} n = i$. This guarantees that the parity for any write request that is aligned on a parity stripe and a parity stripe in size can be computed by using only the data being written without reading old data.

Figure 3 illustrates eight parity placements which satisfy the above properties. Later, we will show that these placements are the most interesting from a relatively small but interesting space of parity placements. The following sections give an informal description for each parity placement.

Appendix A formally defines each parity placement mathematically.

3.1 RAID level 0

The RAID level 0 placement results in the conventional modulo n data striping scheme [11–14]. The RAID level 0 placement is defined here for comparison purposes only.

3.2 RAID level 4

The RAID level 4 placement is derived from the RAID level 0 placement by adding a parity disk to each row. The read performance of a RAID level 4 placement with n disks per row is identical to the read performance of a RAID level 0 placement with $n-1$ disks per row. Because the RAID level 0 placement is widely used and better understood than the other placements, this can be a desirable property. A disadvantage of the RAID level 4 placement is that the parity disks will become a bottleneck for small writes [1] since every write must update a parity disk of which there is only one per row. Also, only $n-1$ disks per row, instead of n disks per row are available for servicing reads since the nth disks do not contain data.

3.3 Right-Asymmetric

The right-asymmetric placement is derived from the RAID level 0 placement by pushing out data stripe units horizontally as parity stripe units are inserted. For each successive parity stripe, the point at which the parity stripe unit is inserted is rotated one stripe unit towards the right.

3.4 Left-Asymmetric

The left-asymmetric placement is derived from the RAID level 0 placement by pushing data stripe unit out horizontally as parity stripe unit are inserted. For each successive parity stripe, the point at which the parity stripe unit is inserted is rotated one stripe unit towards the left.

3.5 Right-Symmetric

The right-symmetric placement is derived by right rotations of entire parity stripes from the RAID level 4 placement.

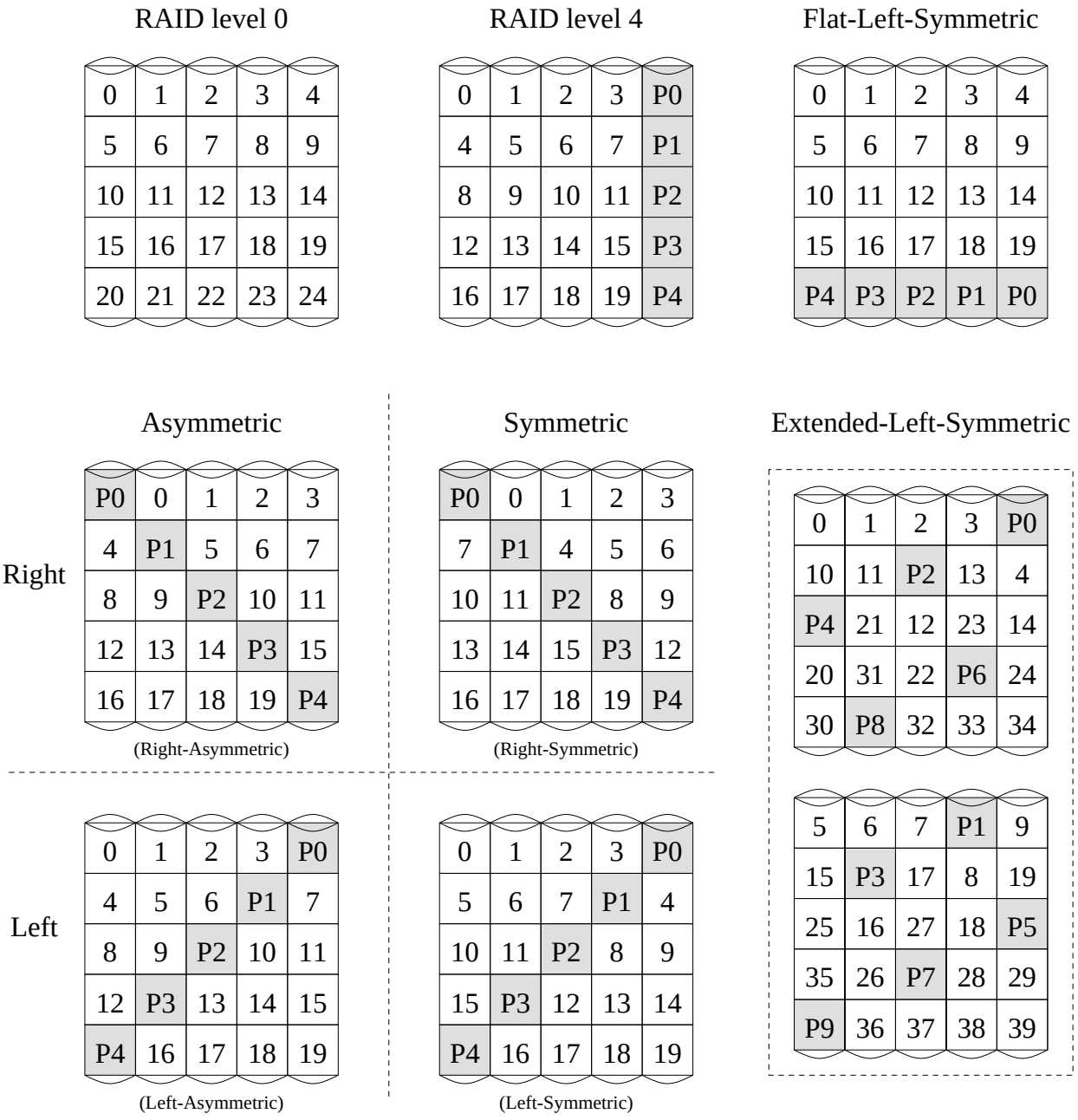

Fig. 3. Parity Placements. *Each square corresponds to a stripe unit. Each column of squares corresponds to a disk. Each matrix corresponds to a single row of disks. In all placements,* $P0$ *computes parity over stripe units* $0, 1, 2$ *and* 3*;* $P1$ *computes parity over stripe units* $4, 5, 6$ *and* 7*; etc. In all cases except the RAID level 0 and RAID level 4 placements, the minimum repeating pattern is shown.*

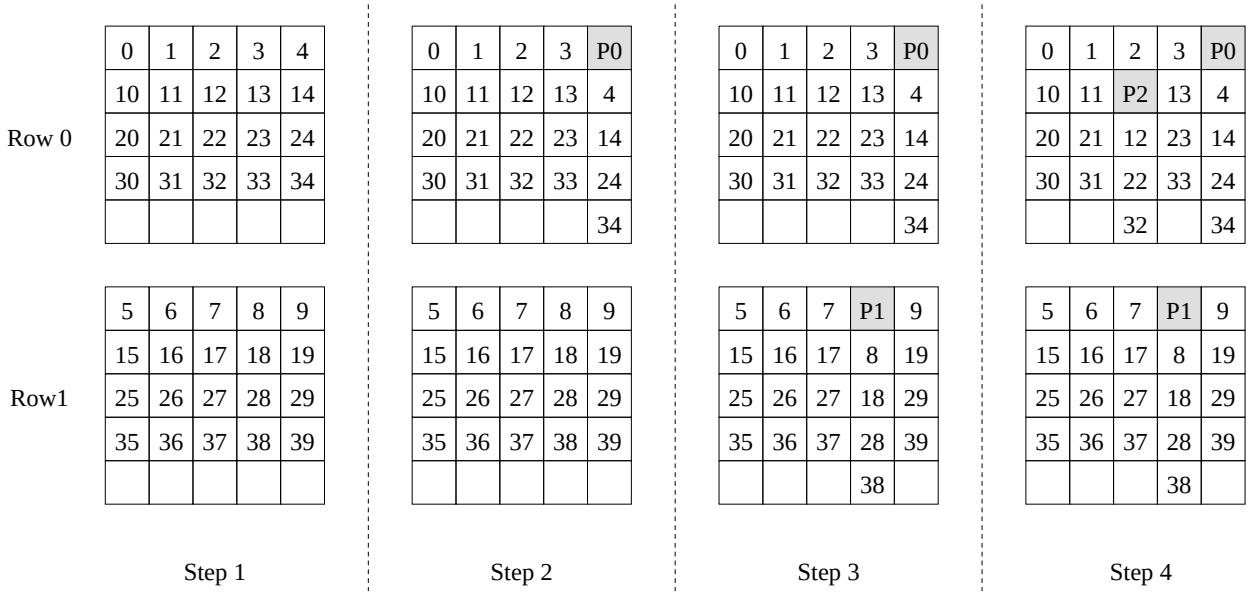

Fig. 4. Derivation of the Extended-Left-Symmetric Placement.

3.6 Left-Symmetric

The left-symmetric placement is derived by left rotations of entire parity stripes from the RAID level 4 placement.

3.7 Extended-Left-Symmetric

The extended-left-symmetric placement is derived from the RAID level 0 placement by pushing out data stripe units vertically as parity stripe units are inserted as shown in Figure 4. For each successive parity stripe, the point at which the parity stripe unit is inserted is rotated one stripe unit towards the left. Figure 4 illustrates the conceptual steps in deriving the extended-left-symmetric placement from the corresponding RAID level 0 placement. In arrays with only one row of disks, the extended-left-symmetric placement is identical to the left-symmetric placement. In arrays with multiple rows of disks, traversing the logical blocks of the extended-left-symmetric placement in sequence results in a traversal of all the disks whereas the left-symmetric placement will skip every nth disk. Note that in Figure 3, where each five element column represents a distinct disk, $P1$ is the parity corresponding to stripe units 4, 5, 6 and 7 and *not* 5, 6, 7 and 9. Likewise, $P2$ corresponds to stripe units 8, 9, 10 and 11. This is to ensure that writes that are a parity stripe in size and parity stripe aligned can always be written without reading additional information from the disks.

3.8 Flat-Left-Symmetric

The flat-left-symmetric placement is derived from the *extended*-left-symmetric placement by grouping all of the parity together and placing them at identical offsets within each disk. When reading large amounts of data, all disk heads within the same row skip over parity at the same time; thus, reducing disk synchronization time. When writing data, however, performance is likely to be worse than the extended-left-symmetric placement since the parity stripe unit is located at a different offset within the disk relative to its corresponding data stripe units. Note that in Figure 3, $P0$ is the parity corresponding to stripe units 0, 1, 2, and 3. Likewise, $P1$ corresponds to 4, 5, 6 and 7.

3.9 Physical vs. Logical Placements

When rotating the placement of parity, the question arises whether to rotate relative to physical stripe numbers or logical stripe numbers. Physical stripe numbers are sequential within each disk. The logical stripe numbers are assigned in the same sequence as the data striping, thus, spanning multiple rows of disks. Figure 5 illustrates the physical and logical variations of the left-symmetric placement when applied to an array with two rows. In the physical placement, the parity stripe units are rotated one stripe unit to the left as the physical stripe number is incremented. In the logical placement, the parity stripe units are rotated one stripe unit to the left as the logical stripe number is incremented. Note that for a single row of disks, there is no difference between a physical placement and a logical placement (physical stripe number = logical stripe number).

Specifying parity stripe unit relative to logical addresses can have surprising and undesirable consequences. For example, *rotating parity stripe units relative to logical addresses under certain array configurations will cause all of the parity stripe unit to be concentrated on a small subset of the disks!* Figure 5 illustrates this problem. To guarantee that this does not occur with a logical placement, one must ensure that the number of rows per array and the number of disks per row are relatively prime. For example, a two-by-five configuration is acceptable but a two-by-four configuration is not since two and four are both divisible by two. Physical placements, by contrast, always guarantee a uniform distribution of parity over all disks. Thus, physical placements are generally preferable over logical placements. All of the parity placements presented so far, with the exception of the extended-left-symmetric and flat-left-symmetric placements, which suffer from the problem just described, are physical placements.

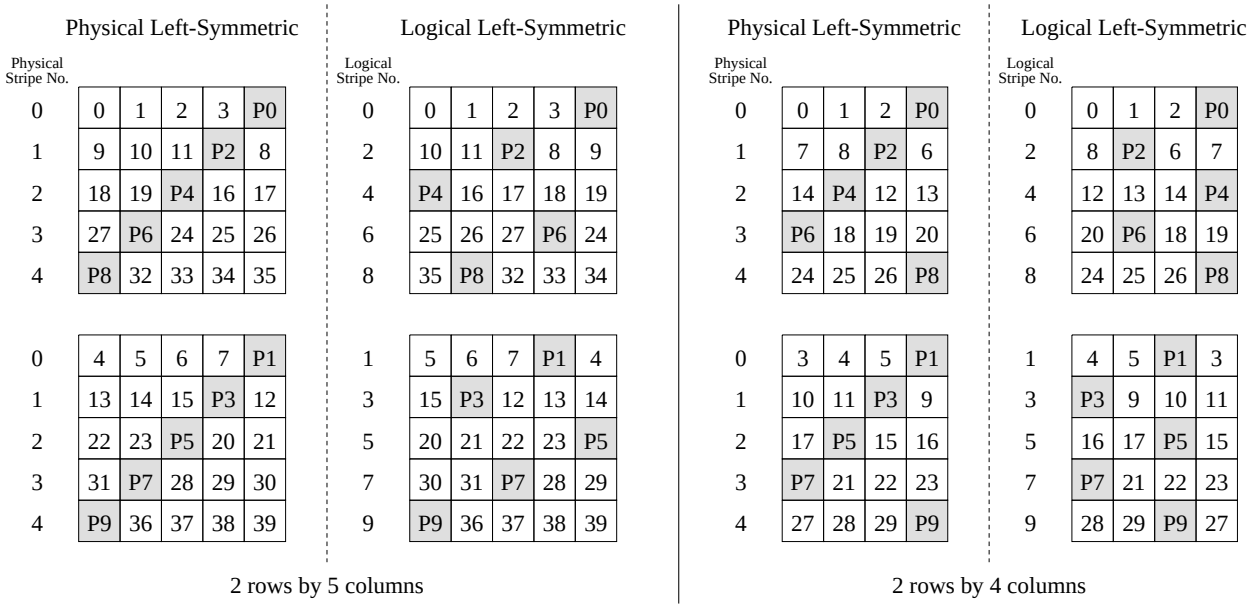

Fig. 5. Physical Versus Logical Placement. *This figure illustrates the physical and logical variations of the left-symmetric placement for a two-by-five array and a two-by-four array. In the physical placement, the parity stripe units are rotated one stripe unit to the left as the physical stripe number is incremented whereas in the logical placement, the parity stripe units are rotated one stripe unit to the left as the logical stripe number is incremented.*

4 Simulation of Parity Placements

This section evaluates the performance of the RAID level 0, RAID level 4, right-asymmetric, left-asymmetric, right-symmetric, left-symmetric, extended-left-symmetric and flat-left-symmetric parity placement schemes (all the placements illustrated in Figure 3) via simulation.

4.1 The RAID Simulator

The simulator consists of a module for computing the various parity placements, a model of disk behavior and a program to generate synthetic I/O requests. The only hardware resources modeled are disks, which are rotationally synchronized. The request types used are random reads and random writes of various sizes. Sequential requests are not used because the sequentiality is much less meaningful when there is more than one process generating requests to the same disk (the disk will thrash between the two sequential request streams). It is also desirable in most systems to convert small sequential requests to large random requests. The load on the system is controlled by specifying the number of concurrent processes which issue requests. The number of processes, request type and request size are fixed for each simulation run.

cylinders per disk	949
tracks per cylinder	14
sectors per track	48
bytes per sector	512
track skew	4
revolution time	13.9 ms
single cylinder seek time	2.0 ms
average seek time	12.5 ms
max stroke seek time	25.0 ms
max sustained transfer rate	1.7 MB/s

Table 1. Disk Parameters.

4.2 The Disk

The disk model is loosely based upon the IBM 0661 3.5 inch SCSI disk drive. Table 1 tabulates the parameters of the simulated disk. Figure 6 plots the seek profile of the simulated disk.

4.3 Simulation Parameters

The following is a list of the input variables used in the simulation and their corresponding ranges.

- Parity placement scheme (one of the eight previously described).
- Number of rows of disks in array (1 or 2).
- Number of disks per row (5 or 9).
- Size of the stripe unit (4KB, 8KB, 16KB, 32KB, or 64KB).
- Request type (read or write).
- The size of the request (2KB to 1MB).
- Request alignment (aligned or unaligned). Aligned requests are aligned on their size boundaries. Unaligned requests are aligned on sector boundaries (512 bytes).
- The degree of concurrency; that is, the number of processes generating requests (1, 2 or 16). The term *low load* refers to a degree of concurrency of one. The term *high load* refers to a degree of concurrency of sixteen.

The primary output variable of interest is the throughput, measured in megabytes per second. Because we are simulating a closed queueing system, the response time is always inversly proportional

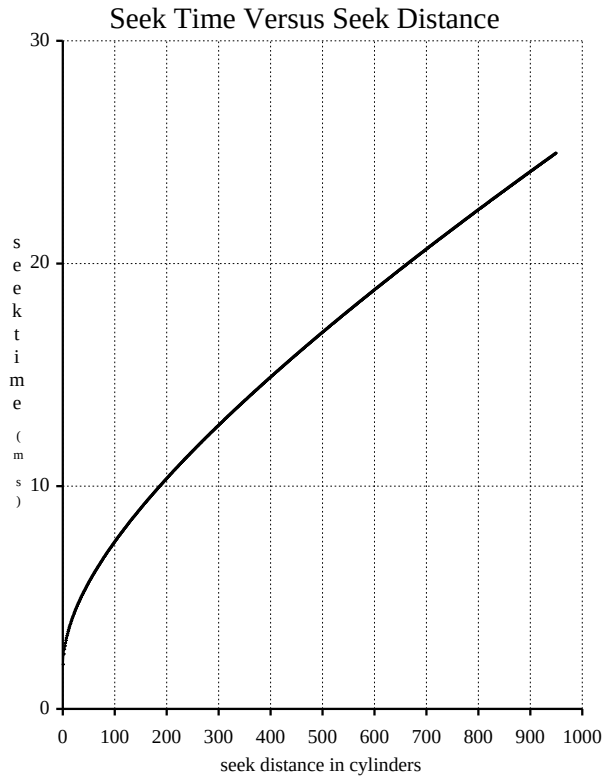

Fig. 6. Seek Profile. *A plot of the seek time in milliseconds versus the seek distance in cylinders for the simulated disk.*

$$seekTime \quad = \quad \begin{cases} 0, & \text{if } x = 0 \\ a(x-1)^{0.5} + b(x-1) + c, & \text{if } x > 0 \end{cases}$$

Where x *is the seek distance in cylinders and* a, b *and* c *are constants chosen to satisfy the single cylinder, max stroke and average seek times. For the simulated disk,* $a = 0.4623, b = 0.0092$ *and* $c = 2$.

to the throughput. In the interests of reducing the parameter space, all workloads are homogeneous, mixtures of reads and writes as well as non-constant distributions of request sizes are not used. Another reason for using homogeneous workloads is that they are more likely to accentuate the differences in performance of the various parity placements. We leave heterogeneous workloads as future work.

The cross product of all input variables was considered too large to simulate and analyze properly. We also did not have reasonable ranges for the input variables when we started. Thus, only specific subsets of the cross product were investigated. The selection of the subsets and the investigation of the results proceeded in an iterative manner to reduce the possibility of missing interesting points in the cross product. In all, approximately 20,000 points in the cross product were investigated.

4.4 Simulation Results

Figures 7, 8, 9 and 10 illustrate the simulated performance of each parity placement under a set of particular inputs and is characteristic of other input combinations which are not illustrated here. Note that at low load, the graphs display a sawtooth pattern with a period approximately equal to $2\ rows \times 5\ columns \times 32KB\ stripeunits = 320KB$. The first dip occurs when requests become large enough to wrap around the array. At this point, some of the disks must read/write two stripe units while other disks read/write only one stripe unit. Some of the disks end up waiting for the other disks, resulting in inefficient resource utilization. This behavior is repeated each time the request wraps around the array, resulting in a periodic behavior. Note that at low load and relatively large requests sizes, the choice of parity placement results in up to a 20 to 30 percent difference in performance for the array configuration simulated. The following sections examines each of the Figures 7, 8, 9 and 10 in greater detail.

4.4.1 Reads at Low Load

When performing reads at low load, roughly four groups of placements can be distinguished based on performance. From highest performance to lowest performance they are grouped as follows:

1. RAID level 0, extended-left-symmetric and flat-left-symmetric.
2. Left-symmetric.
3. RAID level 4, right-asymmetric and left-asymmetric.

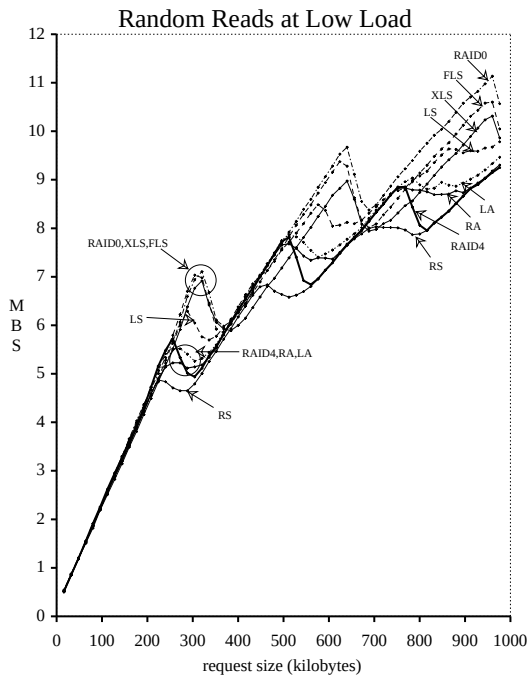

Fig. 7. Reads at Low Load.

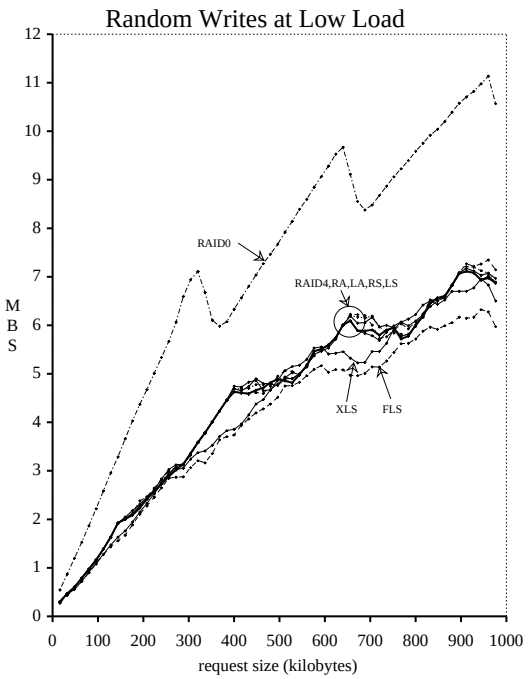

Fig. 8. Writes at Low Load.

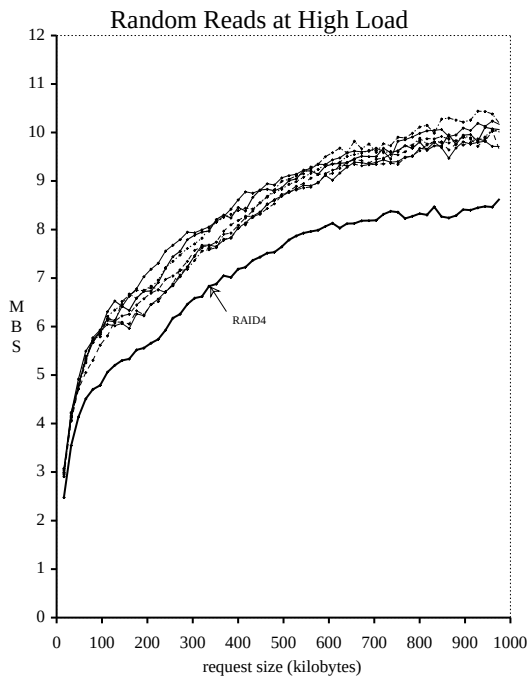

Fig. 9. Reads at High Load.

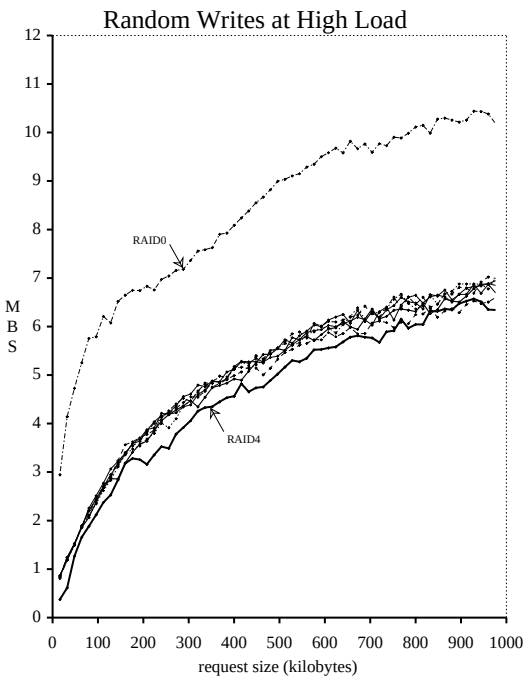

Fig. 10. Writes at High Load.

The simulation parameters are two rows, five disks per row, 32KB stripe units, and unaligned requests. The degree of concurrency for low load is one and the degree of concurrency for high load is sixteen.

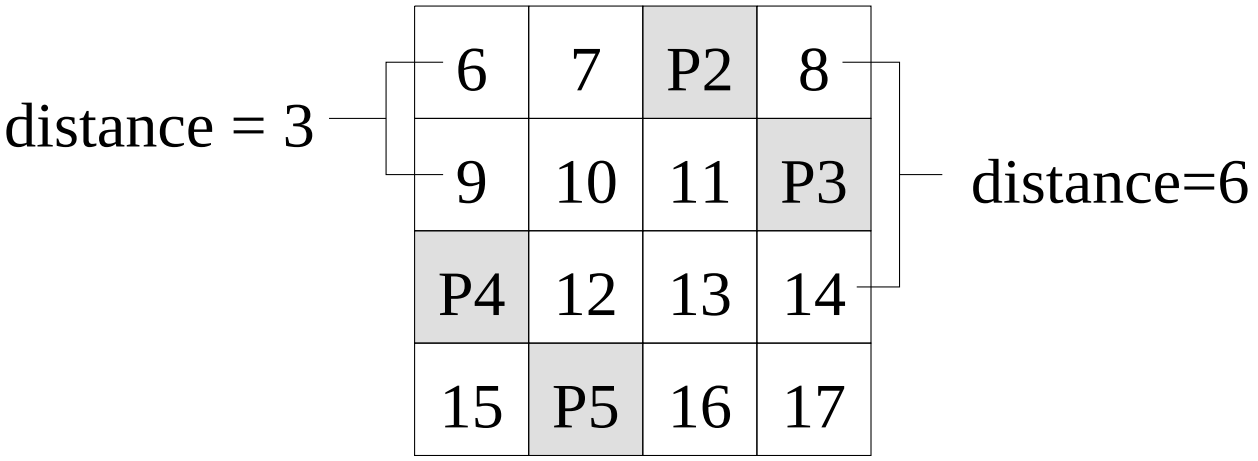

Fig. 11. Placement Distance.

4. Right-symmetric.

We define the *minimum placement distance* of a placement, which gives a good prediction of read performance at low load, as the minimum arithmetic difference between logical data stripe unit addresses of stripe units that are logically sequential on the same disk. Figure 11 illustrates the concept of distance. Parity stripe units are ignored when determining minimum placement distances. The minimum placement distance gives an indication of the number of disks a parity placement can stripe data over before reusing a disk and hence the amount of parallelism available for servicing a single request. The larger the minimum placement distance, the better the read performance at low load. We will refer to a placement with a minimum placement distance of N as a *distance N placement*. The maximum minimum placement distance for an array with N disks is N. Table 2 tabulates the minimum placement distances for the parity placements so far considered.

Among placements with the same distance, the RAID level 0 placement outperforms the extended-left-symmetric placement because the RAID level 0 placement never has to skip over parity stripe units. The flat-left-symmetric placement outperforms the extended-left-symmetric placement because the flat-left-symmetric placement aligns the parity stripe units together, so that on large reads, all parity stripe units can be skipped simultaneously. The left-asymmetric placement outperforms the RAID level 4 placement and the right-asymmetric placement outperforms the right-symmetric placement because, even though their minimum placement distances are the same, the average placement distances are different. In each of the above two cases, the placement with the larger average placement distance performs better. The minimum placement distance is a

	minimum placement distance
RAID level 0	mn
Flat-Left-Symmetric	mn
X-Left-Symmetric	mn
Left-Symmetric	$m(n-1)+1$
Left-Asymmetric	$m(n-1)$
RAID level 4	$m(n-1)$
Right-Asymmetric	$m(n-1)-1$
Right-Symmetric	$m(n-1)-1$

Table 2. Parity Placement Distances. *The symbol m denotes the number of rows in the array and the symbol n denotes the number of disks per row. Note that the left-symmetric and extended-left-symmetric placements are identical placements when the number of rows is one.*

safer but less discriminatory indicator of performance than the average placement distance in that it is possible to contrive placements with very good average placement distances but very poor performance.

4.4.2 Writes at Low Load

When performing writes at low load, roughly four groups of placements can be distinguished based on performance. From highest performance to lowest performance they are grouped as follows:

1. RAID level 0.
2. RAID level 4, right-asymmetric, left-asymmetric, right-symmetric and left-symmetric.
3. Extended-left-symmetric.
4. Flat-left-symmetric.

As expected, the RAID level 0 placement – no redundancy – displays the highest performance since it does not maintain parity. The extended-left-symmetric placement performs worse than the placements in the second group because it places the parity of the previous stripe on the same disk as the data of the current stripe. For example, note that in the extended-left-symmetric placement illustrated in Figure 3, $P0$ is on the same disk as stripe unit 4. Thus, if two sequential parity stripes are written, a single disk must service both a parity access and a data access. The extended-left-symmetric placement performs better than the flat-left-symmetric placement because it aligns parity with its corresponding data stripe units, whereas the flat-left-symmetric placement

aligns the parity together. Note that this was the same reason the flat-left-symmetric-placement outperformed the extended left-symmetric-placement on reads at low load.

4.4.3 Reads at High Load

The RAID level 4 placement has the worst read performance at high load because it does not distribute parity and data over all disks. Thus, only $n-1$ rather than n disks are available for servicing read requests. The performance of the other placements, which distribute parity and data, are comparable. At high load, the most important criterion for high read performance is uniform disk utilization. Thus, placements which distribute the load uniformly are desirable.

4.4.4 Writes at High Load

As expected, the RAID level 0 placement displays the highest write performance at high load since it does not maintain parity. Note that because the redundant placements must write parity, their write performance asymptotically approachs $(n-1)/n$, 4/5 in the case of Figure 10, of the maximum RAID level 0 performance at large request sizes. The RAID level 4 placement displays the worst write performance at high load due to contention for the parity disks. For small request sizes (< 32KB) and these particular simulation parameters, the difference in performance between the RAID level 4 placement and the other placements is over a factor of two. With more disks per row and at higher load, we would expect the performance difference to be even larger. Thus, for high write performance at high load, placements which distribute the parity uniformly are desirable.

4.4.5 Performance Summary

In this section, we have examined the performance characteristics of eight parity placements. The RAID level 0 placement is unsuitable since it does not support parity. The left-symmetric placement outperforms the RAID level 4, right-asymmetric, left-asymmetric and right-symmetric placements on reads at low load over almost all request sizes and has comparable or better performance in all of the other cases. Thus, the choice to be made is between the left-symmetric, extended-left-symmetric and flat-left-symmetric placements. Which placement is used for a particular system depends on the importance of read performance at low load versus write performance at low load. For the reader's benefit, the performance of the above three placements at low load is redisplayed in Figure 12 with the other placements omitted. To make the differences easier to evaluate, Table 3 tabulates the absolute throughput and throughput relative to the left-symmetric placement for a

few key requests sizes ($2\ rows \times 5\ columns \times 32KB\ stripeunits = 320KB$) from Figure 12.

Note that the performance of the left-symmetric, extended-left-symmetric and flat-left-symmetric placements converge as the request size increases for reads but remains relatively unchanged for writes. Note also that the flat-left-symmetric placement achieves only 90% of the performance of the left-symmetric placement for 16KB request sizes.

We propose the following list of desirable placement properties, roughly in the order of their importance:

1. Stripe units belonging to the same parity stripe should not occupy the same column. (In many RAID systems, the disks within a column have a common failure mode, the string interface for example.) This is referred to as the "orthogonal RAID" property. All of the proposed placements have this property.

2. In a RAID with n disks per row, the ith parity stripe unit should correspond to stripe unit j such that $j \operatorname{div} n = i$. This guarantees that any write request that is stripe aligned and a stripe in size can be written without reading old data. All of the proposed placements have this property.

3. Parity and data should be distributed over all disks. All of the proposed placements except the RAID level 4 placement have this property.

4. The minimum placement distance should be maximized. Only the extended-left-symmetric and flat-left-symmetric placements are maximum distance placements. The left-symmetric placement is a maximum distance placement for a single row of disks but not for multiple rows of disks.

5. If the number of rows is n then stripe units belonging to any n consecutive parity stripes should not occupy the same disk. This avoids the write contention experienced by the extended-left-symmetric and flat-left-symmetric placements which are the only proposed placements which do not satisfy this property.

Unfortunately, properties 4 and 5 are at odds with one another. Given an array with N disks, it is not generally possible to devise a placement which is both distance N and satisfies Property 5 for a general m-by-n disk array. With only a single row of disks, the left-symmetric, extended-left-symmetric and flat-left-symmetric placements are optimal with respect to both properties. When

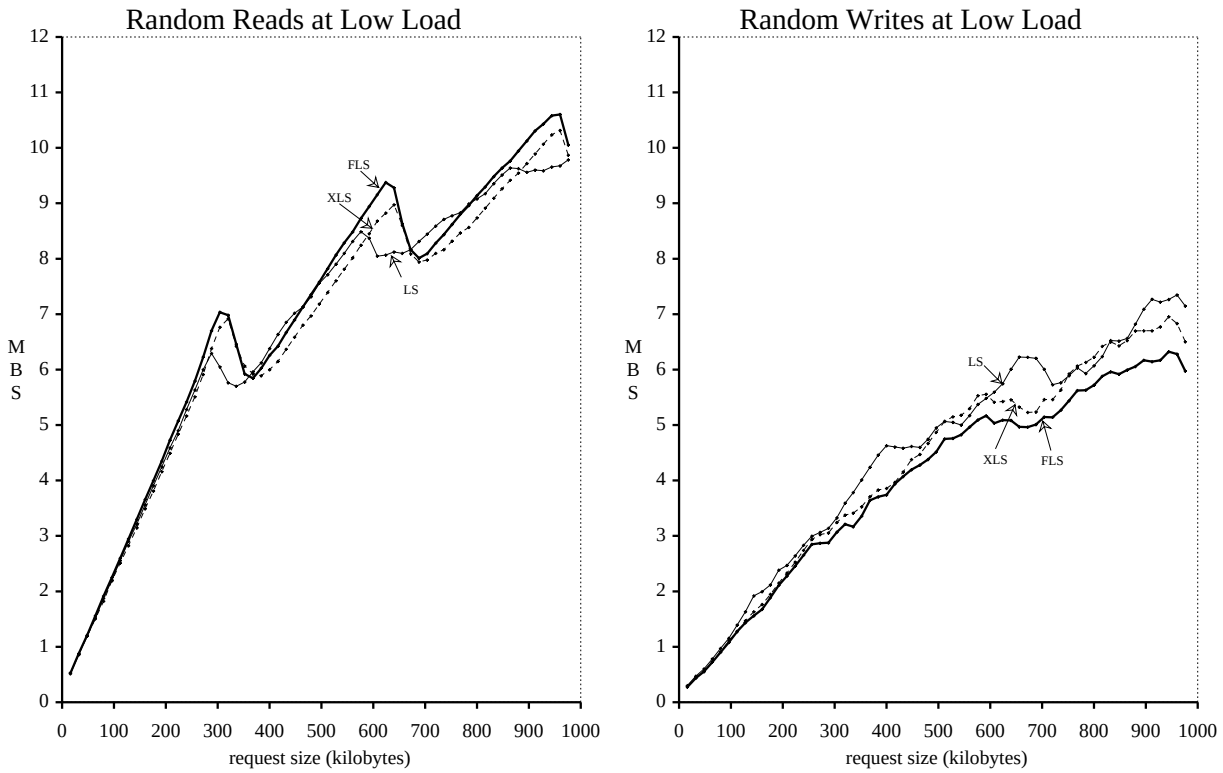

Fig. 12. Parity Placement Performance Summary. *The simulation parameters are two rows, five disks per row, 32KB stripe units, unaligned requests, and the degree of concurrency for low load is one.*

Random Reads at Low Load

	16KB		320KB		640KB		960K	
	MBS	% of LS	MBS	% of LS	MBS	% of LS	MBS	% of LS
Left-Symmetric	0.52	100%	5.77	100%	8.12	100%	9.68	100%
X-Left-Symmetric	0.53	102%	6.92	120%	8.98	111%	10.31	107%
Flat-Left-Symmetric	0.52	100%	6.98	121%	9.28	114%	10.60	110%

Random Writes at Low Load

	16KB		320KB		640KB		960K	
	MBS	% of LS	MBS	% of LS	MBS	% of LS	MBS	% of LS
Left-Symmetric	0.30	100%	3.59	100%	6.01	100%	7.35	100%
X-Left-Symmetric	0.29	97%	3.38	94%	5.46	91%	6.83	93%
Flat-Left-Symmetric	0.27	90%	3.21	89%	5.08	85%	6.28	85%

Table 3. Parity Placement Performance Summary. MBS *denotes throughput in megabytes per second.* Percentage of LS *denotes throughput relative to the left-symmetric parity placement.*

selecting a parity placement for a general disk array, one must sacrifice either read performance or write performance.

4.5 Flat-Left-Symmetric vs. Left Symmetric

In the previous section, we discovered that, of the redundant placements, the flat-left-symmetric and left symmetric placements, taken together, outperform the other placements. In particular, the flat-left-symmetric placement outperforms the other placements for reads at low load, the left-symmetric placement performs as well as the other placements for writes at low load and both placements perform as well as other placements at reads and writes at high load. However, since the flat-left-symmetric-placement performs better than the left-symmetric placement at reads at low load and visa versa for writes at low load, it still remains to be seen which placement is desirable for a workload that is a mixture of reads and writes.

In this section, we simulate workloads that are mixtures of reads and writes to determine the specific read/write percentage and request sizes over which each of the two placements performs better than the other at low load. We will use the same simulation parameters as used in the previous section (two rows of disks, five disks per row, 32KB stripe units, and unaligned requests) except that instead of using workloads that are purely reads or purely writes, we will vary the percentage of requests that are writes from 0 percent to 100 percent in 10 percent increments.

Figure 13 illustrates the relative performance of the flat-left-symmetric placement vs. the left symmetric placement as a function of the percentage of write requests and the request size. Note that the flat-left-symmetric placement performs significantly better than the left-symmetric placement for request sizes that are close multiples of $2\ rows \times 5\ columns \times 32KB\ stripeunits = 320KB$ but somewhat worse than the left-symmetric placement as the percentage of write requests increases.

To make the relative performance differences between the flat-left-symmetric and left-symmetric placements more concrete, Figure 14 re-illustrates Figure 13 topographically. As can be clearly seen from Figure 14, the performance difference between the flat-left-symmetric and left-symmetric placements is fairly small over most of the workload ranges; however, the left-symmetric placement performs better than the flat-left-symmetric placement over a much larger range of workloads.

In conclusion, we state that the left-symmetric placement is generally preferable to the flat-left-symmetric placement except when the percentage of write requests is fairly small (<30-50%) and the requests can be guaranteed to be of a certain fixed size (in our case 320KB). The left-symmetric

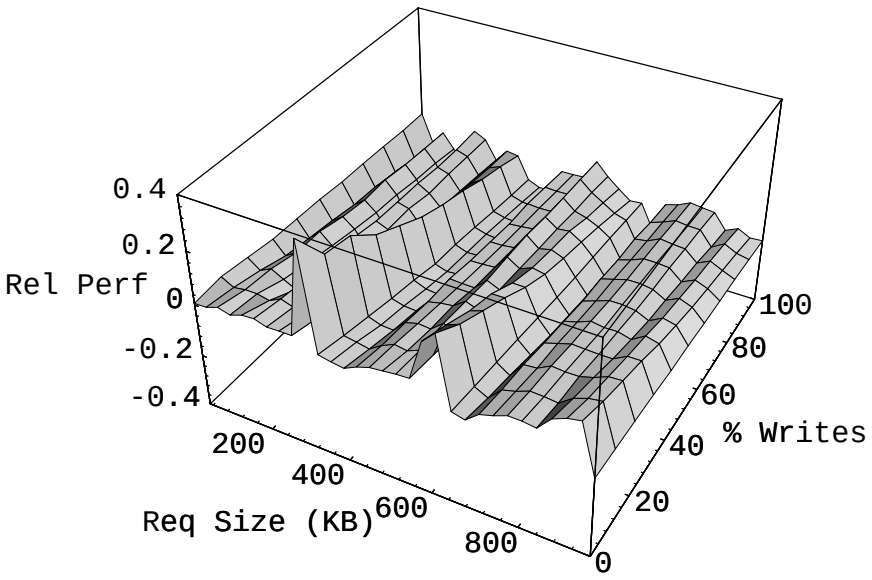

Fig. 13. Relative Performance of Flat-Left-Symmetric vs. Left-Symmetric. *The simulation parameters are two rows, five disks per row, 32KB stripe units, unaligned requests and a degree of concurrency of one. A relative performance of +10% implies that the flat-left-symmetric placement performs 10% better than the left-symmetric placement. Likewise a relative performance of -10% implies that the flat-left-symmetric placement performs 10% worse than the left-symmetric placement.*

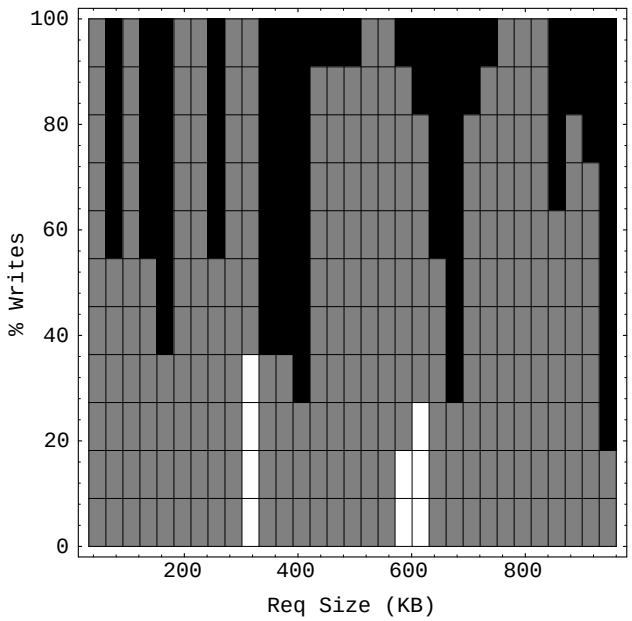

Fig. 14. Relative Performance of Flat-Left-Symmetric vs. Left-Symmetric. *The simulation parameters are two rows, five disks per row, 32KB stripe units, unaligned requests and a degree of concurrency of one. The white regions indicate workloads at which the flat-left-symmetric placement performs at least 10% better than the left-symmetric placement. The black regions indicate workloads at which the flat-left-symmetric placement performs at least 10% worse than the left-symmetric placement. The grey regions represent workloads at which there is less than a 10% difference in the performance of the flat-left-symmetric placement vs. the left-symmetric placement.*

placement becomes even more attractive when we consider that, as pointed out in Section 3.9 which discusses physical placements vs. logical placements, the left-symmetric placement is a physical placement whereas the flat-left-symmetric placement is a logical placement and hence is difficult to apply for general array configurations.

5 A Space of Parity Placements

The previous sections have compared the performance of eight different parity placements and derived properties that are desirable of parity placements in general. In this section, we provide motivation as to why they are the interesting placements to examine. An important question we will address is whether there exists a placement which is superior to the eight placements so far considered. This section attempts to answer this question by systematically generating and examining parity placements. We first look at placements for a single row of disks and then generalize our method to look at placements for multiple rows of disks.

5.1 Single Row Placements

This section takes a brute-force look at parity placements for arrays with a single row of disks by exhaustively considering all three-by-three parity placements. Consider a three-by-three checkerboard where the columns correspond to distinct disks and the squares to stripe units. We define a *parity placement* for a single row of disks as any column indistinct permutation of distinct stripe units, that is, shuffling the columns does not change the parity placement. Since the columns are indistinct, the number of possible ways in which nine distinct stripe units, six of which are data and three of which are parity, can be placed on the checkerboard is $9!/3! = 60480$. This is clearly too many to examine by hand. In order to reduce the number of distinct cases to consider, we will initially ignore the ordering of stripe units within each column and look at the number of ways in which the nine distinct stripe units can be partitioned over the three indistinct columns. Thus, each column forms a set of size three which we will refer to as a *placement group*. The three placement groups taken together will be referred to as a *placement class*.

There exist $9!/(3!3!3!3!) = 280$ distinct placement classes and each placement class corresponds to $3!3!3! = 216$ distinct parity placements. If we restrict placement classes to those which place parity and its corresponding data stripe units on different disks and further specify that parity, and hence data, must be uniformly distributed over all disks, the number of placement classes drops

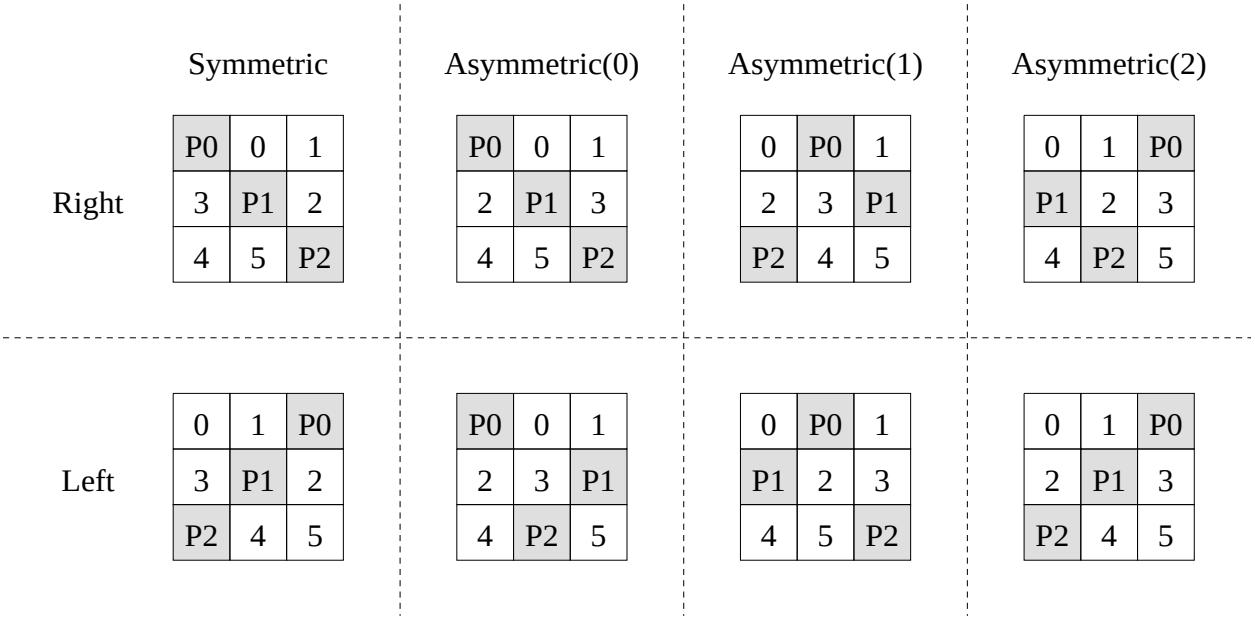

Fig. 15. Three By Three Parity Placement Classes.

down to eight (the eight placement classes were derived by exhaustively generating placement classes and eliminating those that did not satisfy the above requirements). Figure 15 illustrates the eight placement classes which are labeled with instances of their corresponding parity placements. Note that each asymmetric class gives rise to a "family" of placements differing only on the column offset at which the rotation is started.

Now that we have identified eight feasible placement classes, it is time to consider which parity placements derivable from each class are the most useful. Since the data stripe units within each column should obviously be placed in ascending order, the question becomes one of where to place the parity. For all placement classes except the left-symmetric placement class, the parity should almost certainly be aligned with its corresponding data stripe units, that is, there is no advantage to not aligning the data and parity. This ensures that the data and corresponding parity stripe units will be rotationally synchronized, and when writing only exact parity stripes, all of the disk heads will travel equal distances. Placing the data stripe units in ascending order and aligning the parity and its corresponding data results in the family of right-asymmetric placements, the family of left-asymmetric placements, the right-symmetric placement and the left-symmetric placement. For the extended-left-symmetric placement class, there are two reasonable ways to place the parity. The first method, as already described, is to align the parity with its corresponding data. The second method is to align all of the parity with each other. This ensures that on large reads, all

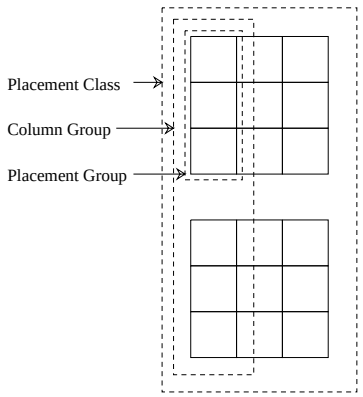

Fig. 16. Parity Placement Definitions.

disk heads will skip over the parity sectors simultaneously. The two methods of placing the parity within each row give rise to the left-symmetric and flat-left-symmetric placements.

Thus, we conclude that the most useful parity placements derivable for a single row of disks from a three-by-three grid are the family of right-asymmetric placements, the family of left-asymmetric placements, the right-symmetric placement, the left-symmetric placement and the flat-left-symmetric placement. Although it would be useful to characterize the placements derivable for general m-by-n disk arrays, we leave these as possibilities for future work.

5.2 Multi Row Placements

This section takes a brute-force look at parity placements for arrays with multiple rows of disks by considering all three-by-three parity placements for two rows of disks. Consider two three-by-three checkerboards where the columns of each checkerboard correspond to distinct disks and the squares to stripe units. Each column within each checker board forms a set of size three which we will refer to as a *placement group*. A placement group represents the stripe units placed together on a single disk. The set of placement groups within the same column is referred to as a *column group*. With two rows of disks, there are two placement groups per column group. The set of all column groups will be referred to as a *placement class*. Figure 16 illustrates the above concepts. A parity placement is derived from a placement class by imposing an ordering on the stripe units within each placement group. Thus, in our example, there are $18!/(3!3!^6 2^3) = 2,858,856$ placement classes and each placement class corresponds to $3!^6 = 46,656$ parity placements (columns are unordered $\Rightarrow 3!$, stripe units within each of six placement groups unordered $\Rightarrow 3!^6$, placement groups within column groups unordered $\Rightarrow 2^3$).

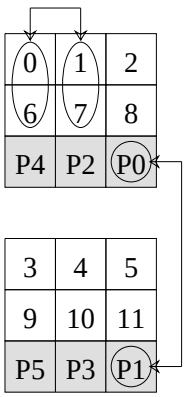

Fig. 17. Distance Six Multi-Row Parity Placement Classes.

In what follows, we explicitly restrict placement classes to those which place parity and its corresponding data stripe units on different disks and uniformly distributes parity and data over all disks. We will refer to such a placement as a *RAID level 5 placement*. Note that the results which follow, unless otherwise indicated, have been derived by brute force, that is, by enumerating all possibilities.

A key question we will now answer in this section is whether there exists a RAID level 5 placement which is distance six and satisfies Property 5, that is, a maximum distance placement which places stripe units belonging to any two consecutive parity stripes on different disks. An enumeration of all possible placement classes indicates that there does not exist such a placement.

The number of RAID level 5 placements which are distance six but do not satisfy Property 5 is sixty-four. The extended-left-symmetric and flat-left-symmetric placements are derived from one of these sixty-four classes. The other sixty-three classes can be generated from the extended-left-symmetric class by swapping the parity stripe units within column groups, and by swapping the set of data stripe units within a placement group with another such set containing data stripe units of the same parity stripe. Figure 17 illustrates each type of swap. There are eight ways to perform the former swap and eight ways to perform the latter swap, resulting in a total of $8 \times 8 = 64$ different placement classes. The performance of placements derived from the other sixty-three placement classes is difficult to predict with certainty but we would not expect them to be significantly different from the extended-left-symmetric and flat-left-symmetric placements.

The number of RAID level 5 placement classes which are distance five and satisfy Property 5, as illustrated by Figure 18, is six. The placement classes differ only in the grouping of placement groups to form column groups. The six placement classes would produce identical performance in our simulations.

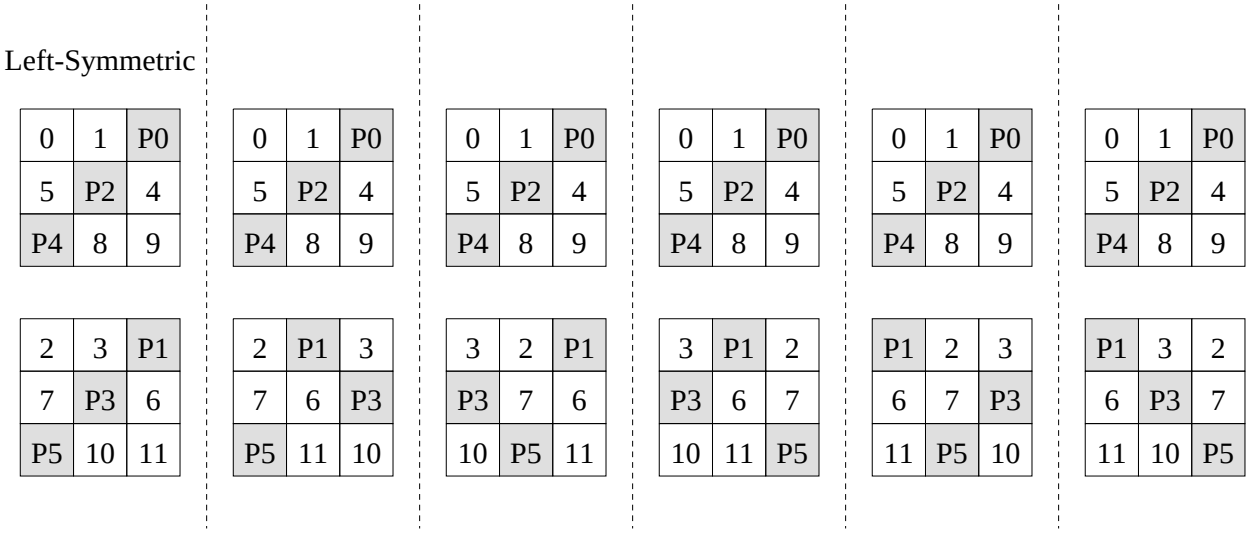

Fig. 18. Distance Five Multi-Row Parity Placement Classes.

5.3 Summary

In this section, we have described why the eight parity placements previously examined represent the class of interesting placements; they have the highest performance potential of any parity placement that can be derived from a three-by-three checkerboard. More specifically, we have shown that within the space of parity placements examined, there does not exist a multi-row parity placement which has maximum placement distance and satisfies Property 5.

6 Summary

We have defined and investigated the performance of a limited space of parity placements. We show that although for small request sizes the choice of parity placement does not have a significant effect on performance, for relatively large request sizes of hundreds of kilobytes at low loads, the choice of parity placement significantly affects performance (up to 20-30 percent for the typical disk array configurations that are common today). We have shown that the left-symmetric, extended-left-symmetric and flat-left-symmetric are the best RAID level 5 parity placements over the examined workloads. Of these, the placement with the highest read performance, flat-left-symmetric, has the lowest write performance and the placement with the lowest read performance, left-symmetric, has the highest write performance.

Based on the simulation results, we have proposed properties that are desirable of parity placements in general. Namely, stripe units belonging to the same parity stripe should not occupy the

same column, write requests that are stripe aligned and a stripe in size should map exactly to a parity stripe, the parity and data should be uniformly distributed, the minimum placement distance should be maximized and consecutive parity stripes should be mapped to disjoint disks.

Finally, we investigated the relative performance of the flat-left-symmetric and left-symmetric parity placements under workloads consisting of a mixture of reads and writes to determine the workload ranges over which one parity placement is better than the other. In general we conclude that the left-symmetric placement is preferable to the flat-left-symmetric placement over a wider range of workloads. The flat-left-symmetric placement is preferable to the left-symmetric placement when the workload consists of very few writes and the request sizes are fixed. In that case, the disk array can be configured such that the parity stripe size matches the request size.

7 Acknowledgements

We would especially like to thank Peter Chen and Garth Gibson for helping us to write parts of the RAID simulator and their invaluable comments. We would also like to thank Ann Drapeau, David Patterson, Mendel Rosenblum, Mary Baker and John Ousterhout for the support they have provided in researching and writing this paper. Finally, we would like to thank our government and industrial affiliates, Array Technologies, DARPA/NASA (NAG2-591), DEC, Eastman Kodak, Hewlett-Packard, IBM, Intel Scientific Computers, California MICRO, NSF (MIP 8715235), Seagate, Sun Microsystems and Thinking Machines Corporation for their financial support.

References

[1] D. A. Patterson, G. Gibson, and R. H. Katz, "A case for redundant arrays of inexpensive disks (RAID)," in *Proc. ACM SIGMOD*, pp. 109–116, June 1988.

[2] R. H. Katz, G. A. Gibson, and D. A. Patterson, "Disk system architectures for high performance computing," in *Proc. IEEE*, vol. 77, pp. 1842–1858, Dec. 1989.

[3] D. A. Patterson, P. M. Chen, G. Gibson, and R. H. Katz, "Introduction to redundant arrays of inexpensive disks (RAID)," in *Proc. IEEE COMPCON*, pp. 112–117, Spring 1989.

[4] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson, "An evaluation of redundant arrays of disks using an Amdahl 5890," in *Proc. SIGMETRICS*, pp. 74–85, May 1990.

[5] P. C. Dibble, "A parallel interleaved file system," Tech. Rep. CS TR 334, University of Rochester, Mar. 1990.

[6] J. Gray, B. Horst, and M. Walker, "Parity striping of disc arrays: Low-cost reliable storage with acceptable throughput," in *Proc. Very Large Data Bases*, pp. 148–161, Aug. 1990.

[7] J. Menon and J. Kasson, "Methods for improved update performance of disk arrays," Research Report RJ 6928 (66034), IBM Research Division, Almaden Research Center, San Jose, California 95120-6099, 1989.

[8] E. L. Miller, "Input/output behavior of supercomputing applications," Tech. Rep. UCB/CSD 91/616, University of California at Berkeley, 1991.

[9] E. K. Lee and R. H. Katz, "Performance consequences of parity placement in disk arrays," in *Proc. ASPLOS*, pp. 190–199, Apr. 1991.

[10] M. Schulze, G. Gibson, R. H. Katz, and D. A. Patterson, "How reliable is a RAID?," in *Proc. IEEE COMPCON*, pp. 118–123, Spring 1989.

[11] M. Y. Kim, "Synchronized disk interleaving," *IEEE Trans. on Computers*, vol. C-35, pp. 978–988, Nov. 1986.

[12] M. Y. Kim and A. N. Tantawi, "Asynchronous disk interleaving," Tech. Rep. RC12497, IBM, Jan. 1987.

[13] M. Livny, S. Khoshafian, and H. Boral, "Multi-disk management algorithms," in *Proc. SIGMETRICS*, pp. 69–77, May 1987.

[14] K. Salem and H. Garcia-Molina, "Disk striping," in *Proc. IEEE Data Engineering*, pp. 336–342, Feb. 1986.

A Parity Mapping Functions

Formally, we define a logical to physical RAID mapping as a pair of functions: the *data-mapping-function* and the *parity-placement-function*. The data-mapping-function is a one-to-one function which maps a logical block address (blockNumber) to a physical disk addresses (disk, stripeUnit). Frequently, we will find it convenient to picture a disk array as a two dimensional array of disks and thus to identify a disk by its row and column numbers. Thus, the physical disk address can be represented as the ordered tuple (row, column, stripeUnit). The parity-placement-function is a many-to-one function which maps a logical block address (blockNumber) to a physical disk address (row, column, stripeUnit). The parity-placement-function specifies for each logical block its corresponding parity stripeUnit. Parity is computed block-wise over all logical blocks which map to the same parity stripeUnit.

The following describes each placement and specifies for each placement the data-mapping-function and the parity-placement-function. The following assumes that any number modulo a positive number is also a possible number. Unfortunately, on some machines, a negative number modulo a positive number is a negative number. The following definitions and clarification of terms will prove useful:

$$
\begin{aligned}
n &= \text{Number of columns in array.} \\
m &= \text{Number of rows in array.} \\
ndata &= \text{Number of data stripe units per parity stripe.} \\
&= n \text{ (RAID level 0) or } n-1 \text{ (all others)} \\
blockNumber &= \text{Logical RAID block address.} \\
stripeUnitID &= blockNumber \bmod ndata \\
rowID &= (blockNumber \operatorname{div} ndata) \bmod m \\
stripeID &= (blockNumber \operatorname{div} ndata) \operatorname{div} m \\
logicalStripeID &= blockNumber \operatorname{div} ndata
\end{aligned}
$$

A.1 RAID level 0

Data-Mapping-Function

$$
\begin{aligned}
row &= rowID \\
col &= stripeUnitID \\
stripeUnit &= stripeID
\end{aligned}
$$

Parity-Placement-Function

Not applicable.

A.2 RAID level 4

Data-Mapping-Function

$$\begin{aligned} row &= rowID \\ col &= stripeUnitID \\ stripeUnit &= stripeID \end{aligned}$$

Parity-Placement-Function

$$\begin{aligned} row &= rowID \\ col &= n-1 \\ stripeUnit &= stripeID \end{aligned}$$

A.3 Right-Asymmetric

Data-Mapping-Function

$$\begin{aligned} row &= rowID \\ col &= \begin{cases} stripeUnitID, & \text{if } stripeID \bmod n > stripeUnitID \\ stripeUnitID + 1, & \text{otherwise} \end{cases} \\ stripeUnit &= stripeID \end{aligned}$$

Parity-Placement-Function

$$\begin{aligned} row &= rowID \\ col &= stripeID \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

A.4 Left-Asymmetric

Data-Mapping-Function

$$\begin{aligned} row &= rowID \\ col &= \begin{cases} stripeUnitID, & \text{if } (-stripeID - 1) \bmod n > stripeUnitID \\ stripeUnitID + 1, & \text{otherwise} \end{cases} \\ stripeUnit &= stripeID \end{aligned}$$

Parity-Placement-Function

$$\begin{aligned} row &= rowID \\ col &= (-stripeID - 1) \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

A.5 Right-Symmetric

Data-Mapping-Function

$$\begin{aligned} row &= rowID \\ col &= (stripeUnitID + stripeID + 1) \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

Parity-Placement-Function

$$\begin{aligned} row &= rowID \\ col &= stripeID \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

A.6 Left-Symmetric

Data-Mapping-Function

$$\begin{aligned} row &= rowID \\ col &= (stripeUnitID - stripeID) \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

Parity-Placement-Function

$$\begin{aligned} row &= rowID \\ col &= (-stripeID - 1) \bmod n \\ stripeUnit &= stripeID \end{aligned}$$

A.7 Extended-Left-Symmetric

We find it convenient to specify the data-mapping-function for the extended-left-symmetric placement in terms or the mapping that would have been obtained with the RAID level 0 placement. Thus,

$$\begin{aligned} row' &= (blockNumber \operatorname{div} n) \bmod m \\ col' &= blockNumber \bmod n \\ stripeUnit' &= (blockNumber \operatorname{div} n) \operatorname{div} m \end{aligned}$$

Data-Mapping-Function

$$
\begin{aligned}
row &= row' \\
col &= col' \\
coladj &= \begin{cases} 1, & \text{if } \exists i \text{ s.t. } 0 \le i \le stripeUnit' \bmod (n-1) \text{ and } (-m \times i - row' - 1) \bmod n = col' \\ 0, & \text{otherwise} \end{cases} \\
stripeUnit &= stripeUnit' + stripeUnit' \operatorname{div} ndata + coladj
\end{aligned}
$$

Parity-Placement-Function

$$
\begin{aligned}
row &= rowID \\
col &= (-logicalStripeID - 1) \bmod n \\
stripeUnit &= stripeID
\end{aligned}
$$

A.8 Flat-Left-Symmetric

$$
\begin{aligned}
row' &= (blockNumber \operatorname{div} n) \bmod m \\
col' &= blockNumber \bmod n \\
stripeUnit' &= (blockNumber \operatorname{div} n) \operatorname{div} m
\end{aligned}
$$

Data-Mapping-Function

$$
\begin{aligned}
row &= row' \\
col &= col' \\
stripeUnit &= stripeUnit' + stripeUnit' \operatorname{div} ndata
\end{aligned}
$$

Parity-Placement-Function

$$
\begin{aligned}
row &= rowID \\
col &= (-logicalStripeID - 1) \bmod n \\
stripeUnit &= (stripeID \operatorname{div} n) \times n + n - 1
\end{aligned}
$$